

Contents

Machine Learning Server Documentation

Overview

[What's happening to Machine Learning Server?](#)

[About Machine Learning Server](#)

[Release notes](#)

Tutorials

[PySpark and revoscalepy interoperability](#)

[Explore R-to-RevoScaleR](#)

Concepts

[Compute context](#)

[Distributed computing](#)

[Web services](#)

[Pre-trained models](#)

[MicrosoftML](#)

[RevoScaleR](#)

How-to guides

[Introduction](#)

[Data & models](#)

[Python](#)

[Use revoscalepy on Spark](#)

[R](#)

[Set a compute context](#)

[Data access and manipulation](#)

[Data transformations](#)

[XDF files](#)

[Import text data](#)

[Import SQL Server data](#)

[Import ODBC data](#)

[Import HDFS files](#)

- [Use data source objects](#)
- [Transform & subset data](#)
- [Sort data](#)
- [Split data](#)
- [Merge data](#)
- [Summarization](#)
 - [Statistical summaries](#)
 - [Crosstabs](#)
- [Visualization](#)
- [Data modeling](#)
 - [Data modeling overview](#)
 - [Linear regression](#)
 - [Logistic regression](#)
 - [Generalized linear](#)
 - [Decision tree](#)
 - [Decision forest](#)
 - [Estimating with stochastic gradient boosting](#)
 - [Naive Bayes Classification](#)
 - [Clustering Classification](#)
 - [Correlate data](#)
- [Use RevoScaleR on Hadoop](#)
 - [Spark](#)
 - [MapReduce](#)
 - [RevoScaleR functions for distributed analysis](#)
 - [Background jobs](#)
 - [Parallel jobs using rxExec](#)
 - [Enforce a YARN queue assignment](#)
- [Choose a MicrosoftML algorithm](#)
- [Operationalize models & code](#)
 - [Python](#)
 - [Authenticate in Python](#)
 - [Deploy & manage web services](#)

- Consume services synchronously
- Consume services asynchronously
- Create and manage session pools

R

- Deploy & manage web services
- Consume services synchronously
- Consume services asynchronously
- Create and manage session pools

APIs

- About the REST APIs
- Integrate services into apps
- Manage access tokens

Run R code remotely

- Connect to remote server
- Execute code remotely
- Advanced R development
 - Write custom chunking algorithms
 - Write custom analyses for large data sets
 - Convert model objects for use with PMML
 - Manage threads
 - Parallel loops using foreach
 - Parallel execution using doRSR
 - Parallel algorithms with PemaR

Reference

- Python packages
 - Python package overview
 - azureml-model-management-sdk >
 - microsoftml >
 - revoscalepy
 - Package overview
 - rx_btrees
 - rx_cancel_job

rx_cleanup_jobs
rx_create_col_info
RxDataSource
rx_data_step
rx_delete_object
rx_dforest
rx_dtree
rx_exec
rx_exec_by
rx_get_compute_context
rx_get_info
rx_get_job_info
rx_get_job_output
rx_get_job_results
rx_get_jobs
rx_get_job_status
rx_get_partitions
rx_get_pyspark_connection
rx_get_var_info
rx_get_var_names
rx_import
RxInSqlServer
rx_hadoop_command
rx_hadoop_copy_from_local
rx_hadoop_copy_to_local
rx_hadoop_copy
rx_hadoop_file_exists
rx_hadoop_list_files
rx_hadoop_make_dir
rx_hadoop_move
rx_hadoop_remove_dir
rx_hadoop_remove

RxHdfsFileSystem
RxHiveData
rx_lin_mod
rx_list_keys
RxLocalSeq
rx_logit
RxMissingValues
RxNativeFileSystem
RxOdbcData
RxOptions
RxOrcData
RxParquetData
rx_partition
rx_predict
rx_predict_default
rx_predict_rx_dtrees
rx_predict_rx_dforest
rx_privacy_control
rx_read_object
rx_read_xdf
RxRemoteComputeContext
RxRemoteJob
RxRemoteJobStatus
rx_set_compute_context
rx_set_var_info
rx_serialize_model
RxSpark
RxSparkData
RxSparkDataFrame
rx_spark_cache_data
rx_spark_connect
rx_spark_disconnect

[rx_spark_list_data](#)
[rx_spark_remove_data](#)
[RxSqlServerData](#)
[rx_summary](#)
[RxTextData](#)
[rx_wait_for_job](#)
[rx_write_object](#)
[RxXdfData](#)

R packages

[R package overview](#)
[MicrosoftML >](#)
[mrsdeploy >](#)
[olapR >](#)
[RevoIOQ >](#)
[RevoPemaR >](#)
[RevoScaleR](#)

Package overview

Data sets

[AirlineData87to08](#)
[AirlineDemoSmall](#)
[AirOnTime87to12](#)
[CensusUS5Pct2000](#)
[CensusWorkers](#)
[claims](#)
[Kyphosis](#)
[mortDefaultSmall](#)

Functions

[as.gbm](#)
[as.glm](#)
[as.kmeans](#)
[as.lm](#)
[as.naiveBayes](#)

as.randomForest
as.rpart
as.xtabs
prune.rxDTree
rxAddInheritance
rxBTrees
rxCancelJob
rxChiSquaredTest
rxCleanup
rxCompareContexts
rxCompressXdf
RxComputeContext-class
RxComputeContext
rxCovCor
rxCovRegression
rxCreateCollInfo
rxCrossTabs
rxCube
RxDataSource-class
RxDataSource
rxDataStep
rxDForest
rxDForestUtils
RxDistributedHpa-class
rxDistributeJob
rxDTree
rxDTreeBestCp
rxElemArg
rxExec
rxExecBy
rxExecByPartition
rxExecuteSQLDDL

[rxExpression](#)
[rxFactors](#)
[RxFileData-class](#)
[RxFileSystem](#)
[rxFindFileInPath](#)
[rxFindPackage](#)
[RxForeachDoPar-class](#)
[RxForeachDoPar](#)
[rxFormula](#)
[rxGetAvailableNodes](#)
[rxGetEnableThreadPool](#)
[rxGetFuzzyDist](#)
[rxGetFuzzyKeys](#)
[rxGetInfoXdf](#)
[rxGetJobInfo](#)
[rxGetJobOutput](#)
[rxGetJobResults](#)
[rxGetJobs](#)
[rxGetNodeInfo](#)
[rxGetPartitions](#)
[rxGetSparklyrConnection](#)
[rxGetVarInfo](#)
[rxGetVarInfoXdf](#)
[rxGetVarNames](#)
[rxGLM](#)
[rxHadoopCommand](#)
[RxHadoopMR-class](#)
[RxHadoopMR](#)
[rxHdfsConnect](#)
[RxHdfsFileSystem](#)
[rxHistogram](#)
[RxHPCServer-class](#)

[rxImport](#)
[RxInSqlServer-class](#)
[RxInSqlServer](#)
[rxInstalledPackages](#)
[rxInstallPackages](#)
[rxKmeans](#)
[rxLaunchClusterTaskManager](#)
[rxLinePlot](#)
[rxLinMod](#)
[RxLocalParallel-class](#)
[RxLocalParallel](#)
[RxLocalSeq-class](#)
[RxLocalSeq](#)
[rxLocateFile](#)
[rxLogit](#)
[rxLorenz](#)
[rxMakeRNodeNames](#)
[rxMarginals](#)
[rxMergeXdf](#)
[rxMultiTest](#)
[rxNaiveBayes](#)
[RxNativeFileSystem](#)
[rxNew](#)
[rxOAuthParameters](#)
[RxOdbcData-class](#)
[RxOdbcData](#)
[rxOpen-methods](#)
[rxOptions](#)
[rxPackage](#)
[rxPairwiseCrosstab](#)
[rxPingNodes](#)
[rxPartition](#)

[rxPredict](#)
[rxPredict.rxDForest](#)
[rxPredict.rxDTree](#)
[rxPredict.rxNaiveBayes](#)
[rxPrivacyControl](#)
[rxQuantile](#)
[rxReadXdf \(deprecated\)](#)
[rxRealtimeScoring](#)
[rxRemoteCall](#)
[rxRemoteFilePath](#)
[rxRemoteGetId](#)
[rxRemoteHadoopMRCall](#)
[rxRemovePackages](#)
[rxResultsDF](#)
[rxRiskRatio](#)
[rxRng](#)
[rxRoc](#)
[RxSasData-class](#)
[RxSasData](#)
[rxSerializeModel](#)
[rxSetComputeContext](#)
[rxSetFileSystem](#)
[rxSetInfo](#)
[rxSetVarInfoXdf](#)
[rxSortXdf](#)
[RxSpark-class](#)
[RxSpark](#)
[rxSparkCacheData](#)
[RxSparkData](#)
[rxSparkDataOps](#)
[rxSplitXdf](#)
[RxSpssData-class](#)

[RxSpssData](#)
[rxSqlLibPaths](#)
[RxSqlServerData-class](#)
[RxSqlServerData](#)
[rxSqlServerDropTable](#)
[rxStepControl](#)
[rxStepPlot](#)
[rxStopEngine](#)
[rxSummary](#)
[rxSyncPackages](#)
[RxTeradata-class](#)
[RxTeradata](#)
[rxTeradataSql](#)
[RxTextData-class](#)
[RxTextData](#)
[rxTextToXdf](#)
[rxTlcBridge](#)
[rxTransform](#)
[rxTweedie](#)
[rxUnserializeModel](#)
[rxWaitForJob](#)
[rxWriteObject](#)
[RxXdfData-class](#)
[RxXdfData](#)
[rxXdfFileName](#)
[rxXdfToText](#)

[Base R-to-RevoScaleR comparison](#)
[Defunct functions](#)
[Deprecated functions](#)
[Hadoop functions](#)
[SQL Server functions](#)
[Teradata functions](#)

[RevoUtils >](#)

[sqlrutils >](#)

[Resources](#)

[Support timeline](#)

[Known issues](#)

[Opt out of usage data collection](#)

[Additional resources](#)

What's happening to Machine Learning Server?

7/12/2022 • 7 minutes to read • [Edit Online](#)

IMPORTANT

The support for Microsoft Machine Learning Server 9.4.7 will end on July 1, 2022. For more information, see [Support timeline for Microsoft R Server & Machine Learning Server](#).

This article explains options for replacing the functionality of Machine Learning Server and migration strategies for code and data to other platforms.

Machine Learning Server is enterprise software for data science, providing R and Python interpreters, base distributions of R and Python, additional libraries from Microsoft, and operationalization capability.

Cloud analytics options

Many workloads from Machine Learning Server can also be uploaded to the Azure platform. Data *born in the cloud* (originated in cloud-based applications) are prime candidates for these technologies, and data movement services can migrate large-scale data securely and quickly. For more on data movement options, see [Choose an Azure solution for data transfer](#).

Azure has systems and certifications allowing secure data and data processing in a variety of tools. More information on these certifications is located at the [Microsoft Trust Center resource](#).

Azure Synapse Analytics

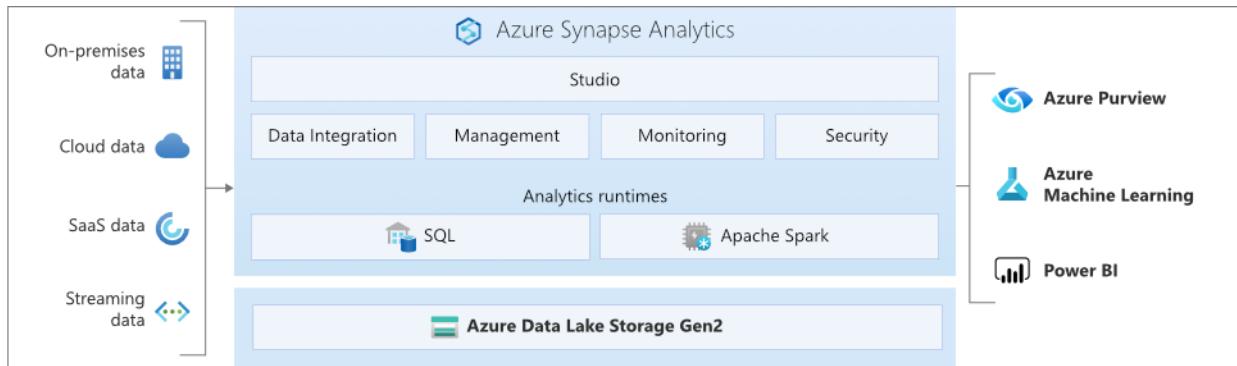
[Azure Synapse Analytics](#) is an enterprise analytics service that accelerates time to insight across data warehouses and big data systems, using distributed processing and data constructs. Azure Synapse Analytics brings together SQL technologies used in enterprise data warehousing, Spark technologies used for big data, pipelines for data integration and ETL/ELT, and deep integration with other Azure services such as [Power BI](#), [Azure Cosmos DB](#), and [Azure Machine Learning](#).

Use Azure Synapse Analytics as a replacement for Machine Learning Server when you need to:

- Leverage both serverless and dedicated resource models. For predictable performance and cost, create dedicated SQL pools to reserve processing power for data stored in SQL tables.
- Process unplanned or *burst* workloads, access an always-available, serverless SQL endpoint.
- Use built-in streaming capabilities to land data from cloud data sources into SQL tables.
- Integrate AI with SQL by using machine learning models to score data using the T-SQL PREDICT function.
- Leverage ML models with SparkML algorithms and Azure Machine Learning integration for Apache Spark 2.4 supported for Linux Foundation Delta Lake.
- Use a simplified resource model that frees you from having to worry about managing clusters.
- Process data that requires fast Spark start-up and aggressive autoscaling.
- Process data using .NET for Spark allowing you to reuse your C# expertise and existing .NET code within a Spark application.
- Work with tables defined on files in the data lake are seamlessly consumed by either Spark or Hive.
- Use SQL with Spark to directly explore and analyze Parquet, CSV, TSV, and JSON files stored in a data lake.
- Enable fast, scalable data loading between SQL and Spark databases.
- Ingest data from 90+ data sources.
- Enable “Code-Free” ETL with Data flow activities.

- Orchestrate notebooks, Spark jobs, stored procedures, SQL scripts, and more.
- Monitor resources, usage, and users across SQL and Spark.
- Use Role-based access control to simplify access to analytics resources.
- Write SQL or Spark code and integrate with enterprise CI/CD processes.

The architecture of Azure Synapse Analytics is as follows:



For more information, see [Machine Learning capabilities in Azure Synapse Analytics](#).

Azure SQL Managed Instance Machine Learning Services

[Machine Learning Services in Azure SQL Managed Instance](#) provides in-database machine learning, supporting both Python and R scripts. The feature includes Microsoft Python and R packages for high-performance predictive analytics and machine learning. The relational data can be used in scripts through stored procedures, T-SQL script containing Python or R statements, or Python or R code containing T-SQL.

Use Azure SQL Managed Instance Machine Learning Services as a replacement for Machine Learning Server when you need to:

- Run R and Python scripts to do data preparation and general purpose data processing.
- Train machine learning models in database.
- Deploy your models and scripts into production in stored procedures.

For more information, see [Machine Learning Services in Azure SQL Managed Instance](#).

Azure Machine Learning

[Azure Machine Learning](#) is a cloud-based service that can be used for any kind of machine learning, from classical machine learning to deep learning, supervised, and unsupervised learning. Whether you prefer to write Python or R code with the SDK or work with no-code/low-code options in the studio, you can build, train, and track machine learning and deep-learning models in an Azure Machine Learning Workspace. With Azure Machine Learning you can start training on your local machine and then scale out to the cloud. The service also interoperates with popular deep learning and reinforcement open-source tools such as PyTorch, TensorFlow, scikit-learn, and Ray RLlib.

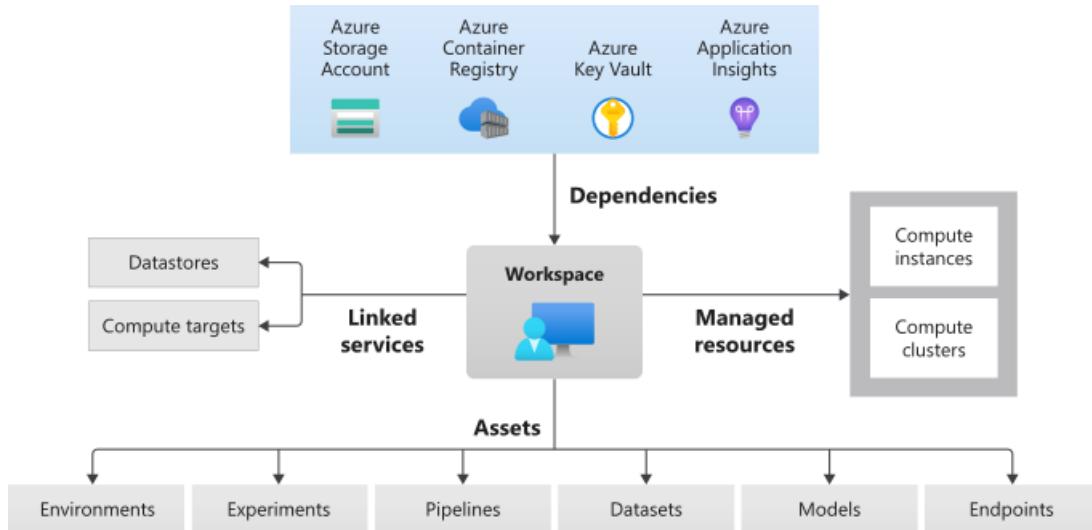
Use Azure Machine Learning as a replacement for Machine Learning Server when you need:

- A designer-based web environment for Machine Learning: drag-n-drop modules to build your experiments and then deploy pipelines in a low-code environment.
- Jupyter notebooks: use our example notebooks or create your own notebooks to leverage our SDK for Python samples for your machine learning.
- R scripts or notebooks in which you use the SDK for R to write your own code or use the R modules in the designer.
- The “Many Models Solution Accelerator” which builds on Azure Machine Learning and enables you to train, operate, and manage hundreds or even thousands of machine learning models.
- Machine learning extensions for Visual Studio Code (preview) provides you with a full-featured development

environment for building and managing your machine learning projects.

- A Machine learning Command-Line Interface (CLI), Azure Machine Learning includes an Azure CLI extension that provides commands for managing with Azure Machine Learning resources from the command line.
- Integration with open-source frameworks such as PyTorch, TensorFlow, and scikit-learn and many more for training, deploying, and managing the end-to-end machine learning process.
- Reinforcement learning with Ray RLlib.
- MLflow to track metrics and deploy models or Kubeflow to build end-to-end workflow pipelines.

The architecture of a Azure Machine Learning deployment is as follows:



For more information, see [What is Azure Machine Learning?](#)

Other cloud analytics options

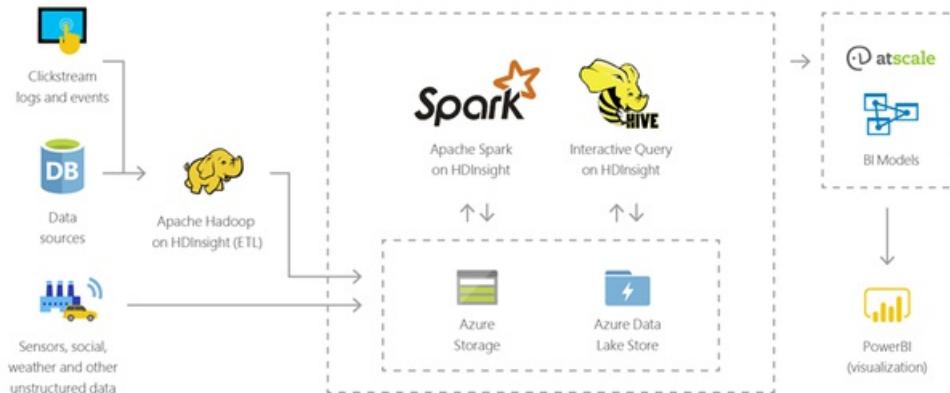
Other options include [Azure HDInsight](#), [Azure Databricks](#), and [Azure Cosmos DB](#).

Azure HDInsight

[Azure HDInsight](#) is a cloud distribution of Hadoop components. Azure HDInsight is a managed, full-spectrum, open-source analytics service in the cloud for enterprises. HDInsight allows you to use open-source frameworks such as Hadoop, Apache Spark, Apache Hive, LLAP, Apache Kafka, Apache Storm, R, and more.

HDInsight includes specific cluster types and cluster customization capabilities, such as the capability to add components, utilities, and languages. For more information, see [Cluster types in HDInsight](#).

The architecture of a Azure HDInsight deployment is as follows:



For more information, see [What is Azure HDInsight?](#)

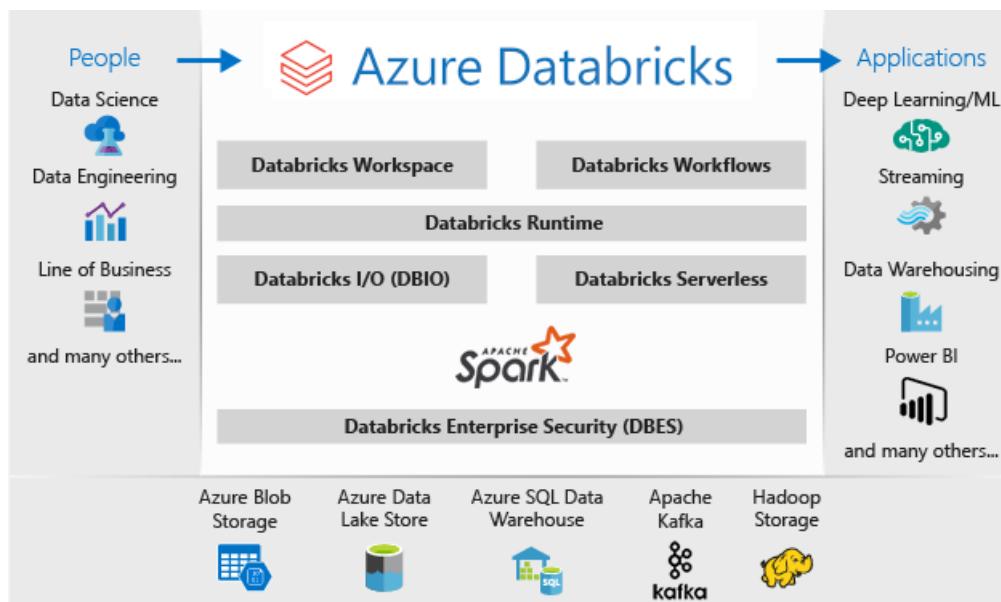
Azure Databricks

[Azure Databricks](#) is a data analytics platform optimized for the Azure cloud services platform. Azure Databricks offers two environments for developing data intensive applications: Azure Databricks SQL Analytics and Azure Databricks Workspace.

Azure Databricks SQL Analytics provides an easy-to-use platform for analysts who want to run SQL queries on their data lake, create multiple visualization types to explore query results from different perspectives, and build and share dashboards.

Azure Databricks Workspace provides an interactive workspace that enables collaboration between data engineers, data scientists, and machine learning engineers. For a big data pipeline, the data (raw or structured) is ingested into Azure through Azure Data Factory in batches, or streamed near real-time using Apache Kafka, Event Hub, or IoT Hub. This data lands in a data lake for long term persisted storage, in Azure Blob Storage or Azure Data Lake Storage. As part of your analytics workflow, use Azure Databricks to read data from multiple data sources and turn it into breakthrough insights using Spark.

The architecture of a Microsoft Azure Databricks deployment is as follows:



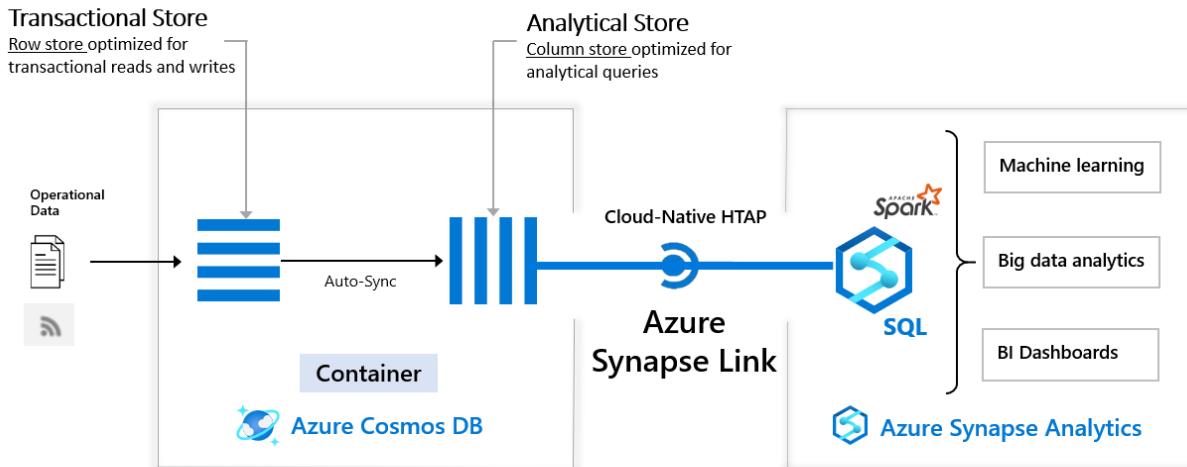
For more information, see [What is Azure Databricks?](#)

Azure Cosmos DB

[Azure Cosmos DB](#) is a fully managed NoSQL database for modern app development. Single-digit millisecond response times, and automatic and instant scalability, guarantee speed at any scale. Business continuity is assured with SLA-backed availability and enterprise-grade security. App development is faster and more productive thanks to turnkey multi region data distribution anywhere in the world, open-source APIs and SDKs for popular languages.

As a fully managed service, Azure Cosmos DB handles database administration with automatic management, updates and patching. It also handles capacity management with serverless and automatic scaling options that respond to application needs to match capacity with demand.

The architecture of a Microsoft Azure CosmosDB deployment is as follows:



For more information on using Notebooks with Python or C# in CosmosDB Deployments, see [Built-in Jupyter Notebooks support in Azure Cosmos DB](#).

For more information, see the [Azure Cosmos DB documentation](#).

See Also

- [Support Timeline for Microsoft R Server & Machine Learning Server](#)
- [Archived articles](#)

What is Machine Learning Server

7/12/2022 • 2 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Microsoft Machine Learning Server 9.4.7 is enterprise software for data science, providing R and Python interpreters, base distributions of R and Python, additional high-performance libraries from Microsoft, and an [operationalization capability](#) for advanced deployment scenarios. Solutions that you develop in R or Python can be deployed as a web service for direct access or as an upstream component to other solutions.

R support is built on a legacy of Microsoft R Server 9.x and Revolution R Enterprise products. Python support was added in the 9.2.1 release.

Machine Learning Server runs [on-premises and in the cloud](#), on a variety of operating systems, and can run in a distributed mode if you want to isolate functions on different computers (specifically, as dedicated web and compute nodes). [Web services](#) are hosted on a server grid on-premises or in the cloud and can be integrated with line-of-business applications. Additionally, Machine Learning Server integrates seamlessly with [Active Directory](#) and [Azure Active Directory](#) and includes [role-based access control](#) to satisfy security and compliance needs of your enterprise. The ability to deploy to an elastic grid lets you scale seamlessly with the needs of your business, both for batch and real-time scoring.

On two data platforms, SQL Server and Apache Spark (on HDFS), the R and Python libraries from Microsoft can compute and analyze data locally, returning just the results, without the need to pull data across the network. This capability is called *remote compute context*. It requires the R and Python libraries and interpreters from Microsoft on both client and server systems, but the client versions are free of charge.

Key features of Machine Learning Server

The following features are included in Machine Learning Server. For feature descriptions in this release, see [What's New in Machine Learning Server](#).

FEATURE CATEGORY	DESCRIPTION
R_SERVER	R packages for solutions written in R, with an open-source distribution of Microsoft R Open and run-time infrastructure for script execution.
PYTHON_SERVER	Python modules for solutions written in Python, with an open-source distribution of Anaconda and run-time infrastructure for script execution.
Pre-trained models	For visual analysis and text sentiment analysis, ready to score data you provide.
Deploy and consume	Operationalize your server and deploy solutions as a web service.

FEATURE CATEGORY	DESCRIPTION
Remote execution	Start remote sessions on a Machine Learning Server on your network from your client workstation.
scale out on premises	Clustered topologies for Apache Spark on Hadoop , and Windows or Linux using the operationalization capability built into Machine Learning Server.

Next steps

[Install Machine Learning Server](#) on a supported platform.

Choose a [quickstart](#) to test-drive capabilities in 10 minutes or less. Move on to tutorials for in-depth exploration.

NOTE

You can use any R or Python IDE to write code using libraries from Machine Learning Server. Microsoft offers [Python for Visual Studio](#) and [R for Visual Studio](#). To use Jupyter Notebook, see [How to configure Jupyter Notebooks](#).

See also

- [What's new in Machine Learning Server](#)
- [Learning Resources](#)

Machine Learning Server 9.4 Release Notes

7/12/2022 • 4 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Machine Learning Server provides powerful R and Python function libraries for data science and machine learning on small-to-massive data sets, in parallel on local or distributed systems, with modern algorithms for predictive analytics, supervised learning, and data mining.

Functionality is delivered through proprietary R and Python packages, internal computational engines built on open-source R and Python, tools, solutions, and samples.

In this article, learn about the capabilities introduced in the latest packages and tools. If you develop in R, you might also want to review feature announcements from recent past releases.

NOTE

For updates to R Client, see [What's New for Microsoft R Client](#). For known issues and workarounds, see [Known issues in 9.4](#).

Announced in 9.4

9.4 updates the R and Python engines and adds support for Spark 2.4 and CDH 6.1. Also, on CDH customers can install either R or Python or both.

Announced in 9.3

New capabilities introduced in 9.3 are listed in the following table.

FEATURE	AREA	DETAILS
Administration command line interface	Operationalization	<p>Refactored tooling for Machine Learning Server configuration. The new command-line interface is similar to Azure CLIs and offers full parity with the previous utility.</p> <p>Use the tool to enable web service deployment, web and compute node designations, and remote execution (R only). You can also manage ports, nodes, credentials; run diagnostic reports; and test the capacity and throughput of web services you create.</p>

FEATURE	AREA	DETAILS
Dedicated session pools	Operationalization	<p>You can construct a dedicated session pool for a specific web service to provide ready-to-use connections with preloaded dependencies for fast access to production code. This capability is in addition to the generic session pools that you can establish server-wide as a shared resource for all web services.</p> <p>Configure in R Configure in Python.</p> <p>For R script, the <code>mrsdeploy</code> function library provides three new functions for managing dedicated sessions: <code>configureServicePool</code>, <code>getPoolStatus</code>, <code>deleteServicePool</code>.</p> <p>For Python, the <code>azureml-model-management-sdk</code> provides the following methods in the <code>mlserver</code> class: <code>create_or_update_service_pool</code>, <code>delete_service_pool</code>, <code>get_service_pool_status</code>.</p>
CDH 5.12	Supported platforms	Version 5.12 of Cloudera distribution of Apache Hadoop (CDH) is now supported for Machine Learning Server for Hadoop .

Announced in 9.2.1

The 9.2.1 release was the first release of Machine Learning Server - based on R Server - expanded with Python libraries for developers and analysts who code in Python.

The following table summarizes the Python and R features that were introduced in the 9.2.1 release. For more information, see [release announcement for Machine Learning Server 9.2.1](#).

FEATURE	LANGUAGE	DETAILS
<code>revoscalepy</code>	Python	The first release of this library, used for distributed computing, local compute context, remote compute context for SQL Server and Spark 2.0-2.1 over the Hadoop Distributed File System (HDFS), and high-performance algorithms for Python. This library is similar to RevoScaleR for R.
<code>microsoftml</code>	Python	The first release of this library, used for machine learning algorithms and data mining. This library is similar to MicrosoftML for R.)
Pre-trained models	Python	Ready-to-use machine learning models for image classification and sentiment detection articulated in Python.

FEATURE	LANGUAGE	DETAILS
azureml-model-management-sdk library	Python	The first release of this library, used to programmatically build web services encapsulating your Python script.
Standard web service support	Python	Contains Python code, models, and model assets. Accepts specific inputs and provides specific outputs for integration with other services and applications.
Real-time web service support	Python	A fully encapsulated web service with no inputs or outputs (operates on dataframes).
Real-time model scoring	R	Now supported on Linux.
Role-based access control (RBAC)	Both	RBAC was extended with a new explicit Reader role.
Administration utility update	Both	The utility simplifies registration of compute nodes with web nodes.

Python development

In Machine Learning Server, Python libraries used in script execute locally, or remotely in either Spark over Hadoop Distributed File System (HDFS) or in a SQL Server compute context. Libraries are built on Anaconda 4.2 over Python 3.5. You can run any 3.5-compatible library on a Python interpreter included in Machine Learning Server.

NOTE

Remote execution is not available for Python scripts. For information about to do this in R, see [Remote execution in R](#).

R development

R function libraries are built on [Microsoft R Open \(MRO\)](#), Microsoft's distribution of open-source R 3.4.1.

The last several releases of R Server added substantial capability for R developers. To review recent additions to R functionality, see [feature announcements](#) for previous versions.

Previous versions

Visit [Feature announcements in R Server](#) version 9.1 and earlier, for descriptions of features added in recent past releases.

See Also

- [Welcome to Machine Learning Server](#)
- [Install Machine Learning Server on Windows](#)
- [Install Machine Learning Server on Linux](#)
- [Install Machine Learning Server on Hadoop](#)
- [Configure Machine Learning Server to operationalize your analytics](#)

Tutorial: PySpark and revoscalepy interoperability in Machine Learning Server

7/12/2022 • 3 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Applies to: Microsoft Learning Server 9.x

[PySpark](#) is Apache Spark's programmable interface for Python. The [revoscalepy](#) module is Machine Learning Server's Python library for predictive analytics at scale. In this tutorial, you learn how to create a logistic regression model using functions from both libraries.

- Import packages
- Connect to Spark using `revoscalepy.rx_spark_connect()`, specifying PySpark interop
- Use PySpark for basic data manipulation
- Use revoscalepy to build a logistic regression model

NOTE

The [revoscalepy](#) module provides functions for data sources and data manipulation. We are using PySpark in this tutorial to illustrate a basic technique for passing data objects between the two programming contexts.

Prerequisites

- A Hadoop cluster with Spark 2.0-2.1 with [Machine Learning Server for Hadoop](#)
- A Python IDE, such as Jupyter Notebooks, [Visual Studio for Python](#), or PyCharm.
- Sample data (AirlineSubsetCsv mentioned in the example) downloaded from our [sample data web site](#) to your Spark cluster.

NOTE

Jupyter Notebook users, update your notebook to include the MMLSPy kernel. Select this kernel in your Jupyter Notebook to use the interoperability feature.

Import the relevant packages

The following commands import the required libraries into the current session.

```
from pyspark import SparkContext  
from pyspark.sql import SparkSession  
from revoscalepy import *
```

Connect to Spark

Setting `interop = 'pyspark'` indicates that you want interoperability.

```
# with PySpark for this Spark session
cc = rx_spark_connect(interop='pyspark', reset=True)

# Get the PySpark context
sc = rx_get_pyspark_connection(cc)
spark = SparkSession(sc)
```

Data acquisition and manipulation

The sample data used in this tutorial is airline arrival and departure data, which you can store in a local file path.

```

# Read in the airline data into a data frame
airlineDF = spark.read.csv('<data source location like "file:///some-file-path/airline.csv">')

# Get a count on rows
airlineDF.count()

# Return the first 10 lines to get familiar with the data
airlineDF.take(10)

# Rename columns for readability
airlineTransformed = airlineDF.selectExpr('ARR_DEL15 as ArrDel15', \
'YEAR as Year', \
'MONTH as Month', \
'DAY_OF_MONTH as DayOfMonth', \
'DAY_OF_WEEK as DayOfWeek', \
'UNIQUE_CARRIER as Carrier', \
'ORIGIN_AIRPORT_ID as OriginAirportID', \
'DEST_AIRPORT_ID as DestAirportID', \
'FLOOR(CRS_DEP_TIME / 100) as CRSDepTime', \
'CRS_ARR_TIME as CRSArrTime')

# Break up the data set into train and test. We use training data for
# all years before 2012 to predict flight delays for Jan 2012
airlineTrainDF = airlineTransformed.filter('Year < 2012')
airlineTestDF = airlineTransformed.filter('(Year == 2012) AND (Month == 1)')

# Define column info for factors
column_info = {
    'ArrDel15': { 'type': 'numeric' },
    #'CRSDepTime': { 'type': 'integer' },
    'CRSDepTime': {
        'type': 'factor',
        'levels': ['0', '2', '3', '4', '5', '6', '7', '8', '9', '10',
                   '11', '12', '13', '14', '15', '16', '17', '18', '19', '20',
                   '21', '22', '23']
    },
    'CRSArrTime': { 'type': 'integer' },
    'Month': {
        'type': 'factor',
        'levels': ['1', '2']
    },
    'DayOfMonth': {
        'type': 'factor',
        'levels': ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10',
                   '11', '12', '13', '14', '15', '16', '17', '18', '19', '20',
                   '21', '22', '23', '24', '25', '26', '27', '28', '29', '30',
                   '31']
    },
    'DayOfWeek': {
        'type': 'factor',
        'levels': ['1', '2', '3', '4', '5', '6', '7']
    },
    #'Carrier': { 'type': 'factor' }
}

# Define a Spark data frame data source, required for passing to revoscalepy
trainDS = RxSparkDataFrame(airlineTrainDF, column_info=column_info)
testDS = RxSparkDataFrame(airlineTestDF, column_info=column_info)

```

Create the model

A logistic regression model requires a symbolic formula, specifying the dependent and independent variables, and a data set. You can output the results using the print function.

```
# Create the formula
formula = "ArrDel15 ~ DayOfMonth + DayOfWeek + CRSDepTime + CRSArrTime"

# Run a logistic regression to predict arrival delay
logitModel = rx_logit(formula, data = trainDS)

# Print the model summary to look at the co-efficients
print(logitModel.summary())
```

Next steps

This tutorial provides an introduction to a basic workflow using PySpark for data preparation and revoscalepy functions for logistic regression. For further exploration, review our [Python samples](#).

See also

- [microsoftml function reference](#)
- [revoscalepy function reference](#)

Basic R commands and RevoScaleR functions: 25 common examples

7/12/2022 • 20 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Applies to: Microsoft R Client, Machine Learning Server

If you are new to both R and Machine Learning Server, this tutorial introduces you to 25 (or so) commonly used R functions. In this tutorial, you learn how to load small data sets into R and perform simple computations. A key point to take away from this tutorial is that you can combine basic R commands and RevoScaleR functions in the same R script.

This tutorial starts with R commands before transitioning to RevoScaleR functions. If you already know R, skip ahead to [Explore RevoScaleR Functions](#).

NOTE

R Client and Machine Learning Server are interchangeable in terms of RevoScaleR as long as [data fits into memory and processing is single-threaded](#). If data size exceeds memory, we recommend pushing the [compute context](#) to Machine Learning Server.

Prerequisites

This is our simplest tutorial in terms of data and tools, but it's also expansive in its coverage of basic R and RevoScaleR functions. To complete the tasks, use the command-line tool **RGui.exe** on Windows or start the **Revo64** program on Linux.

- On Windows, go to \Program Files\Microsoft\R Client\R_SERVER\bin\x64 and double-click **Rgui.exe**.
- On Linux, at the command prompt, type **Revo64**.

The tutorial uses pre-installed sample data so once you have the software, there is nothing more to download or install.

The R command prompt is `>`. You can hand-type commands line by line, or copy-paste a multi-line command sequence.

R is case-sensitive. If you hand-type commands in this example, be sure to use the correct case. Windows users: the file paths in R take a forward slash delimiter (/), required even when the path is on the Windows file system.

Start with R

Because RevoScaleR is built on R, this tutorial begins with an exploration of common R commands.

Load data

R is an environment for analyzing data, so the natural starting point is to load some data. For small data sets, such as the following 20 measurements of the speed of light taken from the famous Michelson-Morley

experiment, the simplest approach uses R's `c` function to combine the data into a vector.

Type or copy the following script and paste it at the `>` prompt at the beginning of the command line:

```
c(850, 740, 900, 1070, 930, 850, 950, 980, 980, 880, 1000, 980, 930, 650, 760, 810, 1000, 1000, 960, 960)
```

When you type the closing parenthesis and press *Enter*, R responds as follows:

```
[1] 850 740 900 1070 930 850 950 980 980 880  
[11] 1000 980 930 650 760 810 1000 1000 960 960
```

This indicates that R has interpreted what you typed, created a vector with 20 elements, and returned that vector. But we have a problem. R hasn't saved what we typed. If we want to use this vector again (and that's the usual reason for creating a vector in the first place), we need to *assign* it. The R assignment operator has the suggestive form `<-` to indicate a value is being assigned to a name. You can use most combinations of letters, numbers, and periods to form names (but note that names can't begin with a number). Here we'll use

```
michelson :
```

```
michelson <- c(850, 740, 900, 1070, 930, 850, 950, 980, 980, 880,  
1000, 980, 930, 650, 760, 810, 1000, 1000, 960, 960)
```

R responds with a `>` prompt. Notice that the named vector is not automatically printed when it is assigned. However, you can view the vector by typing its name at the prompt:

```
> michelson
```

Output:

```
[1] 850 740 900 1070 930 850 950 980 980 880  
[11] 1000 980 930 650 760 810 1000 1000 960 960
```

The `c` function is useful for hand typing in small vectors such as you might find in textbook examples, and it is also useful for combining existing vectors. For example, if we discovered another five observations that extended the Michelson-Morley data, we could extend the vector using `c` as follows:

```
michelsonNew <- c(michelson, 850, 930, 940, 970, 870)  
michelsonNew
```

Output:

```
[1] 850 740 900 1070 930 850 950 980 980 880  
[11] 1000 980 930 650 760 810 1000 1000 960 960  
[21] 850 930 940 970 870
```

Generate random data

Often for testing purposes you want to use randomly generated data. R has a number of built-in distributions from which you can generate random numbers; two of the most commonly used are the normal and the uniform distributions. To obtain a set of numbers from a normal distribution, you use the `rnorm` function. This example generates 25 random numbers in a normal distribution.

```
normalDat <- rnorm(25)
normalDat
```

Output:

```
[1] -0.66184983 1.71895416 2.12166699
[4] 1.49715368 -0.03614058 1.23194518
[7] -0.06488077 1.06899373 -0.37696531
[10] 1.04318309 -0.38282188 0.29942160
[13] 0.67423976 -0.29281632 0.48805336
[16] 0.88280182 1.86274898 1.61172529
[19] 0.13547954 1.08808601 -1.26681476
[22] -0.19858329 0.13886578 -0.27933600
[25] 0.70891942
```

By default, the data are generated from a standard normal with mean 0 and standard deviation 1. You can use the `mean` and `sd` arguments to `rnorm` to specify a different normal distribution:

```
normalSat <- rnorm(25, mean=450, sd=100)
normalSat
```

Output:

```
[1] 373.3390 594.3363 534.4879 410.0630 307.2232
[6] 307.8008 417.1772 478.4570 521.9336 493.2416
[11] 414.8075 479.7721 423.8568 580.8690 451.5870
[16] 406.8826 488.2447 454.1125 444.0776 320.3576
[21] 236.3024 360.6385 511.2733 508.2971 449.4118
```

Similarly, you can use the `runif` function to generate random data from a uniform distribution:

```
uniformDat <- runif(25)
uniformDat
```

Output:

```
[1] 0.03105927 0.18295065 0.96637386 0.71535963
[5] 0.16081450 0.15216891 0.07346868 0.15047337
[9] 0.49408599 0.35582231 0.70424152 0.63671421
[13] 0.20865305 0.20167994 0.37511929 0.54082887
[17] 0.86681824 0.23792988 0.44364083 0.88482396
[21] 0.41863803 0.42392873 0.24800036 0.22084038
[25] 0.48285406
```

Generate a numeric sequence

The default uniform distribution is over the interval 0 to 1. You can specify alternatives by setting the `min` and `max` arguments:

```
uniformPerc <- runif(25, min=0, max=100)
uniformPerc
```

Output:

```
[1] 66.221400 12.270863 33.417174 21.985229  
[5] 92.767213 17.911602 1.935963 53.551991  
[9] 75.110760 22.436347 63.172258 95.977501  
[13] 79.317351 56.767608 89.416080 79.546495  
[17] 8.961152 49.315612 43.432128 68.871867  
[21] 73.598221 63.888835 35.261694 54.481692  
[25] 37.575176
```

Another commonly used vector is the `sequence`, a uniformly spaced run of numbers. For the common case of a run of integers, you can use the infix operator, `:` as follows:

```
1:10
```

Output:

```
[1] 1 2 3 4 5 6 7 8 9 10
```

For more general sequences, use the `seq` function:

```
seq(length = 11, from = 10, to = 30)
```

Output:

```
[1] 10 12 14 16 18 20 22 24 26 28 30
```

```
seq(from = 10,length = 20, by = 4)
```

Output:

```
[1] 10 14 18 22 26 30 34 38 42 46 50 54 58 62 66 70 74  
[18] 78 82 86
```

TIP

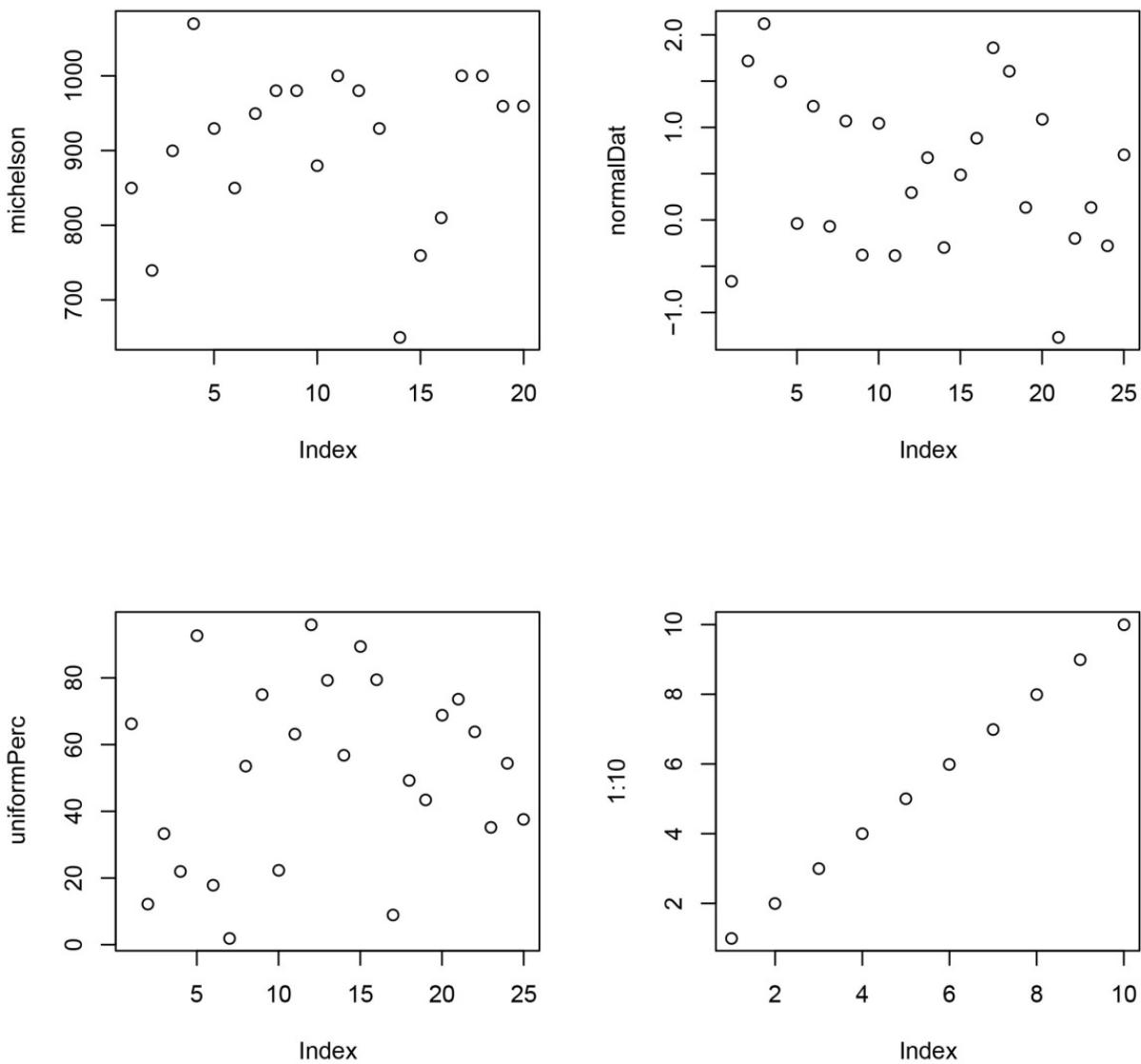
If you are working with big data, you'll still use vectors to manipulate parameters and information about your data, but you'll probably store the data in the RevoScaleR high-performance .xdf file format.

Exploratory Data Analysis

After you have some data, you will want to explore it graphically. For most small data sets, the place to begin is with the `plot` function, which provides a default graphical view of the data:

```
plot(michelson)  
plot(normalDat)  
plot(uniformPerc)  
plot(1:10)
```

For numeric vectors such as ours, the default view is a scatter plot of the observations against their index, resulting in the following plots:



For an exploration of the shape of the data, the usual tools are `stem` (to create a stemplot) and `hist` (to create a histogram):

```
stem(michelson)
```

Output:

```
The decimal point is 2 digit(s) to the right of the |
6 | 5
7 | 46
8 | 1558
9 | 033566888
10 | 0007
```

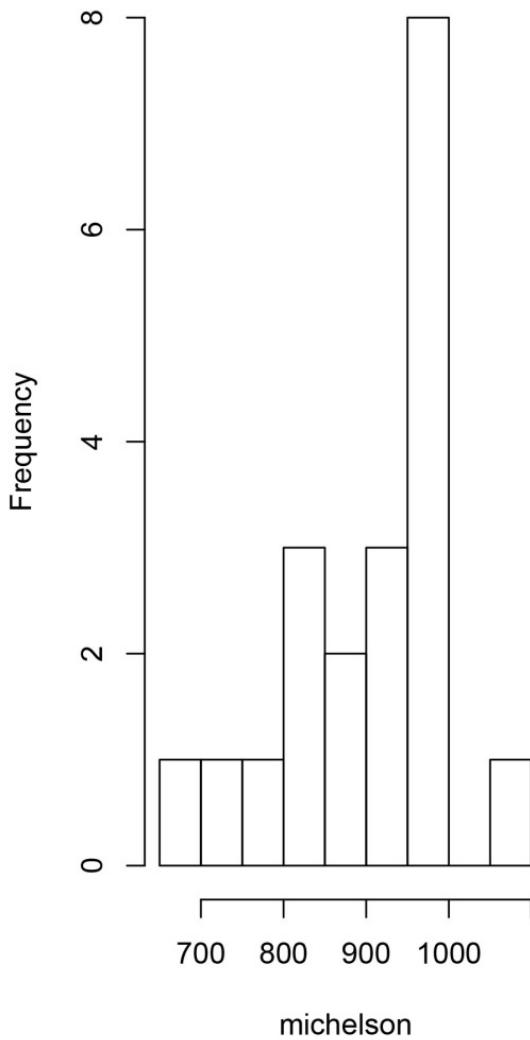
```
hist(michelson)
```

The resulting histogram is shown as the left plot following. We can make the histogram look more like the stemplot by specifying the `nclass` argument to `hist`:

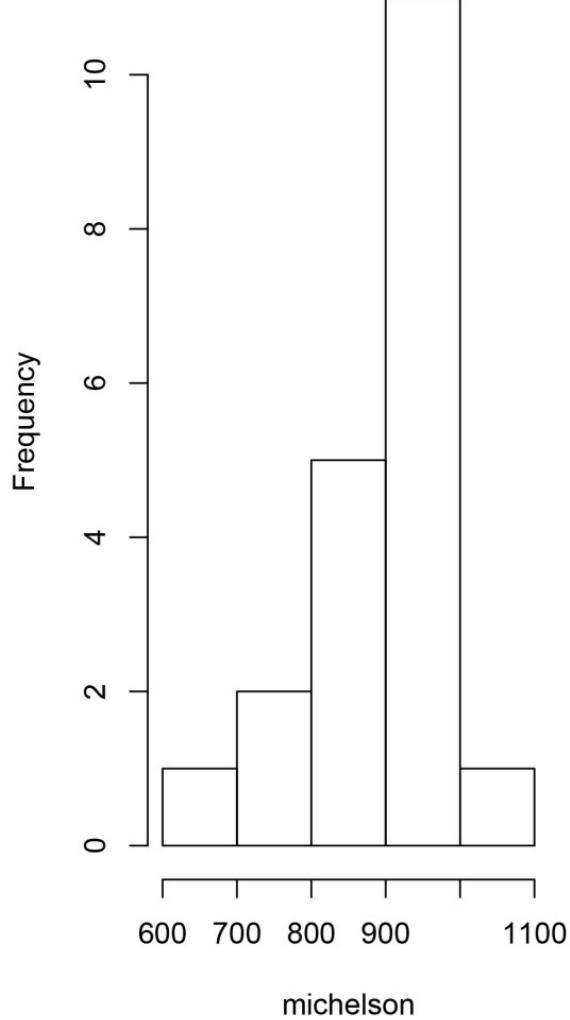
```
hist(michelson, nclass=5)
```

The resulting histogram is shown as the right plot in the figure following.

Histogram of michelson



Histogram of michelson

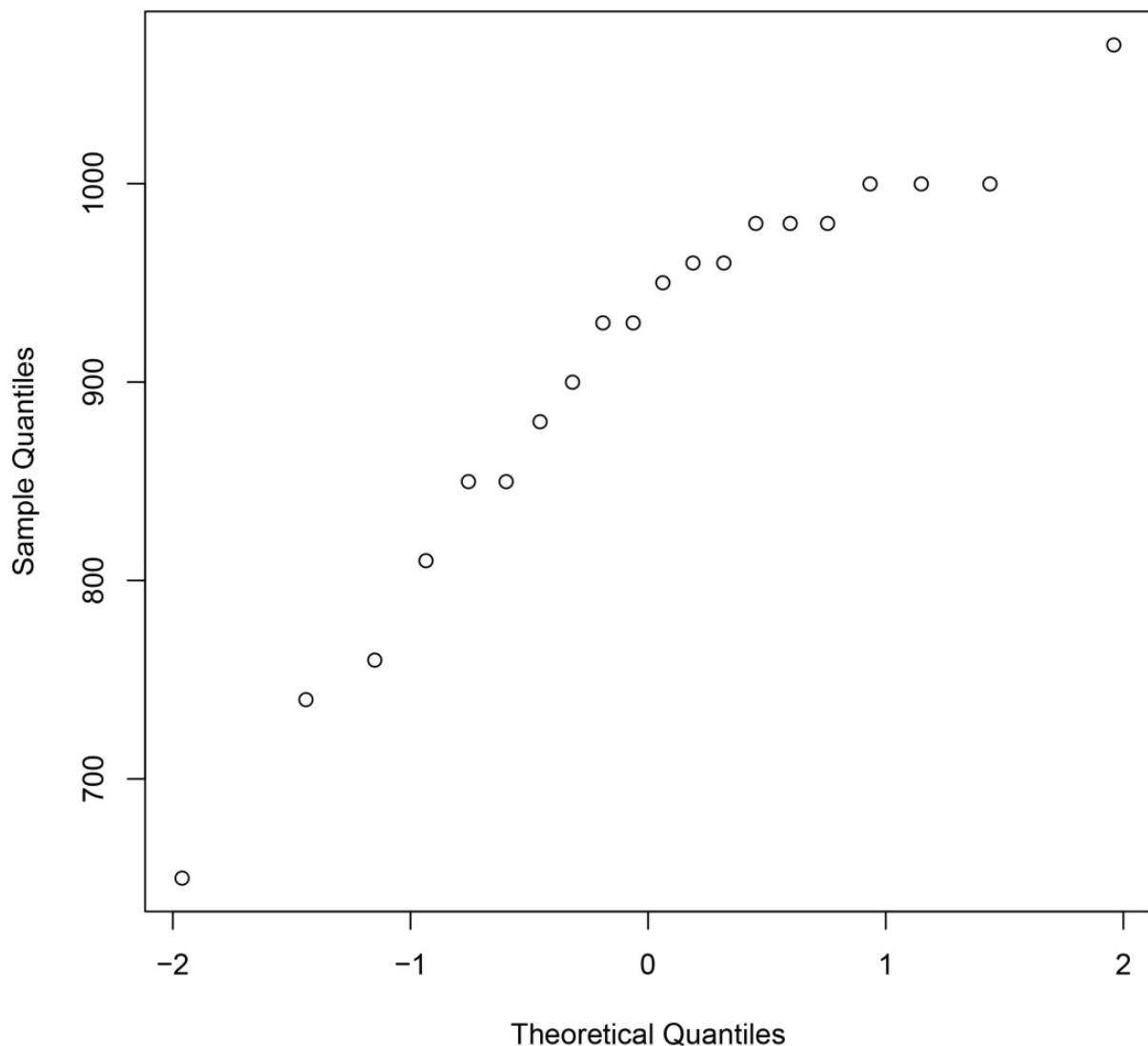


From the histogram and stemplot, it appears that the Michelson-Morley observations are not obviously normal. A normal Q-Q plot gives a graphical test of whether a data set is normal:

```
qqnorm(michelson)
```

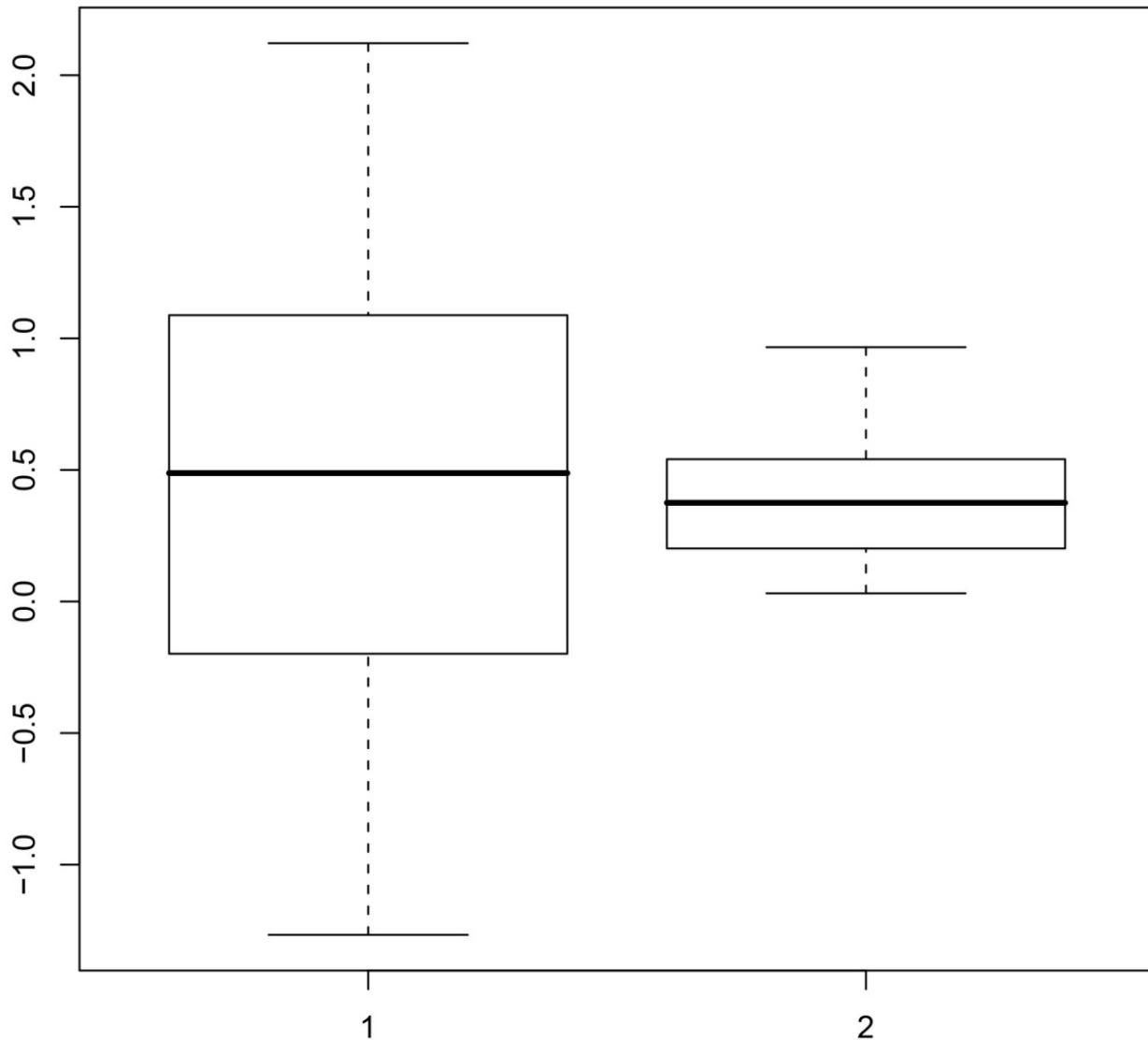
The decided bend in the resulting plot confirms the suspicion that the data are not normal.

Normal Q-Q Plot



Another useful exploratory plot, especially for comparing two distributions, is the boxplot:

```
boxplot(normalDat, uniformDat)
```



TIP

These plots are great if you have a small data set in memory. However, when working with big data, some plot types may not be informative when working directly with the data (for example, scatter plots can produce a large blob of ink) and others may be computational intensive (if sorting is required). A better alternative is the *rxHistogram* function in RevoScaleR that efficiently computes and renders histograms for large data sets. Additionally, RevoScaleR functions such as *rxCube* can provide summary information that is easily amenable to the impressive plotting capabilities provided by R packages.

Summary Statistics

While an informative graphic often gives the fullest description of a data set, numerical summaries provide a useful shorthand for describing certain features of the data. For example, estimators such as the mean and median help to locate the data set, and the standard deviation and variance measure the scale or spread of the data. R has a full set of summary statistics available:

```
> mean(michelson)
[1] 909
> median(michelson)
[1] 940
> sd(michelson)
[1] 104.9260
> var(michelson)
[1] 11009.47
```

The generic summary function provides a meaningful summary of a data set; for a numeric vector it provides the five-number summary plus the mean:

```
summary(michelson)
```

Output:

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
650	850	940	909	980	1070

TIP

The *rxSummary* function in RevoScaleR will efficiently compute summary statistics for a data frame in memory or a large data file stored on disk.

Multivariate Data Sets

In most disciplines, meaningful data sets have multiple variables, typically observations of various quantities and qualities of individual subjects. Such data sets are typically represented as tables in which the columns correspond to variables and the rows correspond to subjects, or cases. In R, such tables can be created as *data frame* objects. For example, at the 2008 All-Star Break, the Seattle Mariners had five players who met the minimum qualifications to be considered for a batting title (that is, at least 3.1 at bats per game played by the team). Their statistics are shown in the following table:

Player	Games	AB	R	H	2B	3B	HR	TB	RBI	BA	OBP	SLG	OPS
"I. Suzuki"	95	391	63	119	11	3	3	145	21	.304	.366	.371	.737
"J. Lopez"	92	379	46	113	26	1	5	156	48	.298	.318	.412	.729
"R. Ibanez"	95	370	41	101	26	1	11	162	55	.273	.338	.438	.776
"Y. Betancourt"	90	326	34	87	22	2	3	122	29	.267	.278	.374	.656
"A. Beltre"	92	352	46	91	16	0	16	155	46	.259	.329	.440	.769

Copy and paste the table into a text editor (such as Notepad on Windows, or emacs or vi on Unix type machines) and save the file as `msStats.txt` in the working directory returned by the `getwd` function, for example:

```
getwd()
```

Output:

```
[1] "/Users/joe"
```

TIP

On Windows, the working directory is probably a bin folder in program files, and by default you don't have permission to save the file at that location. Use `setwd("/Users/TEMP")` to change the working directory to /Users/TEMP and save the file.

You can then read the data into R using the `read.table` function. The argument `header=TRUE` specifies that the first line is a header of variable names:

```
msStats <- read.table("msStats.txt", header=TRUE)
msStats
```

Output:

```
Player Games AB R H X2B X3B HR TB RBI
1 I. Suzuki    95 391 63 119 11   3 3 145 21
2 J. Lopez     92 379 46 113 26   1 5 156 48
3 R. Ibanez    95 370 41 101 26   1 11 162 55
4 Y. Betancourt 90 326 34 87 22   2 3 122 29
5 A. Beltre    92 352 46 91 16   0 16 155 46

BA OBP SLG OPS
1 0.304 0.366 0.371 0.737
2 0.298 0.318 0.412 0.729
3 0.273 0.338 0.438 0.776
4 0.267 0.278 0.374 0.656
5 0.259 0.329 0.440 0.769
```

Notice how `read.table` changed the names of our original "2B" and "3B" columns to be valid R names; R names cannot begin with a numeral.

Use built-in datasets package

Most small R data sets in daily use are data frames. The built-in package, **datasets**, is a rich source of data frames for further experimentation.

To list the data sets in this package, use this command:

```
ls("package:datasets")
```

The output is an alphabetized list of data sets that are readily available for use in R functions. In the next section, we will use the built-in data set *attitude*, available through the **datasets** package.

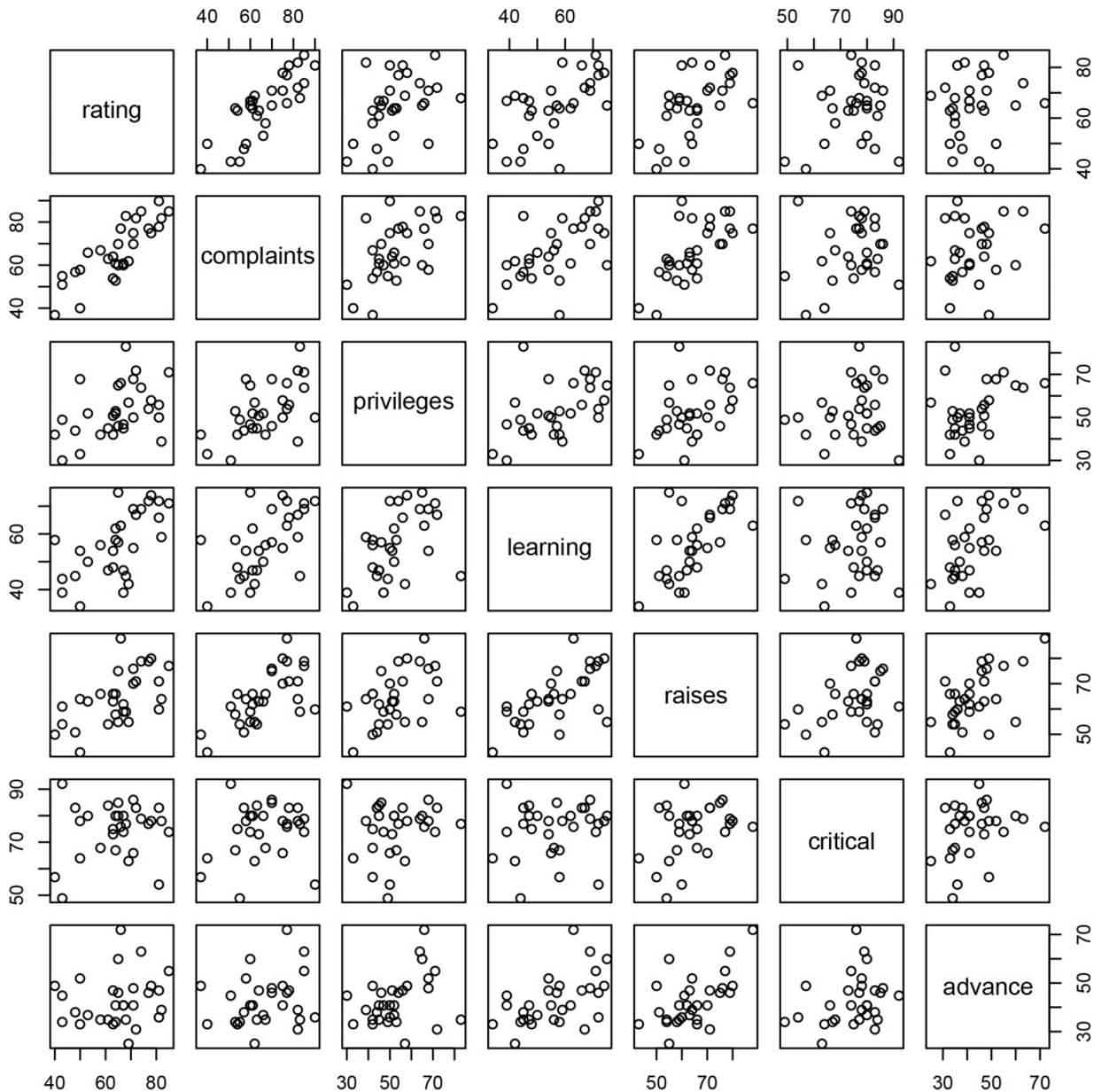
Linear Models

The *attitude* data set is a data frame with 30 observations on 7 variables, measuring the percent proportion of favorable responses to seven survey questions in each of 30 departments. The survey was conducted in a large financial organization; there were approximately 35 respondents in each department.

We mentioned that the `plot` function could be used with virtually any data set to get an initial visualization; let's see what it gives for the *attitude* data:

```
plot(attitude)
```

The resulting plot is a pairwise scatter plot of the numeric variables in the data set.



The first two variables (*rating* and *complaints*) show a strong linear relationship. To model that relationship, we use the `lm` function:

```
attitudeLM1 <- lm(rating ~ complaints, data=attitude)
```

To view a summary of the model, we can use the `summary` function:

```
summary(attitudeLM1)
```

Output:

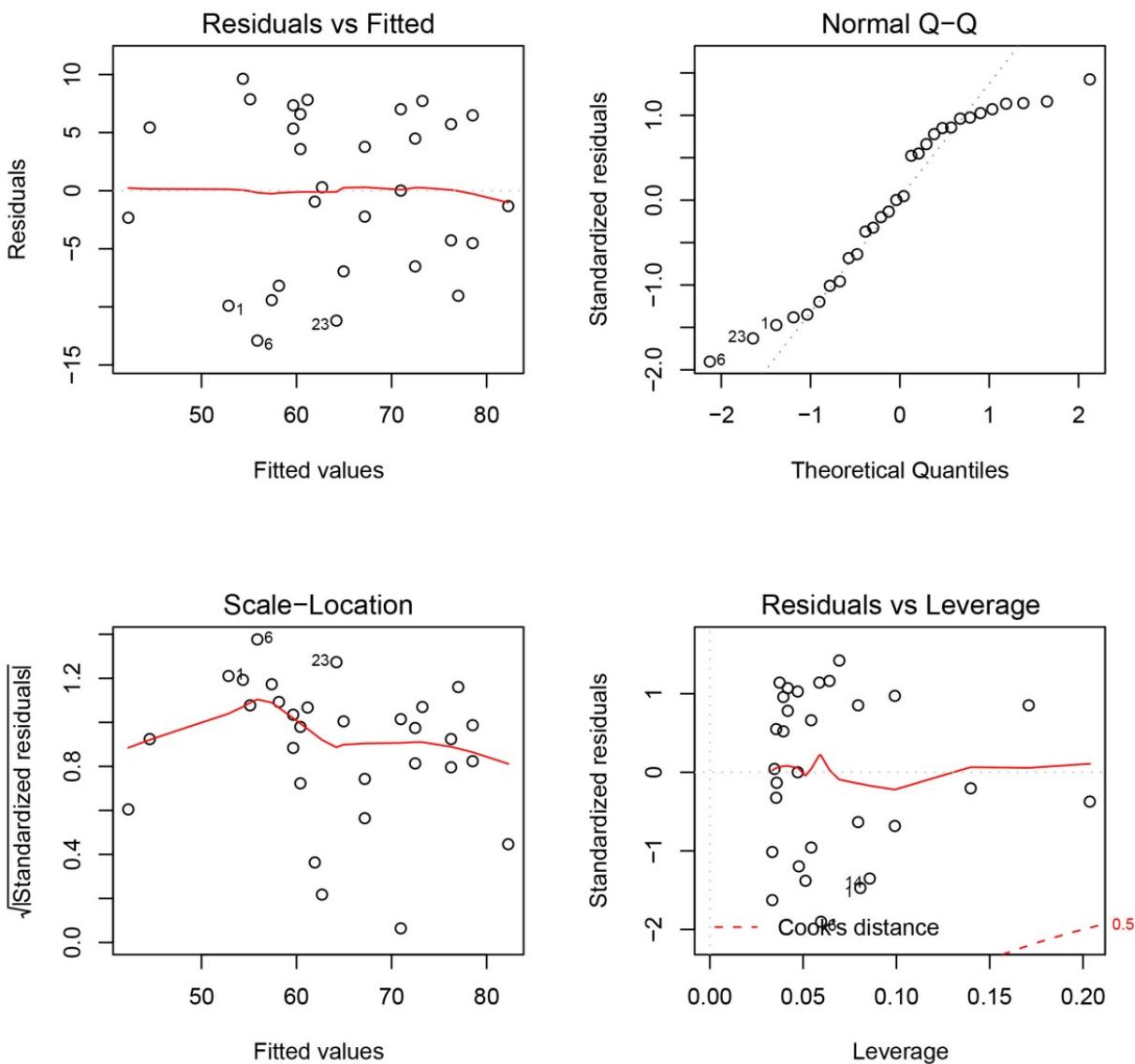
```
Call:  
lm(formula = rating ~ complaints, data = attitude)  
Residuals:  
    Min      1Q  Median      3Q     Max  
-12.8799 -5.9905  0.1783  6.2978  9.6294  
  
Coefficients:  
            Estimate Std. Error t value Pr(>|t|)  
(Intercept) 14.37632   6.61999   2.172   0.0385 *  
complaints   0.75461   0.09753   7.737 1.99e-08 ***  
---  
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1  1  
  
Residual standard error: 6.993 on 28 degrees of freedom  
Multiple R-squared: 0.6813, Adjusted R-squared: 0.6699  
F-statistic: 59.86 on 1 and 28 DF, p-value: 1.988e-08
```

We can also try to visualize the model using `plot`:

```
plot(attitudeLM1)
```

The default plot for a fitted linear model is a set of four plots; by default they are shown one at a time, and you are prompted before each new plot is displayed. To view them all at once, use the `par` function with the `mfrow` parameter to specify a 2×2 layout:

```
par(mfrow=c(2,2))  
plot(attitudeLM1)
```



TIP

The `rxLinMod` function is a full-featured alternative to `lm` that can efficiently handle large data sets. Also look at `rxLogit` and `rxGlm` as alternatives to `glm`, `rxKmeans` as an alternative to `kmeans`, and `rxDTree` as an alternative to `rpart`.

Matrices and `apply` function

A matrix is a two-dimensional data array. Unlike data frames, which can have different data types in their columns, matrices may contain data of only one type. Most commonly, matrices are used to hold numeric data. You create matrices with the `matrix` function:

```
A <- matrix(c(3, 5, 7, 9, 13, 15, 8, 4, 2), ncol=3)
A
```

Output:

	[,1]	[,2]	[,3]
[1,]	3	9	8
[2,]	5	13	4
[3,]	7	15	2

Another matrix example:

```
B <- matrix(c(4, 7, 9, 5, 8, 6), ncol=3)
B
```

Output:

```
[,1] [,2] [,3]
[1,]    4     9     8
[2,]    7     5     6
```

Ordinary arithmetic acts *element-by-element* on matrices:

```
A + A
```

Output:

```
[,1] [,2] [,3]
[1,]    6    18    16
[2,]   10    26     8
[3,]   14    30     4
```

Matrix multiplication uses the expected operators:

```
A * A
```

Output:

```
[,1] [,2] [,3]
[1,]    9    81    64
[2,]   25   169    16
[3,]   49   225     4
```

Matrix multiplication in the linear algebra sense requires a special operator, `%%%`:

```
A %% A

[,1] [,2] [,3]
[1,] 110  264   76
[2,] 108  274  100
[3,] 110  288  120
```

Matrix multiplication requires two matrices to be *conformable*, which means that the number of *columns* of the first matrix is equal to the number of *rows* of the second:

```
B %% A

[,1] [,2] [,3]
[1,] 113  273   84
[2,]  88  218   88

A %% B

Error in A %% B : non-conformable arguments
```

When you need to manipulate the rows or columns of a matrix, an incredibly useful tool is the `apply` function.

With `apply`, you can apply a function to all the rows or columns of matrix at once. For example, to find the column products of A , you could use `apply` as follows:

```
apply(A, 2, prod)  
[1] 105 1755   64
```

The row products are as simple:

```
apply(A, 1, prod)  
[1] 216 260 210
```

To sort the columns of A , just replace *prod* with *sort*.

```
apply(A, 2, sort)  
[,1] [,2] [,3]  
[1,]    3     9     2  
[2,]    5    13     4  
[3,]    7    15     8
```

Lists and *lapply* function

A `list` in R is a flexible data object that can be used to combine data of different types and different lengths for almost any purpose. Arbitrary lists can be created with either the `list` function or the `c` function; many other functions, especially the statistical modeling functions, return their output as list objects.

For example, we can combine a character vector, a numeric vector, and a numeric matrix in a single list as follows:

```
list1 <- list(x = 1:10, y = c("Tami", "Victor", "Emily"),  
z = matrix(c(3, 5, 4, 7), nrow=2))  
list1  
  
$x  
[1] 1 2 3 4 5 6 7 8 9 10  
$y  
[1] "Tami"   "Victor" "Emily"  
$z  
[,1] [,2]  
[1,]    3     4  
[2,]    5     7
```

The function `lapply` can be used to apply the same function to each component of a list in turn:

```
lapply(list1, length)  
  
$x  
[1] 10  
  
$y  
[1] 3  
  
$z  
[1] 4
```

TIP

You will regularly use lists and functions that manipulate them when handling input and output for your big data analyses.

Explore RevoScaleR Functions

The **RevoScaleR** package, included in Machine Learning Server and R Client, provides a framework for quickly writing start-to-finish, scalable R code for data analysis. When you start the R console application on a computer that has Machine Learning Server or R Client, the RevoScaleR function library is loaded automatically.

Load Data with *rxImport*

The *rxImport* function allows you to import data from fixed or delimited text files, SAS files, SPSS files, or a SQL Server, Teradata, or ODBC connection. There's no need to have SAS or SPSS installed on your system to import those file types, but you need a [locally installed ODBC driver](#) for your database to access data on a local or remote computer.

Let's start simply by using a delimited text file available in the [built-in sample data directory](#) of the **RevoScaleR** package. We store the location of the file in a character string (*inDataFile*), then import the data into an in-memory data set (data frame) called *mortData*:

```
inDataFile <- file.path(rxGetOption("sampleDataDir"), "mortDefaultSmall2000.csv")
mortData <- rxImport(inData = inDataFile)
```

If we anticipate repeating the same analysis on a larger data set later, we could prepare for that by putting placeholders in our code for output files. An output file is an XDF file, native to R Client and Machine Learning Server, persisted on disk and structured to hold modular data. If we included an output file with *rxImport*, the output object returned from *rxImport* would be a small object representing the .xdf file on disk (an *RxXdfData* object), rather than an in-memory data frame containing all of the data.

For now, let's continue to work with the data in memory. We can do this by omitting the *outFile* argument, or by setting the *outFile* parameter to NULL. The following code is equivalent to the importing task of that above:

```
mortOutput <- NULL
mortData <- rxImport(inData = inDataFile, outFile = mortOutput)
```

Retrieve metadata

There are a number of basic methods we can use to learn about the data set and its variables that work on the return object of *rxImport*, regardless of whether it is a data frame or *RxXdfData* object.

To get the number of rows, cols, and names of the imported data:

```
nrow(mortData)
ncol(mortData)
names(mortData)
```

Output for these commands is as follows:

```
> nrow(mortData)
[1] 10000
> ncol(mortData)
[1] 6
> names(mortData)
[1] "creditScore" "houseAge" "yearsEmploy" "ccDebt" "year"
[6] "default"
```

To print out the first few rows of the data set, you can use *head()*:

```
head(mortData, n = 3)
```

Output:

```
creditScore houseAge yearsEmploy ccDebt year default
1 691 16 9 6725 2000 0
2 691 4 4 5077 2000 0
3 743 18 3 3080 2000 0
```

The *rxGetInfo* function allows you to quickly get information about your data set and its variables all at one time, including more information about variable types and ranges. Let's try it on *mortData*, having the first three rows of the data set printed out:

```
rxGetInfo(mortData, getVarInfo = TRUE, numRows=3)
```

Output:

```
Data frame: mortData
Data frame: mortData
Number of observations: 10000
Number of variables: 6
Variable information:
Var 1: creditScore, Type: integer, Low/High: (486, 895)
Var 2: houseAge, Type: integer, Low/High: (0, 40)
Var 3: yearsEmploy, Type: integer, Low/High: (0, 14)
Var 4: ccDebt, Type: integer, Low/High: (0, 12275)
Var 5: year, Type: integer, Low/High: (2000, 2000)
Var 6: default, Type: integer, Low/High: (0, 1)
Data (3 rows starting with row 1):
  creditScore houseAge yearsEmploy ccDebt year default
1 691 16 9 6725 2000 0
2 691 4 4 5077 2000 0
3 743 18 3 3080 2000 0
```

Select and transform with *rxDataStep*

The *rxDataStep* function provides a framework for the majority of your data manipulation tasks. It allows for row selection (the *rowSelection* argument), variable selection (the *varsToKeep* or *varsToDelete* arguments), and the creation of new variables from existing ones (the *transforms* argument). Here's an example that does all three with one function call:

```

outFile2 <- NULL
mortDataNew <- rxDataStep(
  # Specify the input data set
  inData = mortData,
  # Put in a placeholder for an output file
  outFile = outFile2,
  # Specify any variables to keep or drop
  varsToDrop = c("year"),
  # Specify rows to select
  rowSelection = creditScore < 850,
  # Specify a list of new variables to create
  transforms = list(
    catDebt = cut(ccDebt, breaks = c(0, 6500, 13000),
      labels = c("Low Debt", "High Debt")),
    lowScore = creditScore < 625))

```

Our new data set, *mortDataNew*, will not have the variable *year*, but adds two new variables: a categorical variable named *catDebt* that uses R's `cut` function to break the *ccDebt* variable into two categories, and a logical variable, *lowScore*, that is TRUE for individuals with low credit scores. These *transforms* expressions follow the rule that they must be able to operate on a chunk of data at a time; that is, the computation for a single row of data cannot depend on values in other rows of data.

With the *rowSelection* argument, we have also removed any observations with high credit scores, above or equal to 850. We can use the *rxGetVarInfo* function to confirm:

```
rxGetVarInfo(mortDataNew)
```

Output:

```

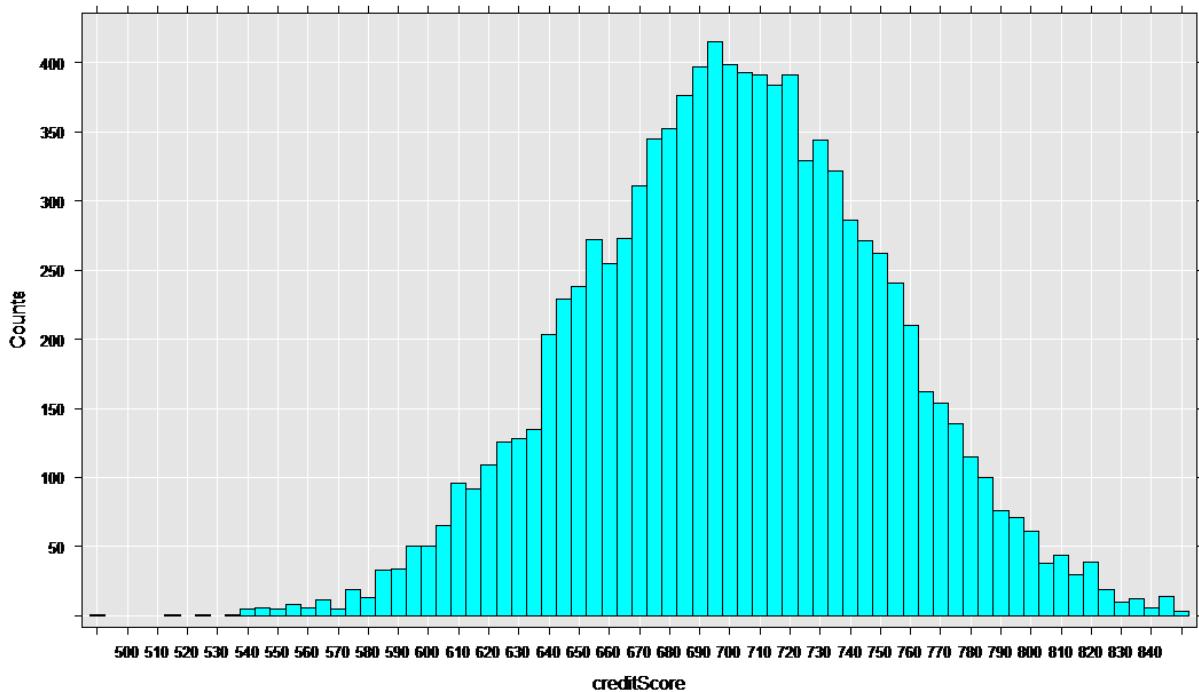
Var 1: creditScore, Type: integer, Low/High: (486, 847)
Var 2: houseAge, Type: integer, Low/High: (0, 40)
Var 3: yearsEmploy, Type: integer, Low/High: (0, 14)
Var 4: ccDebt, Type: integer, Low/High: (0, 12275)
Var 5: default, Type: integer, Low/High: (0, 1)
Var 6: catDebt
  2 factor levels: Low Debt High Debt
Var 7: lowScore, Type: logical, Low/High: (0, 1)

```

Visualize with *rxHistogram*, *rxCube*, and *rxLinePlot*

The *rxHistogram* function shows us the distribution of any of the variables in our data set. For example, let's look at credit score:

```
rxHistogram(~creditScore, data = mortDataNew )
```

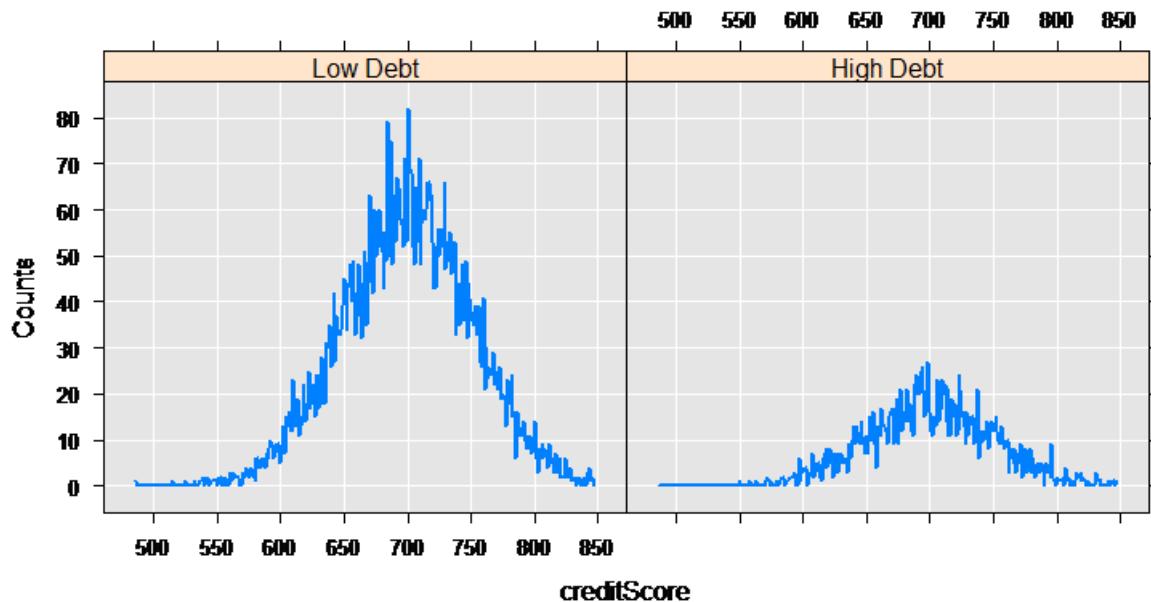


The `rxCube` function computes category counts, and can operate on the interaction of categorical variables. Using the `F()` notation to convert a variable into an on-the-fly categorical factor variable (with a level for each integer value), we can compute the counts for each credit score for the two groups who have low and high credit card debt:

```
mortCube <- rxCube(~F(creditScore):catDebt, data = mortDataNew)
```

The `rxLinePlot` function is a convenient way to plot output from `rxCube`. We use the `rxResultsDF` helper function to convert cube output into a data frame convenient for plotting:

```
rxLinePlot(Counts~creditScore|catDebt, data=rxResultsDF(mortCube))
```



Analyze with `rxLogit`

RevoScaleR provides the foundation for a variety of high performance, scalable data analyses. Here we do a

logistic regression, but you probably also want to take look at computing summary statistics (*rxSummary*), computing cross-tabs (*rxCrossTabs*), estimating linear models (*rxLinMod*) or generalized linear models (*rxGlm*), and estimating variance-covariance or correlation matrices (*rxCovCor*) that can be used as inputs to other R functions such as principal components analysis and factor analysis. Now, let's estimate a logistic regression on whether or not an individual defaulted on their loan, using credit card debt and years of employment as independent variables:

```
myLogit <- rxLogit(default~ccDebt+yearsEmploy , data=mortDataNew)
summary(myLogit)
```

We get the following output:

```
Call:
rxLogit(formula = default ~ ccDebt + yearsEmploy, data = mortDataNew)

Logistic Regression Results for: default ~ ccDebt + yearsEmploy
Data: mortDataNew
Dependent variable(s): default
Total independent variables: 3
Number of valid observations: 9982
Number of missing observations: 0
-2*LogLikelihood: 100.6036 (Residual deviance on 9979 degrees of freedom)

Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.614e+01 2.074e+00 -7.781 2.22e-16 ***
ccDebt 1.414e-03 2.139e-04 6.610 3.83e-11 ***
yearsEmploy -3.317e-01 1.608e-01 -2.063 0.0391 *
---
Signif. codes: 0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Condition number of final variance-covariance matrix: 1.4455
Number of iterations: 9
```

Scale your analysis

Until now, our exploration has been limited to small data sets in memory. Let's scale up to a data set with a million rows rather than just 10000. These larger text data files are available [online](#). Windows users should download the zip version, *mortDefault.zip*, and Linux users *mortDefault.tar.gz*.

After downloading and unpacking the data, set your path to the correct location in the code below. It is more efficient to store the imported data on disk, so we also specify the locations for our imported and transformed data sets:

```
# bigDataDir <- "C:/MicrosoftR/Data" # Specify the location
inDataFile <- file.path(bigDataDir, "mortDefault",
"mortDefault2000.csv")
outFile <- "myMortData.xdf"
outFile2 <- "myMortData2.xdf"
```

That's it! Now you can reuse all of the importing, data step, plotting, and analysis code preceding on the larger data set.

```
# Import data
mortData <- rxImport(inData = inDataFile, outFile = outFile)
```

Output:

```
Rows Read: 500000, Total Rows Processed: 500000, Total Chunk Time: 1.043 seconds
Rows Read: 500000, Total Rows Processed: 1000000, Total Chunk Time: 1.001 seconds
```

Because we have specified an output file when importing the data, the returned `mortData` object is a small object in memory representing the .xdf data file, rather than a full data frame containing all of the data in memory. It can be used in **RevoScaleR** analysis functions in the same way as data frames.

```
# Some quick information about my data
rxGetInfo(mortData, getVarInfo = TRUE, numRows=5)
```

Output:

```
File name: C:\\\\MicrosoftR\\\\Data\\\\myMortData.xdf
Number of observations: 1e+06
Number of variables: 6
Number of blocks: 2
Compression type: zlib
Variable information:
Var 1: creditScore, Type: integer, Low/High: (459, 942)
Var 2: houseAge, Type: integer, Low/High: (0, 40)
Var 3: yearsEmploy, Type: integer, Low/High: (0, 15)
Var 4: ccDebt, Type: integer, Low/High: (0, 14639)
Var 5: year, Type: integer, Low/High: (2000, 2000)
Var 6: default, Type: integer, Low/High: (0, 1)
Data (5 rows starting with row 1):
creditScore houseAge yearsEmploy ccDebt year default
1 615 10 5 2818 2000 0
2 780 34 5 3575 2000 0
3 735 12 1 3184 2000 0
4 713 15 5 6236 2000 0
5 689 10 5 6817 2000 0
```

The data step:

```
mortDataNew <- rxDataStep(
  # Specify the input data set
  inData = mortData,
  # Put in a placeholder for an output file
  outFile = outFile2,
  # Specify any variables to keep or drop
  varsToDrop = c("year"),
  # Specify rows to select
  rowSelection = creditScore < 850,
  # Specify a list of new variables to create
  transforms = list(
    catDebt = cut(ccDebt, breaks = c(0, 6500, 13000),
      labels = c("Low Debt", "High Debt")),
    lowScore = creditScore < 625))
```

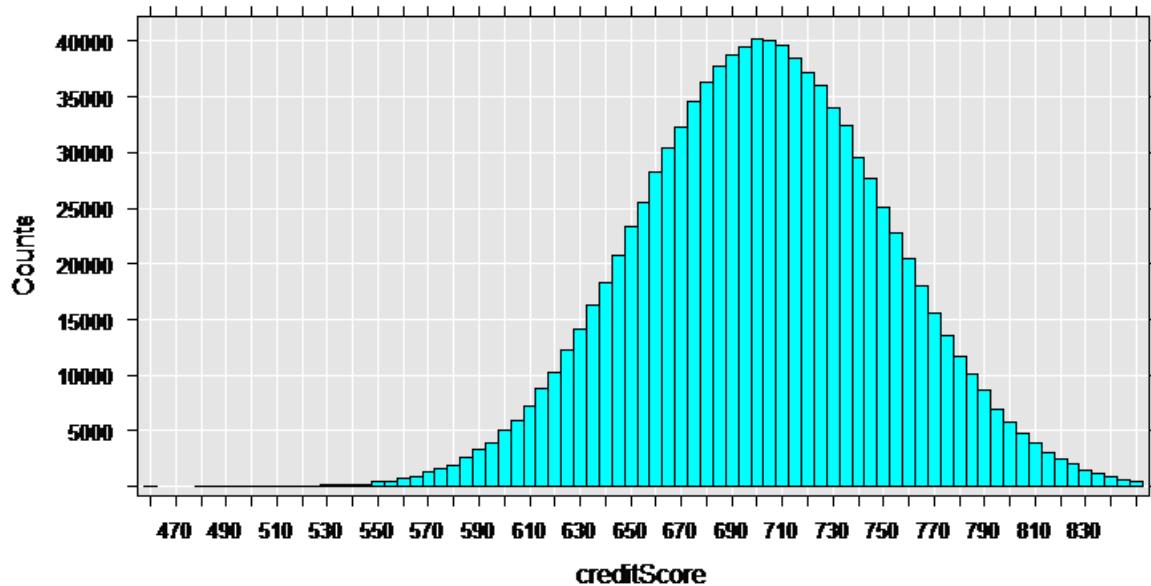
Output:

```
Rows Read: 500000, Total Rows Processed: 500000, Total Chunk Time: 0.673 seconds
Rows Read: 500000, Total Rows Processed: 1000000, Total Chunk Time: 0.448 seconds
>
```

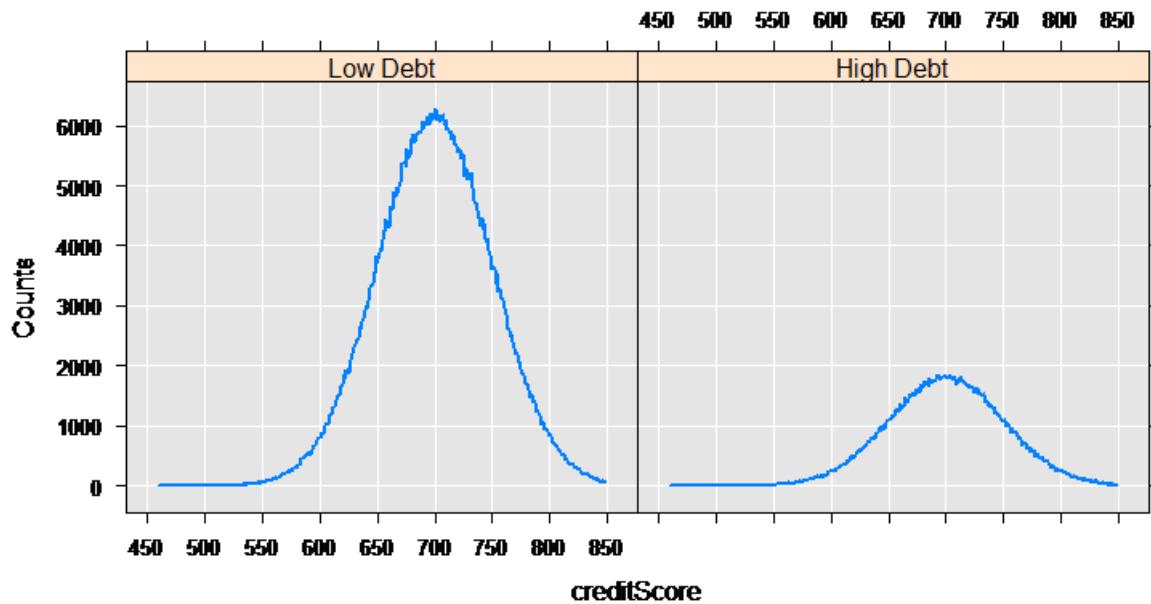
Looking at the data:

```
# Looking at the data
rxHistogram(~creditScore, data = mortDataNew )

Rows Read: 499294, Total Rows Processed: 499294, Total Chunk Time: 0.329 seconds
Rows Read: 499293, Total Rows Processed: 998587, Total Chunk Time: 0.335 seconds
Computation time: 0.678 seconds.
```



```
myCube = rxCube(~F(creditScore):catDebt, data = mortDataNew)
rxLinePlot(Counts~creditScore|catDebt, data=rxResultsDF(myCube))
```



```
# Compute a logistic regression
myLogit <- rxLogit(default~ccDebt+yearsEmploy , data=mortDataNew)
summary(myLogit)
```

Output:

```
Call:
rxLogit(formula = default ~ ccDebt + yearsEmploy, data = mortDataNew)

Logistic Regression Results for: default ~ ccDebt + yearsEmploy
File name:
C:\Users\RUser\myMortData2.xdf
Dependent variable(s): default
Total independent variables: 3
Number of valid observations: 998587
Number of missing observations: 0
-2*LogLikelihood: 8837.7644 (Residual deviance on 998584 degrees of freedom)

Coefficients:

Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.725e+01 2.330e-01 -74.04 2.22e-16 ***
ccDebt 1.509e-03 2.327e-05 64.85 2.22e-16 ***
yearsEmploy -3.054e-01 1.713e-02 -17.83 2.22e-16 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Condition number of final variance-covariance matrix: 1.3005
Number of iterations: 9
```

Next Steps

Continue on to these tutorials to work with larger data set using the RevoScaleR functions:

- [Flight delays data analysis](#)
- [Loan data analysis](#)
- [Census data analysis](#)

Compute context for script execution in Machine Learning Server

7/12/2022 • 5 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

In Machine Learning Server, a *compute context* refers to the physical location of the computational engine handling a given workload. The default is local. However, if you have multiple machines, you can switch from local to remote, pushing execution of data-centric [RevoScaleR \(R\)](#), [revoscalepy \(Python\)](#), [MicrosoftML \(R\)](#) and [microsoftml \(Python\)](#) functions to a computational engine on another system. For example, script running locally in R Client can shift execution to a remote Machine Learning Server in a Spark cluster to process data there.

The primary reason for shifting compute context is to eliminate data transfer over your network, bringing computations to where the data resides. This is particularly relevant for big data platforms like Hadoop, where data is distributed over multiple nodes, or for data sets that are simply too large for a client workstation.

Compare "local" to "remote"

CONTEXT	USAGE
Local	Default, supported by all products (including R Client), on all platforms. Script executes on local interpreters using local machine resources.
Remote	Specifically targets a Machine Learning Server on selected data platforms: Spark over the Hadoop Distributed File System (HDFS) and SQL Server. Clients, or servers acting in the capacity of a client, can initiate a remote compute context, but the target remote machine itself must be a Machine Learning Server installation.

Compare "remote execution" to "remote compute context"

Although similarly named, remote execution is distinct from a remote compute context.

CONCEPT	LANGUAGE	USAGE	CONFIGURATION
Remote compute context	R and Python	Data-centric and function-specific. Script or code that runs in a remote compute context can include functions from our proprietary libraries: RevoScaleR (R) , MicrosoftML (R) , revoscalepy (Python) , and microsoftml (Python) .	None required. If you have server or client installs at the same functional level, you can write script that shifts the compute context.

CONCEPT	LANGUAGE	USAGE	CONFIGURATION
Remote execution	R only	Machine-oriented, using two or more Machine Learning Server instances interchangeably, or shifting execution from R Client to a more powerful Machine Learning Server on Windows or Linux. Remote execution is data and library agnostic: you can call functions from any library, including base R and third-party vendors.	An operationalization feature, enabled as a post-installation task. For more information, see remote execution .

RevoScaleR compute context

Remote computing is available for specific data sources on selected platforms. The following tables document the supported combinations.

CONTEXT NAME	ALIAS	USAGE
RxLocalSeq	local	All server and client configurations support a local compute context.
RxSpark	spark	Remote compute context. Target is a Spark cluster on Hadoop.
RxInSqlServer	sqlserver	Remote compute context. Target server is a single database node (SQL Server 2016 R Services or SQL Server 2017 or later Machine Learning Services). Computation is parallel, but not distributed.
RxLocalParallel	localpar	Compute context is often used to enable controlled, distributed computations relying on instructions you provide rather than a built-in scheduler on Hadoop. You can use compute context for manual distributed computing.
RxForeachDoPar	dopar	Use for manual distributed computing.

Data sources per compute context

Given a compute context, the following table shows which data sources are available (x indicates available):

DATA SOURCE	RXLOCALSEQ	RXSPARK	RXINSQLSERVER
RxTextData	X	X	
RxXdfData	X	X	
RxHiveData	X	X	

DATA SOURCE	RXLOCALSEQ	RXSPARK	RXINSQLSERVER
RxParquetData	X	X	
RxOrcData	X	X	
RxOdbcData	X		
RxSqlServerData	X		X
RxSasData	X		
RxSpssData	X		

NOTE

Within a data source type, you might find differences depending on the file system type and compute context. For example, the .xdf files created on the Hadoop Distributed File System (HDFS) are somewhat different from .xdf files created in a non-distributed file system such as Windows or Linux. For more information, see [How to use RevoScaleR on Spark](#).

revoscalepy compute context

Remote computing is available for specific data sources on selected platforms. The following tables document the supported combinations for revoscalepy.

CONTEXT NAME	ALIAS	USAGE
RxLocalSeq	local	All server and client configurations support a local compute context.
rx-spark-connect	spark	Remote compute context. Target is a Spark 2.0-2.1 cluster over Hadoop Distributed File System (HDFS).
RxInSqlServer	sqlserver	Remote compute context. Target server is a single database node (SQL Server 2017 Machine Learning with Python support). Computation is parallel, but not distributed.

Data sources per compute context

Given a compute context, the following table shows which data sources are available (x indicates available):

DATA SOURCE	RXLOCALSEQ	RX-GET-SPARK-CONNECT	RXINSQLSERVER
RxTextData	X	X	
RxXdfData	X	X	
RxHiveData	X	X	
RxParquetData	X	X	

DATA SOURCE	RXLOCALSEQ	RX-GET-SPARK-CONNECT	RXINSQLSERVER
RxOrcData	X	X	
RxSparkDataFrame	X	X	
RxOdbcData	X	X	
RxSqlServerData	X		X

When to switch context

The primary use case for switching the compute context is to bring calculations and analysis to the data itself. As such, the use cases for a remote compute context leverage database platforms, such as SQL Server, or data located on the Hadoop Distributed File System (HDFS) using Spark or MapReduce for processing layer.

USE CASE	DESCRIPTION
Client to Server	Write and run script locally in R Client, pushing specific computations to a remote Machine Learning Server instance. You can shift calculations to systems with more powerful processing capabilities or database assets.
Server to Server	Push platform-specific computations to a server on a different platform. Supported platforms include SQL Server, Hadoop (Spark). You can implement a distributed processing architecture: RxLocalSeq, RxSpark, RxInSqlServer.

Context and distributed computing

Many analytical functions in RevoScaleR, MicrosoftML, revoscalepy, and microsoftml can execute in parallel. On a multi-core computer, such functions run multi-threaded. On a distributed platform like Hadoop, the functions distribute workload execution to all available cores and nodes. This capability translates into high-performance computing for predictive and statistical analysis of big data, and is a major motivation for pushing a compute context to a remote Hadoop cluster. For more information, see [Distributed and parallel computing in Machine Learning Server](#).

Next steps

Step-by-step instructions on how to get, set, and manage compute context in [How to set and manage compute context in Machine Learning Server](#).

See also

- [Introduction to Machine Learning Server](#)
- [Install Machine Learning Server on Windows](#)
- [Install Machine Learning Server on Linux](#)
- [Install Machine Learning Server on Hadoop](#)

Distributed and parallel computing in Machine Learning Server

7/12/2022 • 3 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Machine Learning Server's computational engine is built for distributed and parallel processing, automatically partitioning a workload across multiple nodes in a cluster, or on the available threads on multi-core machine. Access to the engine is through functions in our proprietary packages: [RevoScaleR \(R\)](#), [revoscalepy \(Python\)](#), and machine learning algorithms in [MicrosoftML \(R\)](#) and [microsoftml \(Python\)](#), respectively.

RevoScaleR and revoscalepy provide data structures and data operations used in downstream analyses and predictions. Both libraries are designed to process large data one chunk at a time, independently and in parallel. Each computing resource needs access only to that portion of the total data source required for its particular computation. This capability is amplified on distributed computing platforms like Spark. Instead of passing large amounts of data from node to node, the computations are farmed out to nodes in the cluster, executing on the node providing the data.

Architectures supporting workload distribution

On a single server with multiple cores, jobs run in parallel, assuming the workload can be divided into smaller pieces and executed on multiple threads.

On a distributed platform like Hadoop, you might write script that runs locally on one node, such as an edge node in the cluster, but shift execution to worker nodes for bigger jobs. When executed on a distributed platform like Spark over Hadoop Distributed File System (HDFS), both revoscalepy and RevoScaleR automatically use all available cores on all nodes in a cluster.

Distributed and parallel processing is revo-managed, where the engine assigns a job to an available computing resource (a node in cluster, or a thread on a multi-core machine), thereby becoming the logical master node for that job. The master node is responsible for the following operations:

1. Distributes the computation to itself and the other computing resources
2. Gathers the results of the independent, parallel computations
3. Finalizes and returns the results

To shift execution to the worker nodes in a cluster, you have to set the [compute context](#) to the platform. For example, you might use the local compute context on an edge node to prepare data or set up variables, and then shift context to RxSpark or RxHadoopMR to run data analysis on worker nodes.

Shifting to a Spark or HadoopMR compute context comes with a list of supported data sources for that platform. Assuming the data inputs you want to analyze are supported in a Spark or Hadoop compute context, your scripts for distributed analysis can include any of the functions noted in this article. For a list of supported data sources by compute context, see [Compute context for script execution in Machine Learning Server](#).

NOTE

Distributed computing is conceptually similar to parallel computing, but in Machine Learning Server, it specifically refers to workload distribution across multiple physical servers. Distributed platforms contribute the following infrastructure used for managing the entire operation: a job scheduler for allocating jobs, data nodes to run the jobs, and a master node for tracking the work and coordinating the results. In practical terms, you can think of distributed computing as a capability provided by [Machine Learning Server for Hadoop and Spark](#).

Functions for multi-threaded data operations

Import, merge, and step transformations are multi-threaded on a parallel architecture.

REVOSCALER (R)	REVOSCALEPY (PYTHON)
RxImport	rx-import
RxDataStep	rx-data-step
RxMerge	not available

Functions for distributed analysis

The following analytical functions execute in parallel, with the results unified as a single response in the return object:

REVOSCALER (R)	REVOSCALEPY (PYTHON)
rxSummary	rx-summary
rxLinMod	rx-lin-mod
rxLogit	rx-logit
rxGlm	not available
rxCovCor	not available
rxCube	not available
rxCrossTabs	not available
rxKmeans	not available
rxDTree	rx-dtree
rxForest	rx-dforest
rxBTrees	rx-btrees
rxNaiveBayes	not available

TIP

For examples and practical tips on working with high-performance analytical functions, see [Running distributed analyses using RevoScaleR](#).

User-defined distributed analysis

An alternative approach is to use the [rxExec \(R\)](#) or [rx-exec \(Python\)](#) function, which can run arbitrary R functions in a distributed fashion on available cores, and all available nodes in a cluster. You can create new functions or call existing functions in sequence, packaged as a single execution performed by rxExec.

When using rxExec, you largely control how the computational tasks are distributed and you are responsible for any aggregation and final processing of results.

See also

- [Running distributed analyses using RevoScaleR](#)
- [Compute context](#)
- [What is RevoScaleR](#)

What are web services in Machine Learning Server?

7/12/2022 • 6 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Applies to: Machine Learning Server

In Machine Learning Server, a web service is an R or Python code execution on the [operationalization compute node](#).

Data scientists can deploy R and Python code and models as web services into Machine Learning Server to give other users a chance to use their code and predictive models. Once hosted there, these web services are exposed and available for consumption.

Web services can be consumed directly in R or Python, programmatically using [REST APIs](#), or via [Swagger-generated client libraries](#). They can be consumed synchronously, in real-time, or in batch mode. They can also be deployed from one platform and consumed on another.

Web services facilitate the consumption and integration of the operationalized models and code they contain. Once you've built a predictive model, in many cases the next step is to operationalize the model. That is to generate predictions from the pre-trained model on demand. In this scenario, where new data often become available one row at a time, latency becomes the critical metric. It is important to respond with the single prediction (or score) as quickly as possible.

Each web service is uniquely defined by its name and version. You can use the functions in the [mrsdeploy R package](#) or the [azureml-model-management-sdk Python package](#) to gain access a service's lifecycle from an R or Python script.

Requirement! Before you can deploy and work with web services, you must have access to a Machine Learning Server instance [configured to host web services](#).

There are two types of web services: standard and real-time.

Standard web services

These web services offer fast execution and scoring of arbitrary Python or R code and models. They can contain code, models, and model assets. They can also take specific inputs and provide specific outputs for those users who are integrating the services inside their applications.

Standard web services, like all web services, are identified by their name and version. Additionally, they can also be defined by any Python or R code, models, and any necessary model assets. When deploying a standard web service, you should also define the required inputs and any output the application developers use to integrate the service in their applications.

See a standard web service deployment example: [R](#) | [Python](#)

Real-time web services

Real-time web services do not support arbitrary code and only accept models created with the supported functions from packages installed with the product. See the following sections for the list of supported functions by language and package.

Real-time web services offer even lower latency to produce results faster and score more models in parallel. The improved performance boost comes from the fact that these web services do not depend on an interpreter at consumption time even though the services use the objects created by the model. Therefore, fewer additional resources and less time is spent spinning up a session for each call. Additionally, the model is only loaded once in the compute node and can be scored multiple times.

For real-time services, you do **not** need to specify:

- inputs and outputs (dataframes are assumed)
- code (only serialized models are supported)

See [real-time web service deployment examples](#): [R](#) | [Python](#)

Supported R functions for real time

A model object created with these supported functions:

R PACKAGE	SUPPORTED FUNCTIONS
RevoScaleR	rxBTrees, rxDTree, rxDForest, rxLogit, rxLinMod
MicrosoftML	Machine learning and transform tasks: rxFastTrees, rxFastForest, rxLogisticRegression, rxOneClassSvm, rxNeuralNet, rxFastLinear, featurizeText, concat, categorical, categoricalHash, selectFeatures, featurizeImage, getSentiment, loadImage, resizeImage, extractPixels, selectColumns, and dropColumns While mlTransform featurization is supported in real-time scoring, R transforms are not supported. Instead, use sp_execute_external_script.

There are additional restrictions on the input dataframe format for microsoftml models:

1. The dataframe must have the same number of columns as the formula specified for the model.
2. The dataframe must be in the exact same order as the formula specified for the model.
3. The columns must be of the same data type as the training data. Type casting is not possible.

Supported Python functions for real time

PYTHON PACKAGE	SUPPORTED FUNCTIONS
revoscalepy	rx_btrees, rx_dforest, rx_dtrees, rx_logit, rx_lin_mod
microsoftml	Machine learning and transform tasks: categorical, categorical_hash, concat, extract_pixels, featurize_text, featurize_image, get_sentiment, rx_fast_trees, rx_fast_forest, rx_fast_linear, rx_logistic_regression, rx_neural_network, rx_oneclass_svm, load_image, resize_image, select_columns, and drop_columns. See the preceding input dataframe format restrictions .

Versioning

Every time a web service is published, a version is assigned to the web service. Versioning enables users to better manage the release of their web services and helps the people consuming your service to find it easily.

At publish time, specify an alphanumeric string that is meaningful to those users who consume the service. For example, you could use '2.0', 'v1.0.0', 'v1.0.0-alpha', or 'test-1'. Meaningful versions are helpful when you intend to share services with others. We highly recommend a **consistent and meaningful versioning convention** across your organization or team such as semantic versioning. Learn more about semantic versioning here: <http://semver.org/>.

If you do not specify a version, a globally unique identifier (GUID) is automatically assigned. These GUID numbers are long making them harder to remember and use.

Who consumes web services

After a web service has been published, authenticated users can consume that web service on various platforms and in various languages. You can consume directly in R or Python, using APIs, or in your preferred language via Swagger.

You can make it easy for others to find your web services by providing them with the name and version of the web service.

- **Data scientists** who want to explore and consume the services directly [in R](#) and [in Python](#).
- **Quality engineers** who want to bring the models in these web services into validation and monitoring cycles.
- **Application developers** who want to call and integrate a web service into their applications.
Developers can generate client libraries for integration using the Swagger-based JSON file generated during service deployment. Read "[How to integrate web services and authentication into your application](#)" for more details. Services can also be consumed using the [RESTful APIs](#) that provide direct programmatic access to a service's lifecycle.

How are web services consumed

Web services can be consumed using one of these approaches:

APPROACH	DESCRIPTION
Request Response	The service is consumed directly using a single synchronous consumption call. Learn how in R in Python
Asynchronous Batch	Users send a single asynchronous request to the server who in turn makes multiple service calls on their behalf. Learn how in R

Permissions

By default, any authenticated Machine Learning Server user can:

- Publish a new service
- Update and delete web services they have published
- Retrieve any web service object for consumption

- Retrieve a list of any or all web services

Destructive tasks, such as deleting a web service, are available only to the user who initially created the service. However, your administrator can also [assign role-based authorization](#) to further control the permissions around web services. When you list services, you can see your role for each one of them.

See also

In R:

- [Deploy and manage web services in R](#)
- [List, get, and consume web services in R](#)
- [Asynchronous web service consumption via batch](#)

In Python:

- [Deploy and manage web services in Python](#)
- [List, get, and consume web services in Python](#)

Pre-trained machine learning models for sentiment analysis and image detection

7/12/2022 • 2 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

For sentiment analysis of text and image classification, Machine Learning Server offers two approaches for training the models: you can train the models yourself using your data, or install pre-trained models that come with training data obtained and developed by Microsoft. The advantage of pre-trained models is that you can score and classify new content right away.

- Sentiment analysis scores raw unstructured text in positive-negative terms, returning a score between 0 (negative) and 1 (positive), indicating relative sentiment.
- Image detection identifies features of the image. There are several use cases for this model: image recognition, image classification. For image recognition, the model returns n-grams that possibly describe the image. For image classification, the model evaluates images and returns a classification based on possible classes you provided (for example, is the image a fish or a dog).

Pre-trained models are available for both R and Python development, through the [MicrosoftML R package](#) and the [microsoftml Python package](#).

Benefits of using pre-trained models

Pre-trained models have been made available to support customers who need to perform tasks such as sentiment analysis or image featurization, but do not have the resources to obtain the large datasets or train a complex model. Using pre-trained models lets you get started on text and image processing most efficiently.

Currently the models that are available are deep neural network (DNN) models for sentiment analysis and image classification. All four pre-trained models were trained on CNTK. The configuration of each network was based on the following reference implementations:

- Resnet-18
- Resnet-50
- ResNet-101
- AlexNet

For more information about deep residual networks and their implementation using CNTK, go the [Microsoft Research](#) web site and search for these articles:

- Microsoft Researchers' Algorithm Sets ImageNet Challenge Milestone
- Microsoft Computational Network Toolkit offers most efficient distributed deep learning computational performance

How to install the models

Pre-trained models are installed through setup as an optional component of Machine Learning Server or [SQL](#)

[Server Machine Learning](#). You can also get the R version of the models through [Microsoft R Client](#).

1. Run a Machine Learning Server setup program for your target platform: [Install Machine Learning Server](#).
2. When specifying components to install, add at least one language (R Server or Python) and the pre-trained models. Language support is required. The models cannot be installed as a standalone component.
3. After setup completes, verify the models are on your computer. Pre-trained models are local, added to the MicrosoftML and microsoftml library, respectively, when you run setup. The files are \mxlibs<modelname>_updated.model for Python and \mxlibs\x64<modelname>_updated.model for R.

For samples demonstrating use of the pre-trained models, see [R Samples for MicrosoftML](#) and [Python Samples for MicrosoftML](#).

Next steps

Install the models by running the setup program or installation script for the target platform or product:

- [Install Machine Learning Server](#)
- [Install R Client on Windows](#)
- [Install R Client on Linux](#)
- [Install Python client libraries](#)

Review the associated function reference help:

- [featurize_image \(microsoftml Python\)](#)
- [featurize_text \(microsoftml Python\)](#)
- [featurizeImage \(MicrosoftML R\)](#)
- [featurizeText \(MicrosoftML R\)](#)

What is MicrosoftML?

7/12/2022 • 4 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

MicrosoftML adds state-of-the-art data transforms, machine learning algorithms, and pre-trained models to R and Python functionality. The *data transforms* provided by MicrosoftML allow you to compose a custom set of transforms in a pipeline that are applied to your data before training or testing. The primary purpose of these transforms is to allow you to format your data.

The MicrosoftML functions are provided through the **MicrosoftML** package installed with [Machine Learning Server](#), Microsoft R Client, and [SQL Server Machine Learning Services](#).

Functions provide fast and scalable *machine learning algorithms* that enable you to tackle common machine learning tasks such as classification, regression, and anomaly detection. These high-performance algorithms are multi-threaded, some of which execute off disk, so that they can scale up to 100s of GBs on a single-node. They are especially suitable for handling a large corpus of text data or high-dimensional categorical data. It enables you to run these functions locally on Windows or Linux machines or on Azure HDInsight (Hadoop/Spark) clusters.

[Pre-trained models](#) for sentiment analysis and image featurization can also be installed and deployed with MicrosoftML. For more information on the pre-trained models and samples, see [R samples for MicrosoftML](#) and [Python samples for MicrosoftML](#).

Match algorithms to machine learning tasks

Matching data transforms and machine learning algorithms to appropriate data science tasks is key to designing successful intelligent applications.

Machine learning tasks

The **MicrosoftML** package implements algorithms that can perform a variety of machine learning tasks:

- **binary classification:** algorithms that learn to predict which of two classes an instance of data belongs to. These provide supervised learning in which the input of a classification algorithm is a set of labeled examples. Each example is represented as a feature vector, and each label is an integer of value of 0 or 1. The output of a binary classification algorithm is a classifier, which can be used to predict the label of new unlabeled instances.
- **multi-class classification:** algorithms that learn to predict the category of an instance of data. These provide supervised learning in which the input of a classification algorithm is a set of labeled examples. Each example is represented as a feature vector, and each label is an integer between 0 and k-1, where k is the number of classes. The output of a classification algorithm is a classifier, which can be used to predict the label of a new unlabeled instance.
- **regression:** algorithms that learn to predict the value of a dependent variable from a set of related independent variables. Regression algorithms model this relationship to determine how the typical values of dependent variables change as the values of the independent variables are varied. These provide supervised learning in which the input of a regression algorithm is a set of examples with dependent variables of known

values. The output of a regression algorithm is a function, which can be used to predict the value of a new data instance whose dependent variables are not known.

- **anomaly detection:** algorithms that identify outliers that do not belong to some target class or conform to an expected pattern. One-class anomaly detection is a type of unsupervised learning as the input data only contains data that is from the target class and does not contain instances of anomalies to learn from.

Machine learning algorithms

The following table summarizes the MicrosoftML algorithms, the tasks they support, their scalability, and lists some example applications.

ALGORITHM (R/PYTHON)	ML TASK SUPPORTED	SCALABILITY	APPLICATION EXAMPLES
<code>rxFastLiner() / rx_fast-linear()</code> Fast Linear model (SDCA)	binary classification, linear regression	#cols: ~1B; #rows: ~1B; CPU: multi-proc	Mortgage default prediction, Email spam filtering
<code>rxOneClassSvm() / rx_onesvm()</code> OneClass SVM	anomaly detection	cols: ~1K; #rows: RAM-bound; CPU: single-proc	Credit card fraud detection
<code>rxFastTrees() / rx_fast-trees()</code> Fast Tree	binary classification, regression	#cols: ~50K; #rows: RAM-bound; CPU: multi-proc	Bankruptcy prediction
<code>rxFastForest() / rx_fast-forest()</code> Fast Forest	binary classification, regression	#cols: ~50K; #rows: RAM-bound; CPU: multi-proc	Churn Prediction
<code>rxNeuralNet() / rx_neural_network()</code> Neural Network	binary and multiclass classification, regression	#cols: ~10M; #rows: Inf; CPU: multi-proc CUDA GPU	Check signature recognition, OCR, Click Prediction
<code>rxLogisticRegression() / rx_logistic-regression()</code> Logistic regression	binary and multiclass classification	#cols: ~100M; #rows: Inf for single-proc CPU #rows: RAM-bound for multi-proc CPU	Classifying sentiments from feedback

Data transforms

MicrosoftML also provides transforms to help tailor your data for machine learning. They are used to clean, wrangle, train, and score your data. For a description of the transforms, see [Machine learning R transforms](#) and [Machine learning Python transforms](#) reference documentation.

Next steps

For reference documentation on the R individual transforms and functions in the product help, see [MicrosoftML: machine learning algorithms](#).

For reference documentation on the Python individual transforms and functions in the product help, see [MicrosoftML: machine learning algorithms](#).

For guidance when choosing the appropriate machine learning algorithm from the MicrosoftML package, see

the [Cheat Sheet: How to choose a MicrosoftML algorithm](#).

See also

[Machine Learning Server](#)

[R samples for MicrosoftML](#)

[Python samples for MicrosoftML](#)

What is RevoScaleR?

7/12/2022 • 7 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

RevoScaleR is a collection of proprietary functions in Machine Learning Server used for practicing data science at scale. For data scientists, RevoScaleR gives you data-related functions for import, transformation and manipulation, summarization, visualization, and analysis. *At scale* refers to the core engine's ability to perform these tasks against very large datasets, in parallel and on distributed file systems, chunking and reconstituting data when it cannot fit in memory.

RevoScaleR functions are provided through the **RevoScaleR** package installed for free in [Microsoft R Client](#) or commercially in [Machine Learning Server](#) on supported platforms. RevoScaleR is also embedded in Azure HDInsight, Azure Data Science virtual machines, and SQL Server.

The RevoScaleR functions run on a computational engine include in the aforementioned products. As such, the package cannot be downloaded or used independently of the products and services that provide it.

RevoScaleR is engineered to adapt to the computational power of the platform it runs on. On Machine Learning Server for Hadoop, script using RevoScaleR functions that run in parallel will automatically use nodes in the cluster. Whereas on the free R Client, scale is provided at much lower levels (2 processors, data resides in-memory).

RevoScaleR provides enhanced capabilities to many elements of the open-source R programming language. In fact, there are [RevoScaleR equivalents for many common base R functions](#), such as `rxSort` for `sort()`, `rxMerge` for `merge()`, and so forth. Because RevoScaleR is compatible with the open-source R language, solutions often use a combination of base R and RevoScaleR functions. RevoScaleR functions are denoted with an `rx` or `Rx` prefix to make them readily identifiable in your R script that uses the RevoScaleR package.

What can you do with RevoScaleR?

Data scientists and developers can include RevoScaleR functions in custom script or solutions that run locally against R Client or remotely on Machine Learning Server. Solutions leveraging RevoScaleR functions will run wherever the RevoScaleR engine is installed.

Analyzing data using RevoScaleR functions requires three distinct pieces of information:

1. Where the computations should take place (the compute context)
2. Which data to use (the data source)
3. What analysis to perform (the analysis function)

A common workflow is to write the initial code or script against a subset of data on a local computer, change the compute context to specify a large set of data on a big data platform, and then operationalize the solution by deploying it to the target environment, thus making it accessible to users.

At a high level, RevoScaleR functions are grouped as follows:

- Platform-specific utilities.
- Data-related functions are used for import, transformation, summarization, visualization, and analysis. These

functions comprise the bulk of the RevoScaleR function library.

The data manipulation and analysis functions in RevoScaleR are appropriate for small and large datasets, but are particularly useful in three common situations:

1. Analyze data sets that are too big to fit in memory.
2. Perform computations distributed over several cores, processors, or nodes in a cluster.
3. Create scalable data analysis routines that can be developed locally with smaller data sets, then deployed to larger data and/or a cluster of computers.

RevoScaleR enables these scenarios because it operates on chunks of data and uses *updating algorithms*.

Data is stored in an efficient XDF file format designed for rapid reading of arbitrary rows and columns of data. Functions in RevoScaleR are used to import data into XDF before performing analysis, but you can also work directly with data stored in a text, SPSS, or SAS file or an ODBC connection, or extract a subset of a data file into a data frame in memory for further analysis.

To perform an analysis, you must provide the following information: where the computations should take place (the compute context), the data to use (the data source), and what analysis to perform (the analysis function).

Data management and analysis using RevoScaleR

RevoScaleR provides functions for scalable data management and analysis. These functions can be used with data sets in memory, and applied the same way to huge data sets stored on disk. It includes functionality for:

- Accessing external data sets (SAS, SPSS, ODBC, Teradata, and delimited and fixed format text) for analysis in R
- Efficiently storing and retrieving data in a high performance data file
- Cleaning, exploring, and manipulating data
- Fast, basic statistical analyses

RevoScaleR also includes an extensible framework for writing your own analyses for big data sets.

With RevoScaleR, you can analyze data sets far larger than can be kept in memory. This is possible because RevoScaleR uses external memory algorithms that allow it to work on one chunk of data at a time (that is, a subset of the rows and perhaps the variables in the data set), update the results, and proceed through all available data.

Accessing External Data Sets

Data can be stored in a wide-variety of formats. Typically, the first step in any RevoScaleR analysis is to make the data accessible. With RevoScaleR's data import capability, you can access data from a SAS file, SPSS file, fixed format or delimited text file, an ODBC connection, SQL Server, or a Teradata database, bringing it into a data frame in memory, or storing it for fast access in chunks on disk.

Defining Compute Context

RevoScaleR has the concept of *compute context* that sets the location for calculations. The compute context is either local or remote, where remote offloads processing and analysis of chunked data to one or more remote Machine Learning Servers.

Local is the default, and it supports the full range of data source inputs. As its name suggests, a local compute context uses only the physical cores of the local computer. Local compute context is provided by RevoScaleR on both R Client and Machine Learning Server instances.

Remote compute context requires the explicit creation of a compute context object, a single logical object

defining location (a remote network resource that has Machine Learning Server and local data) and modes of processing (such as wait versus no-wait jobs). Remote compute context is supported for RevoScaleR analytical functions that can be performed in a distributed fashion, and is available on these platforms in Machine Learning Server only: HDInsight, Hadoop (Spark), Teradata, SQL Server, and Machine Learning Server (Windows and Linux). For more information, see [Compute Context](#).

Efficiently Storing and Retrieving Data

A key component of RevoScaleR is a data file format (.xdf) that is extremely efficient for both reading and writing data. You can create .xdf files by importing data files or from R data frames, and add rows or variables to an existing .xdf file (appending rows is currently only supported in a local compute context). Once your data is in this file format you can use it directly with analysis functions provided with RevoScaleR, or quickly extract a subsample and read it into a data frame in memory for use in other R functions.

Data Cleaning, Exploration, and Manipulation

When working with a new data set, the first step is to clean and explore. With RevoScaleR, you can quickly obtain information about your data set (e.g., how many rows and variables) and the variables in your data set (such as name, data type, value labels). With RevoScaleR's summary statistics and cube functionality, you can examine summary information about your data and quickly plot histograms or relationships between variables.

RevoScaleR also provides all of the power of R to use in data transformations and manipulations. In RevoScaleR's data step functionality, you can specify R expressions to transform specific variables and have them automatically applied to a single data frame or to each block of data as it is read in from an .xdf file. You can create new variables, recode variables, and set missing values with all the flexibility of the R language.

Statistical Analysis

In addition to descriptive statistics and crosstabs, RevoScaleR provides functions for fitting linear and binary logistic regression models, generalized linear models, k-means models, and decision trees and forests, among others. These functions access .xdf files or other data sources directly or operate on data frames in memory. Because these functions are so efficient and do not require that all of the data be in memory at one time, you can analyze huge data sets without requiring huge computing power. In particular, you can relax assumptions previously required. For example, instead of assuming a linear or polynomial functional form in a model, you can break independent variables into many categories providing a completely flexible functional form. The many degrees of freedom provided by large data sets, combined with RevoScaleR's efficiency, make this approach feasible.

Writing Your Own Analyses for Large Data Sets

All of the main analysis functions in RevoScaleR use updating or external memory algorithms, that is, they analyze a chunk of data, update the results, then move on to the next chunk of data and repeat the process. When all the data has been processed (sometimes multiple times), final results can be calculated and the analysis is complete. You can write your own functions to process a chunk of data and update results, and use RevoScaleR functionality to provide you with access to your data file chunk by chunk. For more information, see [Write custom chunking algorithms in RevoScaleR](#).

See Also

[Introduction to Machine Learning Server](#)

[How-to guides for Machine Learning Server RevoScaleR Functions](#)

How-to guides for data analysis and operationalization

7/12/2022 • 2 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

This section of the Machine Learning Server documentation is for data scientists, analysts, and statisticians. The focus of this content area is on data acquisition, transformation and manipulation, visualization, and analysis in R and Python, as well as the deployment and consumption of models and code. It provides step-by-step guidance for common tasks leveraging the libraries and packages in Machine Learning Server.

If you are new to R, be sure to also use the R Core Team manuals that are part of every R distribution, including *An Introduction to R*, *The R Language Definition*, *Writing R Extensions* and so on. Beyond the standard R manuals, there are many other resources. [Learn about them here](#).

How-to guidance

Data analysis

- [Data acquisition](#)
- [Data summaries](#)
- [Models in ScaleR](#)
- [Crosstabs](#)
- [Linear models](#)
- [Logistic regression](#)
- [Generalized linear](#)
- [Decision trees](#)
- [Decision forest](#)
- [Stochastic gradient boosting](#)
- [Naïve Bayes classifier](#)
- [Correlation and variance/covariance matrices](#)
- [Clustering](#)
- [Converting RevoScaleR model objects for use in PMML](#)
- [Transform functions](#)
- [Visualizing huge data sets](#)

Remote code execution on Machine Learning Server

- [Connect to remote server](#)
- [Create remote session & execute](#)

Operationalization: deploy and consume models and code

- [Publish & manage](#)
- [Consume \(request-response\)](#)

- [Consume \(asynchronous\)](#)
- [Integrate into apps](#)
- [Manage access tokens](#)

See Also

[Learning Resources](#)

[Machine Learning Server](#)

How to use revoscalepy in a Spark compute context

7/12/2022 • 3 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

This article introduces Python functions in a [revoscalepy](#) package with Apache Spark (Spark) running on a Hadoop cluster. Within a Spark cluster, Machine Learning Server leverages these components:

- Hadoop distributed file system for finding and accessing data.
- Yarn for job scheduling and management.
- Spark as the processing framework (versions 2.0-2.1).

The revoscalepy library provides cluster-aware Python functions for data management, predictive analytics, and visualization. When you set the [compute context to rx-spark-connect](#), revoscalepy automatically distributes the workload across all the data nodes. There is no overhead in managing jobs or the queue, or tracking the physical location of data in HDFS; Spark does both for you.

NOTE

For installation instructions, see [Install Machine Learning Server for Hadoop](#).

Start Python

On your cluster's edge node, start a session by typing `mlserver-python` at the command line.

Local compute context on Spark

By default, the local compute context is the implicit computing environment. All `mlserver-python` code runs here until you specify a remote compute context.

Remote compute context on Spark

From the edge node, you can push computations to the data layer by creating a remote Spark compute context. In this context, execution is on all data nodes.

The following example shows how to set a remote compute context to clustered data nodes, execute functions in the Spark compute context, switch back to a local compute context, and disconnect from the server.

```

# Load the functions
from revoscalepy import RxOrcData, rx_spark_connect, rx_spark_list_data, rx_lin_mod, rx_spark_cache_data

# Create a remote compute context
cc = rx_spark_connect()

# Create a col_info object specifying the factors
col_info = {"DayOfWeek": {"type": "factor"}}

# Load data, factored and cached.
df = RxOrcData(file = "/share/sample_data/AirlineDemoSmallOrc", column_info = col_info)
df = rx_spark_cache_data(df, True)

# After the first run, a Spark data object is added into the list
rx_lin_mod("ArrDelay ~ DayOfWeek", data = df)
rx_spark_list_data(True)

# Disconnect. Switches back to a local compute context.
rx_spark_disconnect(cc)
rx_get_compute_context()

```

Specify a data source and location

As part of execution in Spark, your data source must be a file format that Spark understands, such as text, Hive, Orc, and Parquet. You can also create and consume [.xdf files](#), a data file format native to Machine Learning Server that you can read or write to from both Python and R script.

Data source objects provided by revoscalepy in a Spark compute context include [RxTextData](#) [RxXdfData](#), and the [RxSparkData](#) with derivatives for RxHiveData, RxOrcData, RxParquetData and RxSparkDataFrame.

Create a data source

The following example illustrates an Xdf data source object that pulls data from a local sample directory created when you install Machine Learning Server. The "sampleDataDir" argument is a reference to the sampleDataDir folder, known to revoscalepy.

```

import os
import revoscalepy

sample_data_path = revoscalepy.RxOptions.get_option("sampleDataDir")
d_s = revoscalepy.RxXdfData(os.path.join(sample_data_path, "AirlineDemoSmall.xdf"))

```

Import data into a data frame

Data is automatically loaded into a data frame even without `rx_import`, but you can load it explicitly using the `rx_import`, which is useful if you want to include parameters.

In mlserver-python, you can use `head` and `tail` functions, similar to R, to return the first or last part of the data set.

```

airlinedata = rx_import(input_data = d_s, outFile="/tmp/airExample.xdf")
airlinedata.head()

```

Summarize data

To quickly understand fundamental characteristics of your data, use the `rx_summary` function to return basic statistical descriptors. Mean, standard deviation, and min-max values. A count of total observations, missing observations, and valid observations is included.

A minimum specification of the `rx_summary` function consists of a valid data source object and a formula giving the fields to summarize. The formula is symbolic, providing variables used in the model. and typically does not contain a response variable. It should be of the form of `~` terms.

TIP

Get the term list from a data source to see what is available: `revoscalepy.rx_get_var_names(data_source)`

```
import os
from revoscalepy import rx_summary, RxOptions, RxXdfData
sample_data_path = RxOptions.get_option("sampleDataDir")
ds = RxXdfData(os.path.join(sample_data_path, "AirlineDemoSmall.xdf"))
summary = rx_summary("ArrDelay+DayOfWeek", ds)
print(summary)
```

Create models

The following example produces a linear regression, followed by predicted values for the linear regression model.

```
# Linear regression
import os
import tempfile
from revoscalepy import RxOptions, RxXdfData, rx_lin_mod

sample_data_path = RxOptions.get_option("sampleDataDir")
in_mort_ds = RxXdfData(os.path.join(sample_data_path, "mortDefaultSmall.xdf"))

lin_mod = rx_lin_mod("creditScore ~ yearsEmploy", in_mort_ds)
print(lin_mod)
```

```
# Add predicted values
import os
from revoscalepy import RxOptions, RxXdfData, rx_lin_mod, rx_predict, rx_data_step

sample_data_path = RxOptions.get_option("sampleDataDir")
mort_ds = RxXdfData(os.path.join(sample_data_path, "mortDefaultSmall.xdf"))
mort_df = rx_data_step(mort_ds)

lin_mod = rx_lin_mod("creditScore ~ yearsEmploy", mort_df)
pred = rx_predict(lin_mod, data = mort_df)
print(pred.head())
```

See Also

- [Python function reference](#)
- [microsoftml function reference](#)
- [revoscalepy function reference](#)

How to set and manage compute context in Machine Learning Server

7/12/2022 • 8 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

In Machine Learning Server, every session that loads a function library has a [compute context](#). The default is **local**, available on all platforms. No action is required to use a local compute context.

This article explains how to shift script execution to a **remote** Hadoop or Spark cluster, for the purpose of bringing calculations to the data itself and eliminating data transfer over your network. For SQL Server compute contexts, see [Define and use a compute context](#) in the SQL Server documentation.

You can create multiple compute context objects: just use them one at a time. Often, functions operate identically in local and remote context. If script execution is successful locally, you can generally expect the same results on the remote server, subject to these [limitations](#).

Prerequisites

The remote computer must be at the same functional level as the system sending the request. A remote Machine Learning Server 9.2.1 can accept a compute context shift from another 9.2.1 interpreter on Machine Learning Server, a SQL Server 2017 Machine Learning Services instance, and for R developers on Microsoft R Client at the same functional level (3.4.1).

The following table is a recap of platform and data source requirements for each function library.

LIBRARY	PLATFORMS & DATA SOURCES
RevoScaleR	Spark 2.0-2.1 over HDFS, Hadoop MapReduce: Hive, Orc, Parquet, Text, XDF, ODBC SQL Server: tables, views, local text and .xdf files ⁽¹⁾ , ODBC data sources
revoscalepy	Spark 2.0-2.1 over HDFS: Hive, Orc, Parquet, Text, XDF, ODBC SQL Server 2017 Machine Learning with Python: tables, views, local text and .xdf files ⁽¹⁾ , ODBC data sources

⁽¹⁾ You can load text or .xdf files locally, but be aware that code and data runs on SQL Server, which results in [implicit data type conversions](#).

NOTE

RevoScaleR is available in both Machine Learning Server and R Client. You can develop script in R Client for execution on the server. However, because R Client is limited to two threads for processing and in-memory datasets, scripts might require deeper customizations if datasets are large and come with dependencies on data chunking. Chunking is not supported in R Client. In R Client, the `blocksPerRead` argument is ignored and all data is read into memory. Large datasets that exceed memory must be pushed to a compute context of a Machine Learning Server instance that provides data chunking.

Get a compute context

As a starting point, practice the syntax for returning the local compute context. Every platform, product, and language supports the default local compute context.

```
# RevoScaleR is loaded automatically so no import step is required  
rxGetComputeContext()
```

```
from revoscalepy import RxLocalSeq  
localcc = RxLocalSeq()
```

For both languages, the return object should be Return object should be `RxLocalSeq`.

Set a remote compute context

This section uses examples to illustrate the syntax for setting compute context.

For Spark

The compute context used to distribute computations on a Hadoop Spark 2.0-2.1 cluster. For more information, see the [How to use RevoScaleR with Spark](#).

```
myHadoopCluster <- RxSpark(myHadoopCluster)
```

For MapReduce

The compute context used to distribute computations on a Hadoop MapReduce cluster. This compute context can be used on a node (including an edge node) of a Cloudera or Hortonworks cluster with an RHEL operating system, or a client with an SSH connection to such a cluster. For details on creating and using `RxHadoopMR` compute contexts, see the [How to use RevoScaleR with Hadoop](#)

```
myHadoopCluster <- RxHadoopMR(myHadoopCluster)
```

For SQL Server (in-database)

The `RxInSqlServer` compute context is a special case in that it runs computations in-database, but it runs on only a single database node, so the computation is parallel, but not distributed.

For setting up a remote compute context to SQL Server, we provide an example below, but point you to [Define and use a compute context](#) in the SQL Server documentation for additional instructions.

```

# Requires RevoScaleR and SQL Server on same machine
connectionString <- "Server=(placeholder-server-name);Database=RevoAirlineDB;Trusted_Connection=true"

sqlQuery <- "WITH nb AS (SELECT 0 AS n UNION ALL SELECT n+1 FROM nb where n < 9) SELECT
n1.n+10*n2.n+100*n3.n+1 AS n, ABS(CHECKSUM(NewId()))"

myServer <- RxComputeContext("RxInSqlServer", sqlQuery = sqlQuery, connectionString = connectionString)

rxSetComputeContext(computeContext = myServer)

```

Limitations of remote compute context

A primary benefit of remote computing is to perform distributed analysis using the inherent capabilities of the host platform. For a list of analytical functions that are multithreaded and cluster aware, see [Running distributed analyses using RevoScaleR](#). As a counter point, the concept of "remote compute context" is not optimized for data manipulation and transformation workloads. As such, many data-related functions come with local compute requirements. The following table describes the exceptions to remote computing.

LIMIT	DETAILS
Script execution of open-source routines	Scripts use a combination of open-source and proprietary functions. Only revoscalepy and RevoScaleR functions, with the respective interpreters, are multithreaded and distributable. Open-source routines in your script still run locally in a single threaded process.
Single-threaded proprietary functions	Analyses that do not run in parallel include import and export functions , data transformation functions , graphing functions .
Date location and operational limits	Sort and merge must be local. Import is multithreaded, but not cluster-aware. If you execute rxImport in a Hadoop cluster, it runs multithreaded on the current node. Data may have to be moved back and forth between the local and remote environments during the course of overall program execution. Except for file copying in Hadoop, RevoScaleR does not include functions for moving data.

Set a no-wait compute context

By default, all jobs are "waiting jobs" or "blocking jobs", where control of the command prompt is not returned until the job is complete. For jobs that complete quickly, this is an appropriate choice.

For large jobs that run longer, execute the job but use a non-waiting compute context to continue working in your local session. In this case, you can specify the compute context to be *non-waiting* (or *non-blocking*), in which case an object containing information about the pending job is used to retrieve results later.

To set the compute context object to run "no wait" jobs, set the argument *wait* to *FALSE*.

```
myHadoopCluster <- RxSpark(myHadoopCluster, wait=FALSE)
```

Another use for non-waiting compute contexts is for massively parallel jobs involving multiple clusters. You can define a non-waiting compute context on each cluster, launch all your jobs, then aggregate the results from all the jobs once they complete.

For more information, see [Non-Waiting Jobs](#).

Retrieve cluster console output

If you want console output from each of the cluster R processes printed to your user console, specify `consoleOutput=TRUE` in your compute context.

```
myHadoopCluster <- RxSpark(myHadoopCluster, consoleOutput=TRUE)
```

Update a compute context

Once you have a compute context object, modify it using the same function that originally creates it. Pass the name of the original object as its first argument, and then specify only those arguments you wish to modify as additional arguments.

For example, to change only the `suppressWarning` parameter of a Spark compute context `myHadoopCluster` from TRUE to FALSE:

```
myHadoopCluster <- RxSpark(myHadoopCluster, suppressWarning=FALSE)
```

To list parameters and default values, use the `args` function with the name of the compute context constructor, for example:

```
args(RxSpark)
```

which gives the following output:

```
function (object, hdfsShareDir = paste("/user/RevoShare", Sys.info()[[["user"]]],
sep = "/"), shareDir = paste("/var/RevoShare", Sys.info()[[["user"]]],
sep = "/"), clientShareDir = rxGetDefaultTmpDirByOS(), sshUsername = Sys.info()[[["user"]]],
sshHostname = NULL, sshSwitches = "", sshProfileScript = NULL,
sshClientDir = "", nameNode = rxGetOption("hdfsHost"), jobTrackerURL = NULL,
port = rxGetOption("hdfsPort"), onClusterNode = NULL, wait = TRUE,
numExecutors = rxGetOption("spark.numExecutors"), executorCores = rxGetOption("spark.executorCores"),
executorMem = rxGetOption("spark.executorMem"), driverMem = "4g",
executorOverheadMem = rxGetOption("spark.executorOverheadMem"),
extraSparkConfig = "", persistentRun = FALSE, sparkReduceMethod = "auto",
idleTimeout = 3600, suppressWarning = TRUE, consoleOutput = FALSE,
showOutputWhileWaiting = TRUE, autoCleanup = TRUE, workingDir = NULL,
dataPath = NULL, outDataPath = NULL, fileSystem = NULL, packagesToLoad = NULL,
resultsTimeout = 15, ...)
```

You can temporarily modify an existing compute context and set the modified context as the current compute context by calling `rxSetComputeContext`. For example, if you have defined `myHadoopCluster` to be a waiting cluster and want to set the current compute context to be non-waiting, you can call `rxSetComputeContext` as follows:

```
rxSetComputeContext(myHadoopCluster, wait=FALSE)
```

The `rxSetComputeContext` function returns the previous compute context, so it can be used in constructions like the following:

```
oldContext <- rxSetComputeContext(myCluster, wait=FALSE)
...
# do some computing with a non-waiting compute context
...
# restore previous compute context
rxSetComputeContext(oldContext)
```

You can specify the compute context by name, as we have done here, but you can also specify it by calling a compute context constructor in the call to `rxSetComputeContext`. For example, to return to the local sequential compute context after using a cluster context, you can call `rxSetComputeContext` as follows:

```
rxSetComputeContext(RxLocalSeq())
```

In this case, you can also use the descriptive string "local" to do the same thing:

```
rxSetComputeContext("local")
```

Create additional compute contexts

Given a set of distributed computing resources, you might want to create multiple compute context objects to vary the configuration. For example, you might have one compute context for waiting or blocking jobs and another for no-wait or non-blocking jobs. Or you might define one that uses all available nodes and another that specifies a particular set of nodes.

Because the initial specification of a compute context can be somewhat tedious, it is usually simplest to create additional compute contexts by modifying an existing compute context, in precisely the same way as we updated a compute context in the previous section. For example, suppose instead of simply modifying our existing compute context from `wait=TRUE` to `wait=FALSE`, we create a new compute context for non-waiting jobs:

```
myNoWaitCluster <- RxSpark(myHadoopCluster, wait=FALSE)
```

TIP

Store commonly used compute context objects in an R script, or add their definitions to an R startup file.

See also

- [Introduction to Machine Learning Server](#)
- [Distributed computing](#)
- [Compute context](#)
- [Define and use a compute context in SQL Server](#)

Data transformations using RevoScaleR functions (Machine Learning Server)

7/12/2022 • 6 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Loading an initial dataset is typically just the first step in a continuum of data manipulation tasks that persist until your project is completed. Common examples of data manipulation include isolating a subset, slicing a dataset by variables or rows, and converting or merging variables into new forms, often resulting in new data structures along the way.

There are two main approaches to data transformation using the RevoScaleR library:

- Define an external based R *function* and reference it.
- Define an embedded transformation as an input to a *transforms* argument on another RevoScaleR function.

Embedded transformations are supported in **rxImport**, **rxDataStep**, and in analysis functions like **rxLinMod** and **rxCube**, to name a few. Embedded transformations are easier to work with, but external functions allow for a greater degree of complexity and reuse. The following section provides an illustration of both techniques.

Compare approaches

Embedded transformations provide instructions within a formula, through arguments on a function. Using just arguments, you can manipulate data using *transformations*, or select a rowset with the *rowSelection* argument.

Externally defined functions provide data manipulation instructions in an outer *function*, which is then referenced by a RevoScaleR function. An external transformation function is one that takes as input a list of variables and returns a list of (possibly different) variables. Due to the external definition, the object can assume a complex structure and be repurposed across multiple functions supporting transformation.

```

# Load data
> censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers.xdf")

# Option 1: Construct a new variable using an embedded transformation argument
> NewDS <- rxDataStep (inData = censusWorkers, outFile = "c:/temp/newCensusWorkers.xdf",
  transforms = list(ageFactor = cut(age, breaks=seq(from = 20, to = 70, by = 5),
  right = FALSE)), overwrite=TRUE)

# Return variable metadata; ageFactor is a new variable
> rxGetVarInfo(NewDS)

# Option 2: Construct a new variable using an external function and rxDataStep
> ageTransform <- function(dataList)
{
  dataList$ageFactor <- cut(dataList$age, breaks=seq(from = 20, to = 70,
    by = 5), right = FALSE)
  return(dataList)
}

> NewDS <- rxDataStep(inData = censusWorkers, outFile = "c:/temp/newCensusWorkers.xdf",
  transformFunc = ageTransform, transformVars=c("age"), overwrite=TRUE)

# Return variable metadata; it is identical to that of option 1
> rxGetVarInfo(NewDS)

```

For more examples of both approaches, see [How to transform and subset data](#).

Arguments used in transformations

To specify an external function, use the *transformFunc* argument to most RevoScaleR functions. To pass a list of variables to the function, use the *transformVars* argument.

The following table numerates the arguments used in data manipulation tasks.

ARGUMENT	USAGE
<i>transforms</i>	A formula for manipulating or transforming data, passed as an argument to a RevoScaleR function like rxImport or rxDataStep . This expression can be defined outside of the function call using the expression function.
<i>rowSelection</i>	Criteria for selecting a subset of rows in the current data frame.
<i>function</i>	A base R class used to create functions in R script.
<i>transformFunc</i>	Assigns a function to a RevoScaleR function supporting data manipulation and transformations. If this argument is specified, it is evaluated before any other transformations specified in the <i>transforms</i> , <i>rowSelection</i> , or <i>formula</i> arguments. The list of variables to be passed to the transform function is specified as the <i>transformVars</i> argument.
<i>transformVars</i>	A list of variables to pass to your external transform function.
<i>transformObjects</i>	A named list of objects that can be referenced by <i>transforms</i> , <i>transformFunc</i> , or <i>rowSelection</i> arguments.

ARGUMENT	USAGE
<code>transformPackages</code>	A list of packages, in addition to those specified in <code>rxGetOption("transformPackages")</code> , to be preloaded, that provide libraries and functions used in <code>transforms</code> , <code>transformFunc</code> or <code>rowSelection</code> arguments.
<code>transformEnvir</code>	A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation.

Functions supporting transformations

The following functions support embedded transformations or reference an external transformation function.

FUNCTION	USE CASE
<code>rxDataStep</code>	Create a subset rows or variables, or create new variables by transforming existing variables. Also used for easy conversion between data in data frames and .xdf files.
<code>rxImport</code>	Invoke a transformation while loading data into a data frame or .xdf file.
<code>rxSummary</code>	Transform data inline, while computing summary statistics.
<code>rxLinMod</code>	Create or transform variables used in the data set of a linear regression.
<code>rxLogit</code>	Create or transform variables used in the data set of a logistic regression.
<code>rxCube</code>	Create new variables or transform an existing variable used to create the list of variables in <code>rxCube</code> output.

Other functions don't accept transformation logic, but are used for data manipulation operations:

- `rxSetVarInfo` Change variable information, such as the name or description, in an .xdf file or data frame.
- `rxSetInfo` Add or change a data set description.
- `rxFactors` Add or change a factor levels
- `rxMerge` Combines two or more datasets.
- `rxSort` Orders a dataset based on some criteria

How transforms are evaluated

User-defined transforms and transform functions are evaluated in essentially the same way. User-defined transforms are combined into an implicit transform function for evaluation purposes. At evaluation time, the process is the same whether the function is predefined or user-defined.

An evaluation environment is constructed from the base environment together with the utils, stats, methods packages, any packages specified with the `transformPackages` argument, and any specified `transformObjects`, and the closure of the transform function.

Functions are then evaluated in the context of this environment. Functions that are in packages but not part of the evaluation environment can be used if fully qualified (specifically, prefixed by the package name and two or

three colons, depending on whether the function is exported).

If you are using methods packages, you can modify the basic list by setting the `rxOption transformPackages` argument.

Transformations to avoid

Transform functions are very powerful, but there are four types of transformation that should be avoided:

1. Transformations that change the length of a variable. This includes, naturally, most model-fitting functions.
2. Transformations that depend upon all observations simultaneously. Because RevoScaleR works on chunks of data, such transformations will not have access to all the data at once. Examples of such transformations are matrix operations such as `poly` or `solve`.
3. Transformations that have the possibility of creating different mappings of factor codes to factor labels.
4. Transformations that involve sampling with replacement. Again, this is because RevoScaleR works on chunks of data, and sampling with replacement chunk by chunk is not equivalent to sampling with replacement from the full data set.

If you change the length of one variable, you will get errors from subsequent analysis functions that not all variables are the same length. If you try to change the length of all variables (essentially, performing some sort of row selection), you need to pass all of your data through the transform function, and this can be very inefficient. To avoid this problem, use row selection to work on data one slice at a time.

If you create a factor within a transformation function, you may get unexpected results because all of the data is not in memory at one time. When creating a factor within a transformation function, you should always explicitly set the values and labels. For example:

```
dataList$xfac <- as.factor(dataList$x, levels = c(1, 2,3),
  labels = c("One", "Two", "Three"))
```

Next Steps

Continue on to the following data-related articles to learn more about XDF, data source objects, and other data formats:

- [How to transform and subset data](#)
- [XDF files](#)
- [Data Sources](#)
- [Import text data](#)
- [Import ODBC data](#)
- [Import and consume data on HDFS](#)

See Also

[RevoScaleR Functions](#)

[Tutorial: data import and exploration](#) [Tutorial: data visualization and analysis](#)

Create an XDF file in Machine Learning Server

7/12/2022 • 10 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

XDF is the native file format for persisted data in Machine Learning Server and it offers the following benefits:

- Compression, applied when the file is written to disk.
- Columnar storage, one column per variable, for efficient read-write operations of variable data. In data science and machine learning, variables rather than rowsets are the data structures typically used in analysis.
- Modular data access and management so that you can work with chunks of data at a time.

XDF files are not strictly required for statistical analysis and data mining, but when data sets are large or complex, XDF offers stability in the form of persisted data under your control, plus the ability to subset and transform data for repeated analysis.

To create an XDF file, use the `rxImport` function in RevoScaleR to pipe external data to Machine Learning Server. By default, `rxImport` loads data into an in-memory data frame, but by specifying the `outFile` parameter, `rxImport` creates an XDF file.

Example: Create an XDF

You can create an XDF using any data that can be loaded by `rxImport`, and by specifying an `outFile` consisting of a file path to a writable directory.

This example uses an R console and [sample data](#) to create an XDF using data from a single CSV file. On Windows, you can run `Rgui.exe`, located at `\Program Files\Microsoft\ML Server\R_SERVER\bin\x64`. On Linux, you can type `Revo64` at the command line.

```
# Set the source file location using inData argument  
> mysourcedata <- file.path(rxGetOption("sampleDataDir", "mortDefaultSmall2000.csv"))  
  
# Set the XDF file location using the outFile argument  
> myNewXdf <- file.path("C:/users/temp/mortDefaultSmall2000.xdf")
```

Notice the direction of the path delimiter. By default, R script uses forward slashes as path delimiters.

At this point, the object is created, but the XDF file won't exist until you run `rxImport`.

```
# Create an XDF  
> rxImport(inData = mysourcedata, outFile = myNewXdf)
```

`rxImport` creates the file, builds and populates columns for each variable in the dataset, and then computes metadata for each variable and the XDF as a whole.

Output returned from this operation is as follows:

```
Rows Read: 10000, Total Rows Processed: 10000, Total Chunk Time: 0.022 seconds
```

Use the `rxGetInfo` function to return information about an object. In this case, the object is the XDF file created in a previous step, and the information returned is the precomputed metadata for each variable, plus a summary of observations, variables, blocks, and compression information.

```
rxGetInfo(mySmallXdf, getVarInfo = TRUE)
```

Output is below. Variables are based on fields in the CSV file. In this case, there are 6 variables. Precomputed metadata about each one appears in the output below.

```
File name: C:\Users\TEMP\mortDefaultSmall2000.xdf
Number of observations: 10000
Number of variables: 6
Number of blocks: 1
Compression type: zlib
Variable information:
Var 1: creditScore, Type: integer, Low/High: (486, 895)
Var 2: houseAge, Type: integer, Low/High: (0, 40)
Var 3: yearsEmploy, Type: integer, Low/High: (0, 14)
Var 4: ccDebt, Type: integer, Low/High: (0, 12275)
Var 5: year, Type: integer, Low/High: (2000, 2000)
Var 6: default, Type: integer, Low/High: (0, 1)
```

Set compression levels

You can set data compression via the `xdfCompressionLevel` argument to `rxImport` (and most other RevoScaleR functions that write .xdf files). You specify this as an integer in the range -1 to 9. The value -1 tells `rxImport` to use the current default compression value. The integers 1 through 9 specify increasing levels of compression, where higher numbers perform more compression, but take more time. The value 0 specifies no compression.

The .xdf format allows different blocks to have different compression levels (but not within a single call to `rxImport`). This can be useful, for example, when appending to an existing data set of unknown compression level. You can specify the compression for the new data without affecting the compression of the existing data.

You can specify a standard compression level for all future .xdf file writes by setting `xdfCompressionLevel` using `rxOptions`. For example, to specify a compression level of 3, use `rxOptions` as follows:

```
> rxOptions(xdfCompressionLevel = 3)
```

The default value of this option is 1.

If you have one or more existing .xdf files and would like to compress them, you can use the function `rxCompressXdf`. You can specify a single file or xdf data source, a character vector of files and data sources, or the path to a directory containing .xdf files. For example, to compress all the .xdf files in the C:\data directory, you would call `rxCompressXdf` as follows:

```
> rxCompressXdf("C:/data", xdfCompressionLevel = 1, overwrite = TRUE)
```

Append new observations

If you have observations on the same variables in multiple input files, you can use the `append` argument to `rxImport` to combine them into one file. For example, we could append another copy of the claims text data set

in a second block to the claimCAOrdered2.xdf file:

```
# Appending to an Existing File

inFile <- file.path(rxGetOption("sampleDataDir"), "claims.txt")
colInfoList <- list("car.age" = list(type = "factor", levels = c("0-3",
  "4-7", "8-9", "10+")))
outfileCAOrdered2 <- "claimsCAOrdered2.xdf"

claimsAppend <- rxImport(inFile, outFile = outfileCAOrdered2,
  colClasses = c(number = "integer"),
  colInfo = colInfoList, stringsAsFactors = TRUE, append = "rows")
rxGetInfo(claimsAppend, getVarInfo=TRUE)

File name: C:\YourOutputPath\claimsCAOrdered2.xdf
Number of observations: 256
Number of variables: 6
Number of blocks: 2
Compression type: zlib
Variable information:
Var 1: RowNum, Type: integer, Low/High: (1, 128)
Var 2: age
  8 factor levels: 17-20 21-24 25-29 30-34 35-39 40-49 50-59 60+
Var 3: car.age
  4 factor levels: 0-3 4-7 8-9 10+
Var 4: type
  4 factor levels: A B C D
Var 5: cost, Type: numeric, Storage: float32, Low/High: (11.0000, 850.0000)
Var 6: number, Type: integer, Low/High: (0, 434)
```

Read XDF data into a data frame

It is often convenient to store a large amount of data in an .xdf file and then read a subset of columns and rows of the data into a data frame in memory for analysis. The **rxDataStep** function makes this easy. For example, let's consider taking subsamples from the sample data set CensusWorkers.xdf. Using a *rowSelection* expression and list of *varsToKeep*, we can extract the *age*, *perwt*, and *sex* variables for individuals over the age of 40 living in Washington State:

```
# Reading Data from an .xdf File into a Data Frame

inFile <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers.xdf")
myCensusDF <- rxDataStep(inData=inFile,
  rowSelection = state == "Washington" & age > 40,
  varsToKeep = c("age", "perwt", "sex"))
```

When subsampling rows, we need to be aware that the *rowSelection* is processed on each chunk of data after it is read in. Consider an example where we want to extract every 10th row of the data set. For each chunk we will create a sequence starting with the start row number in that chunk (provided by the internal variable, *.rxStartRow*) with a length equal to the number of rows in the data chunk. We will determine that number of rows by using the length of one of the variables that has been read from the data set, *age*. We will keep only the rows where the remainder after dividing the row number by 10 is 0:

```
myCensusSample <- rxDataStep(inData=inFile,
  rowSelection= (seq(from=.rxStartRow,length.out=.rxNumRows) %% 10) == 0 )
```

We can also create transformed variables while we are reading in the data. For example, create an 10-year *ageGroup* variable, starting with 20 (the minimum age in the data set):

```
myCensusDF2 <- rxDataStep(inData=inFile,
  varsToKeep = c("age", "perwt", "sex"),
  transforms=list(ageGroup = cut(age, seq(from=20, to=70, by=10))))
```

Split an XDF into multiple files

RevoScaleR makes it possible to analyze huge data sets easily and efficiently, and for most purposes the most efficient computations are done on a single .xdf file. However, there are many circumstances when you will want to work with only a portion of your data. For example, you may want to distribute your data over the nodes of a cluster; in such a case, RevoScaleR's analysis functions will process each node's data separately, combining all the results for the final return value. You might also want to split your data into training and test data so that you can fit a model using the training data and validate it using the test data.

Use the function **rxSplit** to split your data. For example, to split variables of interest in the large 2000 U.S. Census data into five files for distribution on a five node cluster, you could use **rxSplit** as follows (change the location of the **bigDataDir** to the location of your downloaded file):

```
# Splitting Data Files
rxSetComputeContext("local")
bigDataDir <- "C:/MRS/Data"
bigCensusData <- file.path(bigDataDir, "Census5PCT2000.xdf")
splitFiles <- rxSplit(bigCensusData, numOutFiles = 5, splitBy = "blocks",
  varsToKeep = c("age", "incearn", "incwelfr", "educrec", "metro", "perwt"))
names(splitFiles)
```

By default, **rxSplit** simply appends a number in the sequence from 1 to *numOutFiles* to the base file name to create the new file names, and in this case the resulting file names, for example, "Census5PCT20001.xdf", are a bit confusing.

You can exercise greater control over the output file names by using the *outFilesBase* and *outFilesSuffixes* arguments. With *outFilesBase*, you can specify either a single character string to be used for all files or a character vector the same length as the desired number of files. The latter option is useful, for example, if you would like to create four files with the same file name, but different paths:

```
nodePaths <- paste("compute", 10:13, sep="")
baseNames <- file.path("C:", nodePaths, "DistCensusData")
splitFiles2 <- rxSplit(bigCensusData, splitBy = "blocks",
  outFilesBase = baseNames,
  varsToKeep = c("age", "incearn", "incwelfr", "educrec", "metro", "perwt"))
names(splitFiles2)
```

This creates the four directories C:/compute10, etc., and creates a file named "DistCensusData.xdf" in each directory. You should adopt an approach like this when using distributed data with the standard RevoScaleR analysis functions such as **rxLinMod** and **rxLogit** in an **RxSpark** or **RxHadoopMR** compute context.

You can supply the *outFilesSuffixes* arguments to exercise greater control over what is appended to the end of each file. Returning to our first example, we can add a hyphen between our base file name and the sequence 1 to 5 using *outFilesSuffixes* as follows:

```
splitFiles3 <- rxSplit(bigCensusData, splitBy = "blocks",
  outFilesSuffixes=paste("-", 1:5, sep=""),
  varsToKeep = c("age", "incearn", "incwelfr", "educrec", "metro", "perwt"))
names(splitFiles3)
```

The *splitBy* argument specifies whether to split your data file row-by-row or block-by-block. The default is

`splitBy="rows"`, which distributes data from each block into different files. The examples above use the faster split by blocks instead. The `splitBy` argument is ignored if you also specify the `splitByFactor` argument as a character string representing a valid factor variable. In this case, one file is created per level of the factor variable.

You can use the `splitByFactor` argument and a `transforms` argument to easily create test and training data sets from an .xdf file. Note that this will take longer than the previous examples because each block of data is being processed:

```
splitFiles4 <- rxSplit(inData = bigCensusData,
  outFilesBase="censusData",
  splitByFactor="testSplitVar",
  varsToKeep = c("age", "incearn", "incwelfr", "educrec", "metro", "perwt"),
  transforms=list(testSplitVar = factor(
    sample(0:1, size=.rxNumRows, replace=TRUE, prob=c(.10, .9)),
    levels=0:1, labels = c("Test", "Train"))),
  names(splitFiles4)
rxSummary(~age, data = splitFiles4[[1]], reportProgress = 0)
rxSummary(~age, data = splitFiles4[[2]], reportProgress = 0)
```

This takes approximately 10% of the data as a test data set, with the remainder going into the training data.

If your .xdf file is relatively small, you may want to set `outFilesBase = ""` so that a list of data frames is returned instead of having files created. You can also use `rxSplit` to split data frames (see the `rxSplit` help page for details).

Re-Block an .xdf File

After a series of data import or row selection steps, you may find that you have an .xdf file with very uneven block sizes. This may make it difficult to efficiently perform computations by "chunk." To find the sizes of the blocks in your .xdf file, use `rxGetInfo` with the `getBlockSizes` argument set to TRUE. For example, let's look at the block sizes for the sample CensusWorkers.xdf file:

```
# Re-Blocking an .xdf File

fileName <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers.xdf")
rxGetInfo(fileName, getBlockSizes = TRUE)
```

The following information is provided:

```
File name: C:\Program Files\Microsoft\MRO-for-RRE\8.0\R-3.2.2\
library\RevoScaleR\SampleData\CensusWorkers.xdf
Number of observations: 351121
Number of variables: 6
Number of blocks: 6
Compression type: zlib
Rows per block: 95420 42503 1799 131234 34726 45439
```

We see that, in fact, the number of rows per block varies from a low of 1799 to a high of 131,234. To create a new file with more even-sized blocks, use the `rowsPerRead` argument in `rxDataStep`:

```
newFile <- "censusWorkersEvenBlocks.xdf"
rxDataStep(inData = fileName, outFile = newFile, rowsPerRead = 60000)
rxGetInfo(newFile, getBlockSizes = TRUE)
```

The new file has blocks sizes of 60,000 for all but the last slightly smaller block:

```
File name: C:\Users\...\censusWorkersEvenBlocks.xdf
Number of observations: 351121
Number of variables: 6
Number of blocks: 6
Compression type: zlib
Rows per block: 60000 60000 60000 60000 60000 51121
```

Next steps

XDF is optimized for distributed file storage and access in the Hadoop Distributed File System (HDFS). To learn more about using XDF in HDFS, see [Import and consume HDFS data files](#).

You can import multiple text files into a single XDF. For instructions, see [Import text data](#).

See also

[Machine Learning Server Install Machine Learning Server on Windows](#)

[Install Machine Learning Server on Linux](#)

[Install Machine Learning Server on Hadoop](#)

Importing text data in Machine Learning Server

7/12/2022 • 19 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

RevoScaleR can use data from a wide range of external data sources, including text files, database files on disk (SPSS and SAS), and relational data sources. This article puts the focus on text files: delimited (.csv) and fixed-format, plus database files accessed through simple file reads.

To store text data for analysis and visualization, you can load it into memory as a *data frame* for the duration of your session, or save it to disk as a .xdf file. For either approach, the RevoScaleR **rxImport** function loads the data.

NOTE

To get the best use of persisted data in XDF, you need Machine Learning Server (as opposed to R Client). Reading and writing *chunked data* on disk is exclusive to Machine Learning Server.

About rxImport

Converting external data into a format understood by RevoScaleR is achieved using the **rxImport** function. Although the function takes several arguments, it's just loading data from a source file that you provide. In the simplest case, you can give it a file path. If the data is delimited by commas or tabs, this is all that is required loading the data. To illustrate, the following example creates a data object loaded with data from a local text-delimited file:

```
> mydataobject <- rxImport("C:/user/temp/mydatafile.csv")
```

Depending on arguments, **rxImport** either loads data as a data frame, or outputs the data to a .xdf file saved to disk. This article covers a range of data access scenarios for text files and file-based data access of SPSS and SAS data. To learn more about other data sources, see related articles in the table of contents or in the link list at the end of this article.

How to import a text file

1. Set the location of the source file. One approach is to use R's `file.path` command.

```
> mySourceFile <- file.path("C:/Users/Temp/my-data-file.txt")
```

To try this out using [built-in samples](#), run the first command to verify the files are available, and the second command to set the location and source file.

```
# Verify the sample files exist and then set mySourceFile to the sample directory  
> list.files(rxGetOption("sampleDataDir"))  
> mySourceFile <- file.path(rxGetOption("sampleDataDir"), "claims.txt")
```

2. Run **rxImport** to load the data into a data frame.

```
> claimsDF <- rxImport(mySourceFile)
```

3. Follow up with the **rxGetInfo** function to get summary information. If we include the argument *getVarInfo=TRUE*, the summary includes the names of the variables and their types:

```
> rxGetInfo(claimsDF, getVarInfo = TRUE)  
  
Data frame: claimsDF  
Number of observations: 128  
Number of variables: 6  
Number of blocks: 1  
Variable information:  
Var 1: RowNum, Type: integer, Low/High: (1, 128)  
Var 2: age, Type: character  
Var 3: car.age, Type: character  
Var 4: type, Type: character  
Var 5: cost, Type: numeric, Storage: float32, Low/High: (11.0000, 850.0000)  
Var 6: number, Type: numeric, Storage: float32, Low/High: (0.0000, 434.0000)
```

For just variables in the data file, use the *names* function:

```
> names(claimsDF)  
[1] "RowNum"    "age"       "car.age"   "type"      "cost"      "number"
```

4. To save the data into a .xdf file rather than importing data into a data frame in memory, add the *outFile* parameter. Specify a path to a writable directory:

```
> claimsDF <- rxImport(inData=mySourceFile, outFile = "c:/users/temp/claims.xdf")
```

A .xdf file is created, and instead of returning a data frame, the **rxImport** function returns a data source object. This is a small R object that contains information about a data source, in this case the name and path of the .xdf file it represents. This data source object can be used as the input data in most RevoScaleR functions. For example:

```
> rxGetInfo(claimsDF, getVarInfo = TRUE)  
> names(claimsDF)
```

5. Optionally, you can simplify the path designation by using the working directory. Use R's working directory commands to get and set the folder. Run the first command to determine the default working directory, and the second command to switch to a writable folder.

```
> getwd()  
> setwd("c:/users/temp")
```

Modifications during import

During import, you can fix problems in the underlying data by specifying arguments for replacement values, changing metadata on the variable, changing data types, and creating new variables based on calculations.

Set a replacement string

If your text data file uses a string other than NA to identify missing values, you can use the *missingValueString* argument to define the replacement string. Only one missing value string is allowed per file. The following example is from the [Airline demo tutorial](#):

```
inFile <- file.path(rxGetOption("sampleDataDir"), "AirlineDemoSmall.csv")
airData <- rxImport(inData = inFile, outFile="airExample.xdf",
  stringsAsFactors = TRUE, missingValueString = "M",
  rowsPerRead = 200000, overwrite = TRUE)
```

Change variable metadata

If you need to modify the name of a variable or the names of the factor levels, or add a description of a variable, you can do this using the *colInfo* argument. For example, the claims data includes a variable *type* specifying the type of car, but the levels A, B, C, and D give us no particular information. If we knew what the types signified, perhaps "Subcompact", "Compact", "Mid-size", and "Full-size", we could relabel the levels as follows:

```
# Specifying Additional Variable Information

inFileAddVars <- file.path(rxGetOption("sampleDataDir"), "claims.txt")
outfileTypeRelabeled <- "claimsTypeRelabeled.xdf"
colInfoList <- list("type" = list(type = "factor", levels = c("A",
  "B", "C", "D"), newLevels=c("Subcompact", "Compact", "Mid-size",
  "Full-size"), description="Body Type"))
claimsNew <- rxImport(inFileAddVars, outFile = outfileTypeRelabeled,
  colInfo = colInfoList)
rxGetInfo(claimsNew, getVarInfo = TRUE)
```

This produces the following output:

```
File name: C:\YourOutputPath\claimsTypeRelabeled.xdf
Number of observations: 128
Number of variables: 6
Number of blocks: 1
Compression type: zlib
Variable information:
Var 1: RowNum, Type: integer, Low/High: (1, 128)
Var 2: age, Type: character
Var 3: car.age, Type: character
Var 4: type, Body Type
  4 factor levels: Subcompact Compact Mid-size Full-size
Var 5: cost, Type: numeric, Storage: float32, Low/High: (11.0000, 850.0000)
Var 6: number, Type: numeric, Storage: float32, Low/High: (0.0000, 434.0000)
```

To specify *newLevels*, you must also specify *levels*, and it is important to note that the *newLevels* argument can only be used to rename levels. It cannot be used to fully recode the factor. That is, the number of *levels* and number of *newLevels* must be the same.

Change data types

The **rxImport** function supports three arguments for specifying variable data types: *stringsAsFactors*, *colClasses*, and *colInfo*. For example, consider storing character data. Often data stored in text files as string data actually represents categorical or *factor* data, which can be more compactly represented as a set of integers denoting the distinct *levels* of the factor. This is common enough that users frequently want to transform *all* string data to factors. This can be done using the *stringsAsFactors* argument:

```

# Specifying Variable Data Types

inFile <- file.path(rxGetOption("sampleDataDir"), "claims.sts")
rxImport(inFile, outFile = "claimsSAF.xdf", stringsAsFactors = TRUE)
rxGetInfo("claimsSAF.xdf", getVarInfo = TRUE)

File name: C:\YourOutputPath\claimsSAF.xdf
Number of observations: 128
Number of variables: 6
Number of blocks: 1
Variable information:
Var 1: RowNum, Type: integer, Low/High: (1, 128)
Var 2: age
  8 factor levels: 17-20 21-24 25-29 30-34 35-39 40-49 50-59 60+
Var 3: car.age
  4 factor levels: 0-3 4-7 8-9 10+
Var 4: type
  4 factor levels: A B C D
Var 5: cost, Type: numeric, Storage: float32, Low/High: (11.0000, 850.0000)
Var 6: number, Type: numeric, Storage: float32, Low/High: (0.0000, 434.0000)

```

You can also specify data types for individual variables using the *colClasses* argument, and even more specific instructions for converting each variable using the *colInfo* argument. Here we use the *colClasses* argument to specify that the variable *number* in the claims data be stored as an integer:

```

outfileColClass <- "claimsCCNum.xdf"
rxImport(infile, outFile = outfileColClass, colClasses=c(number = "integer"))
rxGetInfo("claimsCCNum.xdf", getVarInfo = TRUE)

File name: C:\YourOutputPath\claimsCCNum.xdf
Number of observations: 128
Number of variables: 6
Number of blocks: 1
Compression type: zlib
Variable information:
Var 1: RowNum, Type: integer, Low/High: (1, 128)
Var 2: age, Type: character
Var 3: car.age, Type: character
Var 4: type, Type: character
Var 5: cost, Type: numeric, Storage: float32, Low/High: (11.0000, 850.0000)
Var 6: number, Type: integer, Low/High: (0, 434)

```

We can use the *colInfo* argument to specify the levels of the *car.age* column. This is useful when reading data in chunks, to assure that factors levels are in the desired order.

```

outfileCAOrdered <- "claimsCAOrdered.xdf"
colInfoList <- list("car.age"= list(type = "factor", levels = c("0-3",
  "4-7", "8-9", "10+")))
rxImport(infile, outFile = outfileCAOrdered, colInfo = colInfoList)
rxGetInfo("claimsCAOrdered.xdf", getVarInfo = TRUE)

```

Gives the following output:

```

File name: C:\YourOutputPath\claimsCAOrdered.xdf
Number of observations: 128
Number of variables: 6
Number of blocks: 1
Compression type: zlib
Variable information:
Var 1: RowNum, Type: integer, Low/High: (1, 128)
Var 2: age, Type: character
Var 3: car.age
  4 factor levels: 0-3 4-7 8-9 10+
Var 4: type, Type: character
Var 5: cost, Type: numeric, Storage: float32, Low/High: (11.0000, 850.0000)
Var 6: number, Type: numeric, Storage: float32, Low/High: (0.0000, 434.0000)

```

These various methods of providing column information can be combined as follows:

```

outfileCAOrdered2 <- "claimsCAOrdered2.xdf"
colInfoList <- list("car.age" = list(type = "factor", levels = c("0-3",
  "4-7", "8-9", "10+")))
claimsOrdered <- rxImport(infile, outfileCAOrdered2,
  colClasses = c(number = "integer"),
  colInfo = colInfoList, stringsAsFactors = TRUE)
rxGetInfo(claimsOrdered, getVarInfo = TRUE)

```

Produces the following output:

```

File name: C:\YourOutputPath\claimsCAOrdered2.xdf
Number of observations: 128
Number of variables: 6
Number of blocks: 1
Compression type: zlib
Variable information:
Var 1: RowNum, Type: integer, Low/High: (1, 128)
Var 2: age
  8 factor levels: 17-20 21-24 25-29 30-34 35-39 40-49 50-59 60+
Var 3: car.age
  4 factor levels: 0-3 4-7 8-9 10+
Var 4: type
  4 factor levels: A B C D
Var 5: cost, Type: numeric, Storage: float32, Low/High: (11.0000, 850.0000)
Var 6: number, Type: integer, Low/High: (0, 434)

```

In general, variable specifications provided by the `colInfo` argument are used in preference to `colClasses`, and those in `colClasses` are used in preference to the `stringsAsFactors` argument.

Also note that the .xdf data format supports a wider variety of data types than R, allowing for efficient storage. For example, by default floating point variables are stored as 32-bit floats in .xdf files. When they are read into R for processing, they are converted to doubles (64-bit floats).

Create or modify variables

You can use the `transforms` argument to `rxImport` to create new variables or modify existing variables when you initially read the data into .xdf format. For example, we could create a new variable, `logcost`, by taking the log of the existing cost variable as follows:

```

inFile <- file.path(rxGetOption("sampleDataDir"), "claims.txt")
outfile <- "claimsXform.xdf"
claimsDS <- rxImport(inFile, outFile = outfile, transforms=list(logcost=log(cost)))
rxGetInfo(claimsDS, getVarInfo=TRUE)

```

Gives the following output, showing the new variable:

```
File name: C:\YourOutputPath\claimsXform.xdf
Number of observations: 128
Number of variables: 7
Number of blocks: 1
Compression type: zlib
Variable information:
Var 1: RowNum, Type: integer, Low/High: (1, 128)
Var 2: age, Type: character
Var 3: car.age, Type: character
Var 4: type, Type: character
Var 5: cost, Type: numeric, Storage: float32, Low/High: (11.0000, 850.0000)
Var 6: number, Type: numeric, Storage: float32, Low/High: (0.0000, 434.0000)
Var 7: logcost, Type: numeric, Low/High: (2.3979, 6.7452)
```

Change date formats

The .xdf format can store dates using the standard R **Date** class. When importing data from other data formats that support dates such as SAS or SPSS, the **rxImport** function converts dates data automatically. However, some data sets even in those formats include dates as character string data.

You can store such data more efficiently by converting it to *Date* data using the *transforms* argument. For example, suppose you have a character variable *TransactionDate* with a representative date of the form "14 Sep 2017". You could convert this to a *Date* variable using the following *transforms* argument:

```
transforms=list(TransactionDate=as.Date(TransactionDate, format="%d %b %Y"))
```

The *format* argument is a character string that may contain conversion specifications, as in the example shown. These conversion specifications are described in the *strptime* help file.

Change a delimiter

You cannot create or modify delimiters through **rxImport**, but for text data, you can create an **RxTextData** data source and specify the delimiter using the *delimiter* argument. For more information about creating a data source object, see [Data Sources in Microsoft R](#).

As a simple example, RevoScaleR includes a sample text data file *hyphens.txt* that is not separated by commas or tabs, but by hyphens, with the following contents:

```
Name-Rank-SerialNumber
Smith-Sgt-02912
Johnson-Cpl-90210
Michaels-Pvt-02931
Brown-Pvt-11311
```

By creating an **RxTextData** data source for this file, you can specify the delimiter using the *delimiter* argument:

```
readPath <- rxGetOption("sampleDataDir")
infile <- file.path(readPath, "hyphens.txt")
hyphensTxt <- RxTextData(infile, delimiter="-")
hyphensDF <- rxImport(hyphensTxt)
hyphensDF

  Name Rank SerialNumber
1 Smith Sgt      2912
2 Johnson Cpl    90210
3 Michaels Pvt   2931
4 Brown Pvt     11311
```

In normal usage, the `delimiter` argument is a single character, such as `delimiter = "\t"` for tab-delimited data or `delimiter = ","` for comma-delimited data. However, each column may be delimited by a different character; all the delimiters must be concatenated together into a single character string. For example, if you have one column delimited by a comma, a second by a plus sign, and a third by a tab, you would use the argument `*delimiter = ",+\t". *`

Examples

This section provides example script demonstrating additional import tasks.

Import multiple files

This example demonstrates an approach for importing multiple text files at once. You can use [sample data](#) for this exercise. It includes mortgage default data for consecutive years, with each year's data in a separate file. In this exercise, you will import all of them to a single XDF by appending one after another, using a combination of base R commands and RevoScaleR functions.

Create a source object for a list of files, obtained using the R `list.files` function with a pattern for selecting specific file names:

```
mySourceFiles <- list.files(rxGetOption("sampleDataDir"), pattern = "mortDefaultSmall\\d*.csv",  
full.names=TRUE)
```

Create an object for the XDF file at a writable location:

```
myLargeXdf <- file.path("C:/users/temp/mortgagelarge.xdf")
```

To iterate over multiple files, use the R `lapply` function and create a function to call `rxImport` with the `append` argument. Importing multiple files requires the `append` argument on `rxImport` to avoid overwriting existing content from the first iteration. Each new chunk of data is imported as a block.

```
lapply(mySourceFiles, FUN = function(csv_file) {  
  rxImport(inData = csv_file, outFile=myLargeXdf, append = file.exists(myLargeXdf)) })
```

Partial output from this operation reports out the processing details.

```
Rows Read: 10000, Total Rows Processed: 10000, Total Chunk Time: 0.020 seconds  
Rows Read: 10000, Total Rows Processed: 10000, Total Chunk Time: 0.020 seconds  
Rows Read: 10000, Total Rows Processed: 10000, Total Chunk Time: 0.025 seconds  
Rows Read: 10000, Total Rows Processed: 10000, Total Chunk Time: 0.022 seconds  
Rows Read: 10000, Total Rows Processed: 10000, Total Chunk Time: 0.020 seconds  
Rows Read: 10000, Total Rows Processed: 10000, Total Chunk Time: 0.020 seconds  
Rows Read: 10000, Total Rows Processed: 10000, Total Chunk Time: 0.019 seconds  
Rows Read: 10000, Total Rows Processed: 10000, Total Chunk Time: 0.020 seconds  
Rows Read: 10000, Total Rows Processed: 10000, Total Chunk Time: 0.020 seconds  
Rows Read: 10000, Total Rows Processed: 10000, Total Chunk Time: 0.020 seconds
```

Use `rxGetInfo` to view precomputed metadata. As you would expect, the block count reflects the presence of multiple concatenated data sets.

```
rxGetInfo(myLargeXdf)
```

Results from this command confirm that you have 10 blocks, one for each .csv file. On a distributed file system, you could place these blocks on separate nodes. You could also retrieve or overwrite individual blocks. For more

information, see [Import and consume data on HDFS](#) and [XDF files](#).

```
File name: C:\Users\TEMP\mortgagelarge.xdf
Number of observations: 1e+05
Number of variables: 6
Number of blocks: 10
Compression type: zlib
```

Import fixed-format data

Fixed-format data is text data in which each variable occupies a fixed-width column in the input data file. Column width, rather than a delimiter, gives the data its structure. You can import fixed-format data using the `rxImport` function.

Optionally, fixed-format data might be associated with a *schema file* having a .sts extension. The schema describes the width and type of each column. For complete details on creating a schema file, see page 93 of the [Stat/Transfer PDF Manual](#). If you have a schema file, you can create the input data source simply by specifying the schema file name as the input data file.

The [built-in samples](#) include a fixed-format version of the claims data as the file claims.dat and a schema file named claims.sts. To import the data using this schema file, we use `RxImport` as follows:

```
# Verify the sample files exist
> list.files(rxGetOption("sampleDataDir"))

# (Windows only) Set a working directory for which you have write access
> setwd("c:/users/temp")

# Specify the source schema file, load data, and save output as a new XDF
> inFile <- file.path(rxGetOption("sampleDataDir"), "claims.sts")
> claimsFF <- rxImport(inData=inFile, outFile="claimsFF.xdf")
```

Return summary information using `rxGetInfo`:

```
rxGetInfo(claimsFF, getVarInfo=TRUE)

File name: C:\\YourOutputPath\\claims.xdf

File name: C:\\YourOutputPath\\claims.xdf
Number of observations: 128
Number of variables: 6
Number of blocks: 1
Compression type: zlib
Variable information:
Var 1: RowNum, Type: numeric, Storage: float32, Low/High: (1.0000, 128.0000)
Var 2: age, Type: character
Var 3: car.age, Type: character
Var 4: type, Type: character
Var 5: cost, Type: numeric, Storage: float32, Low/High: (11.0000, 850.0000)
Var 6: number, Type: numeric, Storage: float32, Low/High: (0.0000, 434.0000)
```

To read all string data as factors, set the `stringsAsFactors` argument to `TRUE` in your call to `rxImport`

```
claimsFF2 <- rxImport(inFile, outFile = "claimsFF2.xdf", stringsAsFactors=TRUE)
rxGetInfo(claimsFF2, getVarInfo=TRUE)
```

If you have fixed-format data without a schema file, you can specify the start, width, and type information for each variable using the `colInfo` argument to the `RxTextData` data source constructor, and then read in the data using `rxImport`.

```

inFileNS <- file.path(readPath, "claims.dat")
outFileNS <- "claimsNS.xdf"
colInfo=list("rownum" = list(start = 1, width = 3, type = "integer"),
            "age" = list(start = 4, width = 5, type = "factor"),
            "car.age" = list(start = 9, width = 3, type = "factor"),
            "type" = list(start = 12, width = 1, type = "factor"),
            "cost" = list(start = 13, width = 6, type = "numeric"),
            "number" = list(start = 19, width = 3, type = "numeric"))
claimsNS <- rxImport(inFileNS, outFile = outFileNS, colInfo = colInfo)
rxGetInfo(claimsNS, getVarInfo=TRUE)

```

If you have a schema file, you can still use the *colInfo* argument to specify type information or factor level information. However, in this case, all the variables are read according to the schema file, and then the *colInfo* data specifications are applied. This means that you cannot use your *colInfo* list to omit variables, as you can when there is no schema file.

Fixed-width character data is treated as a special type by RevoScaleR for efficiency purposes. You can use this same type for character data in delimited data by specifying a *colInfo* argument with a *width* argument for the character column. (Typically, you need to find the longest string in the column and specify a width sufficient to include it.)

Import SAS data

The **rxImport** function can also be used to read data from SAS files having a *.sas7bdat* or *.sd7* extension. You do not need to have SAS installed on your computer; simple file access is used to read in the data.

The sample directory contains a SAS version of the claims data as *claims.sas7bdat*. We can read it into *.xdf* format most simply as follows:

```

inFileSAS <- file.path(rxGetOption("sampleDataDir"), "claims.sas7bdat")
xdffileSAS <- "claimsSAS.xdf"
claimsSAS <- rxImport(inData = inFileSAS, outFile = xdffileSAS)
rxGetInfo(claimsSAS, getVarInfo=TRUE)

```

Gives the following output:

```

File name: C:\YourOutputPath\claimsSAS.xdf
Number of observations: 128
Number of variables: 6
Number of blocks: 1
Compression type: zlib
Variable information:
Var 1: RowNum, Type: character
Var 2: age, Type: character
Var 3: car_age, Type: character
Var 4: type, Type: character
Var 5: cost, Type: numeric, Low/High: (11.0000, 850.0000)
Var 6: number, Type: numeric, Low/High: (0.0000, 434.0000)

```

Sometimes, SAS data files on Windows come in two pieces, a *.sas7bdat* file containing the data and a *.sas7bcat* file containing value label information. You can read both the data and the value label information by specifying the *.sas7bdat* file with the *inData* argument and the *.sas7bcat* file with the *formatFile* argument. To do so, in the following code replace *myfile* with your SAS file name:

```

myData <- rxImport(inData = "myfile.sas7bdat",
                    outFile ="myfile.xdf",
                    formatFile = "myfile.sas7bcat")

```

Import SPSS data

The **rxImport** function can also be used to read data from SPSS files. The sample directory contains an SPSS version of the claims data as claims.sav. We can read it into .xdf format as follows:

```
inFileSpss <- file.path(rxGetOption("sampleDataDir"), "claims.sav")
xdfFileSpss <- "claimsSpss.xdf"
claimsSpss <- rxImport(inData = inFileSpss, outFile = xdfFileSpss)
rxGetInfo(claimsSpss, getVarInfo=TRUE)
```

Gives the following output:

```
File name: C:\YourOutputPath\claimsSpss.xdf
Number of observations: 128
Number of variables: 6
Number of blocks: 1
Compression type: zlib
Variable information:
Var 1: RowNum, Type: character
Var 2: age, Type: character
Var 3: car_age, Type: character
Var 4: type, Type: character
Var 5: cost, Type: numeric, Low/High: (11.0000, 850.0000)
Var 6: number, Type: numeric, Low/High: (0.0000, 434.0000)
```

Variables in SPSS data sets often contain value labels with important information about the data. SPSS variables with value labels are typically most usefully imported into R as categorical “factor” data; that is, there is a one-to-one mapping between the values in the data set (such as 1, 2, 3) and the labels that apply to them (Gold, Silver, Bronze).

Interestingly, SPSS allows for value labels on a subset of existing values. For example, a variable with values from 1 to 99 might have value labels of “NOT HOME” for 97, “DIDN’T KNOW” for 98, and “NOT APPLICABLE” for 99. If this variable is converted to a factor in R using the value labels as the factor labels, all of the values from 1 to 96 would be set to missing because there would be no corresponding factor level. Essentially all of the actual data would be thrown away.

To avoid data loss when converting to factors, use the flag *labelsAsLevels=FALSE*. By default, the information from the value labels is retained even if the variables aren’t converted to factors. This information can be returned using **rxGetVarInfo**. If you don’t wish to retain the information from the value labels, you can specify *labelsAsInfo=FALSE*.

Importing wide data

Big data mainly comes in two forms, long or wide, each presenting unique challenges. The common case is long data, having many observations relative to the number of variables in the data set. With wide data, or data sets with a large number of variables, there are specific considerations to take into account during import.

First, we recommend importing wide data into the .xdf format using the **rxImport** function whenever you plan to do repeated analyses on your data set. Doing so allows you to read subsets of columns into a data frame in memory for specific analyses. For more information, see [Transform and subset data](#).

Second, review the data set for categorical variables that can be marked as factors. If possible use the *colInfo* argument to define the levels rather than *stringsAsFactors*. Explicitly setting the levels results in faster processing speeds because you avoid recomputations of variable metadata whenever a new level is encountered. For wide data sets having a very large number of variables, the extra processing due to recomputation can be significant so it’s worth considering the *colInfo* argument as a way to speed up the import.

Third, if you are using *colInfo*, consider creating it as an object prior to using **rxImport**. The following is an

example of defining the colInfo with factor levels for the claims data from earlier examples:

```
colInfoList <- list("age" = list(type = "factor",
  levels = c("17-20", "21-24", "25-29", "30-34", "35-39", "40-49", "50-59", "60+")),
"car.age" = list(type = "factor",
  levels = c("0-3", "4-7", "8-9", "10+")),
"type" = list(type = "factor",
  levels = c("A", "B", "C", "D")))
```

The colInfo list can then be used as the *colInfo* argument in the **rxImport** function to get your data into .xdf format:

```
inFileClaims <- file.path(rxGetOption("sampleDataDir"), "claims.txt")
outFileClaims <- "claimsWithColInfo.xdf"
rxImport(inFile, outFile = outFileClaims, colInfo = colInfoList)
```

Next steps

Continue on to the following data import articles to learn more about XDF, data source objects, and other data formats:

- [XDF files](#)
- [Data Sources](#)
- [Import relational data using ODBC](#)
- [Import and consume data on HDFS](#)

See also

[Tutorial: data import](#) [Tutorial: data manipulation](#)

Import SQL Server relational data

7/12/2022 • 12 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

This article shows you how to import relational data from SQL Server into a data frame or .xdf file in Machine Learning Server. Source data can originate from Azure SQL Database, or SQL Server on premises or on [an Azure virtual machine](#).

Prerequisites

- [Azure subscription](#), [Azure SQL Database](#), [AdventureWorksLT sample database](#)
- [SQL Server](#) (any supported version and edition) with the [AdventureWorksDW sample database](#)
- Machine Learning Server for Windows or Linux
- R console application (RGui.exe on Windows or Revo64 on Linux)

NOTE

Windows users, remember that R is case-sensitive and that file paths use the forward slash as a delimiter.

How to import from Azure SQL Database

The following script imports sample data from Azure SQL database into a local XDF. The script sets the connection string, SQL query, XDF file, and the `rxImport` command for loading data and saving the output:

```
sConnString <- "Driver={ODBC Driver 13 for SQL Server}; Server=tcp:<your-server-name>.database.windows.net,1433; Database=AdventureWorksLT; Uid=<your-user-name>; Pwd=<your-password>; Encrypt=yes; TrustServerCertificate=no; Connection Timeout=30;"  
sQuery <- "select ProductDescriptionID, Description, ModifiedDate from SalesLT.ProductDescription"  
sDataSet <- RxOdbcData(sqlQuery=sQuery, connectionString=sConnString)  
sDataFile <- RxXdfData("c:/users/temp/mysqldata.xdf")  
rxImport(sDataSet, sDataFile, overwrite=TRUE)  
rxGetInfo(sDataFile, getVarInfo=TRUE, numRows=50)
```

You can run this script in an R console application, but a few modifications are necessary before you can do it successfully. Before running the script, review it line by line to see what needs changing.

1 - Set the connection

Create the connection object using information from the Azure portal and ODBC Data Source Administrator.

```
> sConnString <- "Driver={ODBC Driver 13 for SQL Server}; Server=tcp:<your-server-name>.database.windows.net,1433; Database=AdventureWorksLT; Uid=<your-user-name>; Pwd=<your-password>; Encrypt=yes; TrustServerCertificate=no; Connection Timeout=30;"
```

First, get the ODBC driver name. On Windows, search for and then use the [ODBC Data Source Administrator \(64-bit\)](#) app to view the drivers listed in the **Drivers** tab. On Linux, the ODBC driver manager and individual

drivers must be installed manually. For second, see [How to import relational data using ODBC](#).

After **Driver**, all remaining connection properties from **Server** to **Connection Timeout** are obtained from the Azure portal:

1. Sign in to the [Azure portal](#) and locate AdventureWorksLT.
2. In **Overview > Essentials > Connection strings**, click **Show database connection strings**.
3. On the **ODBC** tab, copy the connection information. It should look similar to the following string, except the server name and user name will be valid for your database. The password is always a placeholder, which you must replace with the actual password used for accessing your database.

```
Driver={ODBC Driver 13 for SQL Server}; Server=tcp:<your-server-name>.database.windows.net,1433;
Database=AdventureWorksLT; Uid=<your-user-name>; Pwd=<your-password>; Encrypt=yes;
TrustServerCertificate=no; Connection Timeout=30;
```

4. In the R console, provide the *sConnString* command based on the example syntax, but with valid values for server name, user name, and password.

2 - Set firewall rules

On Azure SQL Database, access is controlled through firewall rules created for specific IP addresses. Creating a firewall rule is a requirement for accessing Azure SQL Database from a client application.

1. On your client machine running Machine Learning Server, sign in to the [Azure portal](#) and locate AdventureWorksLT.
2. Use **Overview > Set server firewall > Add client IP** to create a rule for the local client.
3. Click **Save** once the rule is created.

On a small private network, IP addresses are most likely static, and the IP address detected by the portal is probably correct. To confirm, you can use **IPConfig** on Windows or **hostname -I** on Linux to get your IP address.

On corporate networks, IP addresses can change on computer restarts, through network address translations, or other reasons described in [this article](#). It can be hard to get the right IP address, even through **IPConfig** and **hostname -I**.

One way to get the right IP address is from the error message reported on a connection failure. If you get the "ODBC Error in SQLDisconnect" error message, it will include this text: "Client with IP address is not allowed to access the server". The IP address reported in the message is that one actually used on the connection, and it should be specified as the start and ending IP range in your firewall rule in the Azure portal.

Producing this error is easy. Just run the entire example script (assuming a valid connection string and write permissions to create the XDF), concluding with **rxImport**. When **rxImport** fails, copy the IP address reported in the message, and use it to set the firewall rule in the Azure portal. Wait a few minutes, and then retry the script.

3 - Set the query

In the R console application, create the SQL query object. The example query consists of columns from a single table, but any valid T-SQL query providing a rowset is acceptable. This table was chosen because it includes numeric data.

```
> sQuery <- "SELECT SalesOrderID, SalesOrderDetailID, OrderQty, UnitPrice, UnitPriceDiscount, LineTotal FROM
SalesLT.SalesOrderDetail"
```

Before attempting unqualified *SELECT * FROM* queries, review the columns in your database for unhandled data types in R. In AdventureWorksLT, the *rowguid(uniqueidentifier)* column is not handled. Other unsupported data types are [listed here](#).

Queries with unsupported data types produce the error below. If you get this error and cannot immediately detect the unhandled data type, incrementally add fields to the query to isolate the problem.

```
Unhandled SQL data type!!!
Could not open data source.
Error in doTryCatch(return(expr), name, parentenv, handler)
```

Queries should be data extraction queries (SELECT and SHOW statements) for reading data into a data frame or .xdf file. INSERT queries are not supported. Because queries are used to populate a single data frame or .xdf file, multiple queries (that is, queries separated by a semicolon ",") are not supported. Compound queries, however, producing a single extracted table (such as queries linked by AND or OR, or involving multiple FROM clauses) are supported.

4 - Set the data source

Create the RxOdbcData data object using the connection and query object specifying which data to retrieve. This exercise uses only a few arguments, but to learn more about data sources, see [Data sources in RevoScaleR](#).

```
> sDataSet <- RxOdbcData(sqlQuery=sQuery, connectionString=sConnString)
```

You could substitute RxSqlServerData for RxOdbcData if you want the option of setting the compute context to a remote SQL Server instance (for example, if you want to run rxMerge, rxImport, or rxSort on a remote SQL Server that also has a Machine Learning Server installation on the same machine). Otherwise, RxOdbcData is local compute context only.

5 - Set the output file

Create the XDF file to save the data to disk. Check the folder permissions for write access.

```
> sDataFile <- RxXdfData("c:/users/temp/mysqldata.xdf")
```

By default, Windows does not allow external writes by non-local users. You might want to create a single folder, such as C:/Users/Temp, and give *Everyone* write permissions. Once the XDF is created, you can move the file elsewhere and delete the folder, or revoke the write permissions you just granted.

On Linux, you can use this alternative path:

```
> sDataFile <- RxXdfData(/tmp/mysqldata.xdf")
```

6 - Import the data

Run rxImport with *inData* and *outFile* arguments. Include *overwrite* so that you can rerun the script with different queries without having to delete the file each time.

```
> rxImport(sDataSet, sDataFile, overwrite=TRUE)
```

7 - Return object metadata

Use rxGetInfo to return information about the XDF data source, plus the first 50 rows:

```
> rxGetInfo(sDataFile, getVarInfo=TRUE, numRows=50)
```

8 - Return summary statistics

Use `rxSummary` to produce summary statistics on the data. The `~.` is used to compute summary statistic on numeric fields.

```
> rxSummary(~., sDataFile)
```

Output shows the central tendencies of the data, including mean values, standard deviation, minimum and maximum values, and whether any observations are missing. From the output, you can tell that orders are typically of small quantities and prices.

How to import from SQL Server

The following script demonstrates how to import data from a SQL Server relational database into a local XDF. This script is slightly different from the one for Azure SQL Database. This one uses the AdventureWorks data warehouse (AdventureWorksDW) for its normalized data.

```
sConnString <- "Driver={SQL Server}; Server=(local); Database=AdventureWorksDW2016; Connection Timeout=30;"  
sqquery <-"SELECT * FROM dbo.vDMPrep"  
sDataSet <- RxOdbcData(sqlQuery=sqquery, connectionString=sConnString)  
sDataFile <- RxXdfData("c:/users/temp/mysqldata.xdf")  
rxImport(sDataSet, sDataFile, overwrite=TRUE)  
rxGetInfo(sDataFile, getVarInfo=TRUE)
```

As with the previous exercise, modifications are necessary before you can run this script successfully.

1 - Set the connection

Create the connection object using the SQL Server database driver a local server and the sample database.

```
> sConnString <- "Driver={SQL Server}; Server=(local); Database=AdventureWorksDW2016; Connection  
Timeout=30;"
```

The driver used on the connection is an ODBC driver that is installed by SQL Server. You could use the default database driver provided with operating system, but SQL Server Setup also installs drivers.

On Windows, ODBC drivers can be listed in the **ODBC Data Source Administrator (64-bit)** app on the **Drivers** tab. On Linux, the ODBC driver manager and individual drivers must be installed manually. For pointers, see [How to import relational data using ODBC](#).

The `Server=(local)` refers to a local default instance connected over TCP. A named instance is specified as `computername$instanceName`. A remote server has the same syntax, but you should verify that that remote connections are enabled. The defaults for this setting vary depending on which edition is installed.

2 - Set the query

In the R console application, create the SQL query object. The example query consists of columns from a single view, but any valid T-SQL query providing a rowset is acceptable. This unqualified query works because all columns in this view are supported data types.

```
> sqquery <-"SELECT * FROM dbo.vDMPrep"
```

TIP

You could skip this step and specify the query information through **RxOdbcData** via the *table* argument. Specifically, you could write `sDataSet <- RxOdbcData(table=dbo.vDMPrep, connectionString=sConnString)`.

3 - Set the data source

Create an **RxOdbcData** data source object based on query results. The first example is the simple case.

```
> sDataSet <- RxOdbcData(sqlQuery=squery, connectionString=sConnString)
```

The **RxOdbcData** data source takes arguments that can be used to [modify the data set](#). A revised object includes syntax that converts characters to factors via *stringsAsFactors*, specifies ranges for ages using *colInfo*, and *colClasses* sets the data type to convert to:

```
> colInfoList <- list("Age" = list(type = "factor", levels = c("20-40", "41-50", "51-60", "61+")))
> sDataSet <- RxOdbcData(sqlQuery=squery, connectionString=sConnString, colClasses=c(Model="character",
OrderNumber="character"), colInfo=colInfoList, stringsAsFactors=TRUE)
```

Compare before-and-after variable information. On the original data set, the variable information is as follows:

```
Variable information:
Var 1: EnglishProductName, Type: character
Var 2: Model, Type: character
Var 3: CustomerKey, Type: integer, Low/High: (11000, 29483)
Var 4: Region, Type: character
Var 5: Age, Type: integer, Low/High: (30, 101)
Var 6: IncomeGroup, Type: character
Var 7: CalendarYear, Type: integer, Storage: int16, Low/High: (2010, 2014)
Var 8: FiscalYear, Type: integer, Storage: int16, Low/High: (2010, 2013)
Var 9: Month, Type: integer, Storage: int16, Low/High: (1, 12)
Var 10: OrderNumber, Type: character
Var 11: LineNumber, Type: integer, Storage: int16, Low/High: (1, 8)
Var 12: Quantity, Type: integer, Storage: int16, Low/High: (1, 1)
Var 13: Amount, Type: numeric, Low/High: (2.2900, 3578.2700)
```

After the modifications, the variable information includes default and custom factor levels. Additionally, for *Model* and *OrderNumber*, we preserved the original "character" data type using the *colClasses* argument. We did this because *stringsAsFactors* globally converts character data to factors (levels), and for *OrderNumber* and *Model*, the number of levels was overkill.

```
Variable information:  
Var 1: EnglishProductName  
 3 factor levels: Bikes Accessories Clothing  
Var 2: Model, Type: character  
Var 3: CustomerKey, Type: integer, Low/High: (11000, 29483)  
Var 4: Region  
 3 factor levels: North America Europe Pacific  
Var 5: Age  
 4 factor levels: 20-40 41-50 51-60 61+  
Var 6: IncomeGroup  
 3 factor levels: High Low Moderate  
Var 7: CalendarYear, Type: integer, Storage: int16, Low/High: (2010, 2014)  
Var 8: FiscalYear, Type: integer, Storage: int16, Low/High: (2010, 2013)  
Var 9: Month, Type: integer, Storage: int16, Low/High: (1, 12)  
Var 10: OrderNumber, Type: character  
Var 11: LineNumber, Type: integer, Storage: int16, Low/High: (1, 8)  
Var 12: Quantity, Type: integer, Storage: int16, Low/High: (1, 1)  
Var 13: Amount, Type: numeric, Low/High: (2.2900, 3578.2700)
```

4 - Set the output file

Create the XDF file to save the data to disk. Check the folder permissions for write access.

```
> sDataFile <- RxXdfData("c:/users/temp/mysqldata.xdf")
```

By default, Windows does not allow external writes by non-local users. You might want to create a single folder, such as C:/Users/Tmp, and give *Everyone* write permissions. Once the XDF is created, you can move the file elsewhere and delete the folder, or revoke the write permissions you just granted.

On Linux, you can use this alternative path:

```
> sDataFile <- RxXdfData(/tmp/mysqldata.xdf")
```

5 - Import the data

Run **rxImport** with *inData* and *outFile* arguments. Include *overwrite* so that you can rerun the script with different queries without having to delete the file each time.

```
> rxImport(sDataSet, sDataFile, overwrite=TRUE)
```

6 - Return object metadata

Use **rxGetInfo** to return information about the XDF data source:

```
> rxGetInfo(sDataFile, getVarInfo=TRUE)
```

7 - Return summary statistics

Use **rxSummary** to produce summary statistics on the data. The `~.` is used to compute summary statistic on numeric fields.

```
> rxSummary(~., sDataFile)
```

Output shows the central tendencies of the data, including mean values, standard deviation, minimum and maximum values, and whether any observations are missing. From the output, you can see that Age was probably included in the view by mistake. There are no values for this variable in the observations retrieved

from the data source.

Drill down: Changing data types

Data stored in databases may be stored differently from how you want to store the data in R. As noted in the previous example for on-premises SQL Server, you can add the `colClasses`, `colInfo`, and `stringsAsFactors` arguments to `RxOdbcData` to specify how columns are stored in R.

- The `stringsAsFactors` argument is the simplest to use. If specified, any character string column not otherwise accounted for by the `colClasses` or `colInfo` argument is stored as a factor in R, with levels defined according to the unique character strings found in the column.
- The `colClasses` argument allows you to specify a particular R data type for a particular variable.
- The `colInfo` argument is similar, but it also allows you to specify a set of levels for a factor variable.

IMPORTANT

The data type must be supported, otherwise the unhandled data type error occurs before the conversion. For example, you can't cast a unique identifier as an integer as a bypass mechanism.

The following example uses the SaleOrderHeader table because it provides more columns, and combines all three arguments to modify the data types of several variables in the claims data:

```
> colInfoList <- list("Age" = list(type = "factor", levels = c("20-40", "41-50", "51-60", "61+")))
> sDataSet <- RxOdbcData(sqlQuery=squery, connectionString=sConnString, colClasses=c(Model="character",
OrderNumber="character"), colInfo=colInfoList, stringsAsFactors=TRUE)
```

Next Steps

Continue on to the following data import articles to learn more about XDF, data source objects, and other data formats:

- [SQL Server tutorial for R](#)
- [XDF files](#)
- [Data Sources](#)
- [Import text data](#)
- [Import ODBC data](#)
- [Import and consume data on HDFS](#)

See Also

[RevoScaleR Functions](#)

[Tutorial: data import and exploration](#) [Tutorial: data visualization and analysis](#)

Import relational data using ODBC

7/12/2022 • 7 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

RevoScaleR allows you to read or write data from virtually any database for which you can obtain an ODBC driver, a standard software interface for accessing relational data. ODBC connections are enabled through drivers and a driver manager. Drivers handle the translation of requests from an application to the database. The ODBC Driver Manager sets up and manages the connection between them.

Both drivers and an ODBC Driver Manager must be installed on the computer running Microsoft R. On Windows, the driver manager is built in. On Linux systems, RevoScaleR supports [unixODBC](#), which you will need to install. Once the manager is installed, you can proceed to install individual database drivers for all of the data sources you need to support.

To import data from a relational database management system, do the following:

1. Install an ODBC Driver Manager (Linux only)
2. Install ODBC drivers
3. Create an **RxOdbcData** data source
4. For read operations, use **rxImport** to read data
5. For write operations, use **rxDataStep** to write data back to the data source

About ODBC in Microsoft R

In Microsoft R, an ODBC and **RxOdbcData** dependency exists for data imported from relational database systems like Oracle, PostgreSQL, and MySQL, to name a few. ODBC is primarily required for **RxOdbcData** data sources, but it is also used internally for DBMS-specific connections to SQL Server. If you delete the ODBC driver for SQL Server and then try to use **RxSqlServerData**, the connection fails.

Although ODBC drivers exist for text data, Microsoft R does not use ODBC for sources accessed as file-based reads. To be specific, it's not used for accessing text files, SPSS database files, or SAS database files.

For SQL Server, you can use **RxSqlServerData** or **RxOdbcData** interchangeably, unless you are accessing [R objects stored in a SQL Server table](#), in which case **RxOdbcData** is required. For examples and instructions on using SQL Server data in Microsoft R, see [Import SQL data from Azure SQL Database and SQL Server](#).

How to configure ODBC for relational data access

Follow these steps to configure ODBC for loading data from an external relational database.

1 - Install unixODBC

Linux only

An ODBC Driver Manager manages communication between applications and drivers. On Linux, an ODBC Driver Manager is not typically bundled in a Linux distribution. Although several driver managers are available, Microsoft R supports the [unixODBC Driver Manager](#).

To first check whether unixODBC is installed, do the following:

- On RHEL: `rpm -qa | grep unixODBC`
- On Ubuntu: `apt list --installed | grep unixODBC`
- On SLES: `zypper se unixODBC`

If unixODBC is not installed, issue the following commands to add it:

- On RHEL: `sudo yum install unixodbc-devel unixodbc-bin unixodbc`
- On Ubuntu: `sudo apt-get install unixodbc-devel unixodbc-bin unixodbc`
- On SLES: `sudo zypper install unixODBC`

For more information, SQL Server documentation provides in-depth instructions for [installing unixODBC](#) on a variety of Linux operating systems. Alternatively, you can go to the [unixODBC web site](#), download the software, and follow instructions on the site.

2 - Install drivers

ODBC drivers must be installed on the machine running Machine Learning Server or R Client. You will need a driver that corresponds to the database version you plan to use. To check which drivers are installed, use the instructions below.

On Linux

1. To list existing drivers: `odbcinst -q -d`
2. Driver version information is in the `odbcinst.ini` file. To find the location of that file: `odbcinst -j`
3. Typically, driver information is in `/etc/odbcinst.ini`. To view its contents: `gedit /etc/odbcinst.ini`
4. You can also list any existing ODBC data sources already on the system: `odbcinst -q -s`

On Windows

1. Search for *odbc data sources* to find the ODBC Data Source Administrator (64-bit) application.
2. In ODBC Data Source Administrator > Drivers tab, review the currently installed drivers.

If the driver you need is missing, download and install the driver using instructions provided by the vendor. A partial list of download links is provided below.

ODBC download sites for commonly used databases

Drivers for commonly used databases might be pre-installed with the operating system or database management system, but if not, most can be downloaded from database vendor web sites. The following table provides links to download pages for several data platforms.

DATA PLATFORM	DRIVER DOWNLOAD PAGES AND INSTRUCTIONS
SQL Server on Linux	Installing the Microsoft ODBC Driver for SQL Server on Linux
Oracle Instant Client	Oracle Instant Client Downloads
MySQL	Download Connector/ODBC
PostgreSQL	pgsqlODBC
Cloudera	Cloudera ODBC Driver for Hive

3 - Connect and import

This step uses examples to illustrate connection strings, query strings, **RxOdbcData** data source objects, XDF objects, and import commands for various platform configurations.

Query strings must consist of data extraction queries (SELECT and SHOW statements) that populate a single data frame or .xdf file. As long as a single extracted table is produced, you can use queries linked by AND, OR, and multiple FROM clauses. Unsupported syntax includes INSERT queries or multiple query strings separated by a semicolon.

Recall that **RxOdbcData** provides local compute context only, which means that when you create the object, any read or write operations are executed by Machine Learning Server on the local machine.

Using SQL Server

This example uses a connection string to connect to a local SQL Server instance and the [RevoClaimsDB database](#). For simplicity, the connection is further scoped to a single table, but you could write T-SQL to select a more interesting data set.

```
sConnectStr <- "Driver={ODBC Driver 13 for SQL Server};Server=(local);Database=RevoClaimsDB;Trusted_Connection=Yes"
sQuery = "Select * from dbo.claims"
sDS <- RxOdbcData(sqlQuery=sQuery, connectionString=sConnectStr)
sXdf <- RxXdfData("c:/users/temp/claimsFromODBC.xdf")
rxImport(sDS, sXdf, overwrite=TRUE)
```

This returns the following:

```
> rxGetInfo(sXdf, getVarInfo=TRUE)
File name: c:\Users\TEMP\claimsFromODBC.xdf
Number of observations: 128
Number of variables: 6
Number of blocks: 1
Compression type: zlib
Variable information:
Var 1: "RowNum", Type: character
Var 2: "age", Type: character
Var 3: "car age", Type: character
Var 4: "type", Type: character
Var 5: "cost", Type: character
Var 6: "number", Type: character
```

TIP

If you are selecting an entire table or view, a simpler alternative for the query string is just the name of that object specified as an argument on **RxOdbcData**. For example:

```
sDS <- RxOdbcData(tabble=dbo.claims, connectionString=sConnectStr). With this shortcut, you can omit the sQuery object from your script.
```

By amending the **RxOdbcData** object with additional arguments, you can use *colClasses* to recast numeric "character" data as integers or float, and use *stringsAsFactors* to convert all unspecified character variables to factors:

```
sDS <- RxOdbcData(sqlQuery=sQuery, connectionString=sConnectStr, colClasses=c(RowNum="integer",
number="integer"), stringsAsFactors=TRUE)
sXdf <- RxXdfData("c:/users/temp/claimsFromODBC.xdf")
rxImport(sDS, sXdf, overwrite=TRUE)
```

Data values for age and carage are expressed as ranges in the original data, so those columns (as well as the type column) are appropriately converted to factors through the *stringsAsFactors* argument. Retaining the

character data type for cost is also appropriate for the existing data. While cost has mostly numeric values, it also has multiple instances of "NA".

After re-import, variable metadata should be as follows:

```
> rxGetInfo(sXdf, getVarInfo=TRUE)
Variable information:
Var 1: RowNum, Type: integer, Low/High: (1, 128)
Var 2: age
  8 factor levels: 17-20 21-24 25-29 30-34 35-39 40-49 50-59 60+
Var 3: car age
  4 factor levels: 0-3 4-7 8-9 10+
Var 4: type
  4 factor levels: A B C D
Var 5: cost
  100 factor levels: 289.00 282.00 133.00 160.00 372.00 ... 119.00 385.00 324.00 192.00 123.00
Var 6: number, Type: integer, Low/High: (0, 434)
```

Using Oracle Express

Oracle Express is a free version of the popular Oracle database management system intended for evaluation and education. It uses the same ODBC drivers as the commercial offerings. The follow example demonstrates an Oracle SQL statement to show all the tables in a database (this differs from standard SQL implementations):

```
tablesDS <- RxOdbcData(sqlQuery="select * from user_tables", connectionString =
  "DSN=ORA10GDSN;Uid=system;Pwd=X8dzlkjWQ")
OracleTableDF <- rxImport(tablesDS, overwrite=TRUE)
OracleTableDF[,1]
```

This yields a list of tables similar to the following (showing partial results for brevity, with the first 10 tables):

```
[1] "MVIEWS_ADV_WORKLOAD"      "MVIEWS_ADV_BASETABLE"
[3] "MVIEWS_ADV_SQLDEPEND"    "MVIEWS_ADV_PRETTY"
[5] "MVIEWS_ADV_TEMP"         "MVIEWS_ADV_FILTER"
[7] "MVIEWS_ADV_LOG"          "MVIEWS_ADV_FILTERINSTANCE"
[9] "MVIEWS_ADV_LEVEL"        "MVIEWS_ADV_ROLLUP"
```

Using MySQL on Red Hat Enterprise Linux

As a first step, specify the name of your DSN. On Linux, this is the same name specified for the ODBC configuration.

```
### Test of import from 'centos-database01' ####
sConnectionString = "DSN=ScaleR-ODBC-test"
sUserSQL = "SELECT * FROM airline1987"
airlineXDFName <- file.path(getwd(), "airlineimported.xdf")
airlineODBCSource <- RxOdbcData(sqlQuery = sUserSQL, connectionString = sConnectionString, useFastRead = TRUE)
rxImport(airlineODBCSource, airlineXDFName, overwrite = TRUE)
```

Next Steps

Continue on to the following data import articles to learn more about XDF, data source objects, and other data formats:

- [Import SQL data](#)
- [Import text data](#)
- [Import and consume data on HDFS](#)
- [XDF files](#)

- Data Sources

See Also

[RevoScaleR Functions](#)

[Tutorial: data import and exploration](#) [Tutorial: data manipulation and statistical analysis](#)

Import and consume HDFS data files using RevoScaleR

7/12/2022 • 7 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

This article explains how to load data from the Hadoop Distributed File System (HDFS) into an R data frame or an .xdf file. Example script shows several use cases for using RevoScaleR functions with HDFS data.

Set the file system

By default, data is expected to be found on the native file system (Linux). If all your data is on HDFS, you can use **rxSetFileSystem** to specify this as a global option:

```
rxSetFileSystem(RxHdfsFileSystem())
```

If only some files are on HDFS, keep the native file system default and use the *fileSystem* argument on **RxTextData** or **RxXdfData** data sources to specify which files are on HDFS. For example:

```
hdfsFS <- RxHdfsFileSystem()
txtSource <- RxTextData("/test/HdfsData/AirlineCSV/CSVs/1987.csv", fileSystem=hdfsFS)
xdfSource <- RxXdfData("/test/HdfsData/AirlineData1987", fileSystem=hdfsFS)
```

The **RxHdfsFileSystem** function creates a file system object for the HDFS file system. You can use **RxNativeFileSystem** function does the same thing for the native file system.

Load data from HDFS

Assuming you already have an .xdf file on HDFS, you can load it by creating an **RxXdfData** object that takes the .xdf file as an input.

```
mortDS <- RxXdfData("/share/SampleData/mortDefaultSmall.xdf")
rxGetInfo(mortDS, numRows = 5)
rxSummary(~., data = mortDS, blocksPerRead = 2)
logitObj <- rxLogit(default~F(year) + creditScore + yearsEmploy + ccDebt,
  data = mortDS, blocksPerRead = 2, reportProgress = 1)
summary(logitObj)
```

Write XDF to HDFS

Once you load data from a text file or another source, you can save it as an .xdf file to either HDFS or the native file system. The compute context determines where the file can be saved. To get the current compute context, use **rxGetComputeContext()**. In a local compute context, out files must be written to the native file system. However, by setting the compute context to **RxHadoopMR** or **RxSpark**, you can write to HDFS.

The following example shows how to write a data frame as an .xdf file directly to HDFS using the built-in Iris data set.

1. Set the user name:

```
> username <- `<your-user-name-here>`
```

2. Set folder paths:

```
hdfsDataDirRoot <- paste("/home/<user-dir>/<data-dir>", username, sep="")
localfsDataDirRoot <- paste("/home/<user-dir>/<data-dir>", username, sep="")
setwd(localfsDataDirRoot)
```

3. Set compute context:

```
port <- 8020 # KEEP IF USING THE DEFAULT
host <- system("hostname", intern=TRUE)
hdfsFS <- RxHdfsFileSystem(hostName=host, port=port)

myHadoopCluster <- RxHadoopMR(
  nameNode= host,
  port=port,
  consoleOutput=TRUE)

rxSetComputeContext(myHadoopCluster)
```

4. Write the XDF to a text file on HDFS:

```
air7x <- RxXdfData(file='/user/RevoShare/revolution/AirOnTime7Pct', fileSystem = hdfsFS)
air7t <- RxTextData(file='/user/RevoShare/revolution/AirOnTime7PctText', fileSystem = hdfsFS,
createFileSet=TRUE)

rxDataStep(air7x,air7t)
```

Write a composite XDF

A *composite XDF* refers to a collection of .xdf files rather than a single .xdf out file. You can create a composite .xdf if you want to load, refresh, and analyze data as a collection of smaller files that can be managed independently or used collectively, depending on the need.

A composite set consists of a named parent directory with two subdirectories, *data* and *metadata*, containing split data .xdfd files and metadata .xdfm files, respectively. The .xdfm file contains the metadata for all of the .xdfd files under the same parent folder.

RxXdfData is used to specify a composite set, and **rxImport** is used to read in the data and output the generated .xdfd files.

When the compute context is **RxHadoopMR**, a composite set of XDF is always created. In a local compute context, which you can use on HDFS, you must specify the option *createCompositeSet=TRUE* within the **RxXdfData** if you want the composite set.

To create a composite file

1. Use **RxGetComputeContext()** to determine whether you need to set *createCompositeSet*. In **RxHadoopMR** or **RxSpark**, you can omit the argument. For local compute context, add

createCompositeSet=TRUE to force the composite set.

2. Use **RxTextData** to create a data source object based on a single file or a directory of files.
3. Use **RxXdfData** to create an XDF for use as the *outFile* argument.
4. Use **rxImport** function to read source files and save the output as a composite XDF.

The following example demonstrates creating a composite set of .xdf files within the native file system in a local compute context using a directory of .csv files as input. Prior to trying these examples yourself, copy the sample AirlineDemoSmallSplit folder from the sample data directory to /tmp. Create a second empty folder named testXdf under /tmp to hold the composite file set:

```
# Run this command to verify source path. Output should be the AirlineDemoSmallPart?.csv file list.  
list.files("/tmp/AirlineDemoSmallSplit/")

# Set folders and source and output objects  
AirDemoSrcDir <- "/tmp/AirlineDemoSmallSplit/"  
AirDemoSrcObj <- RxTextData(AirDemoSrcDir)  
AirDemoXdfDir <- "/tmp/testXdf/"  
AirDemoXdfObj <- RxXdfData(AirDemoXdfDir, createCompositeSet=TRUE)

# Run rxImport to convert the data info XDF and save as a composite file  
rxImport(AirDemoSrcObj, outFile=AirDemoXdfObj)
```

This creates a directory named testXdf, with the data and metadata subdirectories, containing the split .xdfd and .xdfm files.

```
list.files(AirDemoXdfDir, recursive=TRUE, full.names=TRUE)
```

Output should be as follows:

```
[1] "/tmp/testXdf/data/testXdf_1.xdfd"  
[2] "/tmp/testXdf/data/testXdf_2.xdfd"  
[3] "/tmp/testXdf/data/testXdf_3.xdfd"  
[4] "/tmp/testXdf/metadata/testXdf.xdfm"
```

Run **rxGetInfo** to return metadata, including the number of composite data files, and the first 10 rows.

```

rxGetInfo(AirDemoXdfObj, getInfo=TRUE, numRows=10)
File name: /tmp/testXdf
Number of composite data files: 3
Number of observations: 6e+05
Number of variables: 3
Number of blocks: 3
Compression type: zlib
Variable information:
Var 1: ArrDelay, Type: character
Var 2: CRSDepTime, Type: numeric, Storage: float32, Low/High: (0.0167, 23.9833)
Var 3: DayOfWeek, Type: character
Data (10 rows starting with row 1):
ArrDelay CRSDepTime DayOfWeek
1       6   9.666666 Monday
2      -8  19.916666 Monday
3      -2  13.750000 Monday
4       1  11.750000 Monday
5      -2   6.416667 Monday
6     -14  13.833333 Monday
7      20  16.416666 Monday
8      -2  19.250000 Monday
9      -2  20.833334 Monday
10     -15 11.833333 Monday

```

Control generated file output

Number of generated .xdfd files depends on characteristics of source data, but it is generally one file per HDFS block. However, if the original source data is distributed among multiple smaller files, each file counts as a block even if the file size is well below HDFS block size. The HDFS block size varies from installation to installation, but is typically either 64MB or 128MB. For more in-depth information about the composite XDF format and its use within a Hadoop compute context, see [Get started with HadoopMR and RevoScaleR](#).

Filenames are based on the parent directory name.

Rows per file are influenced by compute context:

- When compute context is local with *createCompositeSet=TRUE*, the number of blocks put into each .xdfd file in the composite set.
- When compute context is HadoopMR, the number of rows in each .xdfd file is determined by the rows assigned to each MapReduce task, and the number of blocks per .xdfd file is therefore determined by *rowsPerRead*.

Load a composite XDF

You can reference a composite XDF using the data source object used as the *outFile* for **rxImport**. To load a composite XDF residing on the HDFS file system, set **RxXdfData** to the parent folder having data and metadata subdirectories:

```

rxSetFileSystem(RxHdfsFileSystem())
TestXdfObj <- RxXdfData("/tmp/TestXdf/")
rxGetInfo(TestXdfObj, numRows = 5)
rxSummary(~., data = TestXdfObj)

```

Note that in this example we set the file system to HDFS globally so we did not need to specify the file system within the data source constructors.

Using RevoScaleR with rhdfs

If you are using both RevoScaleR and the RHadoop connector package rhdfs, you need to ensure that the two do not interfere with each other. The rhdfs package depends upon the rJava package, which will prevent access to HDFS by RevoScaleR if it is called before RevoScaleR makes its connection to HDFS.

To prevent this interaction, use the function rxHdfsConnect to establish a connection between RevoScaleR and HDFS. An install-time option on Linux can be used to trigger such a call from the Rprofile.site startup file. If the install-time option is not chosen, you can add it later by setting the REVOHADOOPHOST and REVOHADOOPPORT environment variables with the host name of your Hadoop name node and the name node's port number, respectively.

You can also call rxHdfsConnect interactively within a session, provided you have not yet attempted any other rJava or rhdfs commands. For example, the following call will fix a connection between the Hadoop host sandbox-01 and RevoScaleR; if you make a subsequent call to rhdfs, RevoScaleR can continue to use the previously established connection. Note that once rhdfs (or any other rJava call) has been invoked, you cannot change the host or port you use to connect to RevoScaleR:

```
rxHdfsConnect(hostName = "sandbox-01", port = 8020)
```

Next steps

Related articles include best practices for XDF file management, including managing split files, and compute context:

- [XDF files in Microsoft R](#)
- [Compute context in Microsoft R](#)

To further your understanding of RevoScaleR usage with HadoopMR or Spark, continue with the following articles:

- [Data import and exploration on Apache Spark](#)
- [Data import and exploration on Hadoop MapReduce](#)

See Also

[Machine Learning Server Install Machine Learning Server on Linux](#)

[Install Machine Learning Server on Hadoop](#)

Data Sources in RevoScaleR

7/12/2022 • 7 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

A *data source* in RevoScaleR is an R object representing a data set. It is the return object of `rxImport` for read operations and `rxDataStep` for write operations. Although the data itself may be on disk, a data source is an in-memory object that allows you to treat data from disparate sources in a consistent manner within RevoScaleR.

Behind the scenes, `rxImport` often creates data sources implicitly to facilitate data import. You can explicitly create data sources for more control over how data is imported, such as setting arguments to control how many rows are read into the object. This article explains how to create a variety of data sources and use them in an analytical context.

Data Source Constructors

To create data sources directly, use the constructors listed in the following table:

SOURCE DATA	DATA SOURCE CONSTRUCTOR
Text (fixed-format or delimited)	RxTextData
SAS	RxSasData
SPSS	RxSpssData
ODBC Database	RxOdbcData
Teradata Database	RxTeradata
SQL Server Database	RxSqlServerData
Spark data: Hive, Parquet and ORC	RxSparkData or RxHiveData , RxParquetData , RxOrcData
.xdf data files	RxXdfData

[XDF files](#) are the out files for `rxImport` read operations, but you also use them as a data source input when loading all or part of an .xdf into a data frame. Using an XDF data source is recommended for repeated analysis of a single data set. It is almost always faster to import data into an .xdf file and run analyses on the .xdf data source than to load data from an original data source.

When to create a data source

For simple data import, it's not necessary to explicitly create a data source. You can simply specify a file path for a file that `rxImport` can read, and RevoScaleR will read it using the default settings. However, if you need to provide additional options specific to that data source type, you should create a data source using a constructor

from the previous list.

How to create a data source

You can create a data source the same way you create any object in R, by giving it a name and using a constructor. The first argument of any RevoScaleR data source is the source data:

```
# Load sample text file on Linux  
> myTextDS <- RxTextData("/usr/lib64/microsoft-r/3.3/lib64/R/library/RevoScaleR/SampleData/claims.txt")  
  
# Load sample text file on Windows. Remember to replace Window's \ with R's /  
> myTextDS <- RxTextData("C:/Program Files/Microsoft/ML  
Server/R_SERVER/library/RevoScaleR/SampleData/claims.txt")
```

As a coding best practice, create a file object first, and pass that to the data source:

```
> mySrcObj <- file.path(rxGetOption("sampleDataDir"), "claims.txt")  
> myTextDS <- RxTextData(mySrcObj)
```

After the data source object is created, you can return object properties, precomputed metadata, and rows.

```
# Return properties  
> myTextDS  
RxTextData Source  
"C:/Program Files/Microsoft/ML Server/R_SERVER/library/RevoScaleR/SampleData/claims.txt"  
centuryCutoff: 20  
rowsToSniff: 10000  
rowsToSkip: 0  
defaultReadBufferSize: 10000  
isFixedFormat: FALSE  
useFastRead: TRUE  
fileSystem:  
  fileSystemType: native  
  
# Return variable metadata  
> rxGetVarInfo(myTextDS)  
Var 1: RowNum, Type: integer  
Var 2: age, Type: character  
Var 3: car.age, Type: character  
Var 4: type, Type: character  
Var 5: cost, Type: numeric, Storage: float32  
Var 6: number, Type: numeric, Storage: float32  
  
# Return first 10 rows  
> rxGetInfo(myTextDS, numRows=10)  
File name: C:/Program Files/Microsoft/ML Server/R_SERVER/library/RevoScaleR/SampleData/claims.txt  
Data Source: Text  
Data (10 rows starting with row 1):  
RowNum   age car.age type cost number  
1       1 17-20    0-3   A  289     8  
2       2 17-20    4-7   A  282     8  
3       3 17-20    8-9   A  133     4  
4       4 17-20   10+   A  160     1  
5       5 17-20    0-3   B  372    10  
6       6 17-20    4-7   B  249    28  
7       7 17-20    8-9   B  288     1  
8       8 17-20   10+   B   11     1  
9       9 17-20    0-3   C  189     9  
10      10 17-20   4-7   C  288    13
```

Use standard R methods

A number of standard R methods can be used with RevoScaleR data sources. You might be familiar with **names** for viewing variable names, or **head** for viewing the first few rows:

```
> names(myTextDS)
[1] "RowNum" "age" "car.age" "type" "cost" "number"

> head(myTextDS)
RowNum age car.age type cost number
1 1 17-20 0-3 A 289 8
2 2 17-20 4-7 A 282 8
3 3 17-20 8-9 A 133 4
4 4 17-20 10+ A 160 1
5 5 17-20 0-3 B 372 10
6 6 17-20 4-7 B 249 28
```

By loading data into a data frame using **rxImport**, you can use additional functions, such as the **dim** function to return dimensions, and **colnames** or **dimnames** as an alternative to **RxGetVarInfo** to obtain variable names.

```
# Load data into a data frame
newDF <- rxImport(inData = myTextDS)
dim(newDF)
[1] 128   6
```

You can obtain the number of variables using the **length** function:

```
length(newDF)
[1] 6
```

View the **last** rows of a data source using the **tail** function:

```
tail(claimsDS)
RowNum age car.age type cost number
123 123 60+ 8-9 C 227 20
124 124 60+ 10+ C 119 6
125 125 60+ 0-3 D 385 62
126 126 60+ 4-7 D 324 22
127 127 60+ 8-9 D 192 6
128 128 60+ 10+ D 123 6
```

By adding **outFile** to **rxImport**, you create an XDF data source, which you can return using **summary**:

```
newDF <- rxImport(inData = myTextDS, outFile = "claims.xdf", overwrite = TRUE)
Rows Read: 128, Total Rows Processed: 128, Total Chunk Time: 0.009 seconds
summary(newDF)
...
Data: object (RxXdfData Data Source)
File name: claims.xdf
...
```

For .xdf file data sources, **dimnames** returns only column names. Row names are not provided because .xdf files do not contain row names:

```

colnames(newDF)
[1] "RowNum"    "age"      "car.age"   "type"     "cost"     "number"

dimnames(newDF)
[[1]]
NULL
[[2]]
[1] "RowNum"    "age"      "car.age"   "type"     "cost"     "number"

```

Data source by compute context

In the local compute context, all of RevoScaleR's supported data sources are available to you. In a distributed context, the data source object aligns to the compute context. Thus, **RxInSqlServer** only supports **RxSqlServerData** objects. Likewise for **RxInTeradata**, which supports only the **RxTeradata** data sources. For more information, see [Compute context](#).

DATA SOURCE	RXLOCALSEQ	RXSPARK	RXHADOOPMR	RXINSQLSERVER	RXINTERADATA
Delimited Text (RxTextData)	x	x	x		
Fixed-Format Text (RxTextData)	x				
.xdf data files (RxXdfData)	x	x	x		
SAS data files (RxSasData)	x				
SPSS data files (RxSpssData)	x				
ODBC data (RxOdbcData)	x				
SQL Server database (RxSqlServerData)	x			x	
Teradata database (RxTeradata)	x				x
Spark data RxSparkData	x	x			

Examples

The following examples show how to instantiate and use various data sources.

RxSasData

Sample data includes claims.sas7bdat, which you can load without having SAS installed.

```
inFileSAS <- file.path(rxGetOption("sampleDataDir"), "claims.sas7bdat")
sourceDataSAS <- RxSasData(inFileSAS, stringsAsFactors=TRUE)
```

Retrieve variables in the data by calling R's **names** function:

```
names(sourceDataSAS)
[1] "RowNum"    "age"       "car_age"   "type"      "cost"      "number"
```

Compute a regression, passing the data source as the data argument to **rxLinMod**:

```
rxLinMod(cost ~ age + car_age, data = sourceDataSAS)

Rows Read: 128, Total Rows Processed: 128, Total Chunk Time: 0.003 seconds
Computation time: 0.014 seconds.
Call:
rxLinMod(formula = cost ~ age + car_age, data = sourceDataSAS)

Linear Regression Results for: cost ~ age + car_age
Data: sourceDataSAS (RxSasData Data Source)
File name:
C:/Program Files/Microsoft/ML Server/R_SERVER/library/RevoScaleR/SampleData/claims.sas7bdat
Dependent variable(s): cost
Total independent variables: 13 (Including number dropped: 2)
Number of valid observations: 123
Number of missing observations: 5

Coefficients:
cost
(Intercept) 117.38544
age=17-20    88.15174
age=21-24    34.15903
age=25-29    54.68750
age=30-34    2.93750
age=35-39    -20.77430
age=40-49    1.68750
age=50-59    63.12500
age=60+      Dropped
car_age=0-3  159.30531
```

RxSpssData

Similarly, you could use the SPSS version of the claims data as follows:

```
inFileSpss <- file.path(rxGetOption("sampleDataDir"), "claims.sav")
sourceDataSpss <- RxSpssData(inFileSpss, stringsAsFactors=TRUE)
rxLinMod(cost ~ age + car_age, data=sourceDataSpss)
```

RxXdfData

This example shows how to create a data source from the built-in claims.xdf data set:

```
claimsPath <- file.path(rxGetOption("sampleDataDir"), "claims.xdf")
claimsDs <- RxXdfData(claimsPath)
```

Use the open method **rxOpen** to open the data source:

```
rxOpen(claimsDs)
```

Use the method `rxReadNext` to read the next block of data from the data source:

```
claims <- rxReadNext(claimsDs)
```

Use the `rxClose` method to close the data source:

```
rxClose(claimsDs)
```

XDF data sources with `biglm`

Since data sources for xdf files read data in chunks, it is a good match for the CRAN package `biglm`. The `biglm` package does a linear regression on an initial chunk of data, then updates the results with subsequent chunks. Below is a function that loops through an xdf file object and creates and updates the `biglm` results.

```
# Using an Xdf Data Source with biglm

if ("biglm" %in% .packages()){
  require(biglm)
  biglmxdf <- function(dataSource, formula)
  {
    moreData <- TRUE
    df <- rxReadNext(dataSource)
    biglmRes <- biglm(formula, df)
    while (moreData)
    {
      df <- rxReadNext(dataSource)
      if (length(df) != 0)
      {
        biglmRes <- update(bigmRes, df)
      }
      else
      {
        moreData <- FALSE
      }
    }
    return(bigmRes)
  }
}
```

To use the function, we first open the data file. For example, we can again use the large airline data set `AirOnTime87to12.xdf`:

```
bigDataDir <- "C:/MRS/Data"
bigAirData <- file.path(bigDataDir, "AirOnTime87to12/AirOnTime87to12.xdf")
dataSource <- RxXdfData(bigAirData,
  varsToKeep = c("DayOfWeek", "DepDelay", "ArrDelay"), blocksPerRead = 15)
rxOpen(dataSource)
```

Then we will time the computation, doing the regression for all the rows— 148,619,655 if you are using the full data set. Note that it takes several minutes to load the data, even on a very fast machine.

```
system.time(bigmRes <- biglmxdf(dataSource, ArrDelay~DayOfWeek))
rxClose(dataSource)
```

We can see the coefficients by looking at a summary of the object returned:

```
summary(bigLmRes)

} # End of use of biglm
```

It is, of course, much faster to compute a linear model using the **rxLinMod** function, but the **biglm** package provides alternative methods of computation.

Next Steps

Continue on to the following data import articles to learn more about XDF and other data formats:

- [XDF files](#)
- [Import SQL Server data](#)
- [Import text data](#)
- [Import and consume data on HDFS](#)

See Also

[RevoScaleR Functions](#)

[Tutorial: data import and exploration](#) [Tutorial: data manipulation and statistical analysis](#)

How to transform and subset data using RevoScaleR

7/12/2022 • 28 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

A crucial step in many analyses is transforming the data into a form best suited for the chosen analysis. For example, to reduce variations in scale between variables, you might take a log or a power of the original variable before fitting the dataset to a linear model. Additionally, transforming data minimizes passes through the data, which is more efficient.

In RevoScaleR, you can perform data transformations in virtually all of its functions, from `rxImport` to `rxDataStep`, as well as the analysis functions `rxSummary`, `rxLinMod`, `rxLogit`, `rxGlm`, `rxCrossTabs`, `rxCube`, `rxCovCor`, and `rxKmeans`.

In all cases, the basic approach for data transforms is the same. The heart of the RevoScaleR data step is a list of *transforms*, each of which specifies an R expression to be evaluated. The data step typically is an assignment that either creates a new variable or modifies an existing variable from the original data set.

This article uses examples to illustrate common data manipulation tasks. For more background, see [Data Transformations](#).

Subset data by row or variable

A common use of `rxDataStep` is to create a new data set with a subset of rows and variables. The following simple example uses a data frame as the input data set.

The call to `rxDataStep` uses the *rowSelection* argument to select only the rows where the variable *y* is greater than .5, and the *varsToKeep* argument to keeps only the variables *y* and *z*. The *rowSelection* argument is an R expression that evaluates to *TRUE* if the observation should be kept. The *varsToKeep* argument contains a list of variable names to read in from the original data set. Because no *outFile* is specified, a data frame is returned.

```
# Create a data frame
set.seed(59)
myData <- data.frame(
  x = rnorm(100),
  y = runif(100),
  z = rep(1:20, times = 5))

# Subset observations and variables
myNewData <- rxDataStep( inData = myData,
  rowSelection = y > .5,
  varsToKeep = c("y", "z"))

# Get information about the new data frame
rxGetInfo(data = myNewData, getVarInfo = TRUE)
```

You should see the following results:

```

Data frame: myNewData
Number of observations: 52
Number of variables: 2
Variable information:
Var 1: y, Type: numeric, Low/High: (0.5516, 0.9941)
Var 2: z, Type: integer, Low/High: (1, 20)

```

Subsetting is particularly useful if your original data set contains millions of rows or hundreds of thousands of variables. As a smaller example, the `CensusWorkers.xdf` sample data file has six variables and 351,121 rows.

To create a subset containing only workers who have worked less than 30 weeks in the year and five variables, we can again use `rxDataStep` with the `rowSelection` and `varsToKeep` arguments. Since the resulting data set will clearly fit in memory, we omit the `outFile` argument and assign the result of the data step, then use `rxGetInfo` to see our results as usual:

```

readPath <- rxGetOption("sampleDataDir")
censusWorkers <- file.path(readPath, "CensusWorkers.xdf")
partWorkers <- rxDataStep(inData = censusWorkers, rowSelection = wkswork1 < 30,
                           varsToKeep = c("age", "sex", "wkswork1", "incwage", "perwt"))
rxGetInfo(partWorkers, getVarInfo = TRUE)

```

The result is a data set with 14,317 rows:

```

Data frame: partWorkers
Number of observations: 14317
Number of variables: 5
Variable information:
Var 1: age, Age
    Type: integer, Low/High: (20, 65)
Var 2: sex, Sex
    2 factor levels: Male Female
Var 3: wkswork1, Weeks worked last year
    Type: integer, Low/High: (21, 29)
Var 4: incwage, Wage and salary income
    Type: integer, Low/High: (0, 354000)
Var 5: perwt, Type: integer, Low/High: (2, 163)

```

Alternatively, and in this case more easily, you can use `varsToDelete` to prevent variables from being read in from the original data set. This time we'll specify an `outFile`, use the `overwrite` argument to allow the output file to be replaced.

```

partWorkersDS <- rxDataStep(inData = censusWorkers,
                            outFile = "partWorkers.xdf",
                            rowSelection = wkswork1 < 30,
                            varsToDelete = c("state"), overwrite = TRUE)

```

As noted above, if you omit the `outFile` argument to `rxDataStep`, then the results will be returned in a data frame in memory. This is true whether or not the input data is a data frame or an `.xdf` file (assuming the resulting data is small enough to reside in memory). If an `outFile` is specified, a data source object representing the new `.xdf` file is returned, which can be used in subsequent RevoScaleR function calls.

```
rxGetVarInfo(partWorkersDS)
```

Subset and transform in one operation

Still working with the CensusWorkers dataset, this exercise shows how to combine subsetting and transformations in one data step operation. Suppose we want to extract the same five variables as before from the CensusWorkers data set, but also add a factor variable based on the integer variable *age*. For example, to create our factor variable, we can use the following *transforms* argument:

```
transforms = list(ageFactor = cut(age, breaks=seq(from = 20, to = 70,
      by = 5), right = FALSE))
```

In doing a data step operation, RevoScaleR reads in a chunk of data read from the original data set, including only the variables indicated in *varsToKeep*, or omitting variables specified in *varsToDelete*. It then passes the variables needed for data transformations back to R for manipulation:

```
rxDataStep (inData = censusWorkers, outFile = "newCensusWorkers",
  varsToDelete = c("state"), transforms = list(
    ageFactor = cut(age, breaks=seq(from = 20, to = 70, by = 5),
    right = FALSE)))
```

The **rxGetInfo** function reveals the added variable:

```
rxGetInfo("newCensusWorkers", getVarInfo = TRUE)
File name: C:\YourOutputPath\newCensusWorkers.xdf
Number of observations: 351121
Number of variables: 6
Number of blocks: 6
Compression type: zlib
Variable information:
Var 1: age, Age
  Type: integer, Low/High: (20, 65)
Var 2: incwage, Wage and salary income
  Type: integer, Low/High: (0, 354000)
Var 3: perwt, Type: integer, Low/High: (2, 168)
Var 4: sex, Sex
  2 factor levels: Male Female
Var 5: wkswork1, Weeks worked last year
  Type: integer, Low/High: (21, 52)
Var 6: ageFactor
  10 factor levels: [20,25) [25,30) [30,35) [35,40) [40,45) [45,50) [50,55) [55,60) [60,65) [65,70)
```

We can combine the *transforms* argument with the *transformObjects* argument to create new variables from objects in your global environment (or other environments in your current search path).

For example, suppose you would like to estimate a linear model using wage income as the dependent variable, and want to include state-level of per capita expenditure on education as one of the independent variables. We can define a named vector to contain this state-level data as follows:

```
educExp <- c(Connecticut=1795.57, Washington=1170.46, Indiana = 1289.66)
```

We can then use **rxDataStep** to add the per capita education expenditure as a new variable using the *transforms* argument, passing *educExp* to the *transformObjects* argument as a named list:

```
censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers.xdf")
rxDataStep(inData = censusWorkers, outFile = "censusWorkersWithEduc",
  transforms = list(
    stateEducExpPC = educExp[match(state, names(educExp))] ),
  transformObjects= list(educExp=educExp))
```

The `rxGetInfo` function reveals the added variable:

```
rxGetInfo("censusWorkersWithEduc.xdf",getVarInfo=TRUE)
  File name: C:\YourOutputPath\censusWorkersWithEduc.xdf
  Number of observations: 351121
  Number of variables: 7
  Number of blocks: 6
  Compression type: zlib
  Variable information:
  Var 1: age, Age
    Type: integer, Low/High: (20, 65)
  Var 2: incwage, Wage and salary income
    Type: integer, Low/High: (0, 354000)
  Var 3: perwt, Type: integer, Low/High: (2, 168)
  Var 4: sex, Sex
    2 factor levels: Male Female
  Var 5: wkswork1, Weeks worked last year
    Type: integer, Low/High: (21, 52)
  Var 6: state
    3 factor levels: Connecticut Indiana Washington
  Var 7: stateEducExpPC, Type: numeric, Low/High: (1170.4600, 1795.5700)
```

Create a variable

Suppose we want to extract five variables from the *CensusWorkers* data set, but also add a factor variable based on the integer variable `age`. This example shows how to use a transform function to create new variables.

For example, to create our factor variable, we can create the following function:

```
# Use a function to create a factor variable using age data
ageTransform <- function(dataList)
{
  dataList$ageFactor <- cut(dataList$age, breaks=seq(from = 20, to = 70,
    by = 5), right = FALSE)
  return(dataList)
}
```

To test the function, read an arbitrary chunk out of the data set. For efficiency reasons, the data passed to the transformation function is stored as a list rather than a data frame, so when reading from the .xdf file we set the `returnDataFrame` argument to FALSE to emulate this behavior. Since we only use the variable `age` in our transformation function, we restrict the variables extracted to that.

```
censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers.xdf")
testData <- rxDataStep(inata = censusWorkers, startRow = 100, numRows = 10,
returnTransformObjects = FALSE, varsToKeep = c("age"))

as.data.frame(ageTransform(testData))
```

The resulting list of data (displayed as a data frame) shows us that our transformations are working as expected:

```
> as.data.frame(ageTransform(testData))
  age ageFactor
1 20 [20,25)
2 48 [45,50)
3 44 [40,45)
4 29 [25,30)
5 28 [25,30)
6 43 [40,45)
7 20 [20,25)
8 23 [20,25)
9 32 [30,35)
10 42 [40,45)
```

In doing a data step operation, RevoScaleR reads in a chunk of data read from the original data set, including only the variables indicated in *varsToKeep*, or omitting variables specified in *varsToDelete*. It then passes the variables needed for data transformations back to R for manipulation. We specify the variables needed to process the transformation in the *transformVars* argument. Including extra variables does not alter the analysis, but it does reduce the efficiency of the data step. In this case, since *ageFactor* depends only on the *age* variable for its creation, the *transformVars* argument needs to specify just that:

```
rxDataStep(inData = censusWorkers, outFile = "c:/temp/newCensusWorkers.xdf",
  varsToDelete = c("state"), transformFunc = ageTransform,
  transformVars=c("age"), overwrite=TRUE)
```

The **rxGetInfo** function reveals the added and dropped variables:

```
rxGetInfo("newCensusWorkers", getVarInfo = TRUE)

File name: C:\YourOutputPath\newCensusWorkers.xdf
Number of rows: 351121
Number of variables: 6
Number of blocks: 6
Compression type: zlib
Variable information:
Var 1: age, Age
  Type: integer, Low/High: (20, 65)
Var 2: incwage, Wage and salary income
  Type: integer, Low/High: (0, 354000)
Var 3: perwt, Type: integer, Low/High: (2, 168)
Var 4: sex, Sex
  2 factor levels: Male Female
Var 5: wkswork1, Weeks worked last year
  Type: integer, Low/High: (21, 52)
Var 6: ageFactor
  10 factor levels: [20,25) [25,30) [30,35) [35,40) [40,45) [45,50)
    [50,55) [55,60) [60,65) [65,70)
```

Modify variable metadata

To change variable information (rather than the data values themselves), use the function **rxSetVarInfo**. For example, using the CensusData, we can change the names of two variables and add descriptions:

```

# Modifying Variable Information

newVarInfo <- list(
  incwage = list(newName = "WageIncome"),
  state   = list(newName = "State", description = "State of Residence"),
  stateEducExpPC = list(description = "State Per Capita Educ Exp"))
fileName <- "censusWorkersWithEduc.xdf"
rxSetVarInfo(varInfo = newVarInfo, data = fileName)
rxGetVarInfo( fileName )

Var 1: age, Age
      Type: integer, Low/High: (20, 65)
Var 2: WageIncome, Wage and salary income
      Type: integer, Low/High: (0, 354000)
Var 3: perwt, Type: integer, Low/High: (2, 168)
Var 4: sex, Sex
      2 factor levels: Male Female
Var 5: wkswork1, Weeks worked last year
      Type: integer, Low/High: (21, 52)
Var 6: State, State of Residence
      3 factor levels: Connecticut Indiana Washington
Var 7: stateEducExpPC, State Per Capita Educ Exp
      Type: numeric, Low/High: (1170.4600, 1795.5700)

```

Add data to an analysis

It is sometimes useful to access additional information from within a transform function. For example, you might want to match additional data in the process of creating new variables. Transform functions are evaluated in a “sterilized” environment which includes the parent environment of the function closure. To provide access to additional data within the function, you can use the *transformObjects* argument.

For example, suppose you would like to estimate a linear model using wage income as the dependent variable, and want to include state-level of per capita expenditure on education as one of the independent variables. We can define a named vector to contain this state-level data as follows:

```

# Using Additional Objects or Data in a Transform Function

educExpense <- c(Connecticut=1795.57, Washington=1170.46, Indiana = 1289.66)

```

We can then define a transform function that uses this information as follows:

```

transformFunc <- function(dataList)
{
  # Match each individual's state and add the variable for educ. exp.
  dataList$stateEducExpPC = educExp[match(dataList$state, names(educExp))]
  return(dataList)
}

```

We can then use the transform function and our named vector in a call to *rxLinMod* as follows:

```

censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers.xdf")
linModObj <- rxLinMod(incwage~sex + age + stateEducExpPC,
                      data = censusWorkers, pweights = "perwt",
                      transformFun = transformFunc, transformVars = "state",
                      transformObjects = list(educExp = educExpense))
summary(linModObj)

```

When the transform function is evaluated, it will have access to the *educExp* object. The final results show:

```

Call:
rxLinMod(formula = incwage ~ sex + age + stateEducExpPC, data = censusWorkers,
  pweights = "perwt", transformObjects = list(educExp = educExp),
  transformFunc = transformFunc, transformVars = "state")

Linear Regression Results for: incwage ~ sex + age + stateEducExpPC
File name:
  C:\Program Files\Microsoft\MRO-for-RRE\8.0\R-3.2.2\library\RevoScaleR\SampleData\CensusWorkers.xdf
Probability weights: perwt
Dependent variable(s): incwage
Total independent variables: 5 (Including number dropped: 1)
Number of valid observations: 351121
Number of missing observations: 0

Coefficients: (1 not defined because of singularities)
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.809e+04 4.414e+02 -40.99 2.22e-16 ***
sex=Male     1.689e+04 1.321e+02 127.83 2.22e-16 ***
sex=Female   Dropped   Dropped Dropped Dropped
age          5.593e+02 5.791e+00  96.58 2.22e-16 ***
stateEducExpPC 1.643e+01 2.731e-01   60.16 2.22e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 175900 on 351117 degrees of freedom
Multiple R-squared: 0.07755
Adjusted R-squared: 0.07754
F-statistic: 9839 on 3 and 351117 DF,  p-value: < 2.2e-16
Condition number: 1.1176

```

Create a row selection variable

One common use of the `transformFunc` argument is to create a logical variable to use as a row selection variable. For example, suppose you want to create a random sample from a massive data set. You can use the `transformFunc` argument to specify a transformation that creates a random binomial variable, which can then be coerced to logical and used for row selection. The object name `.rxRowSelection` is reserved for the row selection variable; if RevoScaleR finds this object, it is used for row selection.

NOTE

If you both specify a `rowSelection` argument and define a `.rxRowSelection` variable in your transform function, the one specified in your transform function will be overwritten by the contents of the `rowSelection` argument, so that expression takes precedence.

The following code creates a random selection variable to create a data frame with a random 10% subset of the census workers file:

```

# Creating a Row Selection Variable

createRandomSample <- function(data)
{
  data$.rxRowSelection <- as.logical(rbinom(length(data[[1]]), 1, .10))
  return(data)
}
censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers.xdf")
df <- rxImport(file = censusWorkers, transformFunc = createRandomSample,
  transformVars = "age")

```

The resulting data frame, `df`, has approximately 35,000 rows. You can look at the first few rows using `head` as

follows:

```
rxGetInfo(df)
head(df)

  age incwage perwt   sex wkswork1   state
  1  50    9000    30 Male      48 Indiana
  2  41   35000    20 Female     48 Indiana
  3  55   40400    21 Male      52 Indiana
  4  56   45000    30 Female     52 Indiana
  5  46   17200    60 Female     52 Indiana
  6  49   35000    21 Female     52 Indiana
```

Equivalently, we could create the temporary row selection variable using the *rowSelection* argument with the internal *.rxNumRows* variable, which provides the number of rows in the current chunk of data being processed:

```
df <- rxDataStep(inData = censusWorkers,
  rowSelection = as.logical(rbinom(.rxNumRows, 1, .10)) == TRUE)
```

Convert a data frame to XDF

You can use all of the functionality provided by the **rxDataStep** function to create an *.xdf* file from a data frame for further use. For example, create a simple data frame:

```
# Using the Data Step to Create an .xdf File from a Data Frame
set.seed(39)
myData <- data.frame(x1 = rnorm(10000), x2 = runif(10000))
```

Now create an *.xdf* file, using a row selection and creating a new variable. The *rowsPerRead* argument will specify how many rows of the original data frame to process at a time before writing out a block of the new *.xdf* file.

```
rxDataStep(inData = myData, outFile = "testFile.xdf",
  rowSelection = x2 > .1,
  transforms = list( x3 = x1 + x2 ),
  rowsPerRead = 5000 )
rxGetInfo("testFile.xdf")

  File name: C:\Users\...\testFile.xdf
  Number of observations: 8970
  Number of variables: 3
  Number of blocks: 2
```

Convert XDF to Text

If you need to share data with others not using *.xdf* data files, you can export your *.xdf* files to text format using the **rxDataStep** function. For example, we can write the *claims.xdf* file we created earlier to text format as follows:

```
# Converting .xdf Files to Text
claimsCsv <- RxTextData(file="claims.csv")
claimsXdf <- RxXdfData(file="claims.xdf")

rxDataStep(inData=claimsXdf, outFile=claimsCsv)
```

By default, the text file created is comma-delimited, but you can change this by specifying a different delimiter with the *delimiter* argument to the RxTextData function:

```
claimsTxt <- RxTextData(file="claims.txt", delimiter="\t")
rxDataStep(inData=claimsXdf, outFile=claimsTxt)
```

If you have a large number of variables, you can choose to write out only a subsample by using the *VarsToKeep* or *VarsToDelete* argument. Here we write out the claims data, omitting the *number* variable:

```
rxDataStep(inData=claimsXdf, outFile=claimsTxt, varsToDelete="number",
           overwrite=TRUE)
```

Convert strings to factors

Factors are variables that represent categories. An example is "sex", which has the categories "Male" and "Female". There are two parts to a factor variable:

1. A vector of integer indexes with values in the range of 1:K, where K is the number of categories. For example, "sex" has two categories with indexes 1 and 2. The length of the vector corresponds to the number of observations.
2. A vector of K character strings that are used when the factor is displayed. In R, these are normally printed without quote marks, to indicate that the variable is a factor instead of a character string.

If you have character data in an .xdf file or data frame, you can use the **rxFactors** function to convert it to a factor. Let's create a simple data frame with character data. Note that by default, *data.frame* converts character data to factor data. That is, the *stringsAsFactors* argument defaults to *TRUE*. In RevoScaleR's **rxImport** function, *stringsAsFactors* has a default of *FALSE*.

```
# Creating factors from character data
myData <- data.frame(
  id = 1:10,
  sex = c("M", "F", "M", "F", "F", "M", "F", "F", "M", "M"),
  state = c(rep(c("WA", "CA"), each = 5)),
  stringsAsFactors = FALSE)
rxGetVarInfo(myData)
Var 1: id, Type: integer, Low/High: (1, 10)
Var 2: sex, Type: character
Var 3: state, Type: character
```

Now we can use **rxFactors** to convert the character data to factors. We can just specify a vector of variable names to convert as the *factorInfo*:

```
myNewData <- rxFactors(inData = myData, factorInfo = c("sex", "state"))
rxGetVarInfo(myNewData)
Var 1: id, Type: integer, Low/High: (1, 10)
Var 2: sex
  2 factor levels: M F
Var 3: state
  2 factor levels: WA CA
```

Note that by default, the factor levels are in the order in which they are encountered. If you would like to have them sorted alphabetically, specify *sortLevels* to be *TRUE*.

```

myNewData <- rxFactors(inData = myData, factorInfo = c("sex", "state"),
  sortLevels = TRUE)
rxGetVarInfo(myNewData)
Var 1: id, Type: integer, Low/High: (1, 10)
Var 2: sex
  2 factor levels: F M
Var 3: state
  2 factor levels: CA WA

```

If you have variables that are already factors, you may want change the order of the levels, the names of the levels, or how they are grouped. Typically recoding a factor means changing from one set of indexes to another. For example, if the levels of "sex" are currently arranged in the order "M", "F" and you want to change that to "F", "M", you need to change the index for every observation.

You can use the **rxFactors** function to recode factors in RevoScaleR. For example, suppose we have some test scores for a set of male and female subjects. We can generate such data randomly as follows:

```

# Recoding Factors

set.seed(100)
sex <- factor(sample(c("M","F"), size = 10, replace = TRUE),
  levels = c("M", "F"))
DF <- data.frame(sex = sex, score = rnorm(10))

```

If we look at just the sex variable, we see the levels M or F for each observation:

```

DF[["sex"]]

[1] M M F M M M F M F M
Levels: M F

```

To recode this factor so that "Female" is the first level and "Male" the second, we can use **rxFactors** as follows:

```

newDF <- rxFactors(inData = DF, overwrite = TRUE,
  factorInfo = list(Gender = list(newLevels = c(Female = "F",
    Male = "M"),
  varName = "sex")))

```

Looking at the new Gender variable, we see how the levels have changed:

```

newDF$Gender

[1] Male   Male   Female Male   Male   Male   Female Male   Female Male
Levels: Female Male

```

As mentioned earlier, by default, RevoScaleR codes factor levels in the order in which they are encountered in the input file(s). This could lead you to have a State variable ordered as "Maine", "Vermont", "New Hampshire", "Massachusetts", "Connecticut", and so forth.

Usually, you would prefer to have the levels of such a variable sorted in alphabetical order. You can do this with **rxFactors** using the *sortLevels* flag. It is most useful to specify this flag as part of the *factorInfo* list for each variable, although if you have a large number of factors and want most of them to be sorted, you can also set the flag to TRUE globally and then specify *sortLevels=FALSE* for those variables you want to order in a different way.

When using the *sortLevels* flag, it is useful to keep in mind that it is the *levels* that are being sorted, not the data

itself, and that the levels are always character data. If you are using the individual values of a continuous variable as factor levels, you may be surprised by the sorted order of the levels: for example, the levels 1, 3, 20 are sorted as "1", "20", "3".

Another common use of factor recoding is in analyzing survey data gathered using Likert items with five or seven level responses. For example, suppose a customer satisfaction survey offered the following seven-level responses to each of four questions:

1. Completely satisfied
2. Mostly satisfied
3. Somewhat satisfied
4. Neither satisfied nor dissatisfied
5. Somewhat dissatisfied
6. Mostly dissatisfied
7. Completely dissatisfied

In analyzing this data, the survey analyst may recode the factors to focus on those who were largely satisfied (combining levels 1 and 2), largely dissatisfied (combining levels 6 and 7) and somewhere in-between (combining levels 3, 4, and 5). The CustomerSurvey.xdf file in the SampleData directory contains 25 responses to each of the four survey questions. We can read it into a data frame as follows:

```
# Combining factor levels

surveyDF <- rxDataStep(inData =
  file.path(rxGetOption("sampleDataDir"), "CustomerSurvey.xdf"))
```

To recode each question as desired, we can use **rxFactors** as follows:

```
s1 <- levels(surveyDF[[1]])
quarterList <- list(newLevels = list(
  "Largely Satisfied" = s1[1:2],
  "Neither Satisfied Nor Dissatisfied" = s1[3:5],
  "Largely Dissatisfied" = s1[6:7]))
surveyDF <- rxFactors(inData = surveyDF,
  factorInfo <- list(
    Q1 = quarterList,
    Q2 = quarterList,
    Q3 = quarterList,
    Q4 = quarterList))
```

Looking at just Q1, we see the recoded factor:

```

surveyDF[["Q1"]]

[1] Neither Satisfied Nor Dissatisfied Largely Satisfied
[3] Neither Satisfied Nor Dissatisfied Largely Satisfied
[5] Neither Satisfied Nor Dissatisfied Neither Satisfied Nor Dissatisfied
[7] Largely Dissatisfied           Neither Satisfied Nor Dissatisfied
[9] Neither Satisfied Nor Dissatisfied Largely Satisfied
[11] Neither Satisfied Nor Dissatisfied Largely Dissatisfied
[13] Largely Satisfied           Neither Satisfied Nor Dissatisfied
[15] Largely Dissatisfied         Neither Satisfied Nor Dissatisfied
[17] Largely Satisfied           Neither Satisfied Nor Dissatisfied
[19] Neither Satisfied Nor Dissatisfied Neither Satisfied Nor Dissatisfied
[21] Neither Satisfied Nor Dissatisfied Neither Satisfied Nor Dissatisfied
[23] Neither Satisfied Nor Dissatisfied Largely Dissatisfied
[25] Neither Satisfied Nor Dissatisfied
3 Levels: Largely Satisfied ... Largely Dissatisfied

```

Recode factors for compatibility

One important use of factor recoding in RevoScaleR is to ensure that the factor variables in two files are compatible, that is, have the same levels with the same coding. This use comes up in a variety of contexts, including prediction, merging, and distributed computing. For example, suppose you are creating a logistic regression model of whether a given airline flight will be late, and are using the first fifteen years of the airline data as a training set. You then want to test the model on the remaining years of the airline data. You need to ensure that the two files, the training set and the test set, have compatible factor variables. You can generally do this easily using `rxGetInfo` (with `getVarInfo=TRUE`) together with `rxFactors`. Use `rxGetInfo` to find all the levels in all the files, then use `rxFactors` to recode each file so that every factor variable contains all the levels found in any of the files.

The `rxMerge` function automatically checks for factor variable compatibility and recodes on the fly if necessary.

Impute values

A common use case is replace missing values with the variable mean. This example uses generated data in a simple data frame.

```

# Create a data frame with missing values
set.seed(59)
myData1 <- data.frame(x = rnorm(100), y = runif(100))

xmiss <- seq.int(from = 5, to = 100, by = 5)
ymiss <- seq.int(from = 2, to = 100, by = 5)
myData1$x[xmiss] <- NA
myData1$y[ymiss] <- NA
rxGetInfo(myData1, numRows = 5)

```

A call to `rxGetInfo` returns precomputed metadata showing missing values "NA" in both variables:

```

Data frame: myData1
Number of observations: 100
Number of variables: 2
Data (5 rows starting with row 1):
  x          y
1 -1.8621337 0.06206201
2  1.1398069      NA
3  0.3176267 0.84132161
4  1.3998593 0.26298559
5      NA 0.97069679

```

Now use `rxSummary` to compute summary statistics that includes a variable mean, putting both computed

means into a named vector:

```
# Compute summary statistics and extract to a named vector
sumStats <- rxResultsDF(rxSummary(~., myData1))
sumStats
meanVals <- sumStats$Mean
names(meanVals) <- row.names(sumStats)
```

The computed statistics are:

	Mean	StdDev	Min	Max	ValidObs	MissingObs
x	0.07431126	0.9350711	-1.94160646	1.9933814	80	20
y	0.54622241	0.3003457	0.04997869	0.9930338	80	20

Finally, pass the computed means into a **rxDataStep** using the *transformObjects* argument:

```
# Use rxDataStep to replace missings with imputed mean values
myData2 <- rxDataStep(inData = myData1, transforms = list(
  x = ifelse(is.na(x), meanVals["x"], x),
  y = ifelse(is.na(y), meanVals["y"], y)),
  transformObjects = list(meanVals = meanVals))
rxGetInfo(myData2, numRows = 5)
```

The resulting data set information substitutes NA with computed means generated in the previous step:

```
Data frame: myData2
Number of observations: 100
Number of variables: 2
Data (5 rows starting with row 1):
      x         y
1 -1.86213372 0.06206201
2  1.13980693 0.54622241
3  0.31762673 0.84132161
4  1.39985928 0.26298559
5  0.07431126 0.97069679
```

Use internal variables in a transformation

This example shows how to compute moving averages using internal variables in a transformation function.

Four additional variables providing information on the RevoScaleR processing are available for use in your transform functions:

- **.rxStartRow**: The row number from the original data that was read as the first row of the current chunk.
- **.rxChunkNum**: The current chunk being processed.
- **.rxReadFileName**: The name of the .xdf file currently being read.
- **.rxIsTestChunk**: Whether the chunk being processed is being processed as a test sample of data.

These are particularly useful if you need to access additional rows of data when processing a chunk. This is the case, for example, when you want to include lagged data in a calculation. The following example is a transformation function "generator" to compute a simple moving average. By creating this "function within a function" we are able to easily pass arguments into a transformation function. This one takes as arguments the number of days (or time units) for the moving average, the name of the variable that will be used to compute the moving average, and the name of the new variable to create. The transformation function computes the number of lagged observations to read in, then reads in that data to create a long data vector with the lags. The simple moving average calculations are performed and put into a new variable. The additional lagged rows are

removed from the original data vector before returning from the function.

```
# An Example Computing Moving Averages

makeMoveAveTransFunc <- function(numDays = 10, varName="", newVarName="")
{
  function (dataList)
  {
    numRowsToRead <- 0
    varForMoveAve <- 0
    # If no variable is named, use the first one sent
    # to the transformFunc
    if (is.null(varName) || nchar(varName) == 0)
    {
      varForMoveAve <- names(dataList)[1]
    } else {
      varForMoveAve <- varName
    }

    # Get the number of rows in the current chunk
    numRowsInChunk <- length(dataList[[varForMoveAve]])

    # .rxStartRow is the starting row number of the
    # chunk of data currently being processed
    # Read in previous data if we are not starting at row 1
    if (.rxStartRow > 1)
    {
      # Compute the number of lagged rows we'd like
      numRowsToRead <- numDays - 1
      # Check to see if enough data is available
      if (numRowsToRead >= .rxStartRow)
      {
        numRowsToRead <- .rxStartRow - 1
      }
      # Compute the starting row of previous data to read
      startRow <- .rxStartRow - numRowsToRead
      # Read previous rows from the .xdf file
      previousRowsDataList <- RevoScaleR::rxDataStep(
        inData=.rxReadFileName,
        varsToKeep=names(dataList),
        startRow=startRow, numRows=numRowsToRead,
        returnTransformObjects=FALSE)
      # Concatenate the previous rows with the existing rows
      dataList[[varForMoveAve]] <-
        c(previousRowsDataList[[varForMoveAve]],
          dataList[[varForMoveAve]])
    }
    # Create variable for simple moving average
    # It will be added as the last variable in the data list
    newVarIdx <- length(dataList) + 1

    # Initialize with NA's
    dataList[[newVarIdx]] <- rep(as.numeric(NA), times=numRowsInChunk)

    for (i in (numRowsToRead+1):(numRowsInChunk + numRowsToRead))
    {
      j <- i - numRowsToRead
      lowIdx <- i - numDays + 1
      if (lowIdx > 0 && ((lowIdx == 1) ||
        (j > 1 && (is.na(dataList[[newVarIdx]][j-1]))))
      {
        # If it's the first computation or the previous value
        # is missing, take the mean of all the relevant lagged data
        dataList[[newVarIdx]][j] <-
          mean(dataList[[varForMoveAve]][lowIdx:i])
      } else if (lowIdx > 1)
      {
        # Add and subtract from the last computation
      }
    }
  }
}
```

```

dataList[[newVarIdx]][j] <- dataList[[newVarIdx]][j-1] -
  dataList[[varForMoveAve]][lowIdx-1]/numDays +
  dataList[[varForMoveAve]][i]/numDays
}
}
# Remove the extra rows we read in from the original variable
dataList[[varForMoveAve]] <-
  dataList[[varForMoveAve]][(numRowsToRead + 1):(numRowsToRead +
  numRowsInChunk)]

# Name the new variable
if (is.null(newVarName) || (nchar(newVarName) == 0))
{
  # Use a default name if no name specified
  names(dataList)[newVarIdx] <-
    paste(varForMoveAve, "SMA", numDays, sep=".")
} else {
  names(dataList)[newVarIdx] <- newVarName
}
return(dataList)
}
}

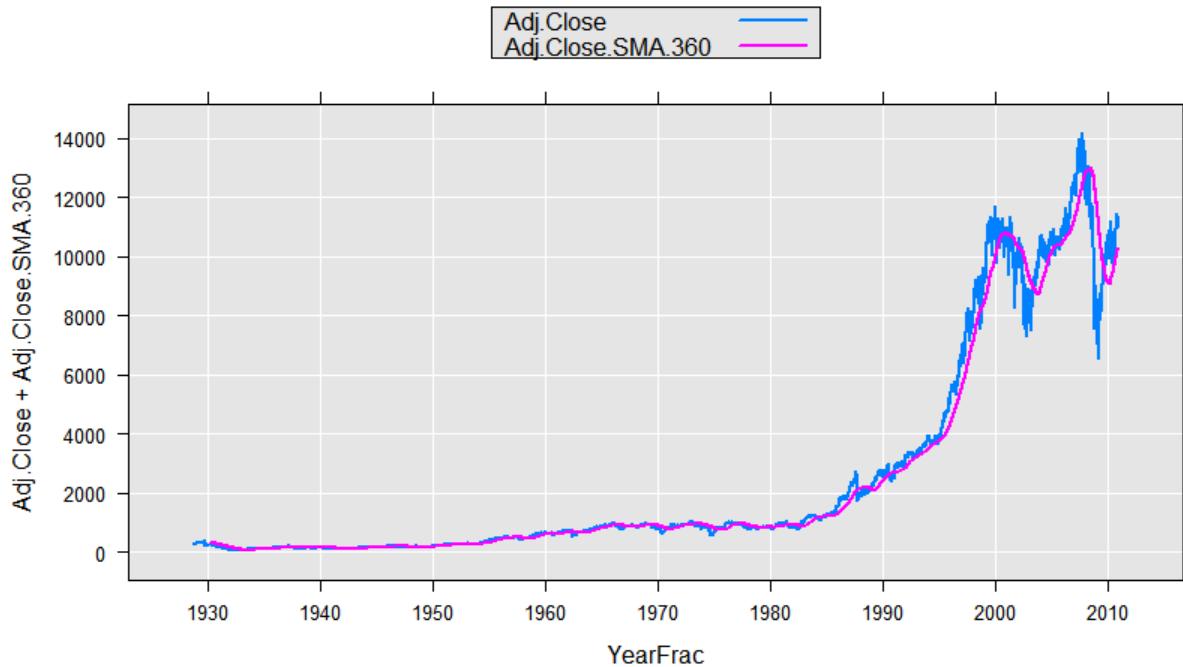
```

We could use this function with `rxDataStep` to add a variable to our data set, or on-the-fly, for example for plotting. Here we will use the sample data set containing daily information on the Dow Jones Industrial Average. We compute a 360-day moving average for the adjusted close (adjusted for dividends and stock splits) and plot it:

```

DJIAdaily <- file.path(rxGetOption("sampleDataDir"), "DJIAdaily.xdf")
rxLinePlot(Adj.Close+Adj.Close.SMA.360~YearFrac, data=DJIAdaily,
  transformFunc=makeMoveAveTransFunc(numDays=360),
  transformVars=c("Adj.Close"))

```



Extended example showing a series of transformations

A `transforms` argument can be a powerful way to effect a sequence of changes on a row by row basis. The `transforms` argument is specified as a list of expressions. Given R expressions operate row-by-row (that is, the computed value of the new variable for an observation is only dependent on values of other variables for that

observation), you can combine them into a single *transforms* argument, as this example demonstrates.

Start with a simple data frame containing a small sample of transaction data:

```
# Transforming Data with rxDataStep

expData <- data.frame(
  BuyDate = c("2011/10/1", "2011/10/1", "2011/10/1",
  "2011/10/2", "2011/10/2", "2011/10/2", "2011/10/2",
  "2011/10/3", "2011/10/4", "2011/10/4"),
  Food =      c( 32, 102, 34, 5, 0, 175, 15, 76, 23, 14),
  Wine =      c( 0, 212, 0, 0, 425, 22, 0, 12, 0, 56),
  Garden =    c( 0, 46, 0, 0, 0, 45, 223, 0, 0, 0),
  House =     c( 22, 72, 56, 3, 0, 0, 0, 37, 48, 23),
  Sex =       factor(c("F", "F", "M", "M", "F", "F", "F", "M", "F")),
  Age =        c( 20, 51, 32, 16, 61, 42, 35, 99, 29, 55),
  stringsAsFactors = FALSE)
```

Apply a series of data transformations:

- Compute the total expenditures for each store visit
- Compute the average category expenditure for each store visit
- Set the variable *Age* to missing if the value is 99
- Create a new logical variable *UnderAge* if the purchaser is under 21
- Find the day of week the purchase was made using R's *as.POSIXlt* function, then convert it to a factor. [Note that when creating factors in a transform, you must specify the levels or you may get unpredictable results. Alternatively use *rxFactors*.]
- Create a factor variable for categories of expenditure levels
- Create a logical variable for expenditures of \$50 or more on either Food or Wine
- Remove the variable *BuyDate* from the data set

The following call to **rxDataStep** will accomplish all of the above, returning a new data frame with the transformed variables:

```
newExpData = rxDataStep( inData = expData,
  transforms = list(
    Total      = Food + Wine + Garden + House,
    AveCat    = Total/4,
    Age        = ifelse(Age == 99, NA, Age),
    UnderAge  = Age < 21,
    Day        = (as.POSIXlt(BuyDate))$wday,
    Day        = factor(Day, levels = 0:6,
      labels = c("Su", "M", "Tu", "W", "Th", "F", "Sa")),
    SpendCat  = cut(Total, breaks=c(0, 75, 250, 10000),
      labels=c("low", "medium", "high"), right=FALSE),
    FoodWine  = ifelse( Food > 50, TRUE, FALSE),
    FoodWine  = ifelse( Wine > 50, TRUE, FoodWine),
    BuyDate   = NULL))
newExpData
```

The new data frame shows all of the transformed data:

	Food	Wine	Garden	House	Sex	Age	Total	AveCat	UnderAge	Day	SpendCat	FoodWine
1	32	0	0	22	F	20	54	13.50	TRUE	Sa	low	FALSE
2	102	212	46	72	F	51	432	108.00	FALSE	Sa	high	TRUE
3	34	0	0	56	M	32	90	22.50	FALSE	Sa	medium	FALSE
4	5	0	0	3	M	16	8	2.00	TRUE	Su	low	FALSE
5	0	425	0	0	M	61	425	106.25	FALSE	Su	high	TRUE
6	175	22	45	0	F	42	242	60.50	FALSE	Su	medium	TRUE
7	15	0	223	0	F	35	238	59.50	FALSE	Su	medium	FALSE
8	76	12	0	37	F	NA	125	31.25	NA	M	medium	TRUE
9	23	0	0	48	M	29	71	17.75	FALSE	Tu	low	FALSE
10	14	56	0	23	F	55	93	23.25	FALSE	Tu	medium	TRUE

If we had a large data set containing expenditure data in an .xdf file, we could use exactly the same transformation code; the only changes in the call to `rxDataStep` would be the `inData` and the addition of an `outFile` for the newly created data set.

Next Steps

Continue on to the following data-related articles to learn more about XDF, data source objects, and other data formats:

- [Data transformations \(introduction\)](#)
- [XDF files](#)
- [Data Sources](#)
- [Import text data](#)
- [Import ODBC data](#)
- [Import and consume data on HDFS](#)

See Also

[RevoScaleR Functions](#)

[Tutorial: data import and exploration](#) [Tutorial: data visualization and analysis](#)

How to sort data using rxSort in RevoScaleR

7/12/2022 • 7 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Many analysis and plotting algorithms require as a first step that the data be sorted. Sorting a massive data set is both memory-intensive and time-consuming, but the `rxSort` function provides an efficient solution. The `rxSort` function allows you to sort by one or many keys. A *stable* sorting routine is used, so that, in the case of ties, remaining columns are left in the same order as they were in the original data set.

As a simple example, we can sort the census worker data by `age` and `incwage`. We will sort first by `age`, using the default increasing sort, and then by `incwage`, which we will sort in decreasing order:

```
# Sorting Data

censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers.xdf")
outXDF <- "censusWorkersSorted.xdf"
rxSort(inData = censusWorkers, outFile = outXDF,
      sortByVars=c("age", "incwage"), decreasing=c(FALSE, TRUE))
```

The first few lines of the sorted file can be viewed as follows:

```
rxGetInfo(outXDF, numRows=10)

File name: C:\YourOutputPath\censusWorkersSorted.xdf
Number of observations: 351121
Number of variables: 6
Number of blocks: 6
Compression type: zlib
Data (10 rows starting with row 1):
  age incwage perwt    sex wkswork1      state
  1  20  336000     3   Male       40 Washington
  2  20  336000    23   Male       46 Washington
  3  20  336000    11   Male       52 Washington
  4  20  314000    33   Male       52   Indiana
  5  20  168000    13   Male       24   Indiana
  6  20  163000    16   Male       26 Washington
  7  20  144000    27 Female     24   Indiana
  8  20   96000    21   Male       48 Washington
  9  20   93000    24   Male       24   Indiana
 10 20   90000     6   Male       52 Washington
```

If the sort keys contain missing values, you can use the `missingsLow` flag to specify whether they are sorted as low values (`missingsLow=TRUE`, the default) or high values (`missingsLow=FALSE`).

Remove duplicates during sort

In many situations, you are sorting a large data set by a particular key, for example, `userID`, but are looking for a sorted list of unique `userID`s. The `removeDupKeys` argument to `rxSort` allows you to remove the duplicate entries from a sorted list. This argument is supported only for `type="auto"` and `type="mergeSort"`; it is ignored

for *type*="varByVar".

When you use *removeDupKeys*=TRUE, the first record containing a unique combination of the *sortByVars* is retained; subsequent matching records are omitted from the sorted results, but, if desired, a count of the matching records is maintained in a new *dupFreqVar* output column. For example, the following artificial data set simulates a small amount of transaction data, with a user name, a state, and a transaction amount. When we sort by the variables *users* and *state* and specify *removeDupKeys*=TRUE, the *transAmt* shown for duplicate entries is the transaction amount for the *first* transaction encountered:

```
set.seed(17)
users <- sample(c("Aiden", "Ella", "Jayden", "Ava", "Max", "Grace", "Riley",
  "Lolita", "Liam", "Emma", "Ethan", "Elizabeth", "Jack",
  "Genevieve", "Avery", "Aurora", "Dylan", "Isabella",
  "Caleb", "Bella"), 100, replace=TRUE)
state <- sample(c("Washington", "California", "Texas", "North Carolina",
  "New York", "Massachusetts"), 100, replace=TRUE)
transAmt <- round(runif(100)*100, digits=3)
df <- data.frame(users=users, state=state, transAmt=transAmt)

rxSort(df, sortByVars=c("users", "state"), removeDupKeys=TRUE,
  dupFreqVar = "DUP_COUNT")
Number of rows written to file: 66, Variable(s): users, state, transAmt, DUP_COUNT, Total number of rows in
file: 66
Time to sort data file: 0.100 seconds
   users      state transAmt DUP_COUNT
1   Aiden    New York   11.010      1
2   Aiden  North Carolina   73.307      1
3   Aiden        Texas   8.037      2
4   Aurora   California   8.787      1
5   Aurora Massachusetts   55.187      1
6   Aurora    New York   91.648      1
7   Aurora        Texas   30.566      1
8   Aurora Washington   27.374      1
9     Ava   California   70.638      2
10   Ava Massachusetts   82.916      2
11   Ava    New York   45.683      2
12   Ava North Carolina   51.748      1
13   Ava        Texas   52.674      1
14   Avery   California   9.756      4
15   Avery Massachusetts   63.715      1
16   Avery    New York   93.430      1
17   Avery North Carolina   1.889      2
18   Bella   California   60.258      2
19   Bella        Texas   32.684      1
20   Bella Washington   60.230      2
21   Caleb Massachusetts   94.527      1
22   Caleb North Carolina   89.259      2
23   Dylan   California   73.665      1
24   Dylan North Carolina   98.384      1
25   Dylan        Texas   27.067      1
26   Dylan Washington   82.141      3
27 Elizabeth   California   95.497      2
28 Elizabeth    New York   35.546      3
29 Elizabeth North Carolina   18.892      2
30   Ella   California   11.644      1
31   Ella North Carolina   72.289      1
32   Ella Washington   66.453      2
33   Emma Massachusetts   28.502      1
34   Emma North Carolina   63.067      1
35   Ethan Massachusetts   31.480      1
36   Ethan    New York   95.639      1
37   Ethan North Carolina   6.561      1
38   Ethan        Texas   29.963      1
39   Ethan Washington   44.187      2
40 Genevieve Massachusetts   90.783      1
41   Grace   New York   18.232      1
```

42	Grace	North Carolina	5.355	2
43	Grace	Washington	91.084	1
44	Isabella	New York	4.115	1
45	Isabella	North Carolina	12.942	2
46	Isabella	Texas	2.227	2
47	Jack	California	40.905	1
48	Jack	Massachusetts	98.080	2
49	Jack	New York	8.071	2
50	Jack	North Carolina	11.304	3
51	Jack	Texas	18.795	2
52	Jayden	Massachusetts	83.949	1
53	Jayden	North Carolina	67.769	1
54	Jayden	Washington	4.360	1
55	Liam	California	3.300	1
56	Liam	Massachusetts	87.585	1
57	Liam	New York	96.599	1
58	Liam	North Carolina	32.997	1
59	Lolita	California	18.102	1
60	Lolita	Washington	30.649	2
61	Max	Massachusetts	21.683	2
62	Max	New York	14.852	2
63	Max	North Carolina	79.982	2
64	Max	Texas	66.749	2
65	Max	Washington	67.326	2
66	Riley	New York	20.527	2

Removing duplicates can be a useful way to reduce the size of a data set without losing information of interest. For example, consider an analysis of using data from the sample *AirlineDemoSmall.xdf* file. It has 600,000 observations and contains the variables *DayOfWeek*, *CRSDepTime*, and *ArrDelay*. We can create a smaller data set reducing the number of observations, and adding a variable that contains the frequency of the duplicated observation.

```
sampleDataDir <- rxGetOption("sampleDataDir")
airDemo <- file.path(sampleDataDir, "AirlineDemoSmall.xdf")
airDedup <- file.path(tempdir(), "rxAirDedup.xdf")
rxSort(inData = airDemo, outFile = airDedup,
       sortByVars = c("DayOfWeek", "CRSDepTime", "ArrDelay"),
       removeDupKeys = TRUE, dupFreqVar = "FreqWt")
rxGetInfo(airDedup)
```

The new data file contains about 1/3 of the observations and one additional variable:

```
File name: C:\YourTempDir\rxAirDedup.xdf
Number of observations: 232451
Number of variables: 4
Number of blocks: 2
Compression type: zlib
```

By using the frequency weights argument, we can use many of the RevoScaleR analysis functions on this smaller data set and get same results as we would using the full data set. For example, a linear model for Arrival Delay can be specified as follows, using the *fweights* argument:

```

linModObj <- rxLinMod(ArrDelay~CRSDepTime + DayOfWeek, data = airDedup,
  fweights = "FreqWt")
summary(linModObj)

Call:
rxLinMod(formula = ArrDelay ~ CRSDepTime + DayOfWeek, data = airDedup,
  fweights = "FreqWt")

Linear Regression Results for: ArrDelay ~ CRSDepTime + DayOfWeek
File name: C:\YourTempDir\rxAirDedup.xdf
Frequency weights: FreqWt
Dependent variable(s): ArrDelay
Total independent variables: 9 (Including number dropped: 1)
Sum of weights of valid observations: 582628
Number of missing observations: 3503

Coefficients: (1 not defined because of singularities)
Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.19458 0.20413 -15.650 2.22e-16 ***
CRSDepTime 0.97862 0.01126 86.948 2.22e-16 ***
DayOfWeek=Monday 2.08100 0.18602 11.187 2.22e-16 ***
DayOfWeek=Tuesday 1.34015 0.19881 6.741 1.58e-11 ***
DayOfWeek=Wednesday 0.15155 0.19679 0.770 0.441
DayOfWeek=Thursday -1.32301 0.19518 -6.778 1.22e-11 ***
DayOfWeek=Friday 4.80042 0.19452 24.679 2.22e-16 ***
DayOfWeek=Saturday 2.18965 0.19229 11.387 2.22e-16 ***
DayOfWeek=Sunday Dropped Dropped Dropped Dropped
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 40.39 on 582620 degrees of freedom
Multiple R-squared: 0.01465
Adjusted R-squared: 0.01464
F-statistic: 1238 on 7 and 582620 DF, p-value: < 2.2e-16
Condition number: 10.6542

```

Using the full data set, we get the following results:

```

linModObjBig <- rxLinMod(ArrDelay~CRSDepTime + DayOfWeek, data = airDemo)
summary(linModObjBig)

Call:
rxLinMod(formula = ArrDelay ~ CRSDepTime + DayOfWeek, data = airDemo)

Linear Regression Results for: ArrDelay ~ CRSDepTime + DayOfWeek
File name: C:\YourSampleDir\AirlineDemoSmall.xdf
Dependent variable(s): ArrDelay
Total independent variables: 9 (Including number dropped: 1)
Number of valid observations: 582628
Number of missing observations: 17372

Coefficients: (1 not defined because of singularities)
Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.19458 0.20413 -15.650 2.22e-16 ***
CRSDepTime 0.97862 0.01126 86.948 2.22e-16 ***
DayOfWeek=Monday 2.08100 0.18602 11.187 2.22e-16 ***
DayOfWeek=Tuesday 1.34015 0.19881 6.741 1.58e-11 ***
DayOfWeek=Wednesday 0.15155 0.19679 0.770 0.441
DayOfWeek=Thursday -1.32301 0.19518 -6.778 1.22e-11 ***
DayOfWeek=Friday 4.80042 0.19452 24.679 2.22e-16 ***
DayOfWeek=Saturday 2.18965 0.19229 11.387 2.22e-16 ***
DayOfWeek=Sunday Dropped Dropped Dropped Dropped
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 40.39 on 582620 degrees of freedom
Multiple R-squared: 0.01465
Adjusted R-squared: 0.01464
F-statistic: 1238 on 7 and 582620 DF, p-value: < 2.2e-16
Condition number: 10.6542

```

The rxQuantile Function and the Five-Number Summary

Sorting data is, in the general case, a prerequisite to finding exact quantiles, including medians. However, it is possible to compute approximate quantiles by counting binned data then computing a linear interpolation of the empirical cdf. If the data are integers, or can be converted to integers by exact multiplication, and integral bins are used, the computation is exact. The RevoScaleR function rxQuantile does this computation, and by default returns an approximate five-number summary:

```

# The rxQuantile Function and the Five-Number Summary

readPath <- rxGetOption("sampleDataDir")
AirlinePath <- file.path(readPath, "AirlineDemoSmall.xdf")
rxQuantile("ArrDelay", AirlinePath)
Rows Processed: 600000
0% 25% 50% 75% 100%
-86   -9    0   16 1490

```

See Also

[Machine Learning Server Install Machine Learning Server on Windows](#)
[Install Machine Learning Server on Linux](#)
[Install Machine Learning Server on Hadoop](#)

How to split datasets for model training and testing

7/12/2022 • 7 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

This article explains how to split a dataset in two for training and testing a model, but the same technique applies for any use case where subdividing data is required. For example, to manually partition data across multiple nodes in a cluster, you could use the functions and workflow described in this article for that purpose.

Functions for splitting datasets

R developers, use [rxSplit \(RevoScaleR\)](#). The following example is from [Run R code in R Client and Machine Learning Server](#) quickstart.

```
#Randomly split data (80% for training, 20% for testing).
rxSplit(inData = outFileFinal,
        outFilesBase = paste0(td, "/modelData"),
        outFileSuffixes = c("Train", "Test"),
        splitByFactor = "splitVar",
        overwrite = TRUE,
        transforms = list(
            splitVar = factor(sample(c("Train", "Test"),
                                     size = .rxNumRows,
                                     replace = TRUE,
                                     prob = c(.80, .20)),
                           levels = c("Train", "Test"))),
        rngSeed = 17,
        consoleOutput = TRUE)
```

Python developers, we recommend [sci-kit](#) methods for splitting data, as demonstrated in the [Example of binary classification with the microsoftml Python package](#) quickstart.

```
from sklearn.model_selection import train_test_split

bc_train, bc_test = train_test_split(bc_df, test_size=0.2)

print("# of rows in training set = ",bc_train.size)
print("# of rows in test set = ",bc_test.size)
```

Create a distributed dataset on HDFS with rxSplit

In HDFS, the data is distributed automatically, typically to a subset of the nodes, and the computations are also distributed to the nodes containing the required data. To work with data on HDFS, we recommend *composite* .xdf files, which are specialized files designed to be managed by HDFS. For more information, see [Import HDFS > Write a composite XDF](#).

For some computations, such as those involving distributed prediction, it is most efficient to perform the computations on a distributed data set, one in which each node sees only the data it is supposed to work on. You

can split an .xdf file into portions suitable for distribution using the function *rxSplit*. For example, to split the large airline data into five files for distribution on a five node cluster, you could use *rxSplit* as follows:

```
rxOptions(computeContext="local")
bigAirlineData <- "C:/data/AirlineData87to08.xdf"
rxSplit(bigAirlineData, numOutFiles=5)
```

By default, *rxSplit* simply appends a number in the sequence from 1 to *numOutFiles* to the base file name to create the new file names, and in this case the resulting file names, for example, "AirlineData87to081.xdf", are a bit confusing. You can exercise greater control over the output file names by using the *outFilesBase* and *outFilesSuffixes* arguments. With *outFilesBase*, you can specify either a single character string to be used for all files or a character vector the same length as the desired number of files. The latter option is useful, for example, if you would like to create four files with the same file name, but different paths:

```
nodepaths <- paste("compute", 10:13, sep="")
basenames <- file.path("C:", nodepaths, "DistAirlineData")
rxSplit(bigAirlineData, outFilesBase=basenames)
```

This creates the four directories C:/compute10, etc., and creates a file named "DistAirlineData.xdf" in each directory. You will want to do something like this when using distributed data with the standard **RevoScaleR** analysis functions such as *rxLinMod* and *rxLogit*.

You can supply the *outFilesSuffixes* arguments to exercise greater control over what is appended to the end of each file. Returning to our first example, we can add a hyphen between our base file name and the sequence 1 to 5 using *outFilesSuffixes* as follows:

```
rxSplit(bigAirlineData, outFilesSuffixes=paste("-", 1:5, sep=""))
```

The *splitBy* argument specifies whether to split your data file row-by-row or block-by-block. The default is *splitBy="rows"*; to split by blocks instead, specify *splitBy="blocks"*. The *splitBy* argument is ignored if you also specify the *splitByFactor* argument as a character string representing a valid factor variable. In this case, one file is created per level of the factor variable.

The *rxSplit* function works in the local compute context only; once you've split the file you need to distribute the resulting files to the individual nodes using the techniques of the previous sections. You should then specify a compute context with the flag *dataDistType* set to *'split'*. Once you have done this, HPA functions such as *rxLinMod* will know to split their computations according to the data on each node.

Data Analysis with Split Data on HDFS

To use split data in your distributed data analysis, the first step is generally to split the data using *rxSplit*, which as we have seen is a local operation. So the next step is then to copy the split data to your cluster nodes.

Create an XDF source object:

```
hdfsFS <- RxHdfsFileSystem()
bigAirDS <- RxXdfData(airDataDir, fileSystem = hdfsFS )
```

Connect to the cluster:

```
myCluster <- rxSparkConnect(nameNode = "my-name-service-server", port = 8020, wait = TRUE)
```

Set the compute context to your cluster:

```
rxSetComputeContext(myCluster)
```

We are now ready to fit a simple linear model:

```
AirlineLmDist <- rxLinMod(ArrDelay ~ DayOfWeek,  
  data="bigAirDS",  cube=TRUE, blocksPerRead=30)
```

When we print the object, we see that we obtain the same model as when computed with the full data on all nodes:

```
Call:  
rxLinMod(formula = ArrDelay ~ DayOfWeek, data = "DistAirlineData.xdf",  
  cube = TRUE, blocksPerRead = 30)  
  
Cube Linear Regression Results for: ArrDelay ~ DayOfWeek  
File name: C:\data\distributed\DistAirlineData.xdf  
Dependent variable(s): ArrDelay  
Total independent variables: 7  
Number of valid observations: 120947440  
Number of missing observations: 2587529  
  
Coefficients:  
  ArrDelay  
DayOfWeek=Monday    6.669515  
DayOfWeek=Tuesday   5.960421  
DayOfWeek=Wednesday 7.091502  
DayOfWeek=Thursday  8.945047  
DayOfWeek=Friday    9.606953  
DayOfWeek=Saturday  4.187419  
DayOfWeek=Sunday    6.525040
```

With data in the .xdf format, you have your choice of using the full data set or a split data set on each node. For other data sources, you must have the data split across the nodes. For example, the airline data's original form is a set of .csv files, one for each year from 1987 to 2008. (Additional years are now available, but have not been included in our big airline data.) If we copy the year 2000 data to compute10, the year 2001 data to compute11, the year 2002 data to compute12, and the year 2003 data to compute13 with the file name SplitAirline.csv, we can analyze the data as follows:

```
textDS <- RxTextData( file = "C:/data/distributed/SplitAirline.csv",  
  varsToKeep = c( "ArrDelay", "CRSDepTime", "DayOfWeek" ),  
  colInfo = list(ArrDelay = list( type = "integer" ),  
    CRSDepTime = list( type = "integer" ),  
    DayOfWeek = list( type = "integer" ) )  
  )  
rxSummary( ArrDelay ~ F( DayOfWeek, low = 1, high = 7 ), textDS )
```

We can then perform an rxLogit model to classify flights as "Late" as follows:

```

computeLate <- function( dataList )
{
  dataList$Late <- dataList$ArrDelay>15
  return( dataList )
}

rxLogitFitSplitCsv <- rxLogit( Late ~ CRSDepTime + F( DayOfWeek, low = 1,
  high = 7 ), data = textDS,
  transformVars = c( "ArrDelay" ),
  transformFunc=computeLate, verbose=1 )

```

Distributed Predictions on HDFS

You can predict (or score) from a fitted model in a distributed context, but in this case, your data *must* be split.

For example, if we fit our distributed linear model with *covCoef=TRUE* (and *cube=False*), we can compute standard errors for the predicted values:

```

AirlineLmDist <- rxLinMod(ArrDelay ~ DayOfWeek,
  data="DistAirlineData.xdf", covCoef=TRUE, blocksPerRead=30)
rxPredict(AirlineLmDist, data="DistAirlineData.xdf", outData="errDistAirlineData.xdf",
  computeStdErrors=TRUE, computeResiduals=TRUE)

```

NOTE

The `blocksPerRead` argument is ignored if script runs locally using R Client.

The output data is also split, in this case holding fitted values, residuals, and standard errors for the predicted values.

Split Training and Test Datasets on HDFS

One common technique for validating models is to break the data to be analyzed into training and test subsamples, then fit the model using the training data and score it by predicting on the test data. Once you have split your original data set onto your cluster nodes, you can split the data on the individual nodes by calling `rxSplit` again within a call to `rxExec`. If you specify the `RNGseed` argument to `rxExec` (see [Parallel Random Number Generation](#)), the split becomes reproducible:

```

rxExec(rxSplit, inData="C:/data/distributed/DistAirlineData.xdf",
  outFilesBase="airlineData",
  outFileSuffixes=c("Test", "Train"),
  splitByFactor="testSplitVar",
  varsToKeep=c("Late", "ArrDelay", "DayOfWeek", "CRSDepTime"),
  overwrite=TRUE,
  transforms=list(testSplitVar = factor( sample(c("Test", "Train"),
    size=.rxNumRows, replace=TRUE, prob=c(.10, .9)),
    levels= c("Test", "Train"))), rngSeed=17, consoleOutput=TRUE)

```

The result is two new data files, `airlineData.testSplitVar.Train.xdf` and `airlineData.testSplitVar.Test.xdf`, on each of your nodes. We can fit the model to the training data and predict with the test data as follows:

```

AirlineLmDist <- rxLinMod(ArrDelay ~ DayOfWeek,
  data="airlineData.testSplitVar.Train.xdf", covCoef=TRUE, blocksPerRead=30)
rxPredict(AirlineLmDist, data="airlineData.testSplitVar.Test.xdf",
  computeStdErrors=TRUE, computeResiduals=TRUE)

```

NOTE

The `blocksPerRead` argument is ignored if script runs locally using R Client.

Perform Data Operations on Each Node

To create or modify data on each node, use the data manipulation functions within `rxExec`. For example, suppose that after looking at the airline data we decide to create a “cleaner” version of it by keeping only the flights where: there is information on the arrival delay, the flight did not depart more than one hour early, and the actual and scheduled flight time is positive. We can put a call to `rxDataStep` (and any other code we want processed) into a function to be processed on each node via `rxExec`.

```
newAirData <-  function()
{
  airData <- "AirlineData87to08.xdf"
  rxDataStep(inData = airData, outFile = "C:\\\\data\\\\airlineNew.xdf",
             rowSelection = !is.na(ArrDelay) &
               (DepDelay > -60) & (ActualElapsedTime > 0) & (CRSElapsedTime > 0),
             blocksPerRead = 20, overwrite = TRUE)
}
rxExec( newAirData )
```

See also

[Machine Learning Server Install Machine Learning Server on Hadoop](#)

How to merge data using rxMerge in RevoScaleR

7/12/2022 • 7 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Merging allows you to combine the information from two data sets into a third data set that can be used for subsequent analysis. One example is merging account information such as account number and billing address with transaction data such as account number and purchase details to create invoices. In this case, the two files are merged on the common information, that is, the account number.

In RevoScaleR, you merge .xdf files and/or data frames with the rxMerge function. This function supports a number of types of merge that are best illustrated by example. The available types are as follows:

- Inner
- Outer: left, right, and full
- One-to-One
- Union

We describe each of these types in the following sections.

Inner merge

In the default inner merge type, one or more merge key columns is specified, and only those observations for which the specified key columns match exactly are combined to create new observations in the merged data set.

Suppose we have the following data from a dentist's office:

AccountNo	Billee	Patient
0538	Rich C	1
0538	Rich C	2
0538	Rich C	3
0763	Tom D	1
1534	Kath P	1

We can create a data frame with this information:

```
# Merging Data

acct <- c(0538, 0538, 0538, 0763, 1534)
billee <- c("Rich C", "Rich C", "Rich C", "Tom D", "Kath P")
patient <- c(1, 2, 3, 1, 1)
acctDF<- data.frame( acct=acct, billee= billee, patient=patient)
```

Suppose further we have the following information about procedures performed:

```

AccountNo Patient Procedure
0538      3 OffVisit
0538      2 AdultPro
0538      2 OffVisit
0538      3 2SurfCom
0763      1 OffVisit
0763      1 AdultPro
0763      2 OffVisit

```

This data is put into another data frame:

```

acct <- c(0538, 0538, 0538, 0538, 0763, 0763, 0763)
patient <- c(3, 2, 2, 3, 1, 1, 2)
type <- c("OffVisit", "AdultPro", "OffVisit", "2SurfCom", "OffVisit", "AdultPro", "OffVisit")
procedureDF <- data.frame(acct=acct, patient=patient, type=type)

```

Then we use rxMerge to create an inner merge matching on the columns *acct* and *patient*:

```

rxMerge(inData1 = acctDF, inData2 = procedureDF, type = "inner",
matchVars=c("acct", "patient"))

acct billee patient      type
1 538 Rich C          2 AdultPro
2 538 Rich C          2 OffVisit
3 538 Rich C          3 OffVisit
4 538 Rich C          3 2SurfCom
5 763 Tom D           1 OffVisit
6 763 Tom D           1 AdultPro

```

Because the patient 1 in account 538 and patient 1 in account 1534 had no visits, they are omitted from the merged file. Similarly, patient 2 in account 763 had a visit, but does not have any information in the accounts file, so it too is omitted from the merged data set. Also, note that the two input data files are automatically sorted on the merge keys before merging.

Outer merge

There are three types of outer merge: left, right, and full. In a left outer merge, all the lines from the first file are present in the merged file, either matched with lines from the second file that match on the key columns, or if no match, filled out with missing values. A right outer merge is similar, except all the lines from the second file are present, either matched with matching lines from the first file or filled out with missings. A full outer merge includes all lines in both files, either matched or filled out with missings. We can use the same dentist data to illustrate the various types of outer merge:

```
rxMerge(inData1 = acctDF, inData2 = procedureDF, type = "left",
matchVars=c("acct", "patient"))
```

```
acct billee patient      type
1 538 Rich C      1      <NA>
2 538 Rich C      2 AdultPro
3 538 Rich C      2 OffVisit
4 538 Rich C      3 OffVisit
5 538 Rich C      3 2SurfCom
6 763 Tom D       1 OffVisit
7 763 Tom D       1 AdultPro
8 1534 Kath P     1      <NA>
```

```
rxMerge(inData1 = acctDF, inData2 = procedureDF, type = "right",
matchVars=c("acct", "patient"))
```

```
acct billee patient      type
1 538 Rich C      2 AdultPro
2 538 Rich C      2 OffVisit
3 538 Rich C      3 OffVisit
4 538 Rich C      3 2SurfCom
5 763 Tom D       1 OffVisit
6 763 Tom D       1 AdultPro
7 763 <NA>        2 OffVisit
```

```
rxMerge(inData1 = acctDF, inData2 = procedureDF, type = "full",
matchVars=c("acct", "patient"))
```

```
acct billee patient      type
1 538 Rich C      1      <NA>
2 538 Rich C      2 AdultPro
3 538 Rich C      2 OffVisit
4 538 Rich C      3 OffVisit
5 538 Rich C      3 2SurfCom
6 763 Tom D       1 OffVisit
7 763 Tom D       1 AdultPro
8 763 <NA>        2 OffVisit
9 1534 Kath P     1      <NA>
```

One-to-one merge

In the one-to-one merge type, the first observation in the first data set is paired with the first observation in the second data set to create the first observation in the merged data set, the second observation is paired with the second observation to create the second observation in the merged data set, and so on. The data sets must have the same number of rows. It is equivalent to using `*append=***cols*` in a data step.

For example, suppose our first data set contains three observations as follows:

1 a x 2 b y 3 c z

Create a data frame with this data:

```
myData1 <- data.frame( x1 = 1:3, y1 = c("a", "b", "c"), z1 = c("x", "y", "z"))
```

Suppose our second data set contains three different variables:

```
101 d u
102 e v
103 f w
```

Create a data frame with this data:

```
myData2 <- data.frame( x2 = 101:103, y2 = c("d", "e", "f"),
                      z2 = c("u", "v", "w"))
```

A one-to-one merge of these two data sets combines the columns from the two data sets into one data set:

```
rxMerge(inData1 = myData1, inData2 = myData2, type = "oneToOne")

x1 y1 z1  x2 y2 z2
1 1 a   x 101 d   u
2 2 b   y 102 e   v
3 3 c   z 103 f   w
```

Union merge

A union merge is simply the concatenation of two files with the same set of variables. It is equivalent to using *append="rows"* in a data step.

Using the example from one-to-one merge, we rename the variables in the second data frame to be the same as in the first:

```
names(myData2) \<- c("x1", "x2", "x3")
```

Then use a union merge:

```
rxMerge(inData1 = myData1, inData2 = myData2, type = "union")

x1 y1 z1
1 1 a x
2 2 b y
3 3 c z
4 101 d u
5 102 e v
6 103 f w
```

Using rxMerge with .xdf files

You can use **rxMerge** with a combination of .xdf files or data frames. For example, you specify the two paths for two input .xdf files as the *inData1* and *inData2* arguments, and the path to an output file as the *outFile* argument. As a simple example, we can stack two copies of the claims data using the union merge type as follows:

```
claimsXdf <- file.path(rxGetOption("sampleDataDir"), "claims.xdf")

rxMerge(inData1 = claimsXdf, inData2 = claimsXdf, outFile = "claimsTwice.xdf",
       type = "union")
```

A new .xdf file is created containing twice the number of rows of the original claims file.

You can also merge an .xdf file and data frame into a new .xdf file. For example, suppose that you would like to add a variable on state expenditure on education into each observation in the censusWorkers sample .xdf file. First, take a quick look at the state variable in the .xdf file:

```
censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers.xdf")
rxGetVarInfo(censusWorkers, varsToKeep = "state")
```

```
Var 1: state
 3 factor levels: Connecticut Indiana Washington
```

We can create a data frame with per capita educational expenditures for the same three states. (Note that because R alphabetizes factor levels by default, the factor levels in the data frame will be in the same order as those in the .xdf file).

```
educExp <- data.frame(state=c("Connecticut", "Washington", "Indiana"),
  EducExp = c(1795.57,1170.46,1289.66 ))
```

Now use rxMerge, matching by the variable *state*:

```
rxMerge(inData1 = censusWorkers, inData2 = educExp,
  outFile="censusWorkersEd.xdf", matchVars = "state", overwrite=TRUE)
```

The new .xdf file has an additional variable, EducExp:

```
rxGetVarInfo("censusWorkersEd.xdf")

Var 1: age, Age
  Type: integer, Low/High: (20, 65)
Var 2: incwage, Wage and salary income
  Type: integer, Low/High: (0, 354000)
Var 3: perwt, Type: integer, Low/High: (2, 168)
Var 4: sex, Sex
  2 factor levels: Male Female
Var 5: wkswork1, Weeks worked last year
  Type: integer, Low/High: (21, 52)
Var 6: state
  3 factor levels: Connecticut Indiana Washington
Var 7: EducExp, Type: numeric, Low/High: (1170.4600, 1795.5700)
```

See Also

[Machine Learning Server Install Machine Learning Server on Windows](#)

[Install Machine Learning Server on Linux](#)

[Install Machine Learning Server on Hadoop](#)

How to summarize data using RevoScaleR

7/12/2022 • 16 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Summary statistics can help you understand the characteristics and shape of an unfamiliar data set. In RevoScaleR, you can use the `rxGetVarInfo` function to learn more about variables in the data set, and `rxSummary` for statistical measures. The `rxSummary` function also provides a count of observations, and the number of missing values (if any).

This article teaches by example, using built-in sample data sets so that you can practice each skill. It covers the following tasks:

- List variable metadata including name, description, type, labels, Low/High values
- Compute summary statistics: mean, standard deviation, minimum, maximum values
- Compute summary statistics for factor variables
- Create temporary factor variables used in computations
- Compute summary statistics on a row subset
- Execute in-flight data transformations
- Compute and plot a Lorenz curve
- Tips for wide data sets

List variable information

The `rxGetVarInfo` function returns information about the variables in a data frame or .xdf file, including variable names, descriptions, type, and high and low values. The following examples, based on the built-in sample Census data set, demonstrate function usage.

```
# Load data, store variable metadata as censusWorkerInfo, and return variable names.

readPath <- rxGetOption("sampleDataDir")
censusWorkers <- file.path(readPath, "CensusWorkers.xdf")
censusWorkerInfo <- rxGetVarInfo(censusWorkers)
names(censusWorkerInfo)
```

The `names` function is a base R function, which is called on the object to return the following variable names:

```
[1] "age" "incwage" "perwt" "sex" "wkswork1" "state"
```

Once you have a variable name, you can drill down further to examine its metadata. This is the output for `age`:

```
names(censusWorkerInfo$age)

[1] "description" "varType" "storage" "low" "high"
```

Numeric variables include Low/High values. The Low/High values do not necessarily indicate the minimum and

maximum of a numeric or integer variable. Rather, they indicate the values RevoScaleR uses to establish the lowest and highest *factor levels* when treating the variable as a factor. For practical purposes, this is more helpful for data visualization than a true minimum or maximum on the variable. For example, suppose you want to create histograms or hexbin plots, treating the numerical data as categorical, with levels corresponding to the plotting bins. In this scenario, it is often convenient to cut off the highest and lowest data points, and the Low/High values provide this information.

This example demonstrates getting the High value from the age variable:

```
censusWorkerInfo$age$high  
[1] 65
```

Similarly, the Low value is obtained as follows:

```
censusWorkerInfo$age$low  
[1] 20
```

Suppose you are interested in workers between the ages of 30 and 50. You could create a copy of the data file in the working directory, set the High/Low fields as follows and then treat age as a factor in subsequent analysis:

```
outputDir <- rxGetOption("outDataPath")  
tempCensusWorkers <- file.path(outputDir, "tempCensusWorkers.xdf")  
file.copy(from = censusWorkers, to = tempCensusWorkers)  
censusWorkerInfo$age$low <- 35  
censusWorkerInfo$age$high <- 50  
rxSetVarInfo(censusWorkerInfo, tempCensusWorkers)  
rxSummary(~F(age), data = tempCensusWorkers)
```

Results (restricted to the 35 to 50 age range):

```

Call:
rxSummary(formula = ~F(age), data = tempCensusWorkers)

Summary Statistics Results for: ~F(age)
File name:
C:\YourOutDir\tempCensusWorkers.xdf
Number of valid observations: 351121

Category Counts for F_age
Number of categories: 16
Number of valid observations: 158309
Number of missing observations: 192812

F_age Counts
35      9743
36      9888
37      9860
38     10211
39     10378
40     10756
41     10503
42     10511
43     10296
44     10122
45     10074
46      9703
47      9527
48      9093
49      8776
50      8868

```

To reset the low and high values, repeat the previous steps with the original values:

```

censusWorkerInfo$age$low <- 20
censusWorkerInfo$age$high <- 65
rxSetVarInfo(censusWorkerInfo, "tempCensusWorkers.xdf")

```

Compute summary statistics

The `rxSummary` function provides descriptive statistics using a *formula* argument similar to that used in R's modeling functions. The formula specifies the independent variables to summarize.

The basic structure of a formula is a `tilde` symbol "`~`" with one or more independent or right-hand variables, separated by "`+`". To include all of the variables, you can append a dot `(.)` to the tilde as "`~.`".

The `rxSummary` function also takes a data object as the source of the variables.

For example, returning to the `CensusWorkers` sample, run the following script to obtain a data summary of that file:

```

# Formulas in rxSummary

readPath <- rxGetOption("sampleDataDir")
censusWorkers <- file.path(readPath, "CensusWorkers.xdf")
rxSummary(~ age + incwage + perwt + sex + wkswork1, data = censusWorkers)

```

For each term in the formula, the mean, standard deviation, minimum, maximum, and number of valid observations is shown. If `rxSummary` includes `byTerm=FALSE`, the observations (rows) containing missing values for any of the specified variables are omitted in calculating the summary. Cell counts for categorical variables are included.

Results

```
Call:
rxSummary(formula = ~age + incwage + perwt + sex + wkswork1,
  data = censusWorkers)

Summary Statistics Results for: ~age + incwage + perwt + sex + wkswork1
Data: censusWorkers (RxXdfData Data Source)
File name: C:/Program Files/Microsoft/ML Server/R_SERVER/library/RevoScaleR/SampleData/CensusWorkers.xdf
Number of valid observations: 351121

      Name      Mean     StdDev     Min Max  ValidObs MissingObs
age        40.42814   11.385017  20    65 351121      0
incwage  35333.83894 40444.544084   0 354000 351121      0
perwt      20.34423    9.633100   2   168 351121      0
wkswork1   48.62566    6.953843  21    52 351121      0

Category Counts for sex
Number of categories: 2
Number of valid observations: 351121
Number of missing observations: 0

  sex   Counts
Male 189344
Female 161777
```

Compute median values

You might have noticed that rxSummary does not compute a median value for each variable. This is because rxSummary only produces statistics that can be computed in chunks, and a median computation is not a chunkable calculation. Median calculations are predicated on a sort operation, which is expensive on large data sets, and thus excluded from rxSummary.

Assuming your data set fits in memory, you could get the median value of a given variable using the base R [median](#) function:

```
# Load data into a dataframe
mydataframe <- rxImport(censusWorkers)

# As verification, print values of a numeric variable, such as age
mydataframe$age

# Compute the median age
median(mydataframe$age)
```

Results

```
[1] 40
```

For larger disk-bound datasets, you should use visualization techniques to uncover skewness in the underlying data.

Compute summary statistics on factor variables

You can obtain summary statistics on numeric data that are specific to levels within a variable, such as days of the week, months in a year, age or income levels constructed from column values, and so forth.

To do this, specify an interaction between a numeric variable and a factor variable. For example, using the

sample data set AirlineDemoSmall.xdf, use the following command to request a summary of arrival delay by day of week:

```
rxSummary(~ ArrDelay:DayOfWeek, data = file.path(readPath, "AirlineDemoSmall.xdf"))
```

Results

The request produces summary statistics for a factor variable, with output for each level.

```
Call:  
rxSummary(formula = ~ArrDelay:DayOfWeek, data = file.path(readPath,  
"AirlineDemoSmall.xdf"))  
  
Summary Statistics Results for: ~ArrDelay:DayOfWeek  
Data: file.path(readPath, "AirlineDemoSmall.xdf") (RxXdfData Data Source)  
File name: C:/Program Files/Microsoft/ML Server/R_SERVER/library/RevoScaleR/SampleData/AirlineDemoSmall.xdf  
Number of valid observations: 6e+05  
  
Name          Mean      StdDev   Min Max  ValidObs MissingObs  
ArrDelay:DayOfWeek 11.31794 40.68854 -86 1490 582628    17372  
  
Statistics by category (7 categories):  
  
Category          DayOfWeek   Means      StdDev   Min Max  ValidObs  
ArrDelay for DayOfWeek=Monday     Monday  12.025604 40.02463 -76 1017 95298  
ArrDelay for DayOfWeek=Tuesday   Tuesday 11.293808 43.66269 -70 1143 74011  
ArrDelay for DayOfWeek=Wednesday Wednesday 10.156539 39.58803 -81 1166 76786  
ArrDelay for DayOfWeek=Thursday Thursday  8.658007 36.74724 -58 1053 79145  
ArrDelay for DayOfWeek=Friday   Friday  14.804335 41.79260 -78 1490 80142  
ArrDelay for DayOfWeek=Saturday Saturday 11.875326 45.24540 -73 1370 83851  
ArrDelay for DayOfWeek=Sunday   Sunday  10.331806 37.33348 -86 1202 93395
```

NOTE

Interactions provide the one exception to the “responseless” formula mentioned preceding. If you want to obtain the interaction of a continuous variable with one or more factors, you can use a formula of the form `y ~ x:z`, where y is the continuous variable and x and z are factors. This has precisely the same effect as specifying the formula as `~y:x:z`, but is more suggestive of the result: specifically, summary statistics for y at every combination of levels of x and z.

Create a temporary factor variable

You can force RevoScaleR to treat a variable as a factor (with a level for each integer value from the low to high value) by wrapping it with the function call syntax `F()`. Returning to the CensusWorkers.xdf file, in the following example a factor level is temporarily created for each age from 20 through 65:

```
rxSummary(~ incwage:F(age), data = censusWorkers)
```

Results

The results provide not only summary statistics for wage income in the overall data set, but summary statistics on wage income for each age:

```

Call:
rxSummary(formula = ~incwage:F(age), data = censusWorkers)

Summary Statistics Results for: ~incwage:F(age)
Data: censusWorkers (RxXdfData Data Source)
File name: C:/Program Files/Microsoft/ML Server/R_SERVER/library/RevoScaleR/SampleData/CensusWorkers.xdf
Number of valid observations: 351121

Name      Mean     StdDev   Min Max  ValidObs MissingObs
incwage:F(age) 35333.84 40444.54 0  354000 351121  0

Statistics by category (46 categories):

Category          F_age Means     StdDev   Min Max  ValidObs
incwage for F(age)=20 20  12669.94 12396.99 0  336000 6500
incwage for F(age)=21 21  14114.23 12107.81 0  336000 6479
incwage for F(age)=22 22  15982.00 12374.14 0  336000 6676
incwage for F(age)=23 23  18503.92 15093.53 0  336000 6884
incwage for F(age)=24 24  20672.06 14315.67 0  354000 6931
incwage for F(age)=25 25  23856.25 17319.42 0  336000 7273
incwage for F(age)=26 26  25938.17 20707.39 0  354000 7116
incwage for F(age)=27 27  26902.97 20608.09 0  354000 7584
incwage for F(age)=28 28  28531.59 24185.48 0  354000 8184
incwage for F(age)=29 29  30153.10 25715.94 0  354000 8889
incwage for F(age)=30 30  30691.10 26955.27 0  354000 9055
. . .

```

If you include an interaction between two factors, the summary provides cell counts for all combinations of levels of the factors. Since the census data has probability weights, you can use the `pweights` argument to get weighted counts:

```
rxSummary(~ sex:state, pweights = "perwt", data = censusWorkers)
```

Results

```

Call:
rxSummary(formula = ~sex:state, data = censusWorkers, pweights = "perwt")

Summary Statistics Results for: ~sex:state
Data: censusWorkers (RxXdfData Data Source)
File name: C:/Program Files/Microsoft/ML Server/R_SERVER/library/RevoScaleR/SampleData/CensusWorkers.xdf
Probability weights: perwt
Number of valid observations: 351121

Category Counts for sex
Number of categories: 6

  sex    state    Counts
  Male   Connecticut 843736
  Female Connecticut 755843
  Male   Indiana    1517966
  Female Indiana    1289412
  Male   Washington 1504840
  Female Washington 1231489

```

Compute summary statistics on a row subset

The following example shows how to restrict the analysis to specific rows (in this case, people aged 30 to 39). You can use the `low` and `high` arguments of the `F()` function to restrict the creation of on-the-fly factor levels to the same range:

```
rxSummary(~ sex:F(age, low = 30, high = 39), data = censusWorkers,  
pweights="perwt", rowSelection = age >= 30 & age < 40)
```

Results

```
Call:  
rxSummary(formula = ~sex:F(age, low = 30, high = 39), data = censusWorkers,  
pweights = "perwt", rowSelection = age >= 30 & age < 40)  
  
Summary Statistics Results for: ~sex:F(age, low = 30, high = 39)  
Data: censusWorkers (RxXdfData Data Source)  
File name: C:/Program Files/Microsoft/ML Server/R_SERVER/library/RevoScaleR/SampleData/CensusWorkers.xdf  
Probability weights: perwt  
Number of valid observations: 93896  
  
Category Counts for sex  
Number of categories: 20  
  
sex      F_age_30_39_T Counts  
Male    30              103242  
Female  30              84209  
Male    31              100234  
Female  31              77947  
Male    32              96325  
Female  32              75469  
Male    33              97734  
Female  33              77133  
Male    34              103380  
Female  34              81812  
Male    35              110358  
Female  35              89681  
Male    36              113444  
Female  36              91394  
Male    37              110828  
Female  37              91563  
Male    38              113838  
Female  38              95988  
Male    39              115552  
Female  39              97209
```

NOTE

The fourth argument in the function, `exclude`, defaults to TRUE. This is reflected in the T that appears in the output variable name.

Write summary statistics to XDF

By-group statistics are often computed for further analysis or plotting. It can be convenient to store these results in a .xdf file, especially when there are a large number of groups. In this example, compute the mean and standard deviation of wage income and number of weeks work for each year of age for both men and women using the CensusWorkers.xdf file with data from three states:

```
# Writing By-Group Summary Statistics to an .xdf File

readPath <- rxGetOption("sampleDataDir")
censusWorkers <- file.path(readPath, "CensusWorkers.xdf")
rxSummary(~ incwage:F(age):sex + wkswork1:F(age):sex, data = censusWorkers,
  byGroupOutFile = "ByAge.xdf",
  summaryStats = c("Mean", "StdDev", "SumOfWeights"),
  pweights = "perwt", overwrite = TRUE)
```

Results

```
Call:
rxSummary(formula = ~incwage:F(age):sex + wkswork1:F(age):sex,
  data = censusWorkers, byGroupOutFile = "ByAge.xdf", summaryStats = c("Mean",
  "StdDev", "SumOfWeights"), pweights = "perwt", overwrite = TRUE)

Summary Statistics Results for: ~incwage:F(age):sex + wkswork1:F(age):sex
Data: censusWorkers (RxXdfData Data Source)
File name: C:/Program Files/Microsoft/ML Server/R_SERVER/library/RevoScaleR/SampleData/CensusWorkers.xdf
Probability weights: perwt
Number of valid observations: 351121

      Name        Mean       StdDev     SumOfWeights
incwage:F_age:sex 35788.4675 40605.12565 7143286
wkswork1:F_age:sex    48.6373   6.94423 7143286

By-group statistics for incwage:F(age):sex contained in C:\YourDir\ByAge.xdf
By-group statistics for wkswork1:F(age):sex contained in C:\YourDir\ByAge.xdf
```

You can take a quick look at the first five rows in the data set to see that the first variables are the two factor variables determining the groups: F_age and sex. The remaining variables are the computed by-group statistics.

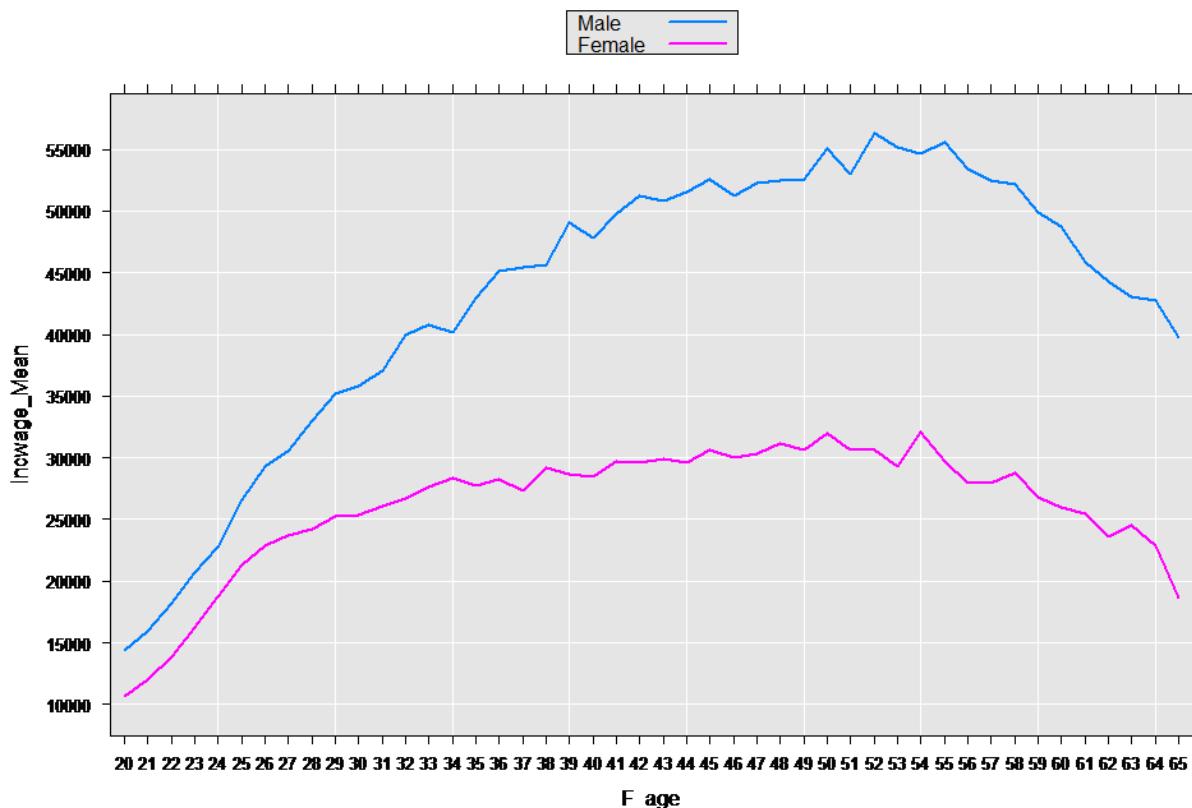
```
rxGetInfo("ByAge.xdf", numRows = 5)
```

Results

```
File name: C:\YourDir\ByAge.xdf
Number of observations: 92
Number of variables: 8
Number of blocks: 1
Compression type: zlib
Data (5 rows starting with row 1):
  F_age sex incwage_Mean incwage_StdDev incwage_SumOfWeights wkswork1_Mean
1    20 Male    14437.68      14118.49        71089    44.29758
2    21 Male    15981.48      13191.73        71150    45.27770
3    22 Male    18258.04      13919.44        75979    46.07166
4    23 Male    20739.91      16511.88        79663    46.75025
5    24 Male    22737.17      15345.41        81412    47.51487
  wkswork1_StdDev wkswork1_SumOfWeights
1          9.755092            71089
2          9.110743            71150
3          8.903617            75979
4          8.451298            79663
5          7.942226            81412
```

You can plot directly from the .xdf file to visualize the results:

```
rxLinePlot(incwage_Mean~F_age, groups = sex, data = "ByAge.xdf")
```



Transform data in-flight

You can use the `transforms` argument to modify your data set before computing a summary. When used in this way, the original data is unmodified and no permanent copy of the modified data is written to disk. The data summaries returned, however, reflect the modified data.

You can also transform data in the formula itself, by specifying simple functions of the original variables. For example, you can get a summary based on the natural logarithm of a variable as follows:

```
# Transform data in rxSummary
rxSummary(~ log(incwage), data = censusWorkers)
```

Results

```
Call:
rxSummary(formula = ~log(incwage), data = censusWorkers)

Summary Statistics Results for: ~log(incwage)
Data: censusWorkers (RxXdfData Data Source)
File name: C:/Program Files/Microsoft/ML Server/R_SERVER/library/RevoScaleR/SampleData/CensusWorkers.xdf
Number of valid observations: 351121

Name      Mean     StdDev    Min      Max      ValidObs MissingObs
log(incwage) 10.19694  0.8387598 1.386294 12.77705  331625    19496
```

Compute and plot Lorenz curves

The Lorenz curves were originally developed to illustrate income inequality. For example, it can show us what percentage of total income is attributed to the lowest earning 10% of the population. The `rxLorenz` function from RevoScaleR provides a "big data" version, using approximate quantiles to quickly compute the cumulative distribution by quantile in a single pass through the data.

The rxLorenz function requires an `orderVarName`, the name of the variable used to compute the quantiles. A separate `valueVarName` can also be specified. This is the name of the variable used to compute the mean values by quantile. By default, the same variable is used for both. We can continue to use the Census Workers data set as an example, computing a Lorenz curve for the distribution of income:

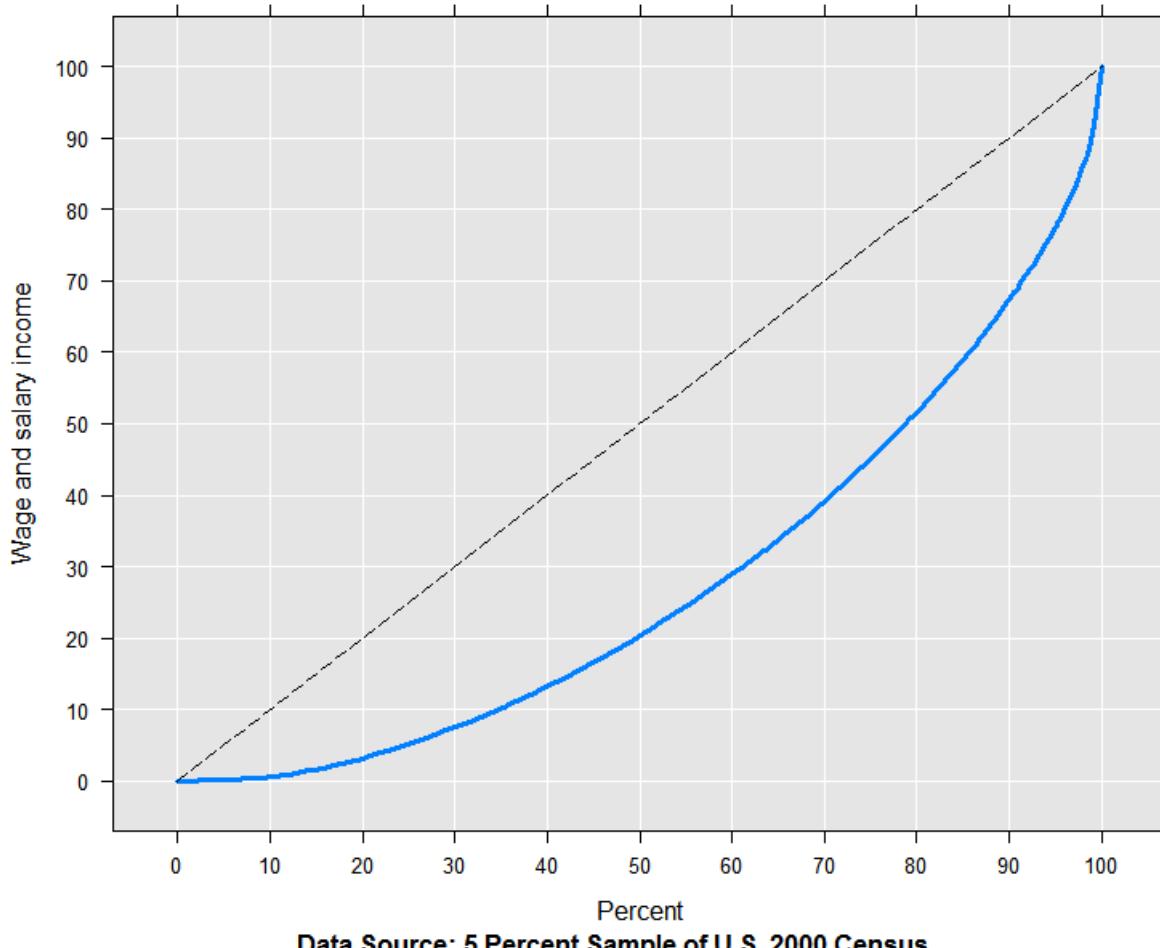
```
lorenzOut <- rxLorenz(orderVarName = "incwage", data = censusWorkers,
  pweights = "perwt")
head(lorenzOut)

cumVals  percents
1 0.0000000  0.000000
2 0.3642878  9.126934
3 0.4005827  9.363072
4 0.5368416  10.181295
5 0.5666976  10.353428
6 0.6241598  10.667766
```

The returned object contains the cumulative values and the percentages. Using the plot method for rxLorenz, we can get a visual representation of the income distribution and compare it with the horizontal line representing perfect equality:

```
plot(lorenzOut)
```

Lorenz Curve for Workers from Three States



Data Source: 5 Percent Sample of U.S. 2000 Census

The Gini coefficient is often used as a summary statistic for Lorenz curves. It is computed by estimating the ratio of the area between the line of equality and the Lorenz curve to the total area under the line of equality (using trapezoidal integration). The Gini coefficient can range from 0 to 1, with 0 representing perfect equality. We can compute it from using the output from rxLorenz:

```
giniCoef <- rxGini(lorenzOut)
giniCoef

[1] 0.4491421
```

Tips for wide data sets

Building a better understanding of the distribution of each variable and the relationships between variables improves decision making during the modeling phase. This is especially true for wide data sets containing hundreds if not thousands of variables. For wide data sets, the main goal of data exploration should be to find any outliers or influential data points, identify redundant variables or correlated variables and transform or combine any variables that seem appropriate.

The functions `rxGetVarInfo` and `rxSummary` provide useful information can help in this effort, but these two functions may need to be used differently when the data contain many variables.

As a first step, import the data to an .xdf so that you can execute functions providing metadata and summary statistics. Recall that `rxGetVarInfo` returns metadata about the data object. After loading data into an XDF data source, the data type information can easily be accessed using the `rxGetVarInfo` function.

Because wide data has so many variables, printed output can be hard to read. As an alternative, save the variable information to an object that can serve as an informal data dictionary. We demonstrate this using the Claims data from the sample data directory:

```
readPath <- rxGetOption("sampleDataDir")
censusWorkers <- file.path(readPath, "CensusWorkers.xdf")
censusDataDictionary <- rxGetVarInfo(censusWorkers)
```

We can then obtain the information for an individual variable as follows:

```
censusDataDictionary$age

Age
Type: integer, Low/High: (20, 65)
```

The `rxSummary` function is a great way to look at the distribution of individual variables and identify outliers. With wide data, you want to store the results of this function into an object. This object contains a data frame with the results of the numeric variables, "sDataFrame", and a list of data frames with the counts for each categorical variable "categorical":

```
readPath <- rxGetOption("sampleDataDir")
censusWorkers <- file.path(readPath, "CensusWorkers.xdf")
censusSummary <- rxSummary(~ age + incwage + perwt + sex + wkwork1,
  data = censusWorkers)
names(censusSummary)

[1] "nobs.valid"      "nobs.missing"     "sDataFrame"       "categorical"
[5] "params"          "formula"         "call"            "categorical.type"
```

Printing the `rxSummary` results to the console wouldn't be useful with so many variables. Saving the results in an object allows us to not only access the summary results programmatically, but also to view the results separately for numeric and categorical variables. We access the `sDataFrame` (printed to show structure) as follows:

```
censusSummary$sDataFrame
  Name      Mean     StdDev Min   Max ValidObs MissingObs
1  age    40.42814  11.385017 20    65   351121       0
2 incwage 35333.83894 40444.544084 0 354000 351121       0
3 perwt    20.34423   9.633100  2    168 351121       0
4 sex        NA        NA NA NA 351121       0
5 wkswork1  48.62566  6.953843 21    52 351121       0
```

To view the categorical variables, we access the categorical component:

```
censusSummary$categorical
[[1]]
sex Counts
1  Male 189344
2 Female 161777
```

Another key piece of data exploration for wide data is looking at the relationships between variables to find variables that are measuring the same information or variables that are correlated. With variables that are measuring the same information, perhaps one stands out as being representative of the group. During data exploration, you must rely on your domain knowledge to group variables into related sets or prioritize variables that are important based on the field or industry. Paring the data set down to related sets allow you to look more closely at redundancy and relatedness within each set.

When looking for correlation between variables the function [rxCrosstabs](#) is useful. In [Crosstabs](#), you see how to use rxCrosstabs and [rxLinePlot](#) to graph the relationship between two variables. Graphs allow for a quick view of the relationship between two variables, which comes in handy when you have many variables to consider. For more information, see [Visualizing Huge Data Sets: An Example from the U.S. Census](#).

See also

- [Tutorial: Data import and exploration using RevoScaleR](#)
- [Tutorial: Transform and subset data using RevoScaleR](#)
- [Sample data for RevoScaleR and revoscalepy](#)
- [Importing text data in Machine Learning Server](#)

Crosstabs using RevoScaleR

7/12/2022 • 20 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Crosstabs, also known as *contingency tables* or *crosstabulations*, are a convenient way to summarize cross-classified categorical data—that is, data that can be tabulated according to multiple levels of two or more factors. If only two factors are involved, the table is sometimes called a *two-way table*. If three factors are involved, the table is sometimes called a *three-way table*.

For large data sets, cross-tabulations of binned numeric data, that is, data that has been converted to a factor where the levels represent ranges of values, can be a fast way to get insight into the relationships among variables. In RevoScaleR, the `rxCube` function is the primary tool to create contingency tables.

For example, the built-in data set `UCBAdmissions` includes information on admissions by gender to various departments at the University of California at Berkeley. We can look at the contingency table as follows:

```
UCBADF <- as.data.frame(UCBAdmissions)
z <- rxCube(Freq ~ Gender:Admit, data = UCBADF)
```

(Because cross-tabulations are explicitly about exploring interactions between variables, multiple predictors must always be specified using the interaction operator ":"; and not the terms operator "+".)

Typing `z` yields the following output:

```
Call:
rxCube(formula = Freq ~ Gender:Admit, data = UCBADF)

Cube Results for: Freq ~ Gender:Admit
Data: UCBADF
Dependent variable(s): Freq
Number of valid observations: 24
Number of missing observations: 0
Statistic: Freq means

  Gender   Admit     Freq Counts
  1  Male Admitted 199.66667      6
  2 Female Admitted  92.83333      6
  3  Male Rejected 248.83333      6
  4 Female Rejected 213.00000      6
```

This data set is widely used in statistics texts because it illustrates Simpson's paradox, which is that in some cases a comparison that holds true in a number of groups is reversed when those groups are aggregated to form a single group. From the preceding table, in which admissions data is aggregated across all departments, it would appear that males are admitted at a higher rate than women. However, if we look at the more granular analysis by department, we find that in four of the six departments, women are admitted at a higher rate than men:

```

z2 <- rxCube(Freq ~ Gender:Admit:Dept, data = UCBADF)
z2

```

This yields the following output:

```

Call:
rxCube(formula = Freq ~ Gender:Admit:Dept, data = UCBADF)

Cube Results for: Freq ~ Gender:Admit:Dept
Data: UCBADF
Dependent variable(s): Freq
Number of valid observations: 24
Number of missing observations: 0
Statistic: Freq means

   Gender Admit Dept Freq Counts
1  Male Admitted A  512     1
2 Female Admitted A   89     1
3  Male Rejected A  313     1
4 Female Rejected A   19     1
5  Male Admitted B  353     1
6 Female Admitted B   17     1
7  Male Rejected B  207     1
8 Female Rejected B    8     1
9  Male Admitted C  120     1
10 Female Admitted C  202     1
11  Male Rejected C  205     1
12 Female Rejected C  391     1
13  Male Admitted D  138     1
14 Female Admitted D  131     1
15  Male Rejected D  279     1
16 Female Rejected D  244     1
17  Male Admitted E   53     1
18 Female Admitted E   94     1
19  Male Rejected E  138     1
20 Female Rejected E  299     1
21  Male Admitted F   22     1
22 Female Admitted F   24     1
23  Male Rejected F  351     1
24 Female Rejected F  317     1

```

Letting the Data Speak Example 1: Analyzing U.S. 2000 Census Data

The CensusWorkers.xdf data set contains a subset of the U.S. 2000 5% Census for individuals aged 20 to 65 who worked at least 20 weeks during the year from three states. Let's examine the relationship between wage income (represented in the data set by the variable *incwage*) and age.

A useful way to observe the relationship between numeric variables is to bin the predictor variable (in our case, age), and then plot the mean of the response for each bin. The simplest way to bin age is to use the F() wrapper within our initial formula; it creates a separate bin for each distinct value of age. (More precisely, it creates a bin of length one from the low value of age to the high value of age—if some ages are missing in the original data set, bins are created for them anyway.)

We create our original model as follows:

```

# Letting the data speak: Example 1

readPath <- rxGetOption("sampleDataDir")
censusWorkers <- file.path(readPath, "CensusWorkers.xdf")
censusWorkersCube <- rxCube(incwage ~ F(age), data=censusWorkers)

```

We first look at the results in tabular form by typing the returned object name, *censusWorkersCube*, which yields the following output:

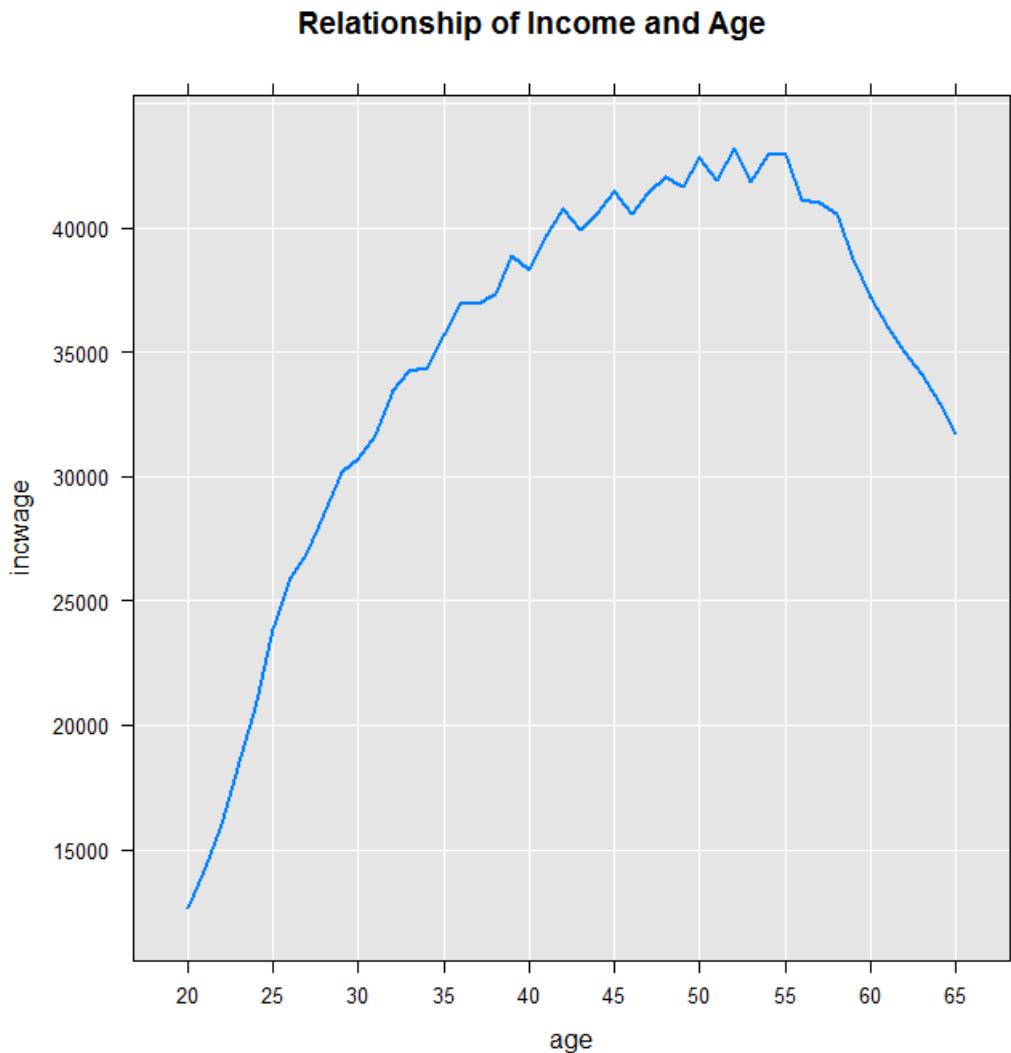
```
Call:  
rxCube(formula = incwage ~ F(age), data = censusWorkers)  
  
Cube Results for: incwage ~ F(age)  
File name:  
C:\Program Files\Microsoft\MRO-for-RRE\8.0\R-3.2.2\library\RevoScaleR\SampleData\CensusWorkers.xdf  
Dependent variable(s): incwage  
Number of valid observations: 351121  
Number of missing observations: 0  
Statistic: incwage means  
  
F_age incwage Counts  
1 20 12669.94 6500  
2 21 14114.23 6479  
3 22 15982.00 6676  
4 23 18503.92 6884  
5 24 20672.06 6931  
6 25 23856.25 7273  
7 26 25938.17 7116  
8 27 26902.97 7584  
9 28 28531.59 8184  
10 29 30153.10 8889  
11 30 30691.10 9055  
12 31 31647.06 8670  
13 32 33459.31 8459  
14 33 34208.33 8574  
15 34 34364.06 9058  
16 35 35739.92 9743  
17 36 36945.24 9888  
18 37 36970.63 9860  
19 38 37331.39 10211  
20 39 38899.67 10378  
21 40 38279.34 10756  
22 41 39678.52 10503  
23 42 40748.10 10511  
24 43 39910.90 10296  
25 44 40524.19 10122  
26 45 41450.27 10074  
27 46 40521.07 9703  
28 47 41371.40 9527  
29 48 42061.04 9093  
30 49 41618.36 8776  
31 50 42789.36 8868  
32 51 41912.11 8506  
33 52 43169.23 8690  
34 53 41864.13 8362  
35 54 42920.45 6275  
36 55 42939.81 6171  
37 56 41157.10 5915  
38 57 40984.69 5881  
39 58 40553.04 5047  
40 59 38738.45 4512  
41 60 37200.02 3775  
42 61 35978.18 3704  
43 62 35000.53 3206  
44 63 34098.00 2563  
45 64 32964.57 2248  
46 65 31698.98 1625
```

As we wanted, the table contains average values of *incwage* for each level of *age*. If we want to create a plot of the results, we can use the *rxResultsDF* function to conveniently convert the output into a data frame. The *F_age* factor variable will automatically be converted back to an integer *age* variable. Then we can plot the data using

rxLinePlot

```
censusWorkersCubeDF <- rxResultsDF(censusWorkersCube)
rxLinePlot(incwage ~ age, data=censusWorkersCubeDF,
           title="Relationship of Income and Age")
```

The resulting plot shows clearly the relationship of income on age:



Transforming Data

Because crosstabs require categorical data for the predictors, you have to do some work to crosstabulate continuous data. In the previous section, we saw that the `F()` wrapper can do a transformation within a formula. The `transforms` argument to `rxCrossTabs` can be used to give you greater control over such transformations.

For example, the `kyphosis` data from the `rpart` package consists of one categorical variable, `Kyphosis`, and three continuous variables `Age`, `Number`, and `Start`. The `Start` variable indicates the topmost vertebra involved in a certain type of spinal surgery, and has a range of 1 to 18. Since there are 7 cervical vertebrae and 12 thoracic vertebrae, we can specify a transform that classifies the start variable as either cervical or thoracic as follows:

```
# Transforming Data

cut(Start, breaks=c(0, 7.5, 19.5), labels=c("cervical", "thoracic"))
```

Similarly, we can create a factorized Age variable as follows (in the original data, age is given in months; with our set of breaks, we cut the data into ranges of years):

```
cut(Age, breaks=c(0, 12, 60, 119, 180, 220), labels=c("<1", "1-4",
"5-9", "10-15", ">15"))
```

We can now crosstabulate the data using the preceding transforms and it is instructive to start by looking at the three two-way tables formed by tabulating Kyphosis with the three predictor variables:

```
library(rpart)
rxCube(~ Kyphosis:Age, data = kyphosis,
       transforms=list(Age = cut(Age, breaks=c(0, 12, 60, 119,
180, 220), labels=c("<1", "1-4", "5-9", "10-15", ">15"))))
Call:
rxCube(formula = ~Kyphosis:Age, data = kyphosis, transforms = list(Age = cut(Age,
breaks = c(0, 12, 60, 119, 180, 220), labels = c("<1", "1-4",
"5-9", "10-15", ">15"))))

Cube Results for: ~Kyphosis:Age
Data: kyphosis
Number of valid observations: 81
Number of missing observations: 0

  Kyphosis   Age Counts
1 absent     <1    13
2 present    <1     0
3 absent    1-4    13
4 present   1-4     4
5 absent    5-9    17
6 present   5-9     6
7 absent   10-15    19
8 present   10-15    7
9 absent    >15     2
10 present   >15    0
rxCube(~ Kyphosis:F(Number), data = kyphosis)
Call:
rxCube(formula = ~Kyphosis:F(Number), data = kyphosis)

Cube Results for: ~Kyphosis:F(Number)
Data: kyphosis
Number of valid observations: 81
Number of missing observations: 0

  Kyphosis F_Number Counts
1 absent        2    12
2 present       2     0
3 absent        3    19
4 present       3     4
5 absent        4    16
6 present       4     2
7 absent        5    12
8 present       5     5
9 absent        6     2
10 present      6     2
11 absent       7     2
12 present      7     3
13 absent       8     0
14 present      8     0
15 absent       9     1
16 present      9     0
17 absent      10     0
18 present      10     1

rxCube(~ Kyphosis:Start, data = kyphosis,
       transforms=list(Start = cut(Start, breaks=c(0, 7.5, 19.5),
labels=c("cervical", "thoracic"))))
Call:
rxCube(formula = ~Kyphosis:Start, data = kyphosis, transforms = list(Start = cut(Start,
breaks = c(0, 7.5, 19.5), labels = c("cervical", "thoracic"))))
```

```
Cube Results for: ~Kyphosis:Start
Data: kyphosis
Number of valid observations: 81
Number of missing observations: 0

  Kyphosis    Start Counts
1 absent cervical     8
2 present cervical    9
3 absent thoracic    56
4 present thoracic   8
```

From these, we see that the probability of the post-operative complication Kyphosis seems to be greater if the *Start* is a cervical vertebra and as more vertebrae are involved in the surgery. Similarly, it appears that the dependence on age is non-linear: it first increases with age, peaks in the range 5-9, and then decreases again.

Cross-Tabulation with rxCrossTabs

The *rxCrossTabs* function is an alternative to the *rxCube* function, which performs the same calculations, but displays its results in format similar to the standard R *xtabs* function. For some purposes, this format can be more informative than the matrix-like display of *rxCube*, and in some situations can be more compact as well.

As an example, consider again the admission data example:

```

# Cross-Tabulation with rxCrossTabs

z3 <- rxCrossTabs(Freq ~ Gender:Admit:Dept, data = UCBADF)
z3
Call:
rxCrossTabs(formula = Freq ~ Gender:Admit:Dept, data = UCBADF)

Cross Tabulation Results for: Freq ~ Gender:Admit:Dept
Data: UCBADF
Dependent variable(s): Freq
Number of valid observations: 24
Number of missing observations: 0
Statistic: sums

Freq, Dept = A (sums):
Admit
Gender Admitted Rejected
Male      512     313
Female    89      19
::::::::::

Freq, Dept = B (sums):
Admit
Gender Admitted Rejected
Male      353     207
Female    17      8
::::::::::

Freq, Dept = C (sums):
Admit
Gender Admitted Rejected
Male      120     205
Female    202     391
::::::::::

Freq, Dept = D (sums):
Admit
Gender Admitted Rejected
Male      138     279
Female    131     244
::::::::::

Freq, Dept = E (sums):
Admit
Gender Admitted Rejected
Male      53      138
Female    94      299
::::::::::

Freq, Dept = F (sums):
Admit
Gender Admitted Rejected
Male      22      351
Female    24      317

```

You can see the row, column, and total percentages by calling the summary function on the rxCrossTabs object:

```

summary(z3)
Call:
rxCrossTabs(formula = Freq ~ Gender:Admit:Dept, data = UCBADF)

```

Cross Tabulation Results for: Freq ~ Gender:Admit:Dept
Data: UCBADF
Dependent variable(s): Freq
Number of valid observations: 24
Number of missing observations: 0
Statistic: sums

Freq, Dept = A (sums):

	Admitted	Rejected	Row Total
Male	512.000000	313.000000	825.00000
Row%	62.060606	37.939394	
Col%	85.191348	94.277108	
Tot%	54.876742	33.547696	88.42444
Female	89.000000	19.000000	108.00000
Row%	82.407407	17.592593	
Col%	14.808652	5.722892	
Tot%	9.539121	2.036442	11.57556
Col Total	601.000000	332.000000	
Grand Total	933.000000		

:::::::::::::::::::::

Freq, Dept = B (sums):

	Admitted	Rejected	Row Total
Male	353.000000	207.000000	560.000000
Row%	63.035714	36.964286	
Col%	95.405405	96.279070	
Tot%	60.341880	35.384615	95.726496
Female	17.000000	8.000000	25.000000
Row%	68.000000	32.000000	
Col%	4.594595	3.720930	
Tot%	2.905983	1.367521	4.273504
Col Total	370.000000	215.000000	
Grand Total	585.000000		

:::::::::::::::::::::

Freq, Dept = C (sums):

	Admitted	Rejected	Row Total
Male	120.00000	205.00000	325.00000
Row%	36.92308	63.07692	
Col%	37.26708	34.39597	
Tot%	13.07190	22.33115	35.40305
Female	202.00000	391.00000	593.00000
Row%	34.06408	65.93592	
Col%	62.73292	65.60403	
Tot%	22.00436	42.59259	64.59695
Col Total	322.00000	596.00000	
Grand Total	918.00000		

:::::::::::::::::::::

Freq, Dept = D (sums):

	Admitted	Rejected	Row Total
Male	138.00000	279.00000	417.00000
Row%	33.09353	66.90647	
Col%	51.30112	53.34608	
Tot%	17.42424	35.22727	52.65152
Female	131.00000	244.00000	375.00000
Row%	34.93333	65.06667	
Col%	48.69888	46.65392	
Tot%	16.54040	30.80808	47.34848
Col Total	269.00000	523.00000	
Grand Total	792.00000		

:::::::::::::::::::::

Freq, Dept = E (sums):

	Admitted	Rejected	Row Total
--	----------	----------	-----------

```

Male      53.000000 138.00000 191.00000
Row%     27.748691 72.25131
Col%     36.054422 31.57895
Tot%     9.075342 23.63014 32.70548
Female    94.000000 299.00000 393.00000
Row%     23.918575 76.08142
Col%     63.945578 68.42105
Tot%     16.095890 51.19863 67.29452
Col Total 147.000000 437.00000
Grand Total 584.000000

```

```
:::::::::::::::::::
```

```

Freq, Dept = F (sums):
      Admitted Rejected Row Total
Male      22.000000 351.00000 373.0000
Row%     5.898123 94.10188
Col%     47.826087 52.54491
Tot%     3.081232 49.15966 52.2409
Female    24.000000 317.00000 341.0000
Row%     7.038123 92.96188
Col%     52.173913 47.45509
Tot%     3.361345 44.39776 47.7591
Col Total 46.000000 668.00000
Grand Total 714.000000

```

You can see, for example, that in Department A, 62 percent of male applicants are admitted, but 82 percent of female applicants are admitted, and in Department B, 63 percent of male applicants are admitted, while 68 percent of female applicants are admitted.

A Large Data Example

The power of *rxCrossTabs* is most evident when you need to tabulate a data set that won't fit into memory. For example, in the large airline data set AirOnTime87to12.xdf, you can obtain the mean arrival delay by carrier and day of week as follows (if you have downloaded the data set, modify the first line as follows to reflect your local path):

The `blocksPerRead` argument is ignored if run locally using R Client. [Learn more...](#)

```

# A Large Data Example

bigDataDir <- "C:/MRS/Data"
bigAirData <- file.path(bigDataDir, "AirOnTime87to12/AirOnTime87to12.xdf")
arrDelayXT <- rxCrossTabs(ArrDelay ~ UniqueCarrier:DayOfWeek,
                           data = bigAirData, blocksPerRead = 30)
print(arrDelayXT)

```

Gives the following output:

```

Call:
rxCrossTabs(formula = ArrDelay ~ UniqueCarrier:DayOfWeek, data = bigAirData,
blocksPerRead = 30)

Cross Tabulation Results for: ArrDelay ~ UniqueCarrier:DayOfWeek
File name: C:\MRS\Data\AirOnTime87to12\AirOnTime87to12.xdf
Dependent variable(s): ArrDelay
Number of valid observations: 145576737
Number of missing observations: 3042918
Statistic: sums

ArrDelay (sums):
  DayOfWeek
UniqueCarrier Mon Tues Wed Thur Fri Sat Sun
  AA 15956852 13367087 16498840 20554660 20714146 9359729 14577582
  US 11797366 11688903 14065606 17379113 19541862 5865427 10264583
  AS 3144578 2677858 3182356 3980209 4415144 2433581 3039129
  CO 8464507 7966834 9537366 11901028 11749616 3553719 6562487
  DL 18146092 15962559 19474389 24077435 24933864 10483280 16414060
  EA 782103 832332 796811 1152825 1405399 638911 670924
  HP 3577460 3170343 3700890 4734543 5015896 2864314 3985741
  NW 7750970 7818040 9256994 11199718 10294116 3726129 6504924
  PA (1) 191137 235924 225260 290844 345238 174284 229677
  PI 1164688 1391526 1456173 1515403 1568266 939820 986642
  PS 88144 111282 122520 133567 173422 44362 88891
  TW 3356944 3459185 4060151 5027427 5267750 1669048 2377671
  UA 15941096 14731587 17801128 21060697 20920843 9174567 13688577
  WN 12438738 8978339 12215989 21556781 26787623 4972506 15973176
  ML (1) 20735 50927 55881 75030 62855 44549 18173
  KH 20744 -26425 -24265 30078 98529 33595 43026
  MQ 7065052 5152746 5893882 7136735 8087443 2947023 5540099
  B6 1886261 1340744 1736450 2373151 2930423 1012698 1969546
  DH 795614 527649 708129 801968 986930 227907 504644
  EV 5733212 3684210 4005374 5262924 5874647 1753361 4418290
  FL 2666677 1694294 1810548 2928247 3068538 819827 2188420
  OO 4717107 3106319 3438056 4725854 5481441 2797745 4764041
  XE 4870453 3904752 4532069 5349375 5315818 1636826 3531446
  TZ 228508 147963 197371 224693 275340 39722 148940
  HA -72468 -92714 -66578 4840 153830 -2082 -22196
  OH 2276399 1567510 1830571 2336032 2702519 922531 1659708
  F9 551932 484426 566122 858027 729273 337695 526887
  YV 1959906 1419073 1463954 1930992 2152270 1270104 1830749
  9E 787776 579608 590038 709161 869358 304151 586378
  VX 10208 37079 12956 42661 73457 2943 39987

```

Using Sparse Cubes

An additional tool that may be useful when using `rxCube` and `rxCrossTabs` with large data is the `useSparseCube` parameter. Compiling cross-tabulations of categorical data can sometimes result in a large number of cells with zero counts, yielding at its core a "sparse matrix". In the usual case, memory is allocated for every cell in the cube, but large cubes may overwhelm memory resources. If we instead allocate space only for cells with positive counts, such operations may often proceed successfully.

As an example, let's look at the airline data again and construct a case where the cross-tabulation yields many zero entries. As the overwhelming number of flights in the data set were not canceled, by appending the `Cancelled` predictor in the formula, we would expect a large number of categorical predictor combinations to have zero observations. Because the `Cancelled` predictor is a logical rather than a factor variable, we need to use the `F()` function to convert it.

```

bigDataDir <- "C:/MRS/Data"
bigAirData <- file.path(bigDataDir, "AirOnTime87to12/AirOnTime87to12.xdf")

arrDelaySparse <- rxCube(ArrDelay ~ UniqueCarrier:DayOfWeek:F(Cancelled),
  data = bigAirData, blocksPerRead = 30, useSparseCube = TRUE)
print(arrDelaySparse)

```

This gives the following output. We get 210 rows with F_Cancelled = 0. By default, if useSparseCube=TRUE, rows with zero counts are removed from the result.

```

Call:
rxCube(formula = ArrDelay ~ UniqueCarrier:DayOfWeek:F(Cancelled),
  data = bigAirData, useSparseCube = TRUE, blocksPerRead = 30)

Cube Results for: ArrDelay ~ UniqueCarrier:DayOfWeek:F(Cancelled)
File name: C:/data/AirOnTime87to12/AirOnTime87to12.xdf
Dependent variable(s): ArrDelay
Number of valid observations: 145576737
Number of missing observations: 3042918
Statistic: ArrDelay means

  UniqueCarrier DayOfWeek F_Cancelled ArrDelay    Counts
1     AA          Mon      0   6.54609466 2437614
2     US          Mon      0   5.20151371 2268064
3     AS          Mon      0   6.37231471 493475
4     CO          Mon      0   6.49381077 1303473
5     DL          Mon      0   6.63422763 2735223
6     EA          Mon      0   6.13500730 127482
7     HP          Mon      0   6.85447467 521916
8     NW          Mon      0   5.09910096 1520066
9     PA (1)      Mon      0   4.24550765 45021
10    PI          Mon      0   9.32556128 124892
11    PS          Mon      0   7.11355016 12391
12    TW          Mon      0   6.21971889 539726
13    UA          Mon      0   7.51524443 2121168
14    WN          Mon      0   4.11051602 3026077
15    ML (1)      Mon      0   2.05724774 10079
... [rows omitted] ...
201   FL          Sun      0   6.91461396 316492
202   OO          Sun      0   6.30954516 755053
203   XE          Sun      0   7.72464706 457166
204   TZ          Sun      0   5.19715263 28658
205   HA          Sun      0   -0.28030915 79184
206   OH          Sun      0   7.12036827 233093
207   F9          Sun      0   5.61844996 93778
208   YV          Sun      0   8.35988986 218992
209   9E          Sun      0   4.18506623 140112
210   VX          Sun      0   5.06036446 7902

```

While this particular example will likely run successfully to completion even on a minimally equipped modern computer without setting the *useSparseCube* flag to *TRUE*, it illustrates how one can quickly start to see the number of zero entries accumulate in an *rxCube* computation. With larger data sets and a larger number of categorical variable combinations, however, this setting may allow computations of cubes that would not otherwise fit in memory.

For the *rxCrossTabs* function, the *useSparseCube* option works exactly the same internally. However, because *rxCrossTabs* always returns a table, it may require more memory to format its result than *rxCube*. If you have an extremely large contingency table, we recommend *rxCube* with *useSparseCube=TRUE* for the greatest chance of completing the computation. The *useSparseCube* flag may also be used with *rxSummary*.

Tests of Independence on Cross-Tabulated Data

One common use of contingency tables is to test whether the tabulated variables are independent. RevoScaleR includes several tests of independence, all of which expect data in the standard R *xtabs* format. You can get data in this format from the *rxCrossTabs* function by using the argument *returnXtabs=TRUE*.

```
# Tests of Independence on Cross-Tabulated Data

bigDataDir <- "C:/MRS/Data"
bigAirData <- file.path(bigDataDir, "AirOnTime87to12/AirOnTime87to12.xdf")
arrDelayXTab <- rxCrossTabs(ArrDel15~ UniqueCarrier:DayOfWeek,
    data = bigAirData, blocksPerRead = 30, returnXtabs=TRUE)
```

The `blocksPerRead` argument is ignored if run locally using R Client. [Learn more...](#)

You can then use this as input to any of the following functions:

- *rxChiSquaredTest*: performs Pearson's chi-squared test of independence.
- *rxFisherTest*: performs Fisher's exact test of independence.
- *rxKendallCor*: performs a Kendall tau test of independence. There are three flavors of test, a, b, and c; by default, the b flavor, which accounts for ties, is used.

(In fact, regular *rxCrossTabs* or *rxCube* output can be used as input to these functions, but they are converted to *xtabs* format first, so it is somewhat more efficient to have *rxCrossTabs* return the *xtabs* format directly.)

Here we use the *arrDelayXTab* data created preceding and perform a Pearson's chi-squared test of independence on it:

```
rxChiSquaredTest(arrDelayXTab)
```

Gives the following output:

```
Chi-squared test of independence between UniqueCarrier and DayOfWeek
X-squared df p-value
105645.8 174      0
```

For large contingency tables such as this one, the chi-squared test is the tool of choice. For smaller tables, particularly those with cells with expected counts fewer than five, Fisher's exact test is useful. On a large table, however, Fisher's exact test may not be an option. For example, if we try it on our airline table, it returns an error:

```
rxFisherTest(arrDelayXTab)

Error in FUN(tbl[, , i], ...) : FEXACT error 40.
Out of workspace.
```

To show the Fisher test, we return to the admissions data from the beginning of the article. This time we use *rxCrossTabs* to return an *xtabs* object:

```
UCBADF <- as.data.frame(UCBAdmissions)
admissCTabs <- rxCrossTabs(Freq ~ Gender:Admit, data = UCBADF,
    returnXtabs=TRUE)
```

We then call *rxFisherTest* on the resulting table:

```

rxFisherTest(admissCTabs)
Fisher's Exact Test for Count Data
estimate 1 95% CI Lower 95% CI Upper      p-value
  1.840856    1.621356    2.091246 4.835903e-22
HA: two.sided
H0: odds ratio = 1

```

The chi-squared test works equally well on this example:

```

rxChiSquaredTest(admissCTabs)
Chi-squared test of independence between Gender and Admit
X-squared df      p-value
  91.6096  1 1.055797e-21

```

In both cases, we are given indisputable evidence of the independence of our two predictor factors. For this example, we could have as easily used the standard R functions *chisq.test* and *fisher.test*. The RevoScaleR enhancements, however, permit *rxChiSquaredTest* and *rxFisherTest* to work on *xtabs* objects with multiple tables. For example, if we expand our examination of the admissions data to include the department info, we obtain a multi-way contingency table:

```

admissCTabs2 <- rxCrossTabs(Freq ~ Gender:Admit:Dept, data = UCBADF,
                           returnXtabs=TRUE)

```

The chi-squared and Fisher's exact test results are shown as follows; notice that they provide a test of independence between Gender and Admit for each level of Dept:

```

rxChiSquaredTest(admissCTabs2)

Chi-squared test of independence between Gender and Admit
X-squared df      p-value
Dept==A 16.37177373  1 5.205468e-05
Dept==B  0.08509801  1 7.705041e-01
Dept==C  0.63322380  1 4.261753e-01
Dept==D  0.22159370  1 6.378283e-01
Dept==E  0.80804765  1 3.686981e-01
Dept==F  0.21824336  1 6.403817e-01

rxFisherTest(admissCTabs2)

Fisher's Exact Test for Count Data
odds ratio 95% CI Lower 95% CI Upper      p-value
Dept==A  0.3495628   0.1970420   0.5920417 1.669189e-05
Dept==B  0.8028124   0.2944986   2.0040231 6.770899e-01
Dept==C  1.1329004   0.8452173   1.5162918 3.866166e-01
Dept==D  0.9213798   0.6789572   1.2504742 5.994965e-01
Dept==E  1.2211852   0.8064776   1.8385155 3.603964e-01
Dept==F  0.8280944   0.4332888   1.5756278 5.458408e-01
HA: two.sided
H0: odds ratio = 1

```

Like Fisher's exact test, the Kendall tau correlation test works best on smaller contingency tables. Here is an example of what it returns when applied to our admissions data (the results differ from run to run as the underlying algorithm relies on sampling):

```

rxKendallCor(admissCTabs)

taub p-value
Dept==A -0.13596550 0.000
Dept==B -0.02082575 0.666
Dept==C 0.02865045 0.380
Dept==D -0.01939676 0.585
Dept==E 0.04140240 0.383
Dept==F -0.02319366 0.530
HA: two.sided

```

Odds Ratios and Risk Ratios

Another common task associated with 2×2 contingency tables is the calculation of odds ratios and risk ratios (also known as relative risk). The two functions *rxOddsRatio* and *rxRiskRatio* in RevoScaleR can be used to compute these quantities. The odds ratio and the risk ratio are closely related: the odds ratio computes the relative odds of an event among two or more groups, while the risk ratio computes the relative probabilities of an event. Consider again the contingency table *admissCTabs*:

```

# Odds Ratios and Risk Ratios

admissCTabs

Admit
Gender Admitted Rejected
Male    1198     1493
Female   557      1278

```

In this example, the odds of being admitted as a male are $1198/1493$, or about 4 to 5 against. The odds of being admitted as a female are $557/1278$, or about 4 to 9 against. The odds ratio is $(1198/1493)/(557/1278)$, or 1.8 greater odds that a male will be admitted as opposed to a woman.

```

rxOddsRatio(admissCTabs)

data:
Z = 0.6104, p-value < 2.2e-16
alternative hypothesis: two.sided
95 percent confidence interval:
1.624377 2.086693
sample estimates:
oddsRatio
1.84108

```

The risk ratio, by contrast, compares the probabilities of being *rejected*, that is, $1493/(1198+1493)$ for a man versus $1278/(557+1278)$ for a woman. So here the risk ratio is 0.697 (the probability of a woman being rejected) divided by 0.555 (the probability of a man being rejected), or 1.255:

```

rxRiskRatio(admissCTabs)

data:
Z.Female = 0.2274, p-value < 2.2e-16
alternative hypothesis: two.sided
95 percent confidence interval:
1.199631 1.313560
sample estimates:
riskRatio.Female
1.255303

```

Visualizing Huge Data Sets: An Example from the U.S. Census

7/12/2022 • 12 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

By combining the power and flexibility of the open-source R language with the fast computations and rapid access to huge datasets provided by RevoScaleR, it is easy and efficient to not only do fine-grained calculations "on the fly" for plotting, but to visually drill down into these patterns.

This example focuses on a basic demographic pattern: in general, more boys than girls are born and the death rate is higher for males at every age. So, typically we observe a decline in the ratio of males to females as age increases.

Important! Since Microsoft R Client can only process datasets that fit into the available memory, chunking is not supported. When run locally with R Client, the `blocksPerRead` argument is ignored and all data must be read into memory. When working with Big Data, this may result in memory exhaustion. You can work around this limitation when you push the compute context to a Machine Learning Server instance.

Examining the Data

We can examine this pattern in the United States using the 5% Public Use Microdata Sample (PUMS) of the 2000 United States Census, stored in an .xdf file of about 12 gigabytes.[1] Using the `rxGetInfo` function, we can get a quick summary of the data set:

```
# Visualizing Huge Data Sets: An Example from the U.S. Census
# Examining the Data

bigDataDir <- "C:/MRS/Data"
bigCensusData <- file.path(bigDataDir,"CensusUS5Pct2000.xdf")
rxGetInfo(bigCensusData)

File name: C:\MRS\Data\CensusUS5Pct2000.xdf
Number of observations: 14058983
Number of variables: 264
Number of blocks: 98
Compression type: zlib
```

It contains over 14 million rows, has 264 variables, and has been stored in 98 data blocks in the .xdf file.

First let's use the `rxCube` function to count the number of males and females for each age, using the weighting variable provided by the census bureau. We'll read in the data in chunks of 15 blocks, or about 2 million rows at a time.

```
ageSex <- rxCube(~F(age):sex, pweights = "perwt", data = bigCensusData,
blocksPerRead = 15)
```

The `blocksPerRead` argument is ignored if run locally using R Client. [Learn more...](#)

In the computation, we're treating age as a categorical variable so we'll get counts for each age. Since we'll be converting this factor data back to integers on a regular basis, we'll write a function to do the conversion:

```
factoi <- function(x)
{
  as.integer(levels(x))[x]
}
```

We can now write a simple function in R to take the counts information returned from rxCube and do the arithmetic to compute the sex ratio for each age.

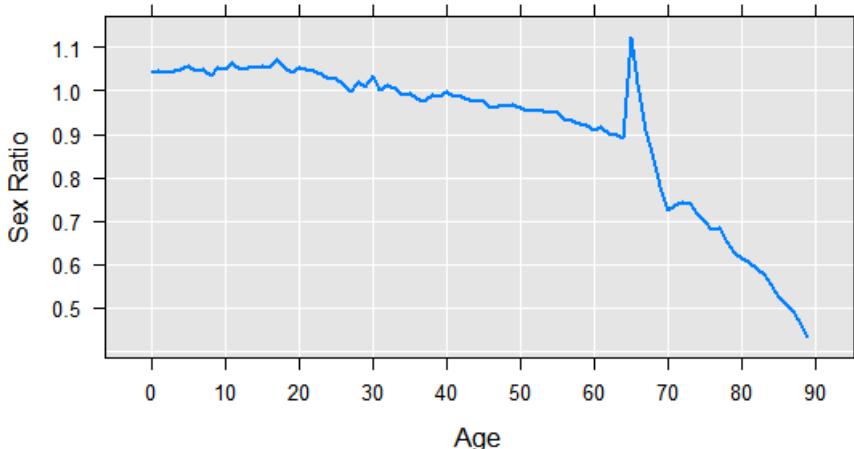
```
getSexRatio <- function(ageSex)
{
  ageSexDF <- as.data.frame(ageSex)
  sexRatioDF <- subset(ageSexDF, sex == 'Male')
  names(sexRatioDF)[names(sexRatioDF) == 'Counts'] <- 'Males'
  sexRatioDF$sex <- NULL
  females <- subset(ageSexDF, sex == 'Female')
  sexRatioDF$Females <- females$Counts
  sexRatioDF$age <- factoi(sexRatioDF$F_age)
  sexRatioDF <- subset(sexRatioDF, Females > 0 & Males > 0 & age <=90)
  sexRatioDF$SexRatio <- sexRatioDF$Males/sexRatioDF$Females
  return(sexRatioDF)
}
```

It returns a data frame with the counts for Males and Females, the SexRatio, and age for all groups in which there are positive counts for both Males and Females. Ages over 90 are also excluded.

Let's use that function and then plot the results:

```
sexRatioDF <- getSexRatio(ageSex)
rxLinePlot(SexRatio~age, data = sexRatioDF,
  xlab = "Age", ylab = "Sex Ratio",
  main = "Figure 1: Sex Ratio by Age, U.S. 2000 5% Census")
```

Figure 1: Sex Ratio by Age, U.S. 2000 5% Census



The graph shows the expected downward trend at the younger ages. But look at what happens at the age of 65! At the age of 65, there are suddenly about 12 men for every 10 women.

We can quickly drill down, and do the same computation for each region:

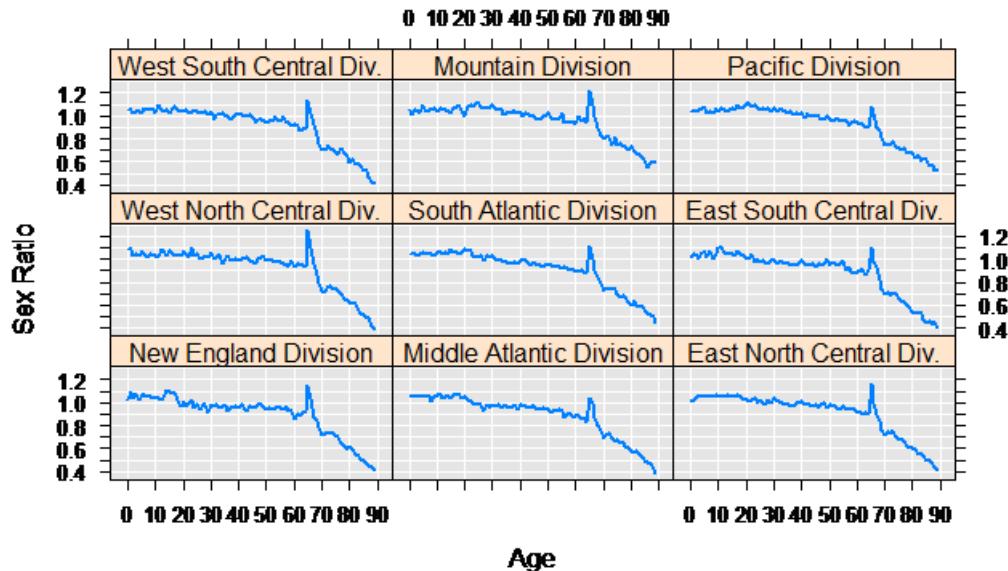
```

ageSex <- rxCube(~F(age):sex:region, pweights = "perwt", data = bigCensusData,
blocksPerRead = 15)
sexRatioDF <- getSexRatio(ageSex)
rxLinePlot(SexRatio~age|region, data = sexRatioDF,
xlab = "Age", ylab = "Sex Ratio",
main = "Figure 2: Sex Ratio by Age and Region, U.S. 2000 5% Census")

```

We see the unlikely "spike" at age 65 in all regions:

Figure 2: Sex Ratio by Age and Region, U.S. 2000 5% Census



Let's try looking at ethnicity, comparing whites with non-whites.

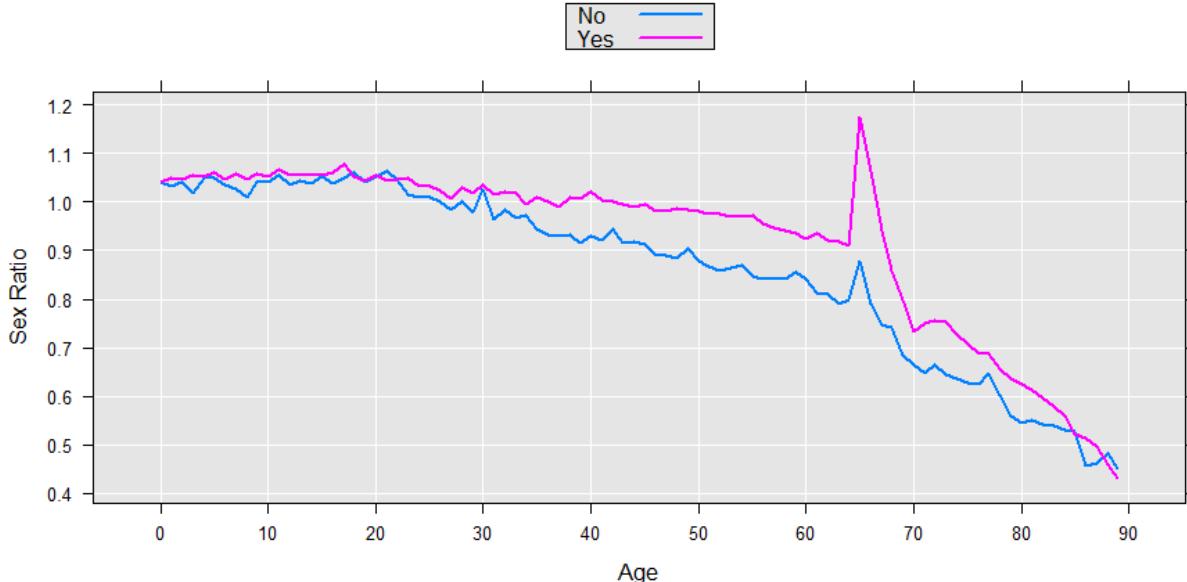
```

ageSex <- rxCube(~F(age):sex:racwht, pweights = "perwt", data = bigCensusData,
blocksPerRead = 15)
sexRatioDF <- getSexRatio(ageSex)
rxLinePlot(SexRatio~age, groups = racwht, data = sexRatioDF,
xlab = "Age", ylab = "Sex Ratio",
main = "Figure 3: Sex Ratio by Age, Conditioned on 'Is White?', U.S. 2000 5% Census")

```

There are interesting differences between the two groups, but again there is the familiar spike at age 65 in both cases.

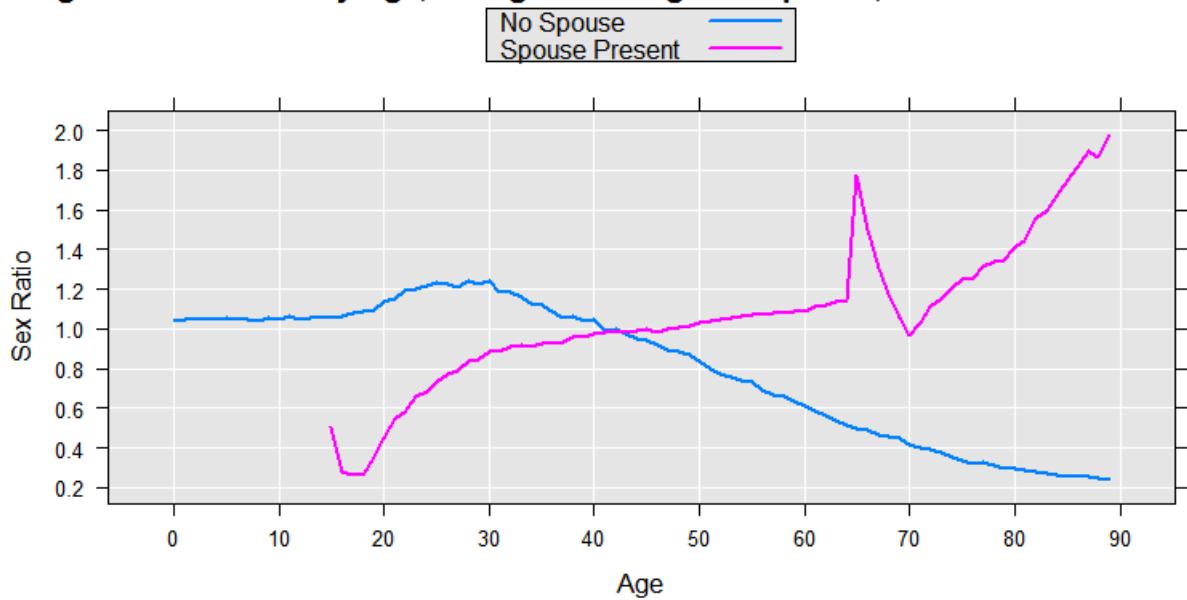
Figure 3: Sex Ratio by Age, Conditioned on 'Is White?', U.S. 2000 5% Census



How about comparing married people with those not living with a spouse? We can create a temporary variable using the transform argument to do this computation:

```
ageSex <- rxCube(~F(age):sex:married, pweights = "perwt", data = bigCensusData,
transforms = list(married = factor(marst == 'Married, spouse present',
levels = c(FALSE, TRUE), labels = c("No Spouse", "Spouse Present"))),
blocksPerRead = 15)
sexRatioDF <- getSexRatio(ageSex)
rxLinePlot(SexRatio~age, groups = married, data = sexRatioDF,
xlab="Age", ylab = "Sex Ratio",
main="Figure 4: Sex Ratio by Age, Living/Not Living with Spouse, U.S. 2000 5% Census")
```

Figure 4: Sex Ratio by Age, Living/Not Living with Spouse, U.S. 2000 5% Census



First, notice that the spike at age 65 is absent for unmarried people. But also look at the very different trends. For married 20 year-olds, there are about 5 men for every 10 women, but for married 60 year-olds, there would be about 11 men for every 10 women. This may at first seem counter-intuitive, but it's consistent with the notion that men tend to marry younger women. Let's explore that next.

Extending the Analysis

We'd like to compare the ages of men with ages of their wives. This is more complicated than the earlier

computations because the spouse's age is stored in a different record in the data file. To handle this, we'll create a new data set using RevoScaleR's data step functionality with a transformation function. This transformation function makes use of RevoScaleR's ability to go back and read additional rows from an .xdf file as it is reading through it in chunks. It also uses internal variables that are provided inside a transformation function:

.rxStartRow, which is the row number from the original data set for the first row of the chunk of data being processed; .rxReadFileName gives the name of the file being read. It first checks the spouse location for the relative position of the spouse in the data set. It then determines which of the observations have spouses in the previous, current, or next chunk of data. Then, in each of the three cases, it looks up the spouse information and adds it to the original observation.

```
# Extending the Analysis

spouseAgeTransform <- function(data)
{
  # Use internal variables
  censusUS2000 <- .rxReadFileName
  startRow <- .rxStartRow

  # Calculate basic information about input data chunk
  numRows <- length(data$sploc)
  endRow <- startRow + numRows - 1

  # Create a new variable. A spouse is present if the spouse locator
  # (relative position of spouse in data) is positive
  data$hasSpouse <- data$sploc > 0

  # Create variables for spouse information
  spouseVars <- c("age", "incwage", "sex")
  data$spouseAge <- rep.int(NA_integer_, numRows)
  data$spouseIncwage <- rep.int(NA_integer_, numRows)
  data$sameSex <- rep.int(NA, numRows)

  # Create temporary row numbers for this block
  rowNum <- seq_len(numRows)
  # Find the temporary row number for the spouse
  spouseRow <- rep.int(NA_integer_, numRows)
  if (any(data$hasSpouse))
  {
    spouseRow[data$hasSpouse] <-
      rowNum[data$hasSpouse] +
      data$sploc[data$hasSpouse] - data$pernum[data$hasSpouse]
  }

  #####
  # Handle possibility that spouse is in previous or next chunk
  # Create a variable indicating if the spouse is in the previous,
  # current, or next chunk
  blockBreaks <- c(-.Machine$integer.max, 0, numRows, .Machine$integer.max)
  blockLabels <- c("previous", "current", "next")
  spouseFlag <- cut(spouseRow, breaks = blockBreaks, labels = blockLabels)
  blockCounts <- tabulate(spouseFlag, nbins = 3)
  names(blockCounts) <- blockLabels

  # At least one spouse in previous chunk
  if (blockCounts[["previous"]] > 0)
  {
    # Go back to the original data set and read the
    # required rows in the previous chunk
    needPreviousRows <- 1 - min(spouseRow, na.rm = TRUE)
    previousData <- rxDataStep(inData = censusUS2000,
      startRow = startRow - needPreviousRows,
      numRows = needPreviousRows, varsToKeep = spouseVars,
      returnTransformObjects = FALSE, reportProgress = 0)

    # Get the spouse locations
    . . . . .
```

```

whichPrevious <- which(spouseFlag == "previous")
spouseRowPrev <- spouseRow[whichPrevious] + needPreviousRows

# Set the spouse information for everyone with a spouse
# in the previous chunk
data$spouseAge[whichPrevious] <- previousData$age[spouseRowPrev]
data$spouseIncwage[whichPrevious] <- previousData$incwage[spouseRowPrev]
data$sameSex[whichPrevious] <-
  data$sex[whichPrevious] == previousData$sex[spouseRowPrev]
}

# At least one spouse in current chunk
if (blockCounts[["current"]] > 0)
{
  # Get the spouse locations
  whichCurrent <- which(spouseFlag == "current")
  spouseRowCurr <- spouseRow[whichCurrent]

  # Set the spouse information for everyone with a spouse
  # in the current chunk
  data$spouseAge[whichCurrent] <- data$age[spouseRowCurr]
  data$spouseIncwage[whichCurrent] <- data$incwage[spouseRowCurr]
  data$sameSex[whichCurrent] <-
    data$sex[whichCurrent] == data$sex[spouseRowCurr]
}

# At least one spouse in next chunk
if (blockCounts[["next"]] > 0)
{
  # Go back to the original data set and read the
  # required rows in the next chunk
  needNextRows <- max(spouseRow, na.rm=TRUE) - numRows
  nextData <- rxDataStep(inData = censusUS2000, startRow = endRow+1,
  numRows = needNextRows, varsToKeep = spouseVars,
  returnTransformObjects = FALSE, reportProgress = 0)

  # Get the spouse locations
  whichNext <- which(spouseFlag == "next")
  spouseRowNext <- spouseRow[whichNext] - numRows

  # Set the spouse information for everyone with a spouse
  # in the next block
  data$spouseAge[whichNext] <- nextData$age[spouseRowNext]
  data$spouseIncwage[whichNext] <- nextData$incwage[spouseRowNext]
  data$sameSex[whichNext] <-
    data$sex[whichNext] == nextData$sex[spouseRowNext]
}

# Now caculate age difference
data$ageDiff <- data$age - data$spouseAge
data
}

```

We can test the transform function by reading in a small number of rows of data. First we will read in a chunk that has a spouse in the previous block and a spouse in the next block, and call the transform function. We will repeat this, expanding the chunk of data to include both spouses, and double check to make sure the results are the same for the equivalent rows:

```

varsToKeep=c("age", "region", "incwage", "racwht", "nchild", "perwt", "sploc",
"pernum", "sex")
testDF <- rxDataStep(inData=bigCensusData, numRows = 6, startRow=9,
varsToKeep = varsToKeep, returnDTransformObjects=FALSE)
.rxStartRow <- 9
.rxReadFileName <- bigCensusData
newTestDF <- as.data.frame(spouseAgeTransform(testDF))
.rxStartRow <- 8
testDF2 <- rxDataStep(inData=bigCensusData, numRows = 8, startRow=8,
varsToKeep = varsToKeep, returnTransformObjects=FALSE)
newTestDF2 <- as.data.frame(spouseAgeTransform(testDF2))
newTestDF[,c("age", "incwage", "sploc", "hasSpouse" , "spouseAge", "ageDiff")]
newTestDF2[,c("age", "incwage", "sploc", "hasSpouse" , "spouseAge", "ageDiff")]

> newTestDF[,c("age", "incwage", "sploc", "hasSpouse" , "spouseAge", "ageDiff")]
age incwage sploc hasSpouse spouseAge ageDiff
1 46 1000 1 TRUE 43 3
2 16 0 0 FALSE NA NA
3 14 NA 0 FALSE NA NA
4 7 NA 0 FALSE NA NA
5 19 1500 0 FALSE NA NA
6 62 42600 2 TRUE 55 7

> newTestDF2[,c("age", "incwage", "sploc", "hasSpouse" , "spouseAge", "ageDiff")]
age incwage sploc hasSpouse spouseAge ageDiff
1 43 150000 2 TRUE 46 -3
2 46 1000 1 TRUE 43 3
3 16 0 0 FALSE NA NA
4 14 NA 0 FALSE NA NA
5 7 NA 0 FALSE NA NA
6 19 1500 0 FALSE NA NA
7 62 42600 2 TRUE 55 7
8 55 0 1 TRUE 62 -7

```

To create the new data set, we'll use the transformation function with `rxDataStep`. Observations for males living with female spouses will be written to a new data .xdf file named `spouseCensus2000.xdf`. It will include information about the age of their spouse.

The `blocksPerRead` argument is ignored if run locally using R Client. [Learn more...](#)

```

spouseCensusXdf <- "spouseCensus2000"
rxDataStep(inData = bigCensusData, outFile=spouseCensusXdf,
varsToKeep=c("age", "region", "incwage", "racwht", "nchild", "perwt"),
transformFunc = spouseAgeTransform,
transformVars = c("age", "incwage", "sploc", "pernum", "sex"),
rowSelection = sex == 'Male' & hasSpouse == 1 & sameSex == FALSE &
age <= 90,
blocksPerRead = 15, overwrite=TRUE)

```

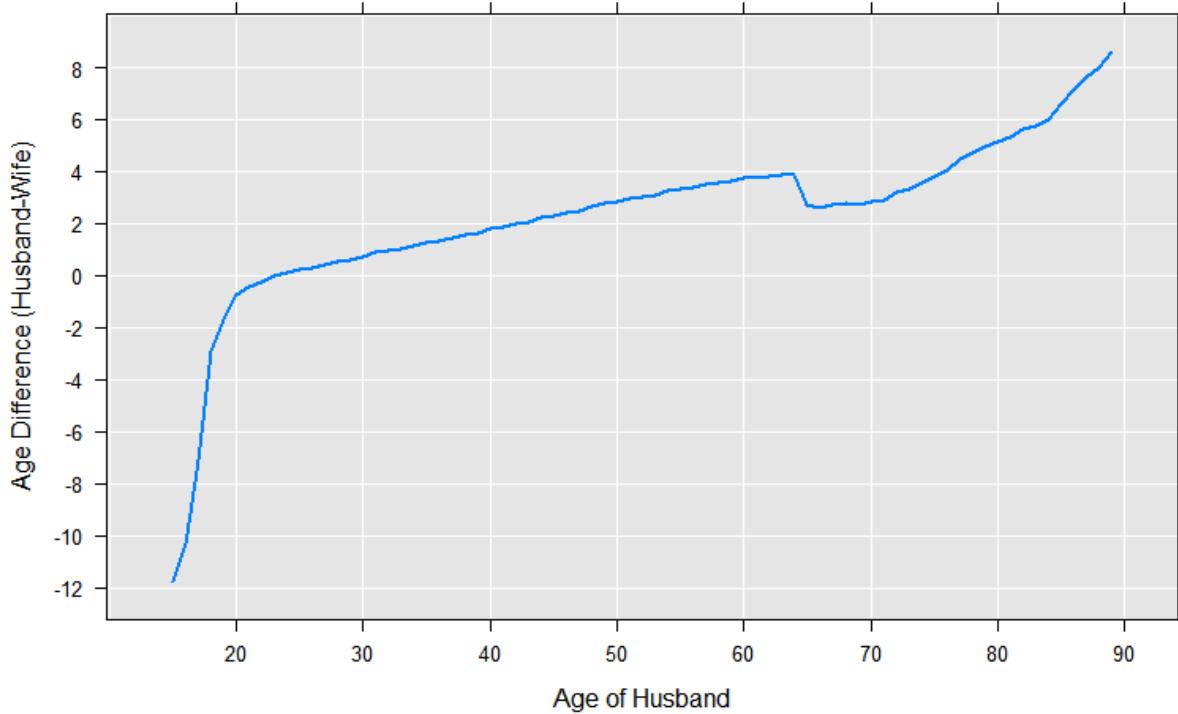
Now for each husband age we can compute the distribution of spouse age. Then, after converting age back to an integer, we can plot the age difference by age of husband:

```

ageDiffData <- rxCube(ageDiff~F(age) , pweights="perwt", data = spouseCensusXdf,
returnDataFrame = TRUE, blocksPerRead = 15)
ageDiffData$ownAge <- factoio(ageDiffData$F_age)
rxLinePlot(ageDiff~ownAge, data = ageDiffData,
xlab="Age of Husband", ylab = "Age Difference (Husband-Wife)",
main="Figure 5: Age Difference of Spouses Living Together, U.S. 2000 5% Census")

```

Figure 5: Age Difference of Spouses Living Together, U.S. 2000 5% Census



Beginning at ages in the early 20's, men tend to be married to younger women. The age difference increases as men get older. But beginning at age 65, our smooth trend stops and we see more erratic behavior, suggesting that our data has misinformation about ages of spouses within households at age 65 and above.

With our new data set we can also calculate the counts for each combination of husband's age and wife's age:

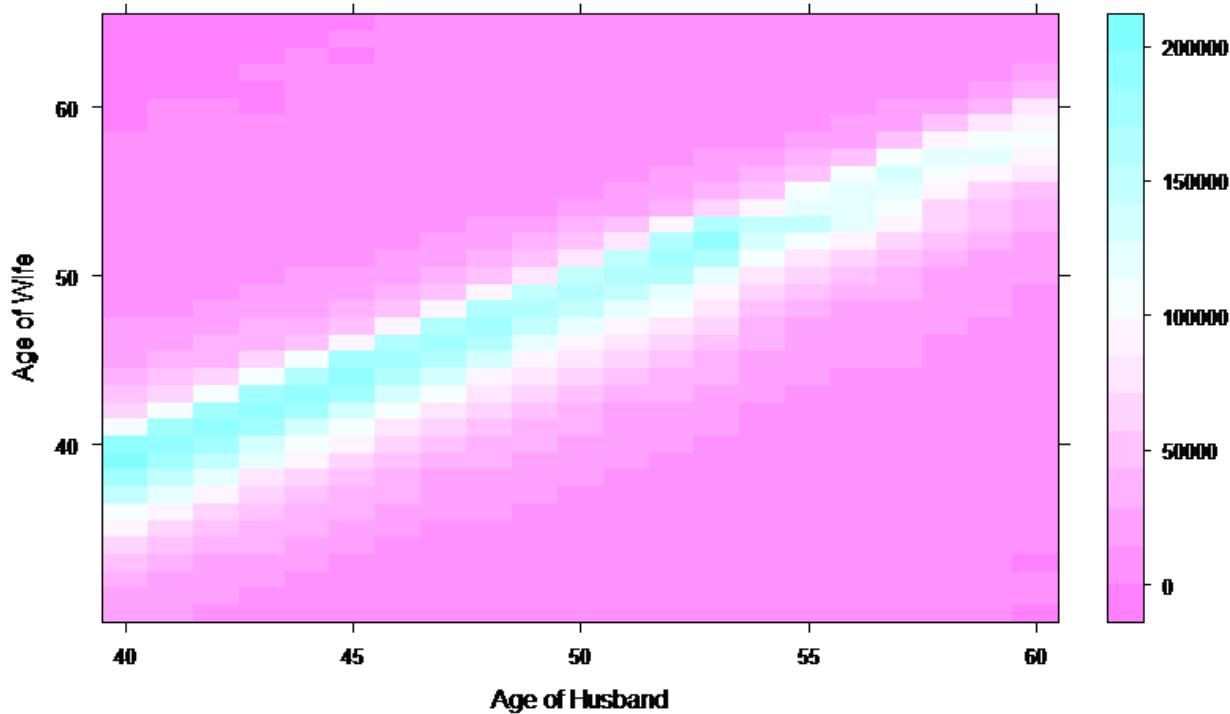
```
aa <- rxCube(~F(age):F(spouseAge), pweights = "perwt", data = spouseCensusXdf,
returnDataFrame = TRUE, blocksPerRead = 7)
```

A level plot is a good way to visualize the results, where the color indicates the count of each category of combination of husband's and wife's age.

```
# Convert factors to integers
aa$age <- factoи(aa$F_age)
aa$spouseAge <- factoи(aa$F_spouseAge)

# Do a level plot showing the counts for husbands aged 40 to 60
ageCompareSubset <- subset(aa, age >= 40 & age <= 60 & spouseAge >= 30 & spouseAge <= 65)
levelplot(Counts~age*spouseAge, data=ageCompareSubset,
xlab="Age of Husband", ylab = "Age of Wife",
main="Figure 6: Counts by Age (40-60) and Spouse Age, U.S. 2000 5% Census")
```

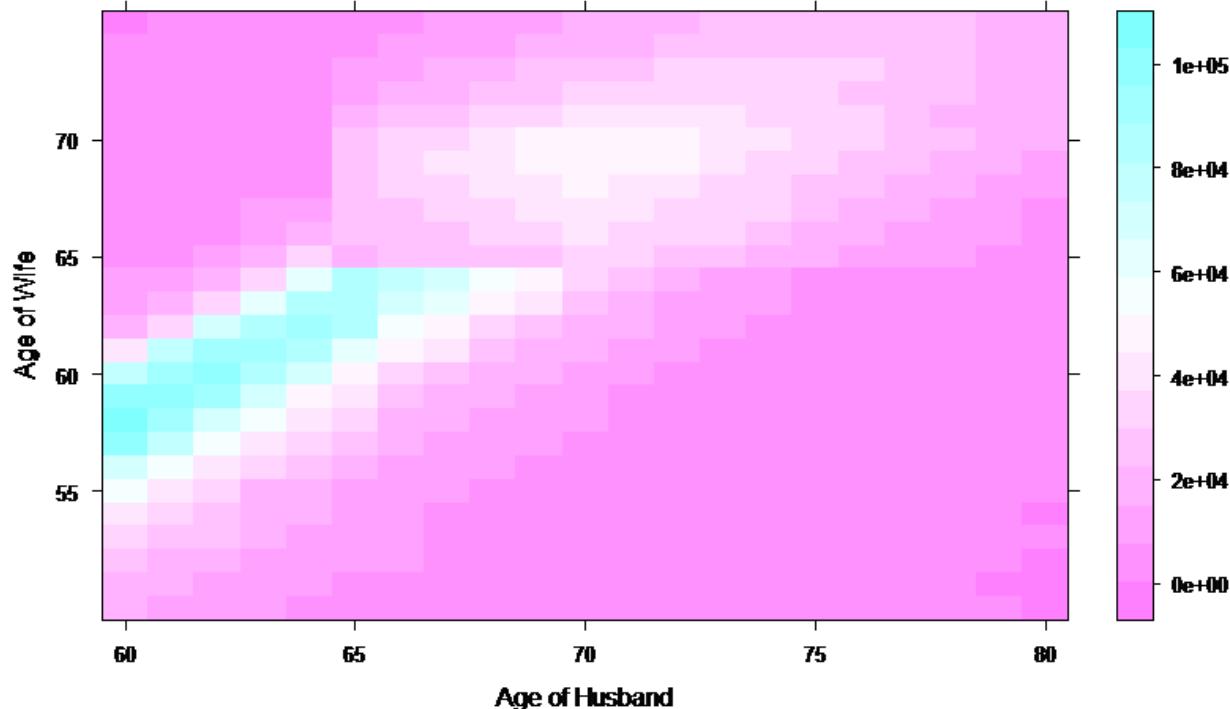
Figure 6: Counts by Age (40-60) and Spouse Age, U.S. 2000 5% Census



In the level plot, there is a very clear pattern with the mode of the relative age of wife dropping gradually as the age of husband increases. Now, repeat with husbands aged 60 to 80.

```
ageCompareSubset <- subset(aa, age >= 60 & age <= 80 & spouseAge >= 50 & spouseAge <= 75)
levelplot(Counts~age*spouseAge, data = ageCompareSubset,
  xlab = "Age of Husband", ylab = "Age of Wife",
  main = "Figure 7: Counts by Age(60-80)and Spouse Age , U.S. 2000 5% Census")
```

Figure 7: Counts by Age(60-80)and Spouse Age , U.S. 2000 5% Census

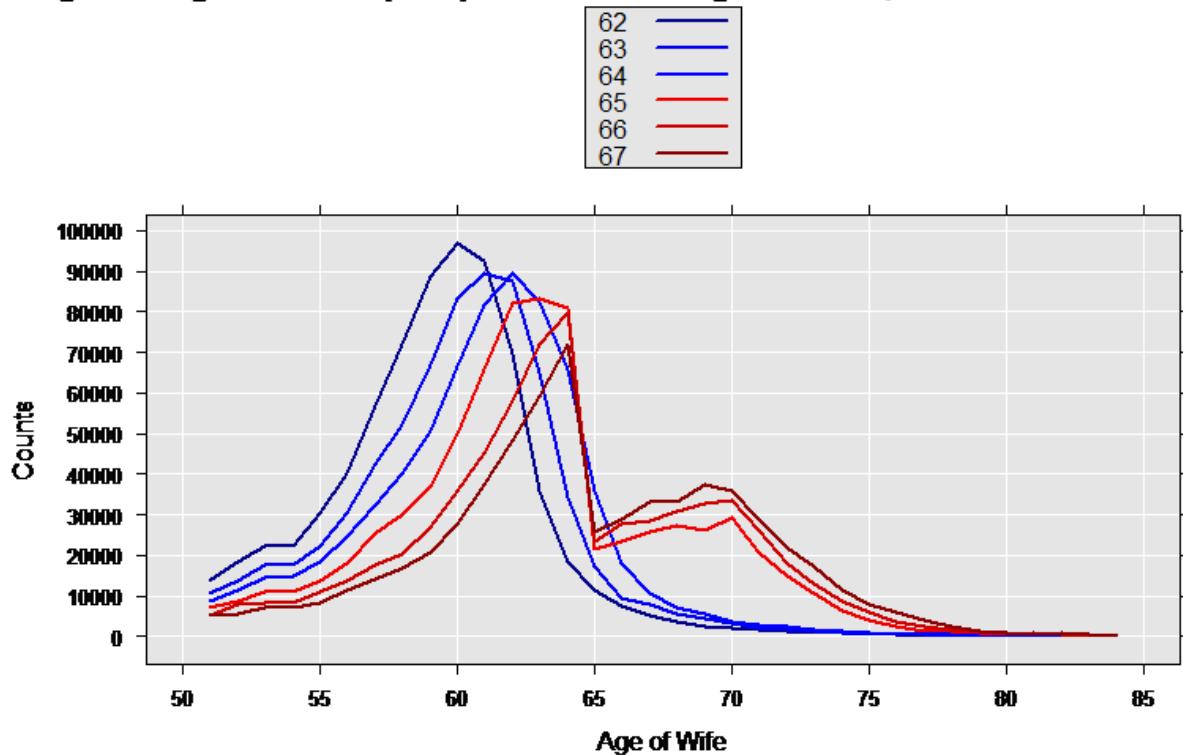


This shows a different story. Notice that there are very few men in the 60 to 80 age range married to 65 year-old women, and in particular, there are very few 65-year-old men married to 65-year-old women. To examine

this further, we can look at line plots of the distributions of wife's ages for men ages 62 to 67:

```
ageCompareSubset <- subset(aa, age > 61 & age < 68 & spouseAge > 50 &
spouseAge < 85)
rxLinePlot(Counts~spouseAge, groups = age, data = ageCompareSubset,
xlab = "Age of Wife", ylab = "Counts",
lineColor = c("Blue4", "Blue2", "Blue1", "Red2", "Red3", "Red4"),
main = "Figure 8: Ages of Wives (> 45) for Husband's Ages 62 to 67, U.S. 2000 5% Census")
```

Figure 8: Ages of Wives (> 45) for Husband's Ages 62 to 67, U.S. 2000 5% Census



The blue lines show husbands ages 62 through 64. The mode of the wife's age is a couple of years younger than the husband, and we have a reasonable looking distribution in both tails. But starting at age 65, with the red lines, we have a very different pattern. It appears that when husbands reach the age of 65 they begin to leave (or lose) their 65-year-old wives in droves – and marry older women. This certainly makes one suspicious of the data. In fact, it turns out the aberration in the data was introduced by disclosure avoidance techniques applied to the data by the census bureau.

Models in RevoScaleR

7/12/2022 • 5 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Specifying a model with RevoScaleR is similar to specifying a model with the standard R statistical modeling functions, but there are some significant differences. Understanding these differences can help you make better use of RevoScaleR. The purpose of this article is to explain these differences at a high level so that when we begin fitting models in script, the terminology does not seem too foreign.

External Memory Algorithms

The first thing to know about RevoScaleR's statistical algorithms is that they are *external memory* algorithms that work on one chunk of data at time. All of RevoScaleR's algorithms share a basic underlying structure:

- An *initialization* step, in which data structures are created and given initial values, a data source is identified and opened for reading, and any other initial conditions are satisfied.
- A *process data* step, in which a chunk of data is read, some calculations are performed, and the result is passed back to the master process.
- An *update results* step, in which the results of the process data step are merged with previous results.
- A *finalize results* step, in which any final processing is performed and a result is returned.

An important consideration in using external memory algorithms is deciding how big a *chunk* of data is. You want to use a chunk size that's as large as possible while still allowing the process data step to complete without swapping. All of the RevoScaleR modeling functions allow you to specify a *blocksPerRead* argument. To make effective use of this, you need to know how many blocks your data file contains and how large the data file is. The number of blocks can be obtained using the *rxGetInfo* function.

The `blocksPerRead` argument is ignored if run locally using R Client. [Learn more...](#)

Formulas in RevoScaleR

RevoScaleR uses a variant of the Wilkinson-Rogers formula notation that is similar to, but not exactly the same as, that used in the standard R modeling functions. The response, or dependent, variables are separated from the predictor, or independent, variables by a tilde (~). In most of the modeling functions, multiple response variables are permitted, specified using the *cbind* function to combine them.

Independent variables (*predictors*) are separated by plus signs (+). Interaction terms can be created by joining two or more variables with a colon (:). Interactions between two categorical variables add one coefficient to the model for every combination of levels of the two variables. For example, if we have the categorical variables sex with two levels and education with four levels, the interaction of sex and education will add eight coefficients to the fitted model. Interactions between a continuous variable and a categorical variable add one coefficient to the model representing the continuous variable for each level of the categorical variable. (However, in RevoScaleR, such interactions cannot be used with the *rxCube* or *rxCrossTabs* functions.) The interaction of two continuous variables is the same as multiplying the two variables.

An asterisk (*) between two variables adds all subsets of interactions to the model. Thus, *sex*education* is equivalent to *sex + education + sex:education*.

The special function syntax $F(x)$ can be used to have RevoScaleR treat a numeric variable x as a categorical variable (factor) for the purposes of the analysis. You can include additional arguments *low* and *high* to specify the minimum and maximum values to be included in the factor variable; RevoScaleR creates a bin for each integer value from the low to high values. You can use the logical flag *exclude* to specify whether values outside that range are omitted from the model (*exclude=TRUE*) or included as a separate level (*exclude=FALSE*).

Similarly, the special function syntax $N(x)$ can be used to have RevoScaleR treat x as a continuous numeric variable. (This syntax is provided for completeness, but is not recommended. In general, if you want to recover numeric data from a factor, you will want to do so from the *levels* of the factor.)

In RevoScaleR, formulas in which the first independent variable is categorical may be handled specially, as *cubes*. See the following section for more details.

Letting the Data Speak For Itself

Classical statistics is largely concerned with fitting predictive models to limited observations. Data analysts and statisticians use exploratory techniques to visualize the data and then make informed guesses as to the appropriate form for a predictive model. Large data analysis, on the other hand, provides the opportunity to let the data speak for itself. With millions of observations in hand, we don't have to guess about the form of relationships between variables: they become clear. A general approach to finding the relationship between two numeric variables x and y might be as follows. First, bin the x values. Then, for each bin, plot the mean of the y values contained in that bin. As the bins become narrower and narrower, the resulting plot becomes a plot of $E(y|x)$ for each value of x .

Cubes and Cube Regression

In RevoScaleR, a *cube* is a type of multi-dimensional array. It may have any number of dimensions, thus making it a hypercube, and each dimension consists of categorical data. The most common operation on a cube is simple cross-tabulation, computing the cell counts for every combination of levels within the cube; this is done using the *rxCube* function. But cubes can also be passed as the first independent variable in a linear or logistic regression, in which case the regression can be computed in a special way. Because the cube consists of categorical data, one part of the moment matrix is diagonal. This makes it possible to compute the regression using a partitioned inverse method, which may be faster and may use less memory than the usual regression method. When a linear or logistic regression is fitted with the argument *cube=TRUE* (and a categorical variable as the first predictor), the partitioned inverse method is used and the model is fitted without an intercept term. Because the first term in the regression is categorical, it is equivalent to a complete set of dummy variables and thus is collinear with a constant term. By dropping the intercept term, the collinearity is resolved and a coefficient can be computed for each level of the categorical predictor. The R-squared, however, is computed as if an intercept were included.

Fitting Linear Models using RevoScaleR

7/12/2022 • 37 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Linear regression models are fitted in RevoScaleR using the `rxLinMod` function. Like other RevoScaleR functions, `rxLinMod` uses an updating algorithm to compute the regression model. The R object returned by `rxLinMod` includes the estimated model coefficients and the call used to generate the model, together with other information that allows RevoScaleR to recompute the model. Because `rxLinMod` is designed to work with arbitrarily large data sets, quantities such as residuals, and fitted values are not included in the return object, although these can be obtained easily once the model has been fitted.

As a simple example, let's use the sample data set `AirlineDemoSmall.xdf` and fit the arrival delay by day of week:

```
# Fitting Linear Models

readPath <- rxGetOption("sampleDataDir")
airlineDemoSmall <- file.path(readPath, "AirlineDemoSmall.xdf")
rxLinMod(ArrDelay ~ DayOfWeek, data = airlineDemoSmall)

Call:
rxLinMod(formula = ArrDelay ~ DayOfWeek, data = airlineDemoSmall)

Linear Regression Results for: ArrDelay ~ DayOfWeek
File name:
C:\Program Files\Microsoft\MRO-for-RRE\8.0\R-3.2.2\library\RevoScaleR\SampleData\AirlineDemoSmall.xdf
Dependent variable(s): ArrDelay
Total independent variables: 8 (Including number dropped: 1)
Number of valid observations: 582628
Number of missing observations: 17372

Coefficients:
          ArrDelay
(Intercept) 10.3318058
DayOfWeek=Monday 1.6937981
DayOfWeek=Tuesday 0.9620019
DayOfWeek=Wednesday -0.1752668
DayOfWeek=Thursday -1.6737983
DayOfWeek=Friday 4.4725290
DayOfWeek=Saturday 1.5435207
DayOfWeek=Sunday Dropped
```

Because our predictor is categorical, we can use the `cube` argument to `rxLinMod` to perform the regression using a partitioned inverse, which may be faster and may use less memory than the standard algorithm. The output object also includes a data frame with the averages or counts for each category:

```
arrDelayLm1 <- rxLinMod(ArrDelay ~ DayOfWeek, cube = TRUE,
data = airlineDemoSmall)
```

Typing the name of the object `arrDelayLm1` yields the following output:

```

Call:
rxLinMod(formula = ArrDelay ~ DayOfWeek, data = airlineDemoSmall,
cube = TRUE)

Cube Linear Regression Results for: ArrDelay ~ DayOfWeek
File name:
  C:\Program Files\Microsoft\MRO-for-RRE\8.0\R-3.2.2\library\ RevoScaleR\SampleData\AirlineDemoSmall.xdf
Dependent variable(s): ArrDelay
Total independent variables: 7
Number of valid observations: 582628
Number of missing observations: 17372

Coefficients:
              ArrDelay
DayOfWeek=Monday 12.025604
DayOfWeek=Tuesday 11.293808
DayOfWeek=Wednesday 10.156539
DayOfWeek=Thursday 8.658007
DayOfWeek=Friday 14.804335
DayOfWeek=Saturday 11.875326
DayOfWeek=Sunday 10.331806

```

Obtaining a Summary of the Model

The print method for the rxLinMod model object shows only the call and the coefficients. You can obtain more information about the model by calling the summary method:

```

# Obtaining a Summary of a Model

summary(arrDelayLm1)

```

This produces the following output, which includes substantially more information about the model coefficients, together with the residual standard error, multiple R-squared, and adjusted R-squared:

```

Call:
rxLinMod(formula = ArrDelay ~ DayOfWeek, data = airlineDemoSmall,
cube = TRUE)

Cube Linear Regression Results for: ArrDelay ~ DayOfWeek
File name:
C:\Program Files\Microsoft\MRO-for-RRE\8.0\R-3.2.2\library\ RevoScaleR\SampleData\AirlineDemoSmall.xdf
Dependent variable(s): ArrDelay
Total independent variables: 7
Number of valid observations: 582628
Number of missing observations: 17372

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    | Counts
DayOfWeek=Monday   12.0256   0.1317  91.32 2.22e-16 *** | 95298
DayOfWeek=Tuesday   11.2938   0.1494  75.58 2.22e-16 *** | 74011
DayOfWeek=Wednesday 10.1565   0.1467  69.23 2.22e-16 *** | 76786
DayOfWeek=Thursday   8.6580   0.1445  59.92 2.22e-16 *** | 79145
DayOfWeek=Friday     14.8043   0.1436 103.10 2.22e-16 *** | 80142
DayOfWeek=Saturday   11.8753   0.1404  84.59 2.22e-16 *** | 83851
DayOfWeek=Sunday     10.3318   0.1330  77.67 2.22e-16 *** | 93395
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 40.65 on 582621 degrees of freedom
Multiple R-squared: 0.001869 (as if intercept included)
Adjusted R-squared: 0.001858
F-statistic: 181.8 on 6 and 582621 DF, p-value: < 2.2e-16
Condition number: 1

```

Using Probability Weights

Probability weights are common in survey data; they represent the probability that a case was selected into the sample from the population, and are calculated as the inverse of the sampling fraction. The variable *perwt* in the census data represents a probability weight. You pass probability weights to the RevoScaleR analysis functions using the *pweights* argument, as in the following example:

```

# Using Probability Weights

rxLinMod(incwage ~ F(age), pweights = "perwt", data = censusWorkers)

```

Yields the following output:

```

Call:
rxLinMod(formula = incwage ~ F(age), data = censusWorkers, pweights = "perwt")

Linear Regression Results for: incwage ~ F(age)
File name:
C:\Program Files\Microsoft\MRO-for-RRE\8.0\R-3.2.2\library\RevoScaleR\SampleData\CensusWorkers.xdf
Probability weights: perwt
Dependent variable(s): incwage
Total independent variables: 47 (Including number dropped: 1)
Number of valid observations: 351121
Number of missing observations: 0

Coefficients:
          incwage
(Intercept) 32111.3012
F_age=20    -19488.0874
F_age=21    -18043.7738
F_age=22    -15928.5538
F_age=23    -13498.2138
F_age=24    -11248.8583
F_age=25    -7948.6603
F_age=26    -5667.3599
F_age=27    -4682.8375
F_age=28    -3024.1648
F_age=29    -1480.8682
F_age=30    -971.4461
F_age=31    143.5648
F_age=32    2009.0019
F_age=33    2861.3031
F_age=34    2797.9986
F_age=35    4038.2131
F_age=36    5511.8633
F_age=37    5148.1841
F_age=38    5957.2190
F_age=39    7652.9870
F_age=40    6969.9630
F_age=41    8387.6821
F_age=42    9150.0006
F_age=43    8936.7590
F_age=44    9267.0820
F_age=45    10148.1702
F_age=46    9099.0659
F_age=47    9996.0450
F_age=48    10408.1712
F_age=49    10281.1324
F_age=50    12029.6751
F_age=51    10247.2529
F_age=52    12105.0654
F_age=53    10957.8211
F_age=54    11881.4307
F_age=55    11490.8457
F_age=56    9442.3856
F_age=57    9331.2347
F_age=58    9353.1980
F_age=59    7526.4912
F_age=60    6078.4463
F_age=61    4636.6756
F_age=62    3129.5531
F_age=63    2535.4858
F_age=64    1520.3707
F_age=65    Dropped

```

Using Frequency Weights

Frequency weights are useful when your data has a particular form: a set of data in which one or more cases is

exactly replicated. If you then compact the data to remove duplicated observations and create a variable to store the number of replications of each observation, you can use that new variable as a frequency weights variable in the RevoScaleR analysis functions.

For example, the sample data set fourthgraders.xdf contains the following 44 rows:

	height	eyecolor	male	reps
14	44	Blue	Male	1
23	44	Brown	Male	3
9	44	Brown	Female	1
38	44	Green	Female	1
8	45	Blue	Male	1
31	45	Blue	Female	5
2	45	Brown	Male	2
51	45	Brown	Female	1
6	45	Green	Male	1
47	45	Hazel	Female	1
37	46	Blue	Male	1
21	46	Blue	Female	3
12	46	Brown	Male	2
5	46	Brown	Female	4
19	46	Green	Male	1
80	46	Green	Female	3
64	47	Blue	Male	1
69	47	Blue	Female	3
39	47	Brown	Male	4
18	47	Brown	Female	3
41	47	Green	Male	2
7	47	Green	Female	4
26	47	Hazel	Male	3
17	48	Blue	Male	4
36	48	Blue	Female	3
13	48	Brown	Male	6
62	48	Brown	Female	4
61	48	Green	Female	2
56	48	Hazel	Male	1
10	49	Blue	Male	1
15	49	Blue	Female	5
22	49	Brown	Male	2
4	49	Brown	Female	5
3	49	Green	Male	3
91	49	Green	Female	1
94	50	Blue	Male	1
96	50	Blue	Female	1
97	50	Brown	Male	1
1	50	Brown	Female	1
28	50	Green	Male	4
52	50	Hazel	Male	1
98	51	Blue	Male	1
57	51	Brown	Female	1
90	51	Green	Female	1

The *reps* column shows the number of replications for each observation; the sum of the *reps* column indicates the total number of observations, in this case 100. We can fit a model (admittedly not very useful) of height on eye color with *rxLinMod* as follows:

```
# Using Frequency Weights

fourthgraders <- file.path(rxGetOption("sampleDataDir"),
  "fourthgraders.xdf")
fourthgradersLm <- rxLinMod(height ~ eyecolor, data = fourthgraders,
  fweights="reps")
```

Typing the name of the object shows the following output:

```
Call:  
rxLinMod(formula = height ~ eyecolor, data = fourthgraders, fweights = "reps")  
  
Linear Regression Results for: height ~ eyecolor  
File name:  
C:\Program Files\Microsoft\MRO-for-RRE\8.0\R-3.2.2\library\RevoScaleR\SampleData\fourthgraders.xdf  
Linear Regression Results for: height ~ eyecolor  
Frequency weights: reps  
Dependent variable(s): height  
Total independent variables: 5 (Including number dropped: 1)  
Number of valid observations: 44  
Number of missing observations: 0  
  
Coefficients:  
height  
(Intercept) 47.33333333  
eyecolor=Blue -0.01075269  
eyecolor=Brown -0.08333333  
eyecolor=Green 0.40579710  
eyecolor=Hazel Dropped
```

Using rxLinMod with R Data Frames

While RevoScaleR is primarily designed to work with data on disk, it can also be used with in-memory R data frames. As an example, consider the R sample data frame "cars", which contains data recorded in the 1920s on the speed of cars and the distances taken to stop.

```
# Using rxLinMod with R Data Frames  
  
rxLinMod(dist ~ speed, data = cars)
```

We get the following output (which matches the output given by the R function lm):

```
Call:  
rxLinMod(formula = dist ~ speed, data = cars)  
  
Linear Regression Results for: dist ~ speed  
Data: cars  
Dependent variable(s): dist  
Total independent variables: 2  
Number of valid observations: 50  
Number of missing observations: 0  
  
Coefficients:  
dist  
(Intercept) -17.579095  
speed 3.932409
```

Using the Cube Option for Conditional Predictions

If the cube argument is set to TRUE and the first term of the independent variables is categorical, *rxLinMod* will compute and return a data frame with category counts. If there are no other independent variables, or if the *cubePredictions* argument is set to TRUE, the data frame will also contain predicted values. Let's create a simple data frame to illustrate:

```

# Using the Cube Option for Conditional Predictions

xfac1 <- factor(c(1,1,1,1,2,2,2,2,3,3,3,3), labels=c("One1", "Two1", "Three1"))
xfac2 <- factor(c(1,1,1,1,1,2,2,2,3,3,3,3), labels=c("One2", "Two2", "Three2"))
set.seed(100)
y <- as.integer(xfac1) + as.integer(xfac2)* 2 + rnorm(12)
myData <- data.frame(y, xfac1, xfac2)

```

If we estimate a simple linear regression of y on xfac1, the coefficients are equal to the within- group means:

```

myLinMod <- rxLinMod(y ~ xfac1, data = myData, cube = TRUE)
myLinMod

Call:
rxLinMod(formula = y ~ xfac1, data = myData, cube = TRUE)

Cube Linear Regression Results for: y ~ xfac1
Data: myData
Dependent variable(s): y
Total independent variables: 3
Number of valid observations: 12
Number of missing observations: 0

Coefficients:
      y
xfac1=One1  3.109302
xfac1=Two1   5.142086
xfac1=Three1 8.250260

```

In addition to the standard output, the returned object contains a *countDF* data frame with information on within-group means and counts in each category. In this simple case the within-group means are the same as the coefficients:

```

myLinMod$countDF
  xfac1      y Counts
1 One1 3.109302     4
2 Two1 5.142086     4
3 Three1 8.250260    4

```

Using rxLinMod with the cube option also allows us to compute conditional within-group means. For example:

```

myLinMod1 <- rxLinMod(y~xfac1 + xfac2, data = myData,
  cube = TRUE, cubePredictions = TRUE)
myLinMod1

Call:
rxLinMod(formula = y ~ xfac1 + xfac2, data = myData, cube = TRUE,
cubePredictions=TRUE)

Cube Linear Regression Results for: y ~ xfac1 + xfac2
Data: myData
Dependent variable(s): y
Total independent variables: 6 (Including number dropped: 1)
Number of valid observations: 12
Number of missing observations: 0

Coefficients:
y
xfac1=One1    7.725231
xfac1=Two1    8.833730
xfac1=Three1  8.942099
xfac2=One2   -4.615929
xfac2=Two2   -2.767359
xfac2=Three2 Dropped

```

If we look at the countDF, we see the within-group means, conditional on the average value of the conditioning variable, xfac2:

```

myLinMod1$countDF
  xfac1      y Counts
1 One1 4.725427     4
2 Two1 5.833926     4
3 Three1 5.942295    4

```

For the variable xfac2, 50% of the observations have the value "One2", 25% of the observations have the value "Two2", and 25% have the value "Three2". We can compute the weighted average of the coefficients for xfac2 as:

```

myCoef <- coef(myLinMod1)
avexfac2c <- .5*myCoef[4] + .25*myCoef[5]

```

To compute the conditional within-group mean shown in the countDF for xfac1 equal to "One1", we add this to the coefficient computed for "One1":

```

condMean1 <- myCoef[1] + avexfac2c
condMean1

xfac1=One1
4.725427

```

Conditional within-group means can also be computed using additional continuous independent variables.

Fitted Values, Residuals, and Prediction

When you fit a model with lm or any of the other core R model-fitting functions, you get back an object that includes as components both the fitted values for the response variable and the model residuals. For models fit with rxLinMod or other RevoScaleR functions, it is usually impractical to include these components, as they can be many megabytes in size. Instead, they are computed on demand using the rxPredict function. This function takes an rxLinMod object as its first argument, an input data set as its second argument, and an output data set as its third argument. If the input data set is the same as the data set used to fit the rxLinMod object, the

resulting predictions are the fitted values for the model. If the input data set is a different data set (but one containing the same variable names used in fitting the rxLinMod object), the resulting predictions are true predictions of the response for the new data from the original model. In either case, residuals for the predicted values can be obtained by setting the flag computeResiduals to TRUE.

For example, we can draw from the 7% sample of the large airline data set (available [online](#)) training and prediction data sets as follows (remember to customize the first line below for your own system):

```
bigDataDir <- "C:/MRS/Data"
sampleAirData <- file.path(bigDataDir, "AirOnTime7Pct.xdf")
trainingDataFile <- "AirlineData06to07.xdf"
targetInfile <- "AirlineData08.xdf"

rxDataStep(sampleAirData, trainingDataFile, rowSelection = Year == 1999 |
Year == 2000 | Year == 2001 | Year == 2002 | Year == 2003 |
Year == 2004 | Year == 2005 | Year == 2006 | Year == 2007)
rxDataStep(sampleAirData, targetInfile, rowSelection = Year == 2008)
```

We can then fit a linear model with the training data and compute predicted values on the prediction data set as follows:

```
arrDelayLm2 <- rxLinMod(ArrDelay ~ DayOfWeek + UniqueCarrier + Dest,
data = trainingDataFile)
rxPredict(arrDelayLm2, data = targetInfile, outData = targetInfile)
```

To see the first few rows of the result, use rxGetInfo as follows:

```
rxGetInfo(targetInfile, numRows = 5)

File name: C:\MRS\Data\OutData\AirlineData08.xdf
Number of observations: 489792
Number of variables: 14
Number of blocks: 2
Compression type: zlib
Data (5 rows starting with row 1):
  Year DayOfWeek UniqueCarrier Origin Dest CRSDepTime DepDelay TaxiOut TaxiIn
1 2008     Mon         9E    ATL   HOU  7.250000     -2      14      5
2 2008     Mon         9E    ATL   HOU  7.250000     -2      16      2
3 2008     Sat         9E    ATL   HOU  7.250000      0      18      5
4 2008     Wed         9E    ATL   SRQ 12.666667      6      20      4
5 2008     Sat         9E   HOU   ATL  9.083333     -9      15      9
  ArrDelay ArrDel15 CRSElapsedTime Distance ArrDelay_Pred
1      -15    FALSE          140      696    8.981548
2       0    FALSE          140      696    8.981548
3       0    FALSE          140      696    5.377572
4       6    FALSE          86      445    7.377952
5      -23    FALSE          120      696    6.552242
```

Prediction Standard Errors, Confidence Intervals, and Prediction Intervals

You can also use rxPredict to obtain prediction standard errors, provided you have included the variance-covariance matrix in the original rxLinMod fit. If you choose to compute the prediction standard errors, you can also obtain either of two kinds of intervals: *confidence intervals* that for a given confidence level tells us how confident we are that the expected value is within the given interval, and *prediction intervals* that specify, for a given confidence level, how likely future observations are to fall within the interval given what has already been observed. Standard error computations are computationally intensive, and they may become prohibitive on

large data sets with a large number of predictors. To illustrate the computation, we start with a small data set, using the example on page 132 of *Introduction to the Practice of Statistics*, (5th Edition). The predictor, nealnc, is the increase in "non-exercise activity" in response to an increase in caloric intake. The response, fatGain, is the associated increase in fat. We first read in the data and create a data frame to use in our analysis:

```
# Standard Errors, Confidence Intervals, and Prediction Intervals

neaInc <- c(-94, -57, -29, 135, 143, 151, 245, 355, 392, 473, 486, 535, 571,
580, 620, 690)
fatGain <- c( 4.2, 3.0, 3.7, 2.7, 3.2, 3.6, 2.4, 1.3, 3.8, 1.7, 1.6, 2.2, 1.0,
0.4, 2.3, 1.1)
ips132df <- data.frame(neaInc = neaInc, fatGain=fatGain)
```

Next we fit the linear model with rxLinMod, setting covCoef=TRUE to ensure we have the variance-covariance matrix in our model object:

```
ips132lm <- rxLinMod(fatGain ~ neaInc, data=ips132df, covCoef=TRUE)
```

Now we use rxPredict to obtain the fitted values, prediction standard errors, and confidence intervals. By setting writeModelVars to TRUE, the variables used in the model will also be included in the output data set. In this first example, we obtain confidence intervals:

```
ips132lmPred <- rxPredict(ips132lm, data=ips132df, computeStdErrors=TRUE,
interval="confidence", writeModelVars = TRUE)
```

The standard errors are by default put into a variable named by concatenating the name of the response variable with an underscore and the string "StdErr":

```
ips132lmPred$fatGain_StdErr

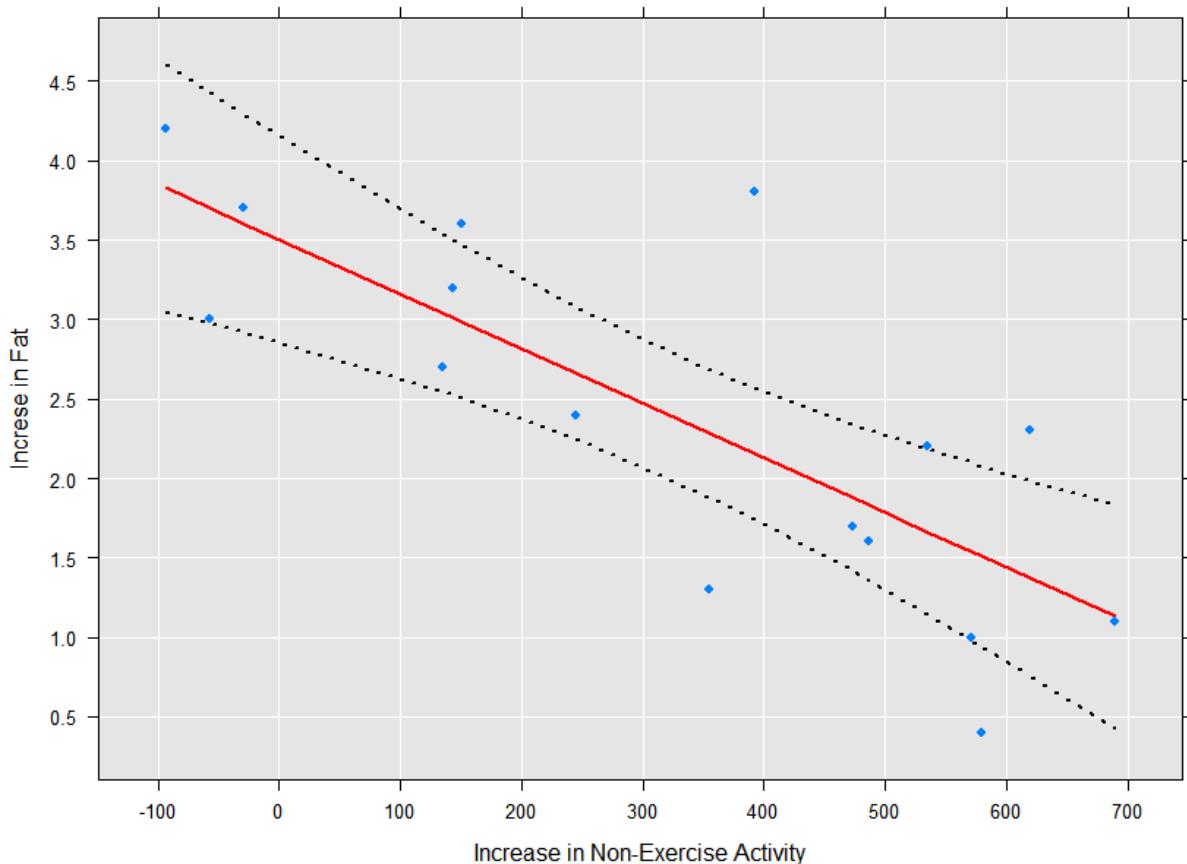
[1] 0.3613863 0.3381111 0.3209344 0.2323853 0.2288433 0.2254018 0.1941840
[8] 0.1863180 0.1915656 0.2151569 0.2202366 0.2418892 0.2598922 0.2646223
[15] 0.2865818 0.3279390
```

We can view the original data, the fitted prediction line, and the confidence intervals as follows:

```
rxLinePlot(fatGain + fatGain_Pred + fatGain_Upper + fatGain_Lower ~ neaInc,
data = ips132lmPred, type = "b",
lineStyle = c("blank", "solid", "dotted", "dotted"),
lineColor = c(NA, "red", "black", "black"),
symbolStyle = c("solid circle", "blank", "blank", "blank"),
title = "Data, Predictions, and Confidence Bounds",
xTitle = "Increase in Non-Exercise Activity",
yTitle = "Increase in Fat", legend = FALSE)
```

The resulting plot is shown below:

Data, Predictions, and Confidence Bounds

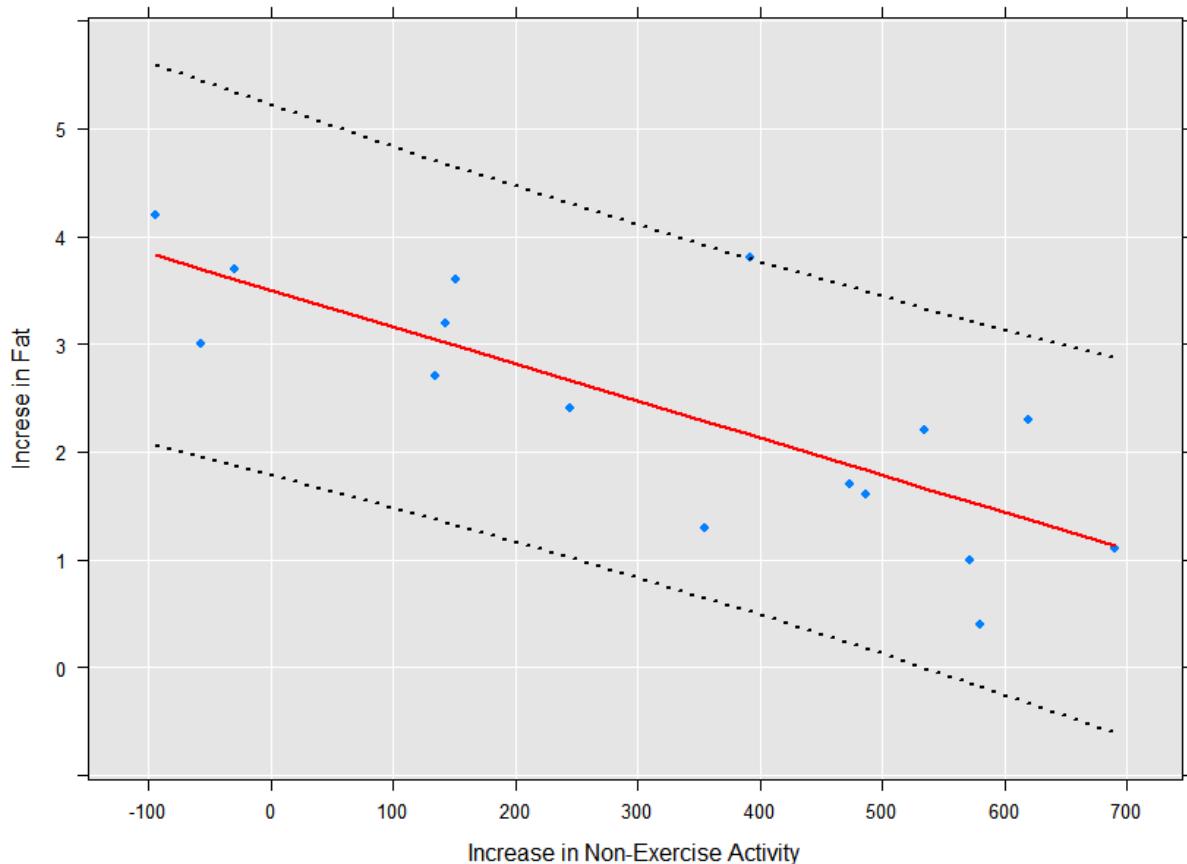


The prediction intervals can be obtained and plotted as follows:

```
ips132lmPred2 <- rxPredict(ips132lm, data=ips132df, computeStdErrors=TRUE,
  interval="prediction", writeModelVars = TRUE)
rxLinePlot(fatGain + fatGain_Pred + fatGain_Upper + fatGain_Lower ~ neaInc,
  data = ips132lmPred2, type = "b",
  lineStyle = c("blank", "solid", "dotted", "dotted"),
  lineColor = c(NA, "red", "black", "black"),
  symbolStyle = c("solid circle", "blank", "blank", "blank"),
  title = "Prediction Intervals",
  xTitle = "Increase in Non-Exercise Activity",
  yTitle = "Increase in Fat", legend = FALSE)
```

The resulting plot is shown below:

Prediction Intervals



We can fit the prediction standard errors on our large airline regression model if we first refit it with covCoef=TRUE:

```
arrDelayLmVC <- rxLinMod(ArrDelay ~ DayOfWeek + UniqueCarrier + Dest,  
  data = trainingDataFile, covCoef=TRUE)
```

We can then obtain the prediction standard errors and a confidence interval as before:

```
rxPredict(arrDelayLmVC, data = targetInfile, outData = targetInfile,  
  computeStdErrors=TRUE, interval = "confidence", overwrite=TRUE)
```

We can then look at the first few lines of targetInfile to see the first few predictions and standard errors:

```

rxGetInfo(targetInfile, numRows=10)

File name: C:\YourOutputPath\AirlineData08.xdf
Number of observations: 489792
Number of variables: 17
Number of blocks: 2
Compression type: zlib
Data (10 rows starting with row 1):
  Year DayOfWeek UniqueCarrier Origin Dest CRSDepTime DepDelay TaxiOut TaxiIn
  1 2008      Mon         9E    ATL   HOU  7.250000     -2     14     5
  2 2008      Mon         9E    ATL   HOU  7.250000     -2     16     2
  3 2008      Sat         9E    ATL   HOU  7.250000      0     18     5
  4 2008      Wed         9E    ATL   SRQ 12.666667      6     20     4
  5 2008      Sat         9E    HOU   ATL  9.083333     -9     15     9
  6 2008      Mon         9E    ATL   IAH 16.416666     -3     25     4
  7 2008      Thur        9E    IAH   ATL 18.500000      9     12     7
  8 2008      Sat         9E    IAH   ATL 18.500000     -3     19     8
  9 2008      Mon         9E    IAH   ATL 18.500000     -5     16     6
 10 2008     Thur        9E    ATL   IAH 13.250000     -2     15     7
  ArrDelay ArrDel15 CRSElapsedTime Distance ArrDelay_Pred ArrDelay_StdErr
  1     -15    FALSE          140      696    8.981548    0.3287748
  2      0    FALSE          140      696    8.981548    0.3287748
  3      0    FALSE          140      696    5.377572    0.3293198
  4      6    FALSE          86       445    7.377952    0.6380760
  5     -23    FALSE          120      696    6.552242    0.2838803
  6     -11    FALSE          145      689    7.530244    0.2952031
  7     -8    FALSE          125      689   12.168922    0.2833356
  8     -16    FALSE          125      689    6.552242    0.2838803
  9     -24    FALSE          125      689   10.156218    0.2833283
 10     21    TRUE           130      689    9.542948    0.2952058
  ArrDelay_Lower ArrDelay_Upper
  1     8.337161    9.625935
  2     8.337161    9.625935
  3     4.732117    6.023028
  4     6.127346    8.628559
  5     5.995847    7.108638
  6     6.951657    8.108832
  7    11.613594   12.724249
  8     5.995847    7.108638
  9     9.600905   10.711532
 10    8.964355   10.121540

```

Stepwise Variable Selection

Stepwise linear regression is an algorithm that helps you determine which variables are most important to a regression model. You provide a minimal, or lower, model formula and a maximal, or upper, model formula, and using forward selection, backward elimination, or bidirectional search, the algorithm determines the model formula that provides the best fit based on an AIC selection criterion.

In SAS, stepwise linear regression is implemented through PROC REG. In open-source R, it is implemented through the function *step*. The problem with using the function *step* in R is that the size of the data set that can be analyzed is severely limited by the requirement that all computations must be done in memory.

RevoScaleR provides an implementation of stepwise linear regression that is not constrained by the use of "in-memory" algorithms. Stepwise linear regression in RevoScaleR is implemented by the functions *rxLinMod* and *rxStepControl*.

Stepwise linear regression begins with an initial model of some sort. Consider, for example, the airline training data set AirlineData06to07.xdf featured in [Fitting Linear Models using RevoScaleR](#):

```

# Stepwise Linear Regression

rxGetVarInfo(trainingDataFile)
  Var 1: Year, Type: integer, Low/High: (1999, 2007)
  Var 2: DayOfWeek
    7 factor levels: Mon Tues Wed Thur Fri Sat Sun
  Var 3: UniqueCarrier
    30 factor levels: AA US AS CO DL ... OH F9 YV 9E VX
  Var 4: Origin
    373 factor levels: JFK LAX HNL OGG DFW ... IMT ISN AZA SHD LAR
  Var 5: Dest
    377 factor levels: LAX HNL JFK OGG DFW ... ESC IMT ISN AZA SHD
  Var 6: CRSDepTime, Type: numeric, Storage: float32, Low/High: (0.0000, 23.9833)
  Var 7: DepDelay, Type: integer, Low/High: (-1199, 1930)
  Var 8: TaxiOut, Type: integer, Low/High: (0, 1439)
  Var 9: TaxiIn, Type: integer, Low/High: (0, 1439)
  Var 10: ArrDelay, Type: integer, Low/High: (-926, 1925)
  Var 11: ArrDel15, Type: logical, Low/High: (0, 1)
  Var 12: CRSElapsedTime, Type: integer, Low/High: (-34, 1295)
  Var 13: Distance, Type: integer, Low/High: (11, 4962)

```

We are interested in fitting a model that will predict arrival delay (*ArrDelay*) as a function of some of the other variables. To keep things simple, we'll start with our by now familiar model of arrival delay as a function of *DayOfWeek* and *CRSDepTime*.

```

initialModel <- rxLinMod(ArrDelay ~ DayOfWeek + CRSDepTime,
  data = trainingDataFile)
initialModel

Call:
rxLinMod(formula = ArrDelay ~ DayOfWeek + CRSDepTime, data = trainingDataFile)

Linear Regression Results for: ArrDelay ~ DayOfWeek + CRSDepTime
File name:
C:\MyWorkingDir\Documents\MRS\LMPrediction\AirlineData06to07.xdf
Dependent variable(s): ArrDelay
Total independent variables: 9 (Including number dropped: 1)
Number of valid observations: 3945964
Number of missing observations: 98378

Coefficients:
  ArrDelay
(Intercept) -4.91416806
DayOfWeek=Mon  0.49172258
DayOfWeek=Tues -1.41878850
DayOfWeek=Wed   0.09677481
DayOfWeek=Thur   2.52841304
DayOfWeek=Fri    3.29474667
DayOfWeek=Sat   -2.86217838
DayOfWeek=Sun      Dropped
CRSDepTime     0.86703378

```

The question is, can we improve this model by adding more predictors? If so, which ones? To use stepwise selection in RevoScaleR, you add the *variableSelection* argument to your call to *rxLinMod*. The *variableSelection* argument is a list, most conveniently created by using the *rxStepControl* function. Using *rxStepControl*, you specify the method (the default, "stepwise", specifies a bidirectional search), the scope (lower and upper formulas for the search), and various control parameters. With our model, we first want to try some more numeric predictors, so we specify our model as follows:

```

airlineStepModel <- rxLinMod(ArrDelay ~ DayOfWeek + CRSDepTime,
  data = trainingDataFile,
  variableSelection = rxStepControl(method="stepwise",
    scope = ~ DayOfWeek + CRSDepTime + CRSElapsedTime +
    Distance + TaxiIn + TaxiOut ))
Call:
rxLinMod(formula = ArrDelay ~ DayOfWeek + CRSDepTime, data = trainingDataFile,
  variableSelection = rxStepControl(method = "stepwise", scope = ~DayOfWeek +
  CRSDepTime + CRSElapsedTime + Distance + TaxiIn + TaxiOut))

Linear Regression Results for: ArrDelay ~ DayOfWeek + CRSDepTime +
CRSElapsedTime + Distance + TaxiIn + TaxiOut
File name:
C:\MyWorkingDir\Documents\MRS\LMPrediction\AirlineData06to07.xdf
Dependent variable(s): ArrDelay
Total independent variables: 13 (Including number dropped: 1)
Number of valid observations: 3945964
Number of missing observations: 98378

Coefficients:
          ArrDelay
(Intercept) -9.87474950
DayOfWeek=Mon  0.09830827
DayOfWeek=Tues -1.81998781
DayOfWeek=Wed  -0.62761606
DayOfWeek=Thur  1.53941124
DayOfWeek=Fri   2.45565510
DayOfWeek=Sat  -2.20111789
DayOfWeek=Sun      Dropped
CRSDepTime     0.75995355
CRSElapsedTime -0.20256102
Distance        0.02135381
TaxiIn          0.16194266
TaxiOut         0.99814931

```

Methods of Variable Selection

Three methods of variable selection are supported by `rxLinMod`:

- "*forward*": starting from the minimal model, variables are added one at a time until no additional variable satisfies the selection criterion, or until the maximal model is reached.
- "*backward*": starting from the maximal model, variables are removed one at a time until the removal of another variable won't satisfy the selection criterion, or until the minimal model is reached.
- "*stepwise*"(the default): a combination of forward and backward selection, in which variables are added to the minimal model, but at each step, the model is reanalyzed to see if any variables that have been added are candidates for deletion from the current model.

You specify the desired method by supplying a named component "*method*" in the list supplied for the `variableSelection` argument, or by specifying the `method` argument in a call to `rxStepControl`/that is then passed as the `variableSelection` argument.

Variable Selection with Wide Data

We've found that generalized linear models do not converge if the number of predictors is greater than the number of observations. If your data has more variables, it won't be possible to include all of them in the maximal model for stepwise selection. We recommend you use domain experience and insights from initial data explorations to choose a subset of the variables to serve as the maximal model before performing stepwise selection.

There are a few things you can do to reduce the number of predictors. If there are many variables that measure the same quantitative or qualitative entity, try to select one variable that represents the entity best. For example,

include a variable that identifies a person's political party affiliation instead of including many variables representing how the person feels about individual issues. If your goal with linear modeling is the interpretation of individual predictors, you want to ensure that correlation between variables in the model is minimal to avoid multicollinearity. This means checking for these correlations before modeling. Sometimes it is useful to combine correlated variables into a composite variable. Height and weight are often correlated, but can be transformed into BMI. Combining variables allows you to reduce the number of variables without losing any information. Ultimately, the variables you select will depend on how you plan to use the results of your linear model.

Specifying Model Scope

You use the *scope* argument in *rxStepControl* (or a named component "scope" in the list supplied for the *variableSelection* argument) to specify which variables should be considered for inclusion in the final model selection and which should be ignored. Whether you specify a separate value for scope also determines which models the algorithm tries next.

You can specify the scope as a simple formula (which will be treated as the upper bound, or maximal model), or as a named list with components "*lower*" and "*upper*" (either of which may be missing). For example, to analyze the iris data with a minimal model involving the single predictor *Sepal.Width* and a maximal model involving *Sepal.Width*, *Petal.Length*, and the interaction between *Petal.Width* and *Species*, we can specify our variable selection model as follows:

```
# Specifying Model Scope

form <- Sepal.Length ~ Sepal.Width + Petal.Length
scope <- list(
  lower = ~ Sepal.Width,
  upper = ~ Sepal.Width + Petal.Length + Petal.Width * Species)

varsel <- rxStepControl(method = "stepwise", scope = scope)
rxlm.step <- rxLinMod(form, data = iris, variableSelection = varsel,
  verbose = 1, dropMain = FALSE, coefLabelStyle = "R")
```

In general, the models considered are determined from the *scope* argument as follows:

- "*lower*" scope only: All models considered will include this lower model up to the base model specified in the *formula* argument to *rxLinMod*.
- "*upper*" scope only: All models considered will include the terms in the base model (specified by the *formula* argument to *rxLinMod*), plus additional terms as specified in the upper scope.
- Both "*lower*" and "*upper*" scope supplied: All models considered will include at least the terms in the lower scope and additional terms are added using terms from the upper scope until the stopping criterion is reached or the full upper scope model is selected.
- No scope supplied: The minimal model includes just the intercept term and the maximal model include at most the terms specified in the base model.

(This convention is identical to that used by R's *step* function.)

Specifying the Selection Criterion

By default, variable selection is determined using the Akaike Information Criterion, or AIC; this is the R standard. If you want a stepwise selection that is more SAS-like, you can specify *stepCriterion*="SigLevel". If this is set *rxLinMod* uses either an F-test (default) or Chi-square test to determine whether to add or drop terms from the model. You can specify which test to perform using the *test* argument. For example, we can refit our airline model using the SigLevel step criterion with an F-test as follows:

```

#
# Specifying selection criterion
#
airlineStepModelSigLevel <- rxLinMod(ArrDelay ~ DayOfWeek + CRSDepTime,
data = trainingDataFile, variableSelection =
rxStepControl( method = "stepwise", scope = ~ DayOfWeek +
CRSDepTime + CRSElapsedTime + Distance + TaxiIn + TaxiOut,
stepCriterion = "SigLevel" ))

```

In this case (and with many other well-behaved models), the results of using the `SigLevel` step criterion are identical to those using AIC.

You can control the significance levels for adding and dropping models using the `maxSigLevelToAdd` and `minSigLevelToDrop`. The `maxSigLevelToAdd` specifies a significance level below which a variable can be considered for inclusion in the model; the `minSigLevelToDrop` specifies a significance level above which a variable currently included can be considered for dropping. By default, for `method="stepwise"`, both the levels are set to 0.15. We can tighten the selection criterion to the 0.10 level as follows:

```

airlineStepModelSigLevel.10 <- rxLinMod(ArrDelay ~ DayOfWeek + CRSDepTime,
data = trainingDataFile, variableSelection =
rxStepControl( method = "stepwise", scope = ~ DayOfWeek +
CRSDepTime + CRSElapsedTime + Distance + TaxiIn + TaxiOut,
stepCriterion = "SigLevel",
maxSigLevelToAdd=.10, minSigLevelToDrop=.10))

```

Plotting Model Coefficients

By default, the values of the parameters at each step of the stepwise selection are not preserved. Using an additional argument, `keepStepCoefs`, in your `rxStepControl` statement saves the values of the coefficients from each step of the regression. This coefficient data can then be plotted using another function, `rxStepPlot`.

Consider the stepwise linear regression on the iris data from [Fitting Linear Models using RevoScaleR](#):

```

#
# Plotting Model Coefficients at Each Step
#
form <- Sepal.Length ~ Sepal.Width + Petal.Length
scope <- list(
lower = ~ Sepal.Width,
upper = ~ Sepal.Width + Petal.Length + Petal.Width * Species)

varsel <- rxStepControl(method = "stepwise", scope = scope, keepStepCoefs=TRUE)
rxlm.step <- rxLinMod(form, data = iris, variableSelection = varsel,
verbose = 1, dropMain = FALSE, coefLabelStyle = "R")

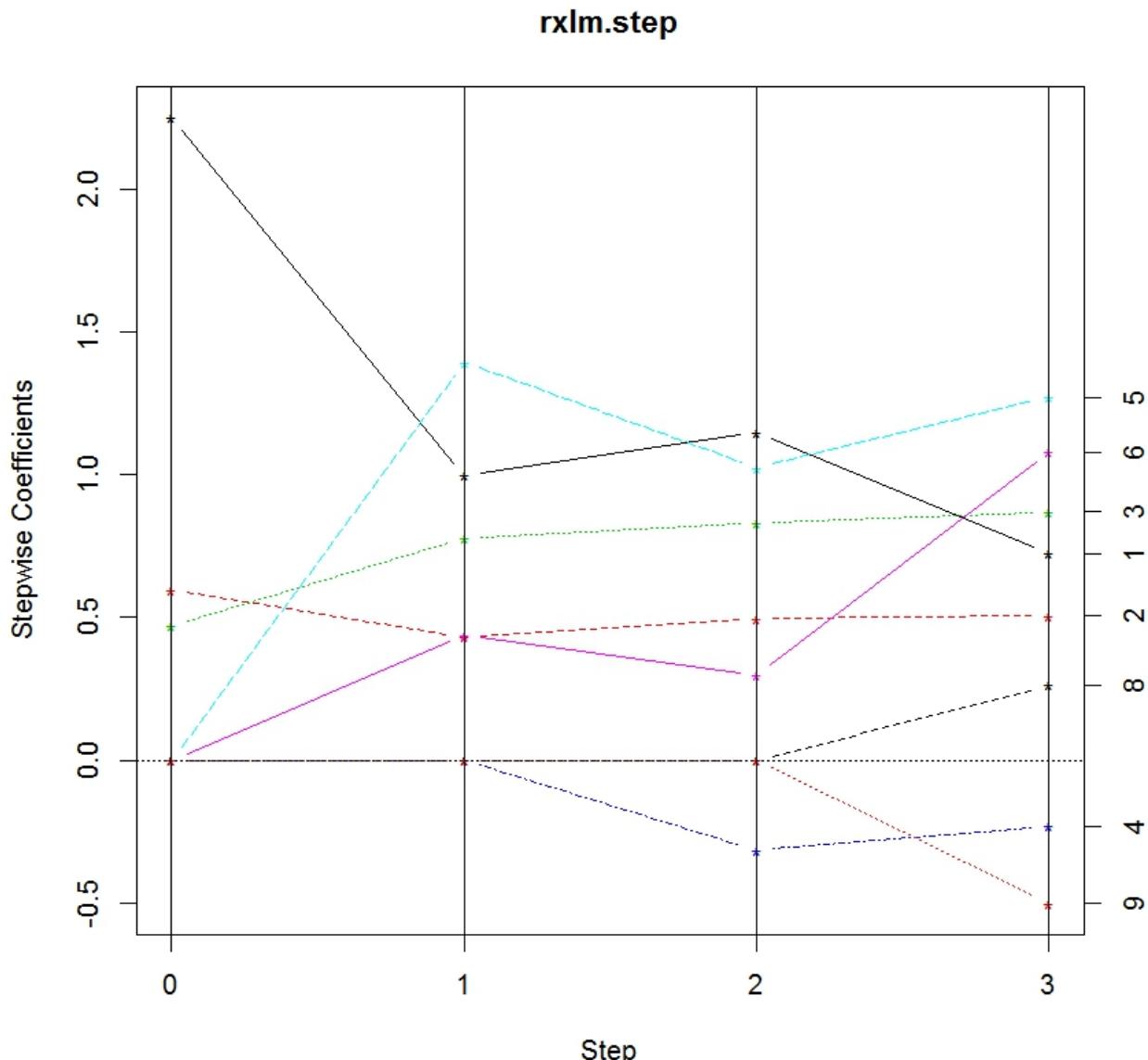
```

Notice the addition of the argument `keepStepCoefs = TRUE` to the `rxStepControl` call. This produces an extra piece of output in the `rxLinMod` object, a dataframe containing the values of the coefficients at each step of the regression. This dataframe, `stepCoefs`, can be accessed as follows:

	0	1	2	3
(Intercept)	2.2491402	0.9962913	1.1477685	0.7228228
Sepal.Width	0.5955247	0.4322172	0.4958889	0.5050955
Petal.Length	0.4719200	0.7756295	0.8292439	0.8702840
Petal.Width	0.0000000	0.0000000	-0.3151552	-0.2313888
Species1	0.0000000	1.3940979	1.0234978	1.2715542
Species2	0.0000000	0.4382856	0.2999359	1.0781752
Species3	0.0000000	NA	NA	NA
Petal.Width:Species1	0.0000000	0.0000000	0.0000000	0.2630978
Petal.Width:Species2	0.0000000	0.0000000	0.0000000	-0.5012827
Petal.Width:Species3	0.0000000	0.0000000	0.0000000	NA

Trying to glean patterns and information from a table can be difficult. So we've added another function, `rxStepPlot`, which allows the user to plot the parameter values at each step. Using the `iris` model object, we plot the coefficients:

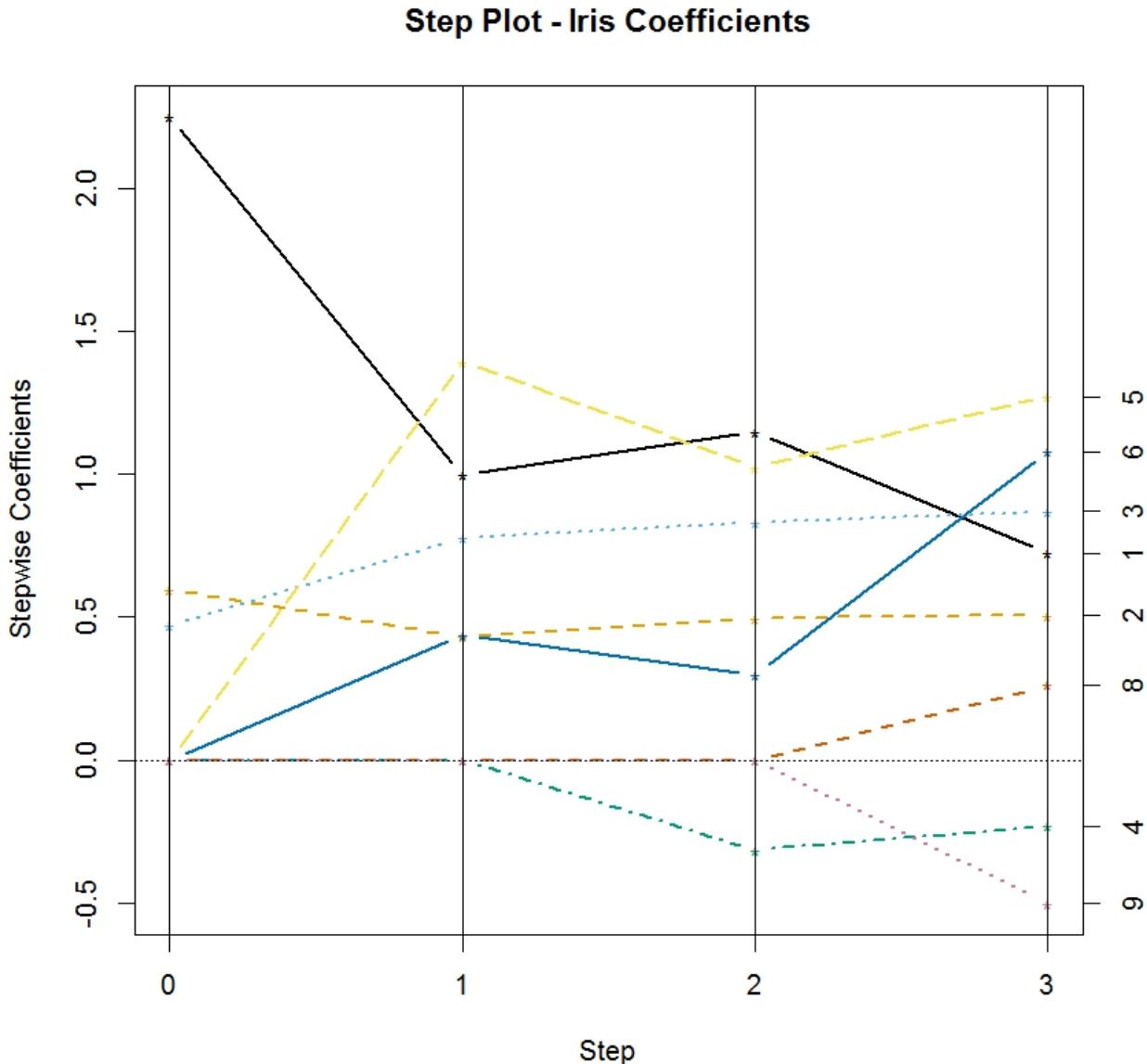
```
rxStepPlot(rxlm.step)
```



From this plot, we can tell when a variable enters the model by noting the step when it becomes non-zero. Lines are labeled with the numbers on the right axis to indicate the parameter. The numbers correspond to the order they appear in the data frame `stepCoef`. You'll notice that the 7th and 10th parameters don't show up in this plot because the `species3` parameter is the reference category for species.

The function `rxStepPlot` is easily customized by using additional graphical parameters from the `matplotlib` function from base R. In the following example, we update our original call to `rxStepPlot` to demonstrate how to specify the colors used for each line, adjust the line width, and add a proper title to the plot:

```
colorSelection <- c("#000000", "#E69F00", "#56B4E9", "#009E73", "#F0E442",
  "#0072B2", "#D55E00", "#CC79A7")
rxStepPlot(rxlm.step, col = colorSelection, lwd = 2,
  main = "Step Plot - Iris Coefficients")
```



By default, the `rxStepPlot` function uses seven line colors. If the number of parameters exceeds the number of colors, they are reused in the same order. However, the line types are set to vary from 1 to 5, so lines that have the same color may differ in line type. The line types can also be specified using the `/ty` argument in the `rxStepPlot` call.

Fixed-Effects Models

Fixed-effects models are commonly associated with studies in which multiple observations are recorded for each test subject, for example, yearly observations of median housing price by city, or measurements of tensile strength from samples of steel rods by batch. To fit such a model with `rxLinMod`, include a factor variable specifying the subject (the cities, or the batch identifier) as the first predictor, and specify `cube=TRUE` to use a partitioned inverse and omit the intercept term.

For example, the MASS library contains the data set *petrol*, which consists of measurements of the yield of a certain refining process with possible predictors including specific gravity, vapor pressure, ASTM 10% point, and volatility measured as the ASTM endpoint for 10 samples of crude oil. The following example (reproduced from [Modern Applied Statistics with S](#)), first scales the numeric predictors, then fits the fixed-effects model:

```
# Fixed-Effects Models

library(MASS)
Petrol <- petrol
Petrol[,2:5] <- scale(Petrol[,2:5], scale=F)
rxLinMod(Y ~ No + EP, data=Petrol, cube=TRUE)

Call:
rxLinMod(formula = Y ~ No + EP, data = Petrol, cube = TRUE)

Cube Linear Regression Results for: Y ~ No + EP
Data: Petrol
Dependent variable(s): Y
Total independent variables: 11
Number of valid observations: 32
Number of missing observations: 0

Coefficients:
Y
No=A 32.5493917
No=B 24.2746407
No=C 27.7820456
No=D 21.1541642
No=E 21.5191269
No=F 20.4355218
No=G 15.0359067
No=H 13.0630467
No=I 9.8053871
No=J 4.4360767
EP 0.1587296
```

Least Squares Dummy Variable (LSDV) Models

RevoScaleR is capable of estimating huge models where fixed effects are estimated by dummy variables, that is, binary variables set to 1 or TRUE if the observation is in a particular category. Creation of these dummy variables is often accomplished by interacting two or more factor variables using ":" in the formula. If the first term in an rxLinMod (or rxLogit) model is purely categorical and the "cube" argument is set to TRUE, the estimation uses a partitioned inverse to save on computation time and memory.

A Quick Review of Interacting Factors

First, let's do a quick, cautionary review of interacting factor variables by experimenting with a small made-up data set.

```
# Least Squares Dummy Variable (LSDV) Models
# A Quick Review of Interacting Factors

set.seed(50)
income <- rep(c(1000,1500,2500,4000), each=5) + 100*rnorm(20)
region <- rep(c("Rural","Urban"), each=10)
sex <- rep(c("Female", "Male"), each=5)
sex <- c(sex,sex)
myData <- data.frame(income, region, sex)
```

The data set has 20 observations with three variables: a numeric variable income and two factor variables representing region and sex. There are three easy ways to compute the within group means of every

combination of age and region using RevoScaleR. First, we can use *rxSummary*:

```
rxSummary(income~region:sex, data=myData)
```

which includes in its output:

Category	region	sex	Means	StdDev	Min
income for region=Rural, sex=Female	Rural	Female	970.7522	98.01983	827.2396
income for region=Urban, sex=Female	Urban	Female	2501.8303	47.10861	2450.1364
income for region=Rural, sex=Male	Rural	Male	1480.4378	92.29320	1355.4250
income for region=Urban, sex=Male	Urban	Male	3944.5127	40.82421	3883.3983

Second, we could use *rxCube*:

```
rxCube(income~region:sex, data=myData)
```

which includes in its output:

region	sex	income	Counts
1 Rural	Female	970.7522	5
2 Urban	Female	2501.8303	5
3 Rural	Male	1480.4378	5
4 Urban	Male	3944.5127	5

Or, we can use *rxLinMod* with *cube=TRUE*. The intercept is automatically omitted, and four dummy variables are created from the two factors: one for each combination of region and sex. The coefficients are simply the within group means:

```
summary(rxLinMod(income~region:sex, cube=TRUE, data=myData))
```

which includes in its output:

Coefficients:						
	Estimate	Std. Error	t value	Pr(> t)		Counts
region=Rural, sex=Female	970.75	33.18	29.26	2.66e-15 ***		5
region=Urban, sex=Female	2501.83	33.18	75.41	2.22e-16 ***		5
region=Rural, sex=Male	1480.44	33.18	44.62	2.22e-16 ***		5
region=Urban, sex=Male	3944.51	33.18	118.90	2.22e-16 ***		5

The same model could be estimated using: *lm(income~region:sex + 0, data=myData)*. Below we refer to these within group means as *MeanIncRuralFemale*, *MeanIncUrbanFemale*, *MeanIncRuralMale*, and *MeanIncUrbanMale*.

If we add an intercept, we encounter perfect multicollinearity and one of the coefficients will be dropped. The intercept is then the mean of a reference group and the other coefficients represent the differences or contrasts between the within group means and the reference group. For example,

```
lm(income~region:sex, data=myData)
```

produces:

```

Coefficients:
(Intercept) region:Rural:sexFemale regionUrban:sexFemale
            3945             -2974            -1443
regionRural:sexMale   regionUrban:sexMale
            -2464                  NA

```

We can see that the dummy variable for urban males was dropped; urban males are the reference group and the Intercept is equal to *MeanIncUrbanMale*. The other coefficients represent contrasts from the reference group, so for example, *regionRural:sexFemale* is equal to *MeanIncRuralFemale – MeanIncUrbanMale*.

Another variation is to use “*” in the formula for factor crossing. Using *a*b* is equivalent to using *a + b + a:b*. For example:

```
lm(income~region*sex, data = myData)
```

which results in:

```

Coefficients:
(Intercept)      regionUrban      sexMale  regionUrban:sexMale
            970.8          1531.1        509.7         933.0

```

The dropping of coefficients in *rxLinMod* can be controlled to obtain the same results:

```
rxLinMod(income~region*sex, data = myData, dropFirst = TRUE, dropMain = FALSE)
```

Coefficients using this model can be more difficult to interpret, and in fact are highly dependent on the order of the factor levels. In this case, we see the following relationship between the estimated coefficients and the within group means

(INTERCEPT)	MEANINCRURALFEMALE
<i>regionUrban</i>	<i>MeanIncUrbanFemale – MeanIncRuralFemale</i>
<i>sexMale</i>	<i>MeanIncRuralMale – MeanIncRuralFemale</i>
<i>regionUrban:sexMale</i>	<i>MeanIncUrbanMale – MeanIncUrbanFemale – MeanIncRuralMale + MeanIncRuralFemale</i>

If we set up our data slightly differently, we get different results. Let's use the same income data but a different naming convention for the factors:

```

region <- rep(c("Rural","Urban"), each=10)
sex <- rep(c("Woman", "Man"), each=5)
sex <- c(sex,sex)
myData1 <- data.frame(income, region, sex)

```

Using the same model, *lm(income~region*sex, data=myData1)*, results in:

```

Coefficients:
(Intercept)      regionUrban      sexWoman
            1480.4          2464.1        -509.7
regionUrban:sexWoman
            -933.0

```

With this superficial modification to the data, the coefficient for the dummy variable for regionUrban has jumped from \$1531 to \$2464. This is due to the change in reference group; in this model the Urban coefficient represents the difference between mean urban and rural male income, while in the previous model it was the difference between mean urban and rural female income. That is, the relationship between the within group means and the new coefficients are:

(INTERCEPT)	MEANINCRURALMALE
<i>regionUrban</i>	$\text{MeanIncUrbanMale} - \text{MeanIncRuralMale}$
<i>sexWoman</i>	$\text{MeanIncRuralFemale} - \text{MeanIncRuralMale}$
<i>regionUrban:sexMale</i>	$\text{MeanIncUrbanFemale} - \text{MeanIncUrbanMale} - \text{MeanIncRuralFemale} + \text{MeanIncRuralMale}$

Omitting the intercept provides yet another combination of results. For example, using rxLinMod with the original factor labeling:

```
rxLinMod(income~region*sex, data=myData, cube=TRUE)
```

results in:

```
Call:
rxLinMod(formula = income ~ region * sex, data = myData, cube = TRUE)

Cube Linear Regression Results for: income ~ region * sex
Data: myData
Dependent variable(s): income
Total independent variables: 8 (Including number dropped: 4)
Number of valid observations: 20
Number of missing observations: 0

Coefficients:
            income
region=Rural    1480.4378
region=Urban     3944.5127
sex=Female      -1442.6824
sex=Male        Dropped
region=Rural, sex=Female 932.9968
region=Urban, sex=Female Dropped
region=Rural, sex=Male   Dropped
region=Urban, sex=Male   Dropped
```

REGION=RURAL	MEANINCRURALMALE
<i>region=Urban</i>	MeanIncUrbanMale
<i>Sex=Female</i>	$\text{MeanIncUrbanFemale} - \text{MeanIncUrbanMale}$
<i>region=Rural, sex=Female</i>	$(\text{MeanIncRuralFemale} - \text{MeanIncRuralMale}) - (\text{MeanIncUrbanFemale} - \text{MeanIncUrbanMale})$

With large data sets it is common to estimate many interaction terms, and if some categories have zero counts, it may not even be obvious what the reference group is. Also note that setting *cube=TRUE* in the preceding model is of limited use: only the first term from the expanded expression (in this case *region*) is estimated using a partitioned inverse.

Using Dummy Variables in rxLinMod: Letting the Data Speak Example 2

In previous articles, we looked at the CensusWorkers.xdf data set and examined the relationship between wage income and age. Now let's add another variable, and examine the relationship between wage income and sex and age.

We can start with a simple dummy variable model, computing the mean wage income by sex:

```
# Using Dummy Variables in rxLinMod

censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers.xdf")
rxLinMod(incwage~sex, data=censusWorkers, pweights="perwt", cube=TRUE)
```

which computes:

```
Call:
rxLinMod(formula = incwage ~ sex, data = censusWorkers, pweights = "perwt",
cube = TRUE)

Cube Linear Regression Results for: incwage ~ sex
File name:
C:\Program Files\Microsoft\MRO-for-RRE\8.0\R-3.2.2\library\RevoScaleR\SampleData\CensusWorkers.xdf
Probability weights: perwt
Dependent variable(s): incwage
Total independent variables: 2
Number of valid observations: 351121
Number of missing observations: 0

Coefficients:
incwage
sex=Male 43472.71
sex=Female 26721.09
```

Similarly, we could look at a simple linear relationship between wage income and age:

```
linMod1 <- rxLinMod(incwage~age, data=censusWorkers, pweights="perwt")
summary(linMod1)
```

resulting in:

```

Call:
rxLinMod(formula = incwage ~ age, data = censusWorkers, pweights = "perwt")

Linear Regression Results for: incwage ~ age
File name:
C:\Program Files\Microsoft\MRO-for-RRE\8.0\R-3.2.2\library\RevoScaleR\SampleData\CensusWorkers.xdf
Probability weights: perwt
Dependent variable(s): incwage
Total independent variables: 2
Number of valid observations: 351121
Number of missing observations: 0

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 12802.980    247.963   51.63 2.22e-16 ***
age          572.980     5.947   96.35 2.22e-16 ***
---
Signif. codes:  0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 180800 on 351119 degrees of freedom
Multiple R-squared:  0.02576
Adjusted R-squared:  0.02576
F-statistic:  9284 on 1 and 351119 DF,  p-value: < 2.2e-16
Condition number: 1

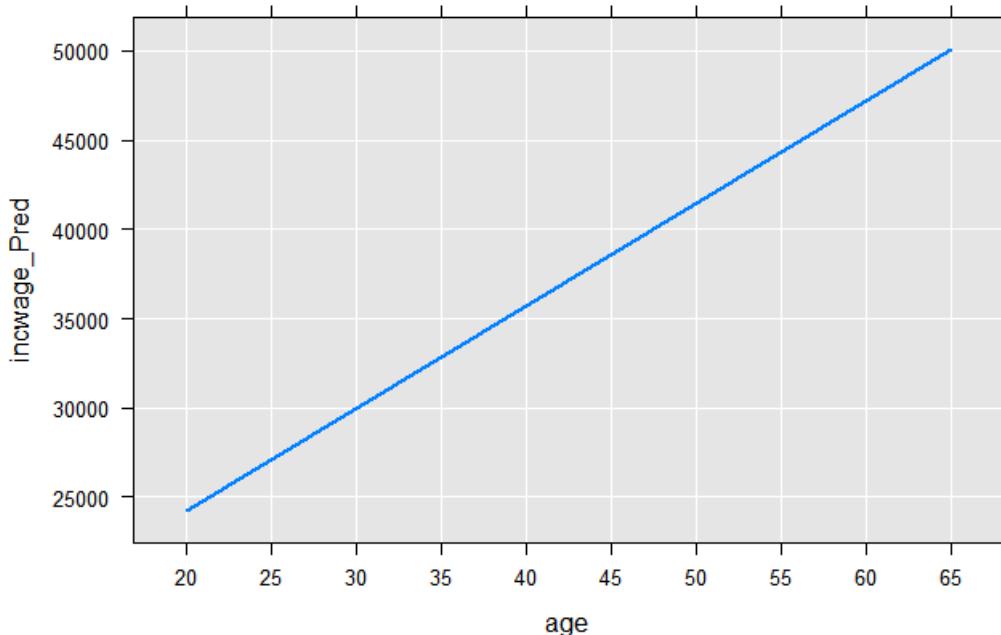
```

Computing the two end points on the regression line, we can plot it:

```

age <- c(20,65)
coefLinMod1 <- coef(linMod1)
incwage_Pred <- coefLinMod1[1] + age*coefLinMod1[2]
plotData1 <- data.frame(age, incwage_Pred)
rxLinePlot(incwage_Pred~age, data=plotData1)

```



The next typical step is to combine the two approaches by estimating separate intercepts for males and females:

```

linMod2 <- rxLinMod(incwage~sex+age, data = censusWorkers, pweights = "perwt",
cube=TRUE)
summary(linMod2)

```

which results in:

```
Call:
rxLinMod(formula = incwage ~ sex + age, data = censusWorkers,
pweights = "perwt", cube = TRUE)

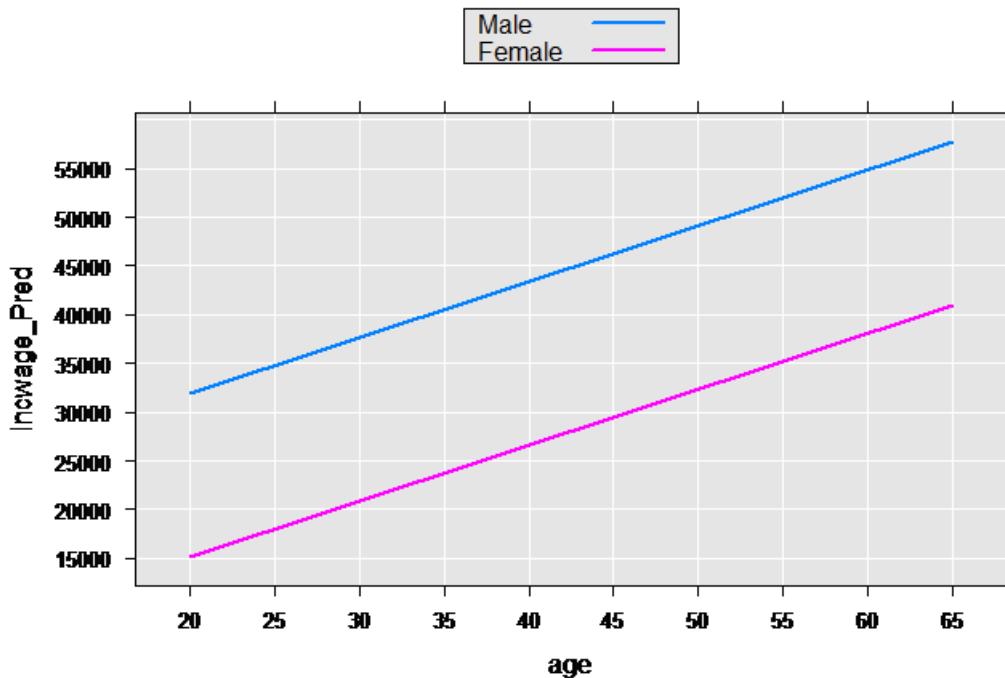
Cube Linear Regression Results for: incwage ~ sex + age
File name:
C:\Program Files\Microsoft\MRO-for-RRE\8.0\R-3.2.2\library\RevoScaleR\SampleData\CensusWorkers.xdf
Probability weights: perwt
Dependent variable(s): incwage
Total independent variables: 3
Number of valid observations: 351121
Number of missing observations: 0

Coefficients:
Estimate Std. Error t value Pr(>|t|) | Counts
sex=Male    20479.178   250.033   81.91 2.22e-16 *** | 3866542
sex=Female   3723.067   252.968   14.72 2.22e-16 *** | 3276744
age          573.232     5.816   98.56 2.22e-16 *** |
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 176800 on 351118 degrees of freedom
Multiple R-squared: 0.06804 (as if intercept included)
Adjusted R-squared: 0.06804
F-statistic: 1.282e+04 on 2 and 351118 DF, p-value: < 2.2e-16
Condition number: 1
```

We create a small sample data set with the same variables we use in `censusWorkers`, but with only four observations representing the high and low value of age for both sexes. Using the `rxPredict` function, the predicted values for each one of these sample observations is computed, which we then plot:

```
age <- c(20,65,20,65)
sex <- factor(rep(c(1, 2), each=2), labels=c("Male", "Female"))
perwt <- rep(1, times=4)
incwage <- rep(0, times=4)
plotData2 <- data.frame(age, sex, perwt, incwage)
plotData2p <- rxPredict(linMod2, data=plotData2, outData=plotData2)
rxLinePlot(incwage_Pred~age, groups=sex, data=plotData2p)
```



These types of models are often relaxed further by allowing both the slope and intercept to vary by group:

```

linMod3 <- rxLinMod(incwage~sex+sex:age, data = censusWorkers,
  pweights = "perwt", cube=TRUE)
summary(linMod3)
Call:
rxLinMod(formula = incwage ~ sex + sex:age, data = censusWorkers,
  pweights = "perwt", cube = TRUE)

Cube Linear Regression Results for: incwage ~ sex + sex:age
File name:
C:\Program Files\Microsoft\MRO-for-RRE\8.0\R-3.2.2\library\RevoScaleR\SampleData\CensusWorkers.xdf
Probability weights: perwt
Dependent variable(s): incwage
Total independent variables: 4
Number of valid observations: 351121
Number of missing observations: 0

Coefficients:
Estimate Std. Error t value Pr(>|t|)    |   Counts
sex=Male      10783.449   328.871   32.79 2.22e-16 *** | 3866542
sex=Female    15131.422   356.806   42.41 2.22e-16 *** | 3276744
age for sex=Male     814.948    7.888  103.31 2.22e-16 *** |
age for sex=Female   288.876    8.556   33.76 2.22e-16 *** |
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 176300 on 351117 degrees of freedom
Multiple R-squared:  0.07343 (as if intercept included)
Adjusted R-squared:  0.07343
F-statistic:  9276 on 3 and 351117 DF,  p-value: < 2.2e-16
Condition number: 1.1764

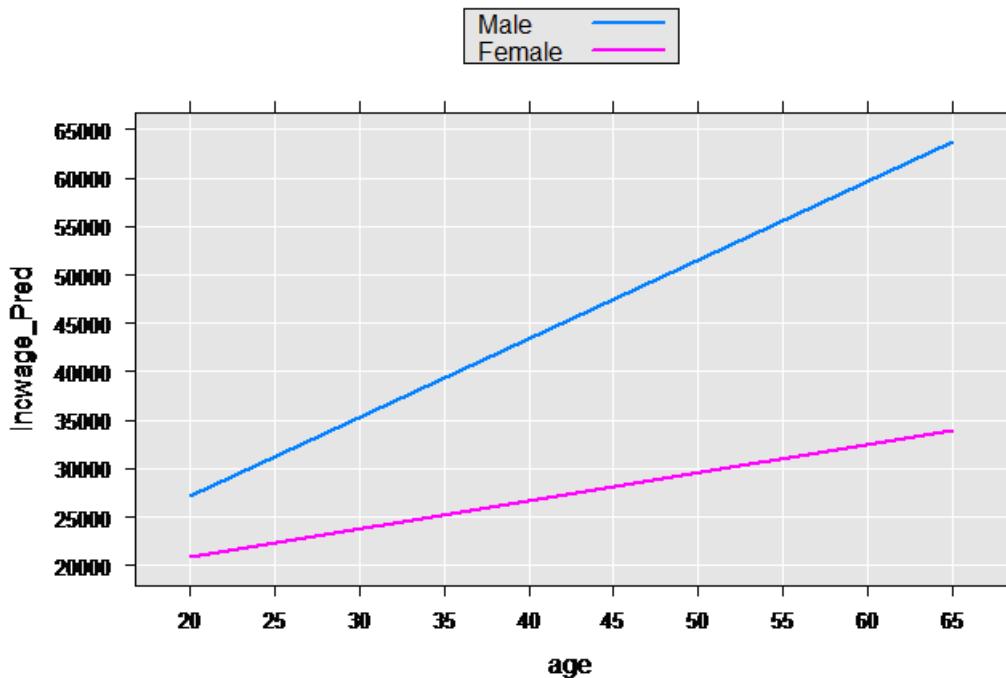
```

Again getting predictions and plotting:

```

plotData3p <- rxPredict(linMod3, data=plotData2, outData=plotData2)
rxLinePlot(incwage_Pred~age, groups=sex, data=plotData3p)

```

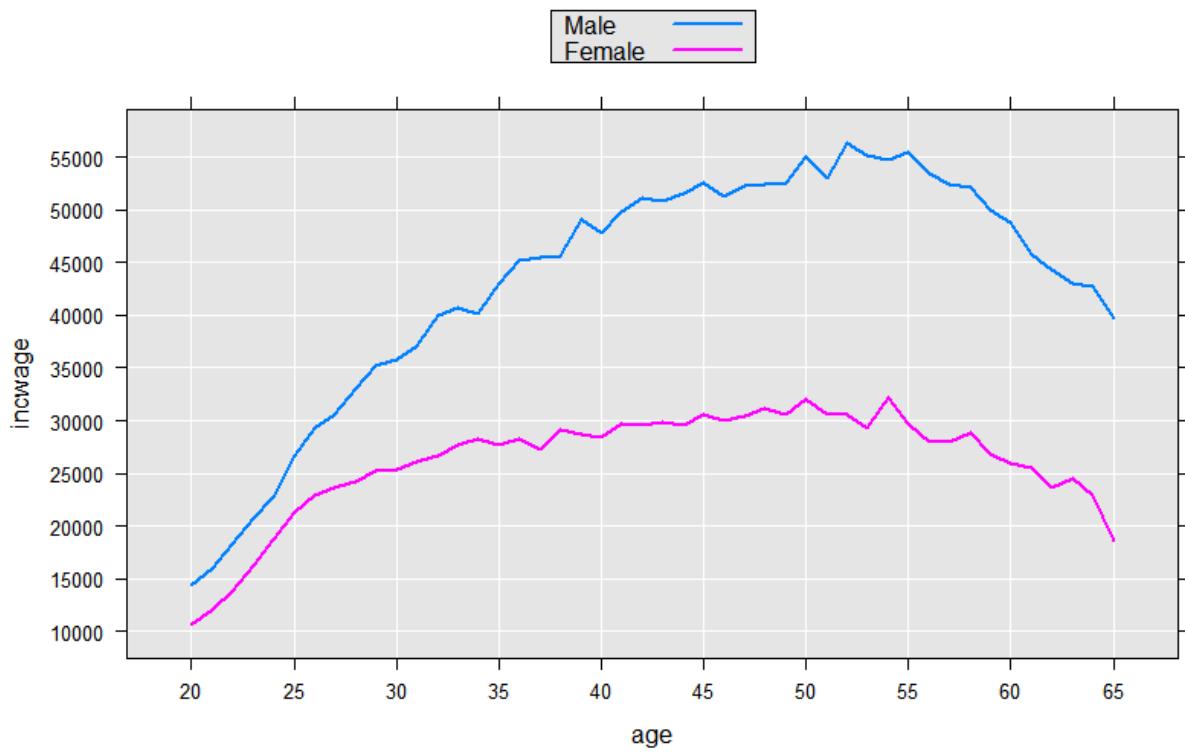


We could continue the process, experimenting with functional forms for age. But, since we have many observations (and therefore many degrees of freedom), we can take advantage of the F() function available in revoScaleR to let the data speak for itself. The F() function creates a factor variable from a numeric variable "on-the-fly", creating a level for every integer value. This allows us to compute and observe the shape of the functional form using a purely dummy variable model:

```
linMod4 <- rxLinMod(incwage~sex:F(age), data=censusWorkers, pweights="perwt",
cube=TRUE)
```

This model estimated a total of 92 coefficients, all for dummy variables representing every age in the range of 20 to 65 for each sex. To visually examine the coefficients we could add observations to our plotData data frame and use rxPredict to compute predicted values for each of the 92 groups represented, but since the model contains only dummy variables in an initial cube term, we can instead use the "counts" data frame returned with the rxLinMod object:

```
plotData4 <- linMod4$countDF
# Convert the age factor variable back to an integer
plotData4$age <- as.integer(levels(plotData4$F.age.))[plotData4$F.age.]
rxLinePlot(incwage~age, groups=sex, data=plotData4)
```



Intercept-Only Models

You may have seen intercept-only models fitted with R's `lm` function, where the model formula is of the form `response ~ 1`. In RevoScaleR these models should be fitted using `rxSummary`, because the intercept-only model simply returns the mean of the response. For example:

```

airlineDF <- rxDataStep(inData =
  file.path(rxGetOption("sampleDataDir"), "AirlineDemoSmall.xdf"))
lm(ArrDelay ~ 1, data = airlineDF)
Call:
lm(formula = ArrDelay ~ 1, data = airlineDF)

Coefficients:
(Intercept)
           11.32

rxSummary(~ ArrDelay, data = airlineDF)
Call:
rxSummary(formula = ~ArrDelay, data = airlineDF)

Summary Statistics Results for: ~ArrDelay
Data: airlineDF
Number of valid observations: 6e+05
Number of missing observations: 0

      Name      Mean     StdDev   Min Max  ValidObs MissingObs
ArrDelay 11.31794 40.68854 -86 1490 582628    17372

```

Fitting Logistic Regression Models using Machine Learning Server

7/12/2022 • 11 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Logistic regression is a standard tool for modeling data with a binary response variable. In R, you fit a logistic regression using the *glm* function, specifying a binomial family and the logit link function. In RevoScaleR, you can use *rxGlm* in the same way (see [Fitting Generalized Linear Models](#)) or you can fit a logistic regression using the optimized *rxLogit* function; because this function is specific to logistic regression, you need not specify a family or link function.

A Simple Logistic Regression Example

As an example, consider the *kyphosis* data set in the *rpart* package. This data set consists of 81 observations of four variables (Age, Number, Kyphosis, Start) in children following corrective spinal surgery; it is used as the initial example of *glm* in the White Book (see [Additional Resources](#) for more information). The variable Kyphosis reports the absence or presence of this deformity.

We can use *rxLogit* to model the probability that kyphosis is present as follows:

```
library(rpart)
rxLogit(Kyphosis ~ Age + Start + Number, data = kyphosis)
```

The following output is returned:

```
Logistic Regression Results for: Kyphosis ~ Age + Start + Number
Data: kyphosis
Dependent variable(s): Kyphosis
Total independent variables: 4
Number of valid observations: 81
Number of missing observations: 0

Coefficients:
  Kyphosis
(Intercept) -2.03693354
Age          0.01093048
Start        -0.20651005
Number       0.41060119
```

The same model can be fit with `glm` (or `rxGlm`) as follows:

```

glm(Kyphosis ~ Age + Start + Number, family = binomial, data = kyphosis)

Call: glm(formula = Kyphosis ~ Age + Start + Number, family = binomial,      data = kyphosis)

Coefficients:
(Intercept)      Age       Start      Number
-2.03693     0.01093    -0.20651     0.41060

Degrees of Freedom: 80 Total (i.e. Null); 77 Residual
Null Deviance: 83.23
Residual Deviance: 61.38 AIC: 69.38

```

Stepwise Logistic Regression

Stepwise logistic regression is an algorithm that helps you determine which variables are most important to a logistic model. You provide a minimal, or lower, model formula and a maximal, or upper, model formula, and using forward selection, backward elimination, or bidirectional search, the algorithm determines the model formula that provides the best fit based on an AIC or significance level selection criterion.

RevoScaleR provides an implementation of stepwise logistic regression that is not constrained by the use of "in-memory" algorithms. Stepwise linear regression in RevoScaleR is implemented by the functions *rxLogit* and *rxStepControl*.

Stepwise logistic regression begins with an initial model of some sort. We can look at the kyphosis data again and start with a simpler model: *Kyphosis ~ Age*:

```

initModel <- rxLogit(Kyphosis ~ Age, data=kyphosis)
initModel

Logistic Regression Results for: Kyphosis ~ Age
Data: kyphosis
Dependent variable(s): Kyphosis
Total independent variables: 2
Number of valid observations: 81
Number of missing observations: 0

Coefficients:
Kyphosis
(Intercept) -1.809351230
Age          0.005441758

```

We can specify a stepwise model using *rxLogit* and *rxStepControl* as follows:

```

KypStepModel <- rxLogit(Kyphosis ~ Age,
  data = kyphosis,
  variableSelection = rxStepControl(method="stepwise",
    scope = ~ Age + Start + Number ))

KypStepModel
  Logistic Regression Results for: Kyphosis ~ Age + Start + Number
  Data: kyphosis
  Dependent variable(s): Kyphosis
  Total independent variables: 4
  Number of valid observations: 81
  Number of missing observations: 0

  Coefficients:
    Kyphosis
  (Intercept) -2.03693354
  Age          0.01093048
  Start        -0.20651005
  Number       0.41060119

```

The methods for variable selection (forward, backward, and stepwise), the definition of model scope, and the available selection criteria are all the same as for stepwise linear regression; see ["Stepwise Variable Selection"](#) and the `rxStepControl` help file for more details.

Plotting Model Coefficients

The ability to save model coefficients using the argument `keepStepCoefs = TRUE` within the `rxStepControl` call and to plot them with the function `rxStepPlot` was described in great detail for stepwise `rxLinMod` in [Fitting Linear Models using RevoScaleR](#). This functionality is also available for stepwise `rxLogit` objects.

Prediction

As described above for linear models, the objects returned by the RevoScaleR model-fitting functions do not include fitted values or residuals. We can obtain them, however, by calling `rxPredict` on our fitted model object, supplying the original data used to fit the model as the data to be used for prediction.

For example, consider the mortgage default example in [Tutorial: Analyzing loan data with RevoScaleR](#). In that example, we used ten input data files to create the data set used to fit the model. But suppose instead we use nine input data files to create the training data set and use the remaining data set for prediction. We can do that as follows (again, remember to modify the first line for your own system):

```

# Logistic Regression Prediction

bigDataDir <- "C:/MRS/Data"
mortCsvDataName <- file.path(bigDataDir, "mortDefault", "mortDefault")
trainingDataFileName <- "mortDefaultTraining"
mortCsv2009 <- paste(mortCsvDataName, "2009.csv", sep = "")
targetDataFileName <- "mortDefault2009.xdf"
ageLevels <- as.character(c(0:40))
yearLevels <- as.character(c(2000:2009))
colInfo <- list(list(name = "houseAge", type = "factor",
  levels = ageLevels), list(name = "year", type = "factor",
  levels = yearLevels))
append= FALSE
for (i in 2000:2008)
{
  importFile <- paste(mortCsvDataName, i, ".csv", sep = "")
  rxImport(inData = importFile, outFile = trainingDataFileName,
  colInfo = colInfo, append = append)
  append = TRUE
}

rxImport(inData = mortCsv2009, outFile = targetDataFileName,
  colInfo = colInfo)

```

We can then fit a logistic regression model to the training data and predict with the prediction data set as follows:

```

logitObj <- rxLogit(default ~ year + creditScore + yearsEmploy + ccDebt,
  data = trainingDataFileName, blocksPerRead = 2, verbose = 1,
  reportProgress=2)
rxPredict(logitObj, data = targetDataFileName,
  outData = targetDataFileName, computeResiduals = TRUE)

```

The `blocksPerRead` argument is ignored if run locally using R Client. [Learn more...](#)

To view the first 30 rows of the output data file, use `rxGetInfo` as follows:

```
rxGetInfo(targetDataFileName, numRows = 30)
```

Prediction Standard Errors and Confidence Intervals

You can use `rxPredict` to obtain prediction standard errors and confidence intervals for models fit with `rxLogit` in the same way as for those fit with `rxLinMod`. The original model must be fit with `covCoef=TRUE`:

```

# Prediction Standard Errors and Confidence Intervals

logitObj2 <- rxLogit(default ~ year + creditScore + yearsEmploy + ccDebt,
  data = trainingDataFileName, blocksPerRead = 2, verbose = 1,
  reportProgress=2, covCoef=TRUE)

```

The `blocksPerRead` argument is ignored if run locally using R Client. [Learn more...](#)

You then specify `computeStdErr=TRUE` to obtain prediction standard errors; if this is TRUE, you can also specify `interval="confidence"` to obtain a confidence interval:

```
rxPredict(logitObj2, data = targetDataFileName,
  outData = targetDataFileName, computeStdErr = TRUE,
  interval = "confidence", overwrite=TRUE)
```

The first ten lines of the file with predictions can be viewed as follows:

```
rxGetInfo(targetDataFileName, numRows=10)

File name: C:\Users\yourname\Documents\MRS\mortDefault2009.xdf
Number of observations: 1e+06
Number of variables: 10
Number of blocks: 2
Compression type: zlib
Data (10 rows starting with row 1):
  creditScore houseAge yearsEmploy ccDebt year default default_Pred
  1           617       20            8   4410 2009      0 6.620773e-06
  2           623       11            7   5609 2009      0 4.610861e-05
  3           758       17            4   7250 2009      0 4.259884e-04
  4           687       22            5   3761 2009      0 3.770789e-06
  5           663       15            6   6746 2009      0 2.312827e-04
  6           676       10            2   7106 2009      0 1.092593e-03
  7           721       23            2   2280 2009      0 8.515912e-07
  8           680       18            7   2831 2009      0 6.011109e-07
  9           734        9            5   3867 2009      0 3.144299e-06
  10          688       16            8   6238 2009      0 5.350031e-05
  default_StdErr default_Lower default_Upper
  1 3.143695e-07 6.032422e-06 7.266507e-06
  2 1.953612e-06 4.243427e-05 5.010109e-05
  3 1.594783e-05 3.958500e-04 4.584203e-04
  4 1.739047e-07 3.444893e-06 4.127516e-06
  5 8.733193e-06 2.147838e-04 2.490486e-04
  6 3.952975e-05 1.017797e-03 1.172880e-03
  7 4.396314e-08 7.696409e-07 9.422675e-07
  8 3.091885e-08 5.434655e-07 6.648706e-07
  9 1.469334e-07 2.869109e-06 3.445883e-06
  10 2.224102e-06 4.931401e-05 5.804197e-05
```

Using ROC Curves to Evaluate Estimated Binary Response Models

A *receiver operating characteristic* (ROC) curve can be used to visually assess binary response models. It plots the *True Positive Rate* (the number of correctly predicted TRUE responses divided by the actual number of TRUE responses) against the *False Positive Rate* (the number of incorrectly predicted TRUE responses divided by the actual number of FALSE responses), calculated at various thresholds. The *True Positive Rate* is the same as the *sensitivity*, and the *False Positive Rate* is equal to one minus the *specificity*.

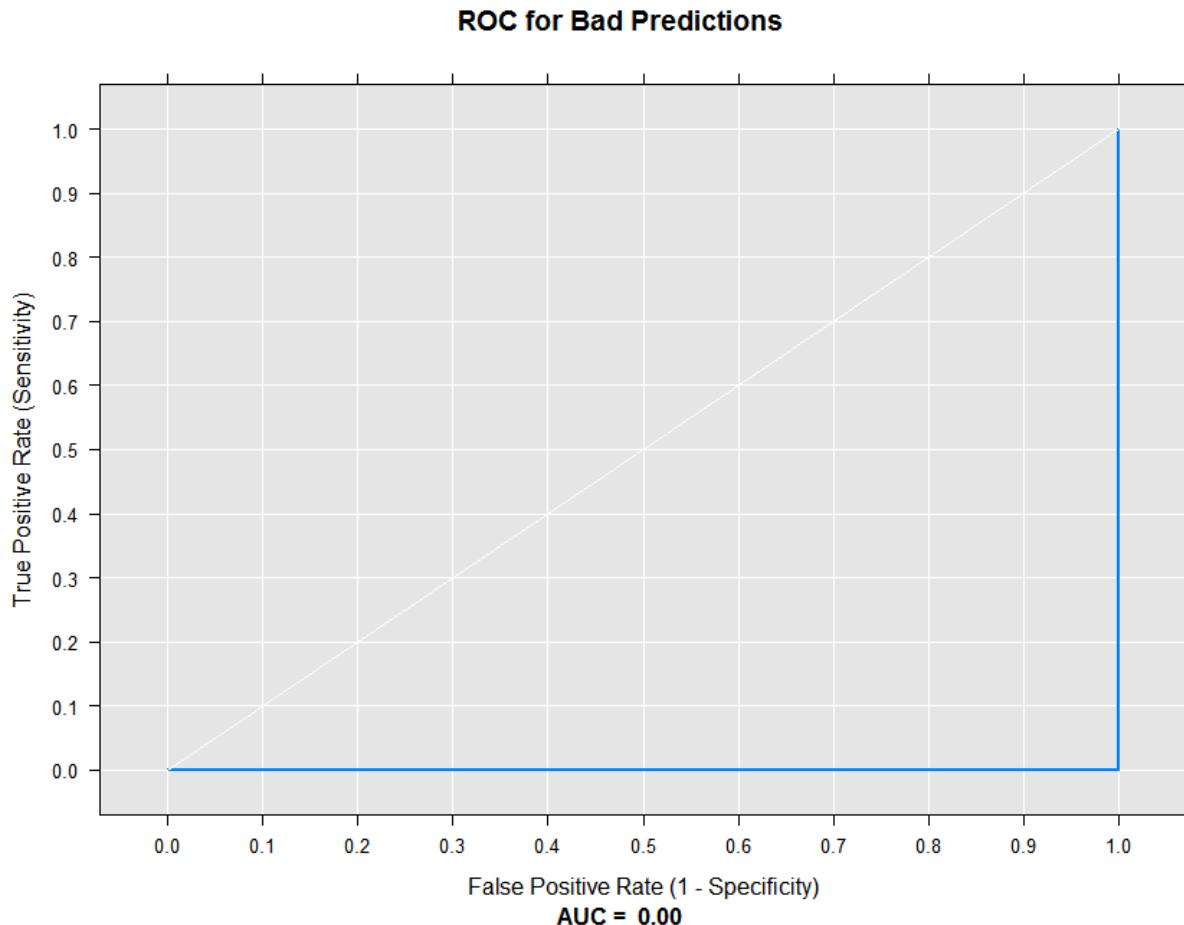
Let's start with a simple example. Suppose we have a data set with 10 observations. The variable *actual* contains the actual responses, or the 'truth'. The variable *badPred* are the predicted responses from a very poor model. The variable *goodPred* contains the predicted responses from a great model.

```
# Using ROC Curves for Binary Response Models

sampleDF <- data.frame(
  actual = c(0, 0, 0, 0, 1, 1, 1, 1, 1),
  badPred = c(.99, .99, .99, .99, .99, .01, .01, .01, .01),
  goodPred = c( .01, .01, .01, .01,.99, .99, .99, .99))
```

We can now call the *rxRocCurve* function to compute the sensitivity and specificity for the 'bad' predictions, and draw the ROC curve. The numBreaks argument indicates the number of breaks to use in determining the thresholds for computing the true and false positive rates.

```
rxRocCurve(actualVarName = "actual", predVarNames = "badPred",
           data = sampleDF, numBreaks = 10, title = "ROC for Bad Predictions")
```

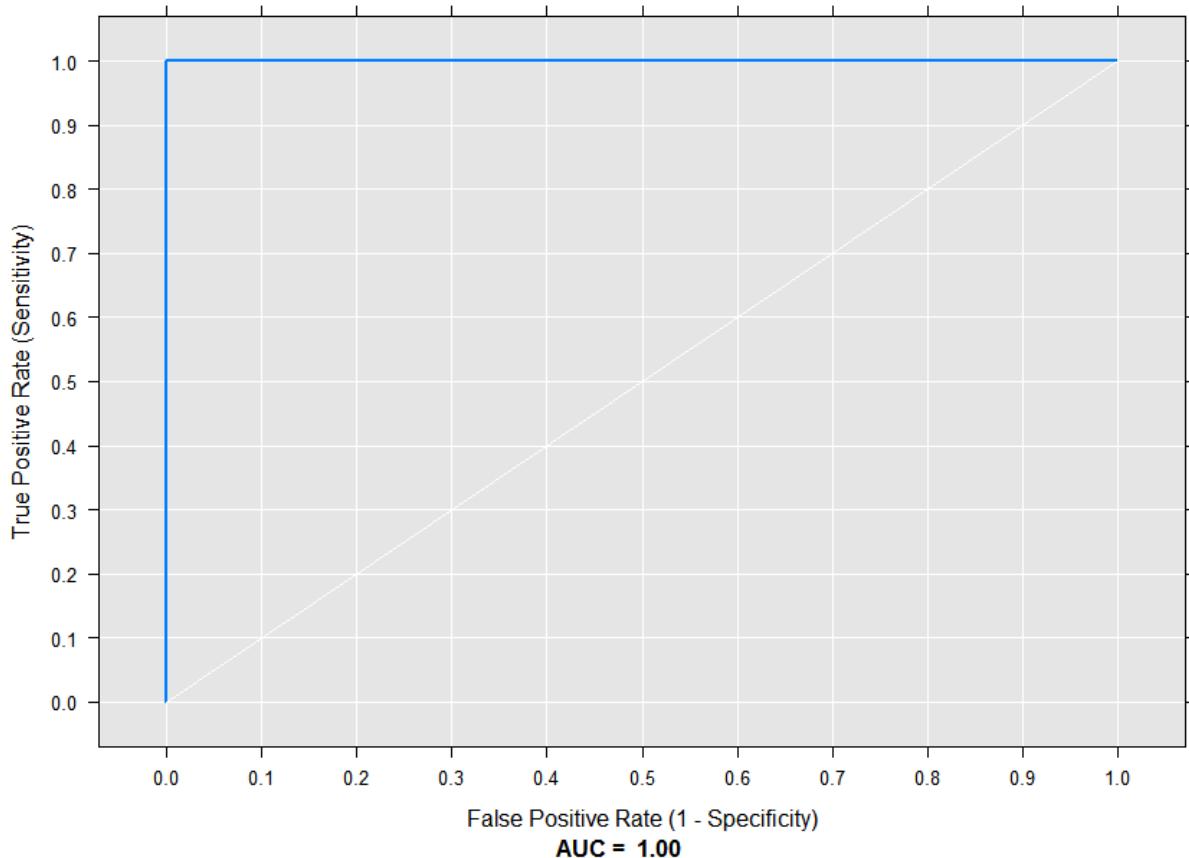


Since all of our predictions are wrong at every threshold, the ROC curve is a flat line at 0. The *Area Under the Curve* (AUC) summary statistic is 0.

At the other extreme, let's draw an ROC curve for our great model:

```
rxRocCurve(actualVarName = "actual", predVarNames = "goodPred",
           data = sampleDF, numBreaks = 10, title = "ROC for Great Predictions")
```

ROC for Great Predictions



With perfect predictions, we see the True Positive Rate is 1 for all thresholds, and the AUC is 1. We'd expect a random guess ROC curve to lie along with white diagonal line.

Now let's use actual model predictions in an ROC curve. We'll use the small mortgage default sample data to estimate a logistic model and then compute predicted values:

The `blocksPerRead` argument is ignored if run locally using R Client. [Learn more...](#)

```
# Using mortDefaultSmall for predictions and an ROC curve

mortXdf <- file.path(rxGetOption("sampleDataDir"), "mortDefaultSmall")
logitOut1 <- rxLogit(default ~ creditScore + yearsEmploy + ccDebt,
  data = mortXdf, blocksPerRead = 5)

predFile <- "mortPred.xdf"

predOutXdf <- rxPredict(modelObject = logitOut1, data = mortXdf,
  writeModelVars = TRUE, predVarNames = "Model1", outData = predFile)
```

Now, let's estimate a different model (with 1 less independent variable), and add the predictions from that model to our output data set:

```
# Estimate a second model without ccDebt
logitOut2 <- rxLogit(default ~ creditScore + yearsEmploy,
  data = predOutXdf, blocksPerRead = 5)

# Add predictions to prediction data file
predOutXdf <- rxPredict(modelObject = logitOut2, data = predOutXdf,
  predVarNames = "Model2")
```

Now we can compute the sensitivity and specificity for both models, using rxRoc:

```
rocOut <- rxRoc(actualVarName = "default",
  predVarNames = c("Model1", "Model2"),
  data = predOutXdf)
rocOut

  threshold predVarName sensitivity specificity
  1      0.00    Model1 1.000000000  0.0000000
  2      0.01    Model1 0.825902335  0.9197118
  3      0.02    Model1 0.647558386  0.9567965
  4      0.03    Model1 0.569002123  0.9721488
  5      0.04    Model1 0.481953291  0.9797647
  6      0.05    Model1 0.437367304  0.9845472
  7      0.06    Model1 0.386411890  0.9877825
  8      0.07    Model1 0.335456476  0.9900130
  9      0.08    Model1 0.305732484  0.9916406
 10     0.09    Model1 0.288747346  0.9930272
 11     0.10    Model1 0.261146497  0.9940520
 12     0.11    Model1 0.237791932  0.9947252
 13     0.12    Model1 0.225053079  0.9953682
 14     0.13    Model1 0.208067941  0.9959107
 15     0.14    Model1 0.197452229  0.9963528
 16     0.15    Model1 0.182590234  0.9967648
 17     0.16    Model1 0.171974522  0.9971064
 18     0.17    Model1 0.161358811  0.9973877
 19     0.18    Model1 0.152866242  0.9975886
 20     0.19    Model1 0.150743100  0.9978298
 21     0.20    Model1 0.144373673  0.9980307
 22     0.21    Model1 0.138004246  0.9982518
 23     0.22    Model1 0.131634820  0.9984527
 24     0.23    Model1 0.131634820  0.9986034
 25     0.24    Model1 0.129511677  0.9987340
 26     0.25    Model1 0.123142251  0.9987843
 27     0.26    Model1 0.116772824  0.9988546
 28     0.27    Model1 0.116772824  0.9989149
 29     0.28    Model1 0.114649682  0.9989752
 30     0.29    Model1 0.108280255  0.9990355
 31     0.30    Model1 0.101910828  0.9991158
 32     0.31    Model1 0.099787686  0.9991661
 33     0.32    Model1 0.091295117  0.9992264
 34     0.33    Model1 0.087048832  0.9992866
 35     0.34    Model1 0.082802548  0.9993469
 36     0.35    Model1 0.080679406  0.9994072
 37     0.36    Model1 0.074309979  0.9994474
 38     0.37    Model1 0.072186837  0.9994675
 39     0.38    Model1 0.070063694  0.9995077
 40     0.39    Model1 0.067940552  0.9995780
 41     0.40    Model1 0.063694268  0.9995881
 42     0.41    Model1 0.063694268  0.9996182
 43     0.42    Model1 0.063694268  0.9996684
 44     0.43    Model1 0.055201699  0.9996986
 45     0.44    Model1 0.050955414  0.9997287
 46     0.45    Model1 0.048832272  0.9997790
 47     0.46    Model1 0.046709130  0.9997790
 48     0.47    Model1 0.042462845  0.9997991
 49     0.48    Model1 0.040339703  0.9998091
 50     0.49    Model1 0.040339703  0.9998292
 51     0.50    Model1 0.040339703  0.9998493
 52     0.51    Model1 0.040339703  0.9998794
 53     0.52    Model1 0.033970276  0.9998895
 54     0.53    Model1 0.031847134  0.9999096
 55     0.54    Model1 0.031847134  0.9999196
 56     0.55    Model1 0.031847134  0.9999297
 57     0.58    Model1 0.029723992  0.9999397
 58     0.59    Model1 0.027600849  0.9999498
 59     0.60    Model1 0.023354565  0.9999598
 60     0.61    Model1 0.016985138  0.9999598
```

```

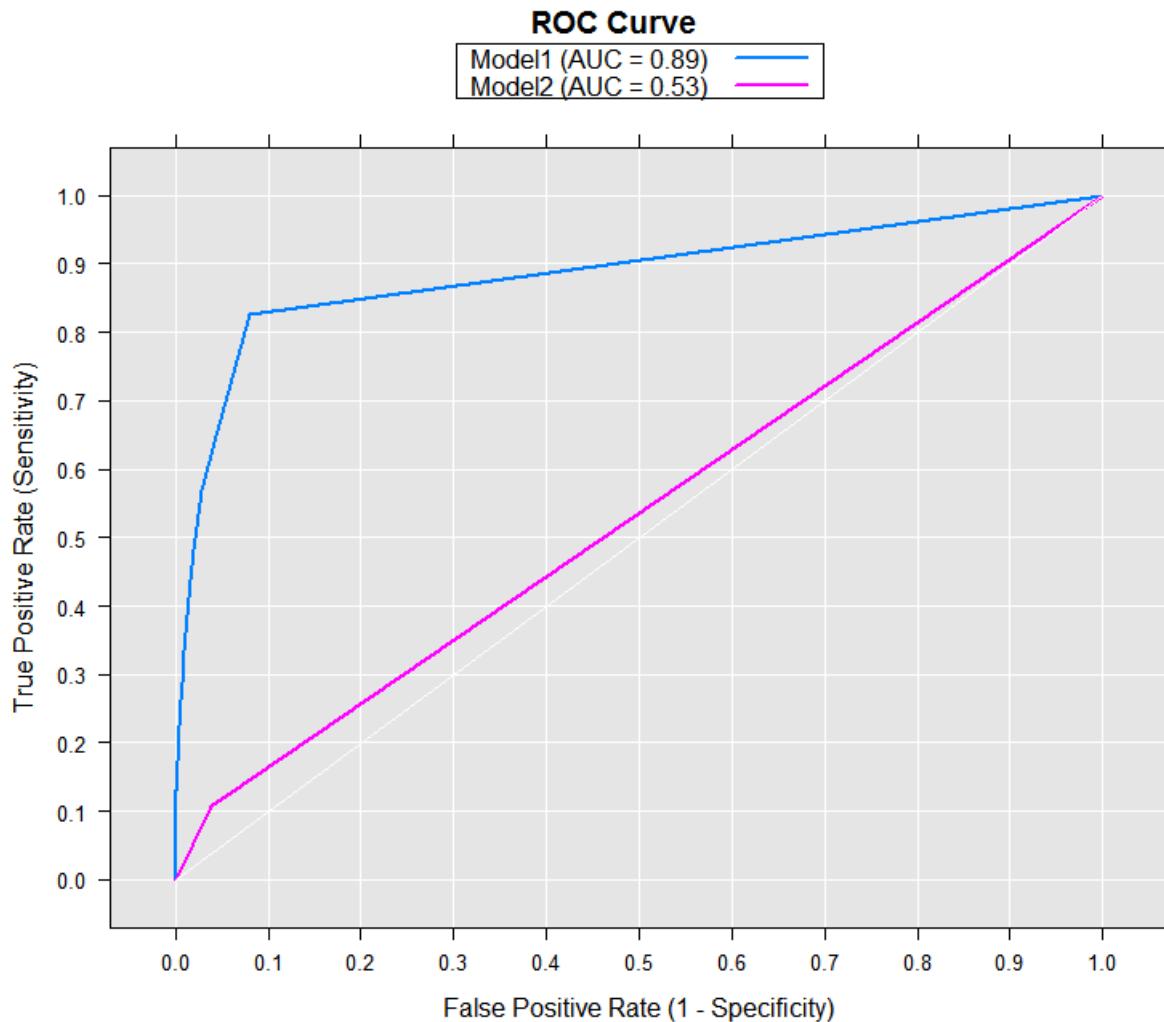
61  0.63  Model1 0.014861996  0.9999598
62  0.65  Model1 0.014861996  0.9999799
63  0.70  Model1 0.014861996  0.9999900
64  0.72  Model1 0.012738854  0.9999900
65  0.74  Model1 0.010615711  0.9999900
66  0.78  Model1 0.010615711  1.0000000
67  0.80  Model1 0.008492569  1.0000000
68  0.83  Model1 0.006369427  1.0000000
69  0.89  Model1 0.004246285  1.0000000
70  0.91  Model1 0.000000000  1.0000000
71  0.00  Model2 1.000000000  0.0000000
72  0.01  Model2 0.108280255  0.9612776
73  0.02  Model2 0.000000000  0.9994474
74  0.03  Model2 0.000000000  1.0000000

```

With the `removeDups` argument set to its default of `TRUE`, rows containing duplicate entries for sensitivity and specificity were removed from the returned data frame. In this case, it results in many fewer rows for Model2 than Model1. We can use the `rxRoc plot` method to render our ROC curve using the computed results.

```
plot(rocOut)
```

The resulting plot shows that the second model is much closer to the "random" diagonal line than the first model.



Generalized Linear Models using RevoScaleR

7/12/2022 • 16 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Generalized linear models (GLM) are a framework for a wide range of analyses. They relax the assumptions for a standard linear model in two ways. First, a functional form can be specified for the conditional mean of the predictor, referred to as the "link" function. Second, you can specify a distribution for the response variable. The rxGlm function in RevoScaleR provides the ability to estimate generalized linear models on large data sets.

The following family/link combinations are implemented in C++ for performance enhancements: binomial/logit, gamma/log, poisson/log, and Tweedie. Other family/link combinations use a combination of C++ and R code.

Any valid R family object that can be used with glm() can be used with rxGlm(), including user-defined. The following table shows all of the supported family/link combinations (in addition to user-defined):

FAMILY	DEFAULT LINK FUNCTION	OTHER AVAILABLE LINK FUNCTIONS
binomial	"logit"	"probit", "cauchit", "log", "cloglog"
gaussian	"identity"	"log", "inverse"
Gamma	"inverse"	"identity", "log"
inverse.gaussian	"1/mu^2"	"inverse", "identity", "log"
poisson	"log"	"identity", "sqrt"
quasi	"identity" with variance = "constant"	"logit", "probit", "cloglog", "inverse", "log", "1/mu^2", "sqrt"
quasibinomial	"logit"	Same as binomial, but dispersion parameter not fixed at one
quasipoisson	"log"	Same as poisson, but dispersion parameter not fixed at one
rxTweedie	requires arguments instead of link function	

A Simple Example Using the Poisson Family

The Poisson family is used to estimate models of count data. Examples from the literature include the following types of response variables:

- Number of drinks on a Saturday night
- Number of bacterial colonies in a Petri dish
- Number of children born to married women

- Number of credit cards a person has

We'll start with a simple example from Kabacoff's **R in Action** book, using data provided with the **robust** R package. The data are from a placebo-controlled clinical trial of 59 epileptics. Patients with partial seizures were enrolled in a randomized clinical trial of the anti-epileptic drug, pro gabide. Counts of epileptic seizures were recorded during the trial. The data set also includes a baseline 8-week seizure count and the age of the patient.

To access this data, first make sure the **robust** package is installed, then use the **data** command to load the data frame:

```
# A Simple Example Using the Poisson Family

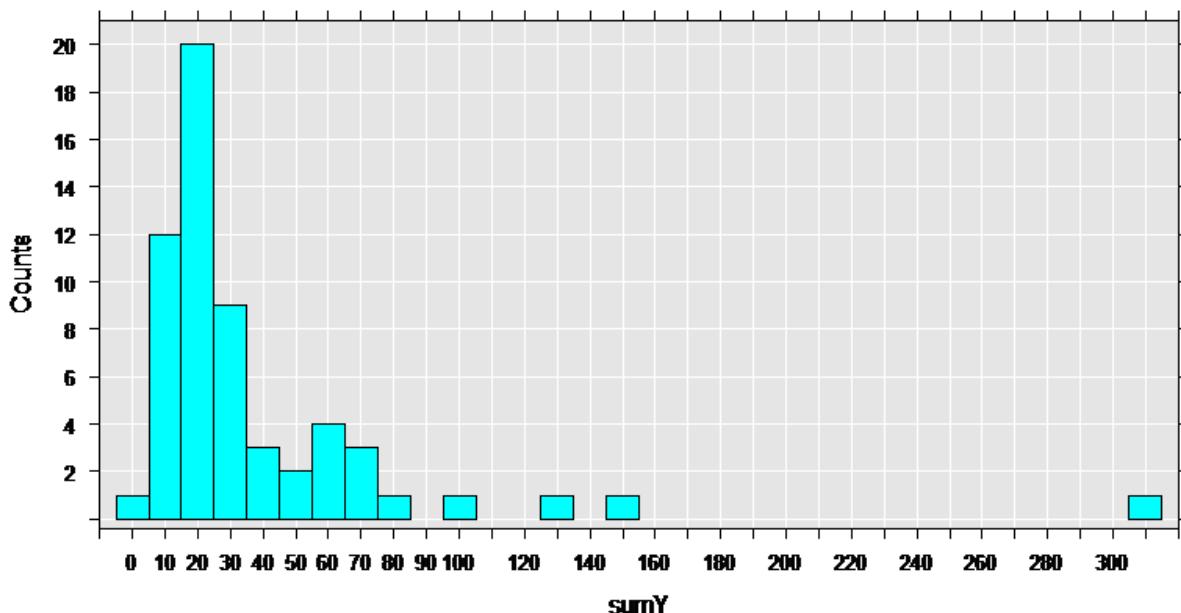
if ("robust" %in% .packages()){
  data(breslow.dat, package = "robust")
```

First, let's get some basic information on the data set, then draw a histogram of the **sumY** variable, containing the total count of seizures during the trial.

```
rxGetInfo(breslow.dat, getVarInfo = TRUE)
rxHistogram(~sumY, numBreaks = 25, data = breslow.dat)
```

The data set has 59 observations, and 12 variables. The variables of interest are **Base**, **Age**, **Trt**, and **sumY**.

```
Data frame: breslow.dat
Number of observations: 59
Number of variables: 12
Variable information:
Var 1: ID, Type: integer, Low/High: (101, 238)
Var 2: Y1, Type: integer, Low/High: (0, 102)
Var 3: Y2, Type: integer, Low/High: (0, 65)
Var 4: Y3, Type: integer, Low/High: (0, 76)
Var 5: Y4, Type: integer, Low/High: (0, 63)
Var 6: Base, Type: integer, Low/High: (6, 151)
Var 7: Age, Type: integer, Low/High: (18, 42)
Var 8: Trt
  2 factor levels: placebo pro gabide
Var 9: Ysum, Type: integer, Low/High: (0, 302)
Var 10: sumY, Type: integer, Low/High: (0, 302)
Var 11: Age10, Type: numeric, Low/High: (1.8000, 4.2000)
Var 12: Base4, Type: numeric, Low/High: (1.5000, 37.7500)
```



To estimate a model with *sumY* as the response variable and the *Base* number of seizures, *Age*, and the treatment as explanatory variables, we can use *rxGlm*. A benefit to using *rxGlm* is that the code scales for use with a much bigger data set.

```
myGlm <- rxGlm(sumY ~ Base + Age + Trt, dropFirst = TRUE,
  data = breslow.dat, family = poisson())
summary(myGlm)
```

Results in:

```
Call:
rxGlm(formula = sumY ~ Base + Age + Trt, data = breslow.dat,
family = poisson(), dropFirst = TRUE)

Generalized Linear Model Results for: sumY ~ Base + Age + Trt
Data: breslow.dat
Dependent variable(s): sumY
Total independent variables: 5 (Including number dropped: 1)
Number of valid observations: 59
Number of missing observations: 0
Family-link: poisson-log

Residual deviance: 559.4437 (on 55 degrees of freedom)

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 1.9488259 0.1356191 14.370 2.22e-16 ***
Base        0.0226517 0.0005093 44.476 2.22e-16 ***
Age         0.0227401 0.0040240  5.651 5.85e-07 ***
Trt=placebo Dropped   Dropped Dropped Dropped
Trt=progabide -0.1527009 0.0478051 -3.194  0.00232 **
---
Signif. codes: 0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

Condition number of final variance-covariance matrix: 3.3382
Number of iterations: 4
```

To interpret the coefficients, it is sometimes useful to transform them back to the original scale of the dependent

variable. In this case:

```
exp(coef(myGlm))

(Intercept)          Base           Age   Trt=placebo Trt=progabide
7.0204403      1.0229102    1.0230007        NA      0.8583864
```

This suggests that, controlling for the base number of seizures and age, people taking progabide during the trial had 85% of the expected number seizures compared with people who didn't.

A common method of checking for overdispersion is to calculate the ratio of the residual deviance with the degrees of freedom. This should be about 1 to fit the assumptions of the model.

```
myGlm$deviance/myGlm$df[2]

[1] 10.1717
```

We can see that the ratio is well above one.

The quasi-poisson family can be used to handle over-dispersion. In this case, instead of assuming that the variance and mean are one, a relationship is estimated from the data:

```
myGlm1 <- rxGlm(sumY ~ Base + Age + Trt, dropFirst = TRUE,
  data = breslow.dat, family = quasipoisson())

summary(myGlm1)
} # End of if for robust package

Call:
rxGlm(formula = sumY ~ Base + Age + Trt, data = breslow.dat,
  family = quasipoisson(), dropFirst = TRUE)

Generalized Linear Model Results for: sumY ~ Base + Age + Trt
Data: breslow.dat
Dependent variable(s): sumY
Total independent variables: 5 (Including number dropped: 1)
Number of valid observations: 59
Number of missing observations: 0
Family-link: quasipoisson-log

Residual deviance: 559.4437 (on 55 degrees of freedom)

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 1.948826 0.465091 4.190 0.000102 ***
Base        0.022652 0.001747 12.969 2.22e-16 ***
Age         0.022740 0.013800 1.648 0.105085
Trt=placebo Dropped   Dropped Dropped Dropped
Trt=progabide -0.152701 0.163943 -0.931 0.355702
---
Signif. codes: 0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for quasipoisson family taken to be 11.76075)

Condition number of final variance-covariance matrix: 3.3382
Number of iterations: 4
```

Notice that the coefficients are the same as when using the poisson family, but that the standard errors are larger. The effect of the treatment is no longer significant.

An Example Using the Gamma Family

The Gamma family is used with data containing positive values with a positive skew. A classic example is estimating the value of auto insurance claims. Using the sample *claims.xdf* data set:

```
# An Example Using the Gamma Family

claimsXdf <- file.path(rxGetOption("sampleDataDir"), "claims.xdf")

claimsGlm <- rxGlm(cost ~ age + car.age + type, family = Gamma,
  dropFirst = TRUE, data = claimsXdf)
summary(claimsGlm)

Call:
rxGlm(formula = cost ~ age + car.age + type, data = claimsXdf,
  family = Gamma, dropFirst = TRUE)

Generalized Linear Model Results for: cost ~ age + car.age + type
File name:
C:\Program Files\Microsoft\MRO-for-RRE\8.0\R-3.2.2\library\RevoScaleR\SampleData\claims.xdf
Dependent variable(s): cost
Total independent variables: 17 (Including number dropped: 3)
Number of valid observations: 123
Number of missing observations: 5
Family-link: Gamma-inverse

Residual deviance: 15.6397 (on 109 degrees of freedom)

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.0032807 0.0005126 6.400 4.02e-09 ***
age=17-20 Dropped Dropped Dropped Dropped
age=21-24 0.0006593 0.0005471 1.205 0.230843
age=25-29 0.0003911 0.0005183 0.755 0.452114
age=30-34 0.0012388 0.0005982 2.071 0.040720 *
age=35-39 0.0017152 0.0006514 2.633 0.009685 **
age=40-49 0.0012649 0.0006007 2.106 0.037516 *
age=50-59 0.0002863 0.0005087 0.563 0.574771
age=60+ 0.0013006 0.0006041 2.153 0.033519 *
car.age=0-3 Dropped Dropped Dropped Dropped
car.age=4-7 0.0003444 0.0003535 0.974 0.332120
car.age=8-9 0.0011005 0.0004161 2.645 0.009375 **
car.age=10+ 0.0034437 0.0006107 5.639 1.36e-07 ***
type=A Dropped Dropped Dropped Dropped
type=B -0.0004443 0.0004880 -0.911 0.364508
type=C -0.0004189 0.0004912 -0.853 0.395668
type=D -0.0016209 0.0004344 -3.732 0.000304 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for Gamma family taken to be 0.1785316)

Condition number of final variance-covariance matrix: 12.4648
Number of iterations: 4
```

But, these estimates are conditional on the fact that a claim was made.

An Example Using the Tweedie Family

The Tweedie family of distributions provides flexible models for estimation. The power parameter *var.power* determines the shape of the distribution, with familiar models as special cases: if *var.power* is set to 0, Tweedie is a normal distribution; when set to 1, it is Poisson; when 2, it is Gamma; when 3, it is inverse Gaussian. If *var.power* is between 1 and 2, it is a compound Poisson distribution and is appropriate for positive data that also contains exact zeros, for example, insurance claims data, rainfall data, or fish-catch data. If *var.power* is greater than 2, it is appropriate for positive data.

In this example, we use a subsample from the 5% sample of the U.S. 2000 census. We consider the annual cost

of property insurance for heads of household ages 21 through 89, and its relationship to age, sex, and region. A variable "perwt" in the data set represents the probability weight for that observation. First, to create the subsample (specify the correct data path for your downloaded data):

```
bigDataDir = "C:/MRS/Data"
bigCensusData <- file.path(bigDataDir, "Census5PCT2000.xdf")
propinFile <- "CensusPropertyIns.xdf"

propinDS <- rxDataStep(inData = bigCensusData, outFile = propinFile,
  rowSelection = (related == 'Head/Householder') & (age > 20) & (age < 90),
  varsToKeep = c("propinsr", "age", "sex", "region", "perwt"),
  blocksPerRead = 10, overwrite = TRUE)
rxGetInfo(propinDS)

File name: C:\YourWorkingDir\CensusPropertyIns.xdf
Number of observations: 5175270
Number of variables: 5
Number of blocks: 10
Compression type: zlib
```

The `blocksPerRead` argument is ignored when run locally using R Client. [Learn more...](#)

An Xdf data source representing the new data file is returned. The new data file has over 5 million observations.

Let's do one more step in data cleaning. The variable *region* has some long factor level character strings, and it also has a number of levels for which there are no observations. We can see this using *rxSummary*.

```
rxSummary(~region, data = propinDS)

Call:
rxSummary(formula = ~region, data = propinDS)

Summary Statistics Results for: ~region
File name: C:\YourWorkingDir\CensusPropertyIns.xdf
Number of valid observations: 5175270

Category Counts for region
Number of categories: 17
Number of valid observations: 5175270
Number of missing observations: 0

  region          Counts
  New England Division      265372
  Middle Atlantic Division    734585
  Mixed Northeast Divisions (1970 Metro)      0
  East North Central Div.      847367
  West North Central Div.      366417
  Mixed Midwest Divisions (1970 Metro)      0
  South Atlantic Division      981614
  East South Central Div.      324003
  West South Central Div.      553425
  Mixed Southern Divisions (1970 Metro)      0
  Mountain Division          328940
  Pacific Division            773547
  Mixed Western Divisions (1970 Metro)      0
  Military/Military reservations      0
  PUMA boundaries cross state lines-1% sample      0
  State not identified        0
  Inter-regional county group (1970 Metro samples)      0
```

We can use the *rxFactors* function rename and reduce the number of levels:

```

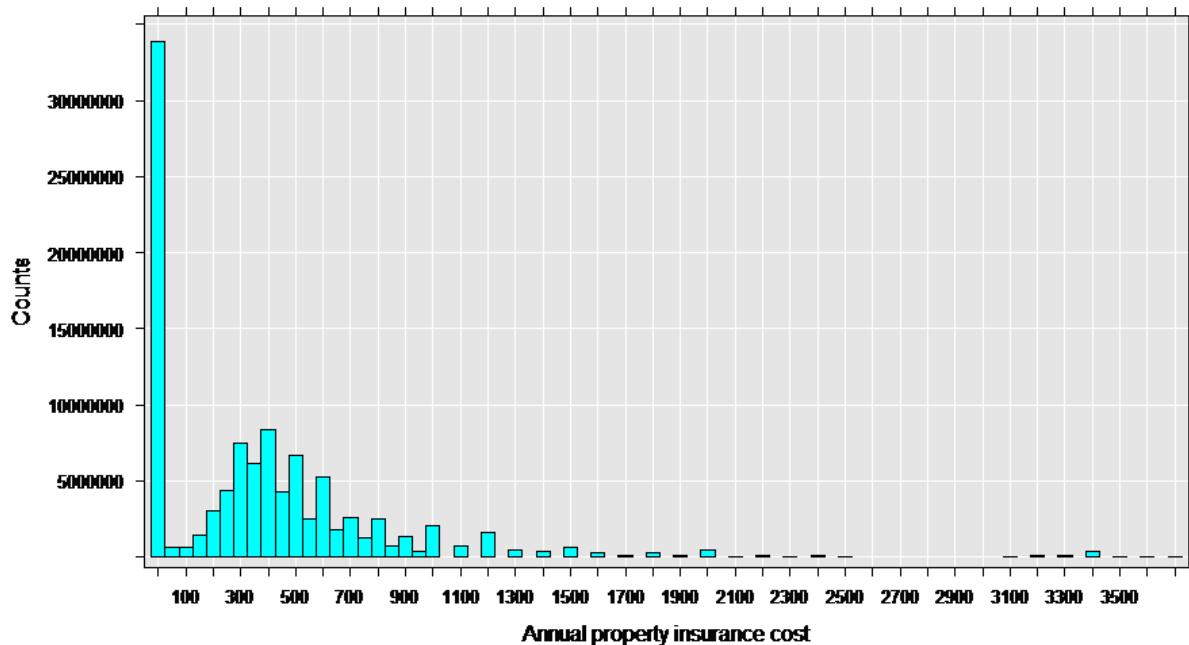
regionLevels <- list( "New England" = "New England Division",
  "Middle Atlantic" = "Middle Atlantic Division",
  "East North Central" = "East North Central Div.",
  "West North Central" = "West North Central Div.",
  "South Atlantic" = "South Atlantic Division",
  "East South Central" = "East South Central Div.",
  "West South Central" = "West South Central Div.",
  "Mountain" = "Mountain Division",
  "Pacific" = "Pacific Division")

rxFactors(inData = propinDS, outFile = propinDS,
  factorInfo = list(region = list(newLevels = regionLevels,
    otherLevel = "Other")),
  overwrite = TRUE)

```

As a first step to analysis, let's look at a histogram of the property insurance cost:

```
rxHistogram(~propinsr, data = propinDS, pweights = "perwt")
```



This data appears to be a good match for the Tweedie family with a variance power parameter between 1 and 2, since it has a "clump" of exact zeros in addition to a distribution of positive values.

We can estimate the parameters using *rxGlm*, setting the *var.power* argument to 1.5. As explanatory variables we'll use sex, an "on-the-fly" factor variable with a level for each age, and region:

```

propinGlm <- rxGlm(propinsr~sex + F(age) + region,
  pweights = "perwt", data = propinDS,
  family = rxTweedie(var.power = 1.5), dropFirst = TRUE)
summary(propinGlm)

Call:
rxGlm(formula = propinsr ~ sex + F(age) + region, data = propinDS,
family = rxTweedie(var.power = 1.5), pweights = "perwt",
dropFirst = TRUE)

Generalized Linear Model Results for: propinsr ~ sex + F(age) + region
File name: C:\YourWorkingDir\CensusPropertyIns.xdf
Probability weights: perwt
Dependent variable(s): propinsr

```

Total independent variables: 82 (Including number dropped: 4)

Number of valid observations: 5175270

Number of missing observations: 0

Family-link: Tweedie-mu^-0.5

Residual deviance: 3292809839.3236 (on 5175192 degrees of freedom)

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	1.231e-01	5.893e-04	208.961	2.22e-16 ***
sex=Male	Dropped	Dropped	Dropped	Dropped
sex=Female	9.026e-03	3.164e-05	285.305	2.22e-16 ***
F_age=21	Dropped	Dropped	Dropped	Dropped
F_age=22	-9.208e-03	7.523e-04	-12.240	2.22e-16 ***
F_age=23	-1.980e-02	6.966e-04	-28.430	2.22e-16 ***
F_age=24	-2.856e-02	6.648e-04	-42.955	2.22e-16 ***
F_age=25	-3.652e-02	6.432e-04	-56.776	2.22e-16 ***
F_age=26	-4.371e-02	6.289e-04	-69.500	2.22e-16 ***
F_age=27	-4.894e-02	6.182e-04	-79.162	2.22e-16 ***
F_age=28	-5.398e-02	6.099e-04	-88.506	2.22e-16 ***
F_age=29	-5.787e-02	6.043e-04	-95.749	2.22e-16 ***
F_age=30	-6.064e-02	6.020e-04	-100.716	2.22e-16 ***
F_age=31	-6.336e-02	6.004e-04	-105.522	2.22e-16 ***
F_age=32	-6.526e-02	5.991e-04	-108.933	2.22e-16 ***
F_age=33	-6.721e-02	5.975e-04	-112.489	2.22e-16 ***
F_age=34	-6.854e-02	5.962e-04	-114.948	2.22e-16 ***
F_age=35	-6.942e-02	5.949e-04	-116.688	2.22e-16 ***
F_age=36	-7.090e-02	5.941e-04	-119.342	2.22e-16 ***
F_age=37	-7.184e-02	5.936e-04	-121.023	2.22e-16 ***
F_age=38	-7.265e-02	5.931e-04	-122.498	2.22e-16 ***
F_age=39	-7.354e-02	5.926e-04	-124.090	2.22e-16 ***
F_age=40	-7.401e-02	5.923e-04	-124.954	2.22e-16 ***
F_age=41	-7.462e-02	5.923e-04	-125.994	2.22e-16 ***
F_age=42	-7.508e-02	5.920e-04	-126.819	2.22e-16 ***
F_age=43	-7.568e-02	5.920e-04	-127.846	2.22e-16 ***
F_age=44	-7.597e-02	5.919e-04	-128.344	2.22e-16 ***
F_age=45	-7.642e-02	5.918e-04	-129.139	2.22e-16 ***
F_age=46	-7.693e-02	5.919e-04	-129.973	2.22e-16 ***
F_age=47	-7.727e-02	5.918e-04	-130.564	2.22e-16 ***
F_age=48	-7.749e-02	5.919e-04	-130.927	2.22e-16 ***
F_age=49	-7.783e-02	5.919e-04	-131.488	2.22e-16 ***
F_age=50	-7.809e-02	5.919e-04	-131.941	2.22e-16 ***
F_age=51	-7.853e-02	5.919e-04	-132.678	2.22e-16 ***
F_age=52	-7.888e-02	5.916e-04	-133.326	2.22e-16 ***
F_age=53	-7.919e-02	5.916e-04	-133.859	2.22e-16 ***
F_age=54	-7.909e-02	5.931e-04	-133.348	2.22e-16 ***
F_age=55	-7.938e-02	5.929e-04	-133.873	2.22e-16 ***
F_age=56	-7.930e-02	5.929e-04	-133.751	2.22e-16 ***
F_age=57	-7.959e-02	5.928e-04	-134.276	2.22e-16 ***
F_age=58	-7.935e-02	5.937e-04	-133.644	2.22e-16 ***
F_age=59	-7.923e-02	5.942e-04	-133.336	2.22e-16 ***
F_age=60	-7.894e-02	5.946e-04	-132.753	2.22e-16 ***
F_age=61	-7.917e-02	5.947e-04	-133.122	2.22e-16 ***
F_age=62	-7.912e-02	5.949e-04	-133.003	2.22e-16 ***
F_age=63	-7.904e-02	5.954e-04	-132.746	2.22e-16 ***
F_age=64	-7.886e-02	5.956e-04	-132.405	2.22e-16 ***
F_age=65	-7.878e-02	5.952e-04	-132.359	2.22e-16 ***
F_age=66	-7.871e-02	5.961e-04	-132.031	2.22e-16 ***
F_age=67	-7.864e-02	5.963e-04	-131.869	2.22e-16 ***
F_age=68	-7.861e-02	5.966e-04	-131.766	2.22e-16 ***
F_age=69	-7.845e-02	5.967e-04	-131.490	2.22e-16 ***
F_age=70	-7.861e-02	5.965e-04	-131.790	2.22e-16 ***
F_age=71	-7.856e-02	5.970e-04	-131.600	2.22e-16 ***
F_age=72	-7.850e-02	5.971e-04	-131.460	2.22e-16 ***
F_age=73	-7.813e-02	5.977e-04	-130.714	2.22e-16 ***
F_age=74	-7.818e-02	5.981e-04	-130.722	2.22e-16 ***
F_age=75	-7.800e-02	5.986e-04	-130.302	2.22e-16 ***
F_age=76	-7.781e-02	5.993e-04	-129.825	2.22e-16 ***
F_age=77	-7.763e-02	6.002e-04	-129.342	2.22e-16 ***

```

F_age=78          -7.735e-02  6.009e-04 -128.728 2.22e-16 ***
F_age=79          -7.724e-02  6.024e-04 -128.221 2.22e-16 ***
F_age=80          -7.646e-02  6.045e-04 -126.495 2.22e-16 ***
F_age=81          -7.651e-02  6.060e-04 -126.244 2.22e-16 ***
F_age=82          -7.643e-02  6.081e-04 -125.693 2.22e-16 ***
F_age=83          -7.600e-02  6.109e-04 -124.411 2.22e-16 ***
F_age=84          -7.546e-02  6.145e-04 -122.798 2.22e-16 ***
F_age=85          -7.529e-02  6.183e-04 -121.775 2.22e-16 ***
F_age=86          -7.441e-02  6.259e-04 -118.882 2.22e-16 ***
F_age=87          -7.422e-02  6.324e-04 -117.363 2.22e-16 ***
F_age=88          -7.339e-02  6.463e-04 -113.553 2.22e-16 ***
F_age=89          -7.310e-02  6.569e-04 -111.284 2.22e-16 ***
region>New England Dropped   Dropped   Dropped   Dropped
region>Middle Atlantic  1.710e-03  6.893e-05  24.807 2.22e-16 ***
region>East North Central 3.552e-03  6.867e-05  51.723 2.22e-16 ***
region>West North Central 4.200e-04  7.697e-05  5.457 4.83e-08 ***
region>South Atlantic    -1.227e-03  6.521e-05 -18.821 2.22e-16 ***
region>East South Central -7.894e-04  7.793e-05 -10.130 2.22e-16 ***
region>West South Central -5.857e-03  6.732e-05 -87.011 2.22e-16 ***
region>Mountain           1.821e-03  8.060e-05  22.596 2.22e-16 ***
region>Pacific             -5.990e-04  6.732e-05 -8.897 2.22e-16 ***
region>Other               Dropped   Dropped   Dropped   Dropped
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

(Dispersion parameter for Tweedie family taken to be 546.4888)

Condition number of final variance-covariance matrix: 5980.277
Number of iterations: 8

A good way to begin examining the results of the estimated model is to look at predicted values for given explanatory characteristics. For example, let's create a prediction data set for the South Atlantic region for all ages and sexes:

```

# Get the region factor levels
varInfo <- rxGetVarInfo(propinDS)
regionLabels <- varInfo$region$levels

# Create a prediction data set for region 5, all ages, both sexes
region <- factor(rep(5, times=138), levels = 1:10, labels = regionLabels)
age <- c(21:89, 21:89)
sex <- factor(c(rep(1, times=69), rep(2, times=69)),
  levels = 1:2,
  labels = c("Male", "Female"))
predData <- data.frame(age, sex, region)

```

Now we'll use that as a basis for a similar prediction data set for the Middle Atlantic region:

```

# Create a prediction data set for region 2, all ages, both sexes
predData2 <- predData
predData2$region <- factor(rep(2, times=138), levels = 1:10,
  labels = varInfo$region$levels)

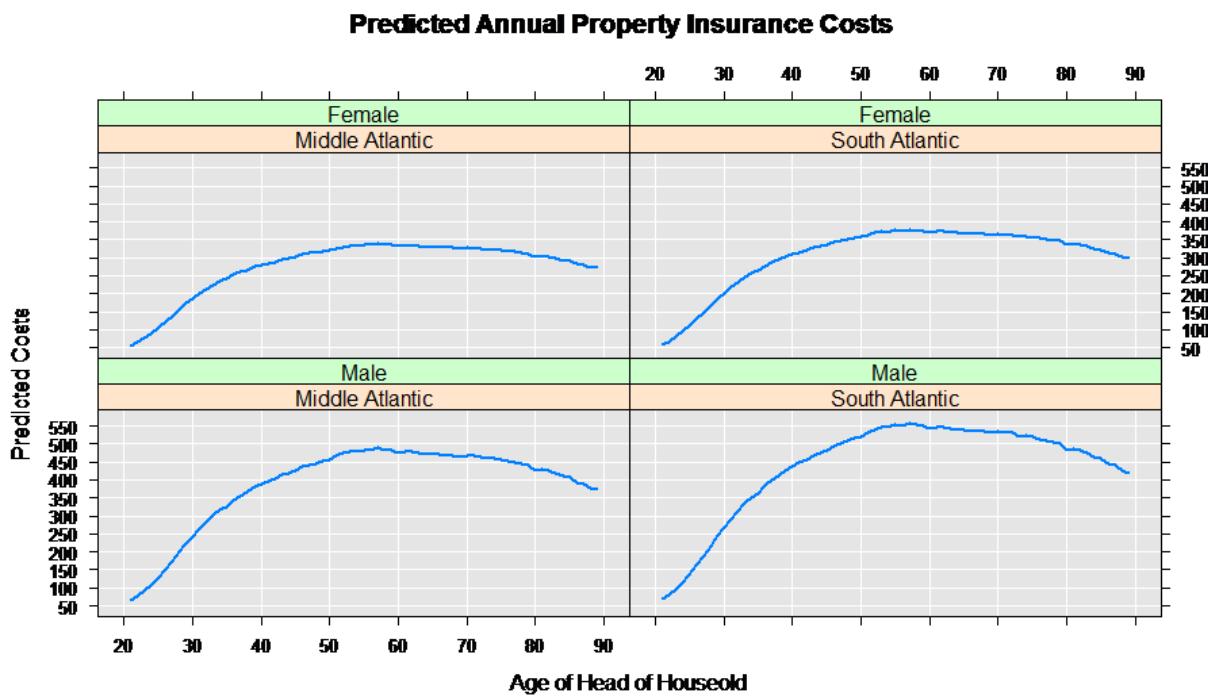
```

Next we combine the two data sets, and compute the predicted values for annual property insurance cost using our estimated *rxGlm* model:

```

predData$predicted <- outData$propinsr_Pred
rxLinePlot( predicted ~age|region+sex, data = predData,
  title = "Predicted Annual Property Insurance Costs",
  xTitle = "Age of Head of Household",
  yTitle = "Predicted Costs")

```



Stepwise Generalized Linear Models

Stepwise generalized linear models help you determine which variables are most important to include in the model. You provide a minimal, or lower, model formula and a maximal, or upper, model formula. Using forward selection, backward elimination, or bidirectional search, the algorithm determines the model formula that provides the best fit based on an AIC selection criterion or a significance level criterion.

As an example, consider again the Gamma family model from earlier in this article:

```
claimsXdf <- file.path(rxGetOption("sampleDataDir"), "claims.xdf")
claimsGlm <- rxGlm(cost ~ age + car.age + type, family = Gamma,
  dropFirst = TRUE, data = claimsXdf)
summary(claimsGlm)
```

We can recast this model as a stepwise model by specifying a variableSelection argument using the rxStepControl function to provide our stepwise arguments:

```

claimsGlmStep <- rxGlm(cost ~ age, family = Gamma, dropFirst=TRUE,
  data=claimsXdf, variableSelection =
  rxStepControl(scope = ~ age + car.age + type ))
summary(claimsGlmStep)

Call:
rxGlm(formula = cost ~ age, data = claimsXdf, family = Gamma,
variableSelection = rxStepControl(scope = ~age + car.age +
type), dropFirst = TRUE)

Generalized Linear Model Results for: cost ~ car.age + type
File name:
C:\Program Files\Microsoft\MRO-for-RRE\8.0\R-3.2.2\library\RevoScaleR\SampleData\claims.xdf
Dependent variable(s): cost
Total independent variables: 9 (Including number dropped: 2)
Number of valid observations: 123
Number of missing observations: 5
Family-link: Gamma-inverse

Residual deviance: 18.0433 (on 116 degrees of freedom)

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.0040354 0.0004661 8.657 3.24e-14 ***
car.age=0-3 Dropped Dropped Dropped Dropped
car.age=4-7 0.0003568 0.0004037 0.884 0.37868
car.age=8-9 0.0011825 0.0004688 2.522 0.01302 *
car.age=10+ 0.0035478 0.0006853 5.177 9.57e-07 ***
type=A Dropped Dropped Dropped Dropped
type=B -0.0004512 0.0005519 -0.818 0.41528
type=C -0.0004135 0.0005558 -0.744 0.45837
type=D -0.0016307 0.0004923 -3.313 0.00123 **
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for Gamma family taken to be 0.2249467)

Condition number of final variance-covariance matrix: 9.1775
Number of iterations: 5

```

We see that in the stepwise model fit, age no longer appears in the final model.

Plotting Model Coefficients

The ability to save model coefficients using the argument `keepStepCoefs = TRUE` within the `rxStepControl` call, and to plot them with the function `rxStepPlot` was described in great detail for stepwise `rxLinMod` in [Fitting Linear Models using RevoScaleR](#). This functionality is also available for stepwise `rxGLM` objects.

Estimating Decision Tree Models

7/12/2022 • 19 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

The *rxDTree* function in RevoScaleR fits tree-based models using a binning-based recursive partitioning algorithm. The resulting model is similar to that produced by the recommended R package *rpart*. Both classification-type trees and regression-type trees are supported; as with *rpart*, the difference is determined by the nature of the response variable: a factor response generates a classification tree; a numeric response generates a regression tree.

The *rxDTree* Algorithm

Decision trees are effective algorithms widely used for classification and regression. Building a decision tree generally requires that all continuous variables be sorted in order to decide where to split the data. This sorting step becomes time and memory prohibitive when dealing with large data. Various techniques have been proposed to overcome the sorting obstacle, which can be roughly classified into two groups: performing data pre-sorting or using approximate summary statistic of the data. While pre-sorting techniques follow standard decision tree algorithms more closely, they cannot accommodate very large data sets. These big data decision trees are normally parallelized in various ways to enable large scale learning: data parallelism partitions the data either horizontally or vertically so that different processors see different observations or variables and task parallelism builds different tree nodes on different processors.

The *rxDTree* algorithm is an approximate decision tree algorithm with horizontal data parallelism, especially designed for handling very large data sets. It uses histograms as the approximate compact representation of the data and builds the decision tree in a breadth-first fashion. The algorithm can be executed in parallel settings such as a multicore machine or a distributed environment with a master-worker architecture. Each worker gets only a subset of the observations of the data, but has a view of the complete tree built so far. It builds a histogram from the observations it sees, which essentially compresses the data to a fixed amount of memory. This approximate description of the data is then sent to a master with constant low communication complexity independent of the size of the data set. The master integrates the information received from each of the workers and determines which terminal tree nodes to split and how. Since the histogram is built in parallel, it can be quickly constructed even for extremely large data sets.

With *rxDTree*, you can control the balance between time complexity and prediction accuracy by specifying the maximum number of bins for the histogram. The algorithm builds the histogram with roughly equal number of observations in each bin and takes the boundaries of the bins as the candidate splits for the terminal tree nodes. Since only a limited number of split locations are examined, it is possible that a suboptimal split point is chosen causing the entire tree to be different from the one constructed by a standard algorithm. However, it has been shown analytically that the error rate of the parallel tree approaches the error rate of the serial tree, even though the trees are not identical. You can set the number of bins in the histograms to control the tradeoff between accuracy and speed: a large number of bins allows a more accurate description of the data and thus more accurate results, whereas a small number of bins reduces time complexity and memory usage.

When integer predictors for which the number of bins equals or exceeds the number of observations, the *rxDTree* algorithm produces the same results as the standard sorting algorithms.

A Simple Classification Tree

In a [previous article](#), we fit a simple logistic regression model to rpart's kypnosis data. That model is easily recast as a classification tree using *rxDTtree* as follows:

```
data("kypnosis", package="rpart")
kyphTree <- rxDTtree(Kyphosis ~ Age + Start + Number, data = kypnosis,
  cp=0.01)
kyphTree

Call:
rxDTtree(formula = Kyphosis ~ Age + Start + Number, data = kypnosis,
  cp = 0.01)
Data: kypnosis
Number of valid observations: 81
Number of missing observations: 0

Tree representation:
n= 81

node), split, n, loss, yval, (yprob)
 * denotes terminal node

1) root 81 17 absent (0.79012346 0.20987654)
 2) Start>=8.5 62 6 absent (0.90322581 0.09677419)
   4) Start>=14.5 29 0 absent (1.00000000 0.00000000) *
   5) Start< 14.5 33 6 absent (0.81818182 0.18181818)
    10) Age< 55 12 0 absent (1.00000000 0.00000000) *
    11) Age>=55 21 6 absent (0.71428571 0.28571429)
     22) Age>=111 14 2 absent (0.85714286 0.14285714) *
     23) Age< 111 7 3 present (0.42857143 0.57142857) *
  3) Start< 8.5 19 8 present (0.42105263 0.57894737) *
```

Recall our conclusions from fitting this model earlier with *rxCube*: the probability of the post-operative complication Kyphosis seems to be greater if the Start is a cervical vertebra and as more vertebrae are involved in the surgery. Similarly, it appears that the dependence on age is non-linear: it first increases with age, peaks in the range 5-9, and then decreases again.

The *rxDTtree* model seems to confirm these earlier conclusions—for Start < 8.5, 11 of 19 observed subjects developed Kyphosis, while none of the 29 subjects with Start >= 14.5 did. For the remaining 33 subjects, Age was the primary splitting factor, and as we observed earlier, ages 5 to 9 had the highest probability of developing Kyphosis.

The returned object *kyphTree* is an object of class *rxDTtree*. The *rxDTtree* class is modeled closely on the *rpart* class, so that objects of class *rxDTtree* have most essential components of an *rpart* object: frame, cptable, splits, etc. By default, however, *rxDTtree* objects do not inherit from class *rpart*. You can, however, use the *rxAddInheritance* function to add *rpart* inheritance to *rxDTtree* objects.

A Simple Regression Tree

As a simple example of a regression tree, consider the *mtcars* data set and let's fit gas mileage (*mpg*) using displacement (*disp*) as a predictor:

```

# A Simple Regression Tree

mtcarTree <- rxDTree(mpg ~ disp, data=mtcars)
mtcarTree

Call:
rxDTTree(formula = mpg ~ disp, data = mtcars)
Data: mtcars
Number of valid observations: 32
Number of missing observations: 0

Tree representation:
n= 32

node), split, n, deviance, yval
* denotes terminal node

1) root 32 1126.0470 20.09063
2) disp>=163.5 18 143.5894 15.99444 *
3) disp< 163.5 14 292.1343 25.35714 *

```

There's a clear split between larger cars (those with engine displacement greater than 163.5 cubic inches) and smaller cars.

A Larger Regression Tree Model

As a more complex example, we return to the censusWorkers data. We create a regression tree predicting wage income from age, sex, and weeks worked, using the perwt variable as probability weights:

```

# A Larger Regression Tree Model

censusWorkers <- file.path(rxGetOption("sampleDataDir"),
  "CensusWorkers.xdf")
rxGetInfo(censusWorkers, getVarInfo=TRUE)
incomeTree <- rxDTTree(incwage ~ age + sex + wkswork1, pweights = "perwt",
  maxDepth = 3, minBucket = 30000, data = censusWorkers)
incomeTree

Call:
rxDTTree(formula = incwage ~ age + sex + wkswork1, data = censusWorkers,
pweights = "perwt", minBucket = 30000, maxDepth = 3)
File: C:\Program Files\Microsoft\MRO-for-RRE\8.0\R-3.2.2\ library\RevoScaleR\SampleData\CensusWorkers.xdf
Number of valid observations: 351121
Number of missing observations: 0

Tree representation:
n= 351121

node), split, n, deviance, yval
* denotes terminal node

1) root 351121 1.177765e+16 35788.47
2) sex=Female 161777 2.271425e+15 26721.09
4) wkswork1< 51.5 56874 5.757587e+14 19717.74 *
5) wkswork1>=51.5 104903 1.608813e+15 30505.87
10) age< 34.5 31511 2.500078e+14 25836.32 *
11) age>=34.5 73392 1.338235e+15 32576.74 *
3) sex=Male 189344 9.008506e+15 43472.71
6) age< 31.5 48449 6.445334e+14 27577.80 *
7) age>=31.5 140895 8.010642e+15 49221.82
14) wkswork1< 51.5 34359 1.550839e+15 37096.62 *
15) wkswork1>=51.5 106536 6.326896e+15 53082.08 *

```

The primary split here (not surprising given our analysis of this data set in the [Tutorial: Analyzing US census data](#)

with RevoScaleR) is sex; women on average earn substantially less than men. The additional splits are also not surprising; older workers earn more than younger workers, and those who work more hours tend to earn more than those who work fewer hours.

Controlling the Model Fit

The *rxDTree* function has a number of options for controlling the model fit. Most of these control parameters are familiar to *rpart* users, but the defaults have been modified in some cases to better support large data tree models. A full listing of these options can be found in the *rxDTree* help file, but the following have been found in our testing to be the most useful at controlling the time required to fit a model with *rxDTree*.

- *xVal*: controls the number of folds used to perform cross-validation. The default of 2 allows for some pruning; once you have closed in a model you may want to increase the value for final fitting and pruning.
- *maxDepth*: sets the maximum depth of any node of the tree. Computations grow rapidly more expensive as the depth increases, so we recommend a *maxDepth* of 10 to 15.
- *maxCompete*: specifies the number of "competitor splits" retained in the output. By default, *rxDTree* sets this to 0, but a setting of 3 or 4 can be useful for diagnostic purposes in determining why a particular split was chosen.
- *maxSurrogate*: specifies the number of surrogate splits retained in the output. Again, by default *rxDTree* sets this to 0. Surrogate splits are used to assign an observation when the primary split variable is missing for that observation.
- *maxNumBins*: controls the maximum number of bins used for each variable. Managing the number of bins is important in controlling memory usage. The default is to use the larger of 101 and the square root of the number of observations for small to moderate size data sets (up to about one million observations), but for larger sets to use 1001 bins. For small data sets with continuous predictors, you may find that you need to increase the *maxNumBins* to obtain models that resemble those from *rpart*.

For large data sets (100000 or more observations), you may need to adjust the following parameters to obtain meaningful models:

- *cp*: a complexity parameter and sets the bar for how much a split must reduce the complexity before being accepted. We have set the default to 0 and recommend using *maxDepth* and *minBucket* to control your tree sizes. If you want to specify a *cp* value, start with a conservative value, such as *rpart*'s 0.01; if you don't see an adequate number of splits, decrease the *cp* by powers of 10 until you do. For our large airline data, we have found interesting models begin with a *cp* of about 1e-4.
- *minSplit*, *minBucket*: determine how many observations must be in a node before a split is attempted (*minSplit*) and how many must remain in a terminal node (*minBucket*).

Large Data Tree Models

Scaling decision trees to very large data sets is possible with *rxDTree* but should be done with caution—the wrong choice of model parameters can easily lead to models that take hours or longer to estimate, even in a distributed computing environment. For example, in the [Tutorial: Load and analyze a large airline data set with RevoScaleR](#), we estimated linear models using the large airline data and used the variable *Origin* as a predictor in several models. The *Origin* variable is a factor variable with 373 levels with no obvious ordering.

Incorporating this variable into an *rxDTree* model that is performing more than two level classification can easily consume hours of computation time. To prevent such unintended consequences, *rxDTree* has a parameter *maxUnorderedLevels*, which defaults to 32; in the case of *Origin*, this parameter would flag an error. However, a factor variable of "Region" which groups the airports of *Origin* by location may well be a useful proxy, and can be constructed to have only a limited number of levels. Numeric and ordered factor predictors are much more easily incorporated into the model.

As an example of a large data classification tree, consider the following simple model using the 7% subsample of the full airline data (uses the variable *ArrDel/15* indicating flights with an arrival delay of 15 minutes or more):

```
# Large Data Tree Models

bigDataDir <- "C:/MRS/Data"
sampleAirData <- file.path(bigDataDir, "AirOnTime7Pct.xdf")
airlineTree <- rxDTree(ArrDel15 ~ CRSDepTime + DayOfWeek, data = sampleAirData,
blocksPerRead = 30, maxDepth = 5, cp = 1e-5)
```

The `blocksPerRead` argument is ignored if run locally using R Client. [Learn more...](#)

The default `cp` of 0 produces a very large number of splits; specifying `cp = 1e-5` produces a more manageable set of splits in this model:

```

airlineTree

Call:
rxDTree(formula = ArrDel15 ~ CRSDepTime + DayOfWeek, data = sampleAirData,
maxDepth = 5, cp = 1e-05, blocksPerRead = 30)
File: C:\MRS\Data\AirOnTime7Pct.xdf
Number of valid observations: 10186272
Number of missing observations: 213483

Tree representation:
n= 10186272

node), split, n, deviance, yval
 * denotes terminal node

1) root 10186272 1630331.000 0.20008640
 2) CRSDepTime< 13.1745 4941190 642452.000 0.15361830
 4) CRSDepTime< 8.3415 1777685 189395.700 0.12123970
 8) CRSDepTime>=0.658 1717573 178594.900 0.11787560
16) CRSDepTime< 6.7665 599548 52711.450 0.09740671
 32) CRSDepTime>=1.625 578762 49884.260 0.09526714 *
 33) CRSDepTime< 1.625 20786 2750.772 0.15698070 *
17) CRSDepTime>=6.7665 1118025 125497.500 0.12885220
 34) DayOfWeek=Sun 134589 11722.540 0.09638975 *
 35) DayOfWeek=Mon,Tues,Wed,Thur,Fri,Sat 983436 113613.80 0.13329490 *
9) CRSDepTime< 0.658 60112 10225.960 0.21736090
18) CRSDepTime>=0.2415 9777 1429.046 0.17776410 *
19) CRSDepTime< 0.2415 50335 8778.609 0.22505220 *
5) CRSDepTime>=8.3415 3163505 450145.400 0.17181290
10) CRSDepTime< 11.3415 1964400 268472.400 0.16335320
20) DayOfWeek=Sun 271900 30839.160 0.13043400
 40) CRSDepTime< 9.7415 126700 13381.800 0.12002370 *
 41) CRSDepTime>=9.7415 145200 17431.650 0.13951790 *
21) DayOfWeek=Mon,Tues,Wed,Thur,Fri,Sat 1692500 237291.300 0.16864170
 42) DayOfWeek=Tues,Wed,Sat 835355 113384.500 0.16196470 *
 43) DayOfWeek=Mon,Thur,Fri 857145 123833.200 0.17514890 *
11) CRSDepTime>=11.3415 1199105 181302.000 0.18567180
22) DayOfWeek=Mon,Tues,Wed,Sat,Sun 852016 124610.900 0.17790390
 44) DayOfWeek=Tues,Sun 342691 48917.520 0.17250230 *
 45) DayOfWeek=Mon,Wed,Sat 509325 75676.600 0.18153830 *
23) DayOfWeek=Thur,Fri 347089 56513.560 0.20474000 *
3) CRSDepTime>=13.1745 5245082 967158.500 0.24386220
6) DayOfWeek=Mon,Tues,Wed,Sat,Sun 3708992 651771.300 0.22746990
12) DayOfWeek=Sat 635207 96495.570 0.18681000
24) CRSDepTime>=20.2745 87013 12025.600 0.16564190 *
25) CRSDepTime< 20.2745 548194 84424.790 0.19016990 *
13) DayOfWeek=Mon,Tues,Wed,Sun 3073785 554008.600 0.23587240
26) CRSDepTime< 16.508 1214018 203375.700 0.21281150
 52) CRSDepTime< 15.1325 709846 114523.300 0.20223400 *
 53) CRSDepTime>=15.1325 504172 88661.120 0.22770400 *
27) CRSDepTime>=16.508 1859767 349565.800 0.25092610
 54) DayOfWeek=Mon,Tues 928523 168050.900 0.23729730 *
 55) DayOfWeek=Wed,Sun 931244 181170.600 0.26451500 *
7) DayOfWeek=Thur,Fri 1536090 311984.200 0.28344240
14) CRSDepTime< 15.608 445085 82373.020 0.24519140
28) CRSDepTime< 14.6825 273682 49360.240 0.23609880 *
29) CRSDepTime>=14.6825 171403 32954.030 0.25970960 *
15) CRSDepTime>=15.608 1091005 228694.300 0.29904720
30) CRSDepTime>=21.9915 64127 11932.930 0.24718140 *
31) CRSDepTime< 21.9915 1026878 216578.100 0.30228620
 62) CRSDepTime< 17.0745 264085 53451.260 0.28182970 *
 63) CRSDepTime>=17.0745 762793 162978.000 0.30936830 *

```

Looking at the fitted objects cptable component, we can look at whether we have overfitted the model:

```
airlineTree$cptable
```

	CP	nsplit	rel	error	xerror	xstd
1	1.270950e-02	0	1.0000000	1.0000002	0.0004697734	
2	2.087342e-03	1	0.9872905	0.9873043	0.0004629111	
3	1.785488e-03	2	0.9852032	0.9852215	0.0004625035	
4	7.772395e-04	3	0.9834177	0.9834381	0.0004608330	
5	6.545095e-04	4	0.9826404	0.9826606	0.0004605065	
6	5.623968e-04	5	0.9819859	0.9820200	0.0004602950	
7	3.525848e-04	6	0.9814235	0.9814584	0.0004602578	
8	2.367018e-04	7	0.9810709	0.9811071	0.0004600062	
9	2.274981e-04	8	0.9808342	0.9808700	0.0004597725	
10	2.112635e-04	9	0.9806067	0.9806567	0.0004596187	
11	2.097651e-04	10	0.9803955	0.9804365	0.0004595150	
12	1.173008e-04	11	0.9801857	0.9803311	0.0004594245	
13	1.124180e-04	12	0.9800684	0.9800354	0.0004592792	
14	1.089414e-04	13	0.9799560	0.9800354	0.0004592792	
15	9.890134e-05	14	0.9798471	0.9799851	0.0004592187	
16	9.125152e-05	15	0.9797482	0.9798766	0.0004591605	
17	4.687397e-05	16	0.9796569	0.9797504	0.0004591074	
18	4.510554e-05	17	0.9796100	0.9797292	0.0004590784	
19	3.603837e-05	18	0.9795649	0.9796812	0.0004590301	
20	2.771093e-05	19	0.9795289	0.9796383	0.0004590247	
21	1.577140e-05	20	0.9795012	0.9796013	0.0004590000	
22	1.122899e-05	21	0.9794854	0.9795671	0.0004589736	
23	1.025944e-05	22	0.9794742	0.9795560	0.0004589678	
24	1.000000e-05	23	0.9794639	0.9795455	0.0004589660	

We see a steady decrease in cross-validation error (xerror) as the number of splits increase, but note that at about nsplit=11 the rate of change slows dramatically. The optimal model is probably very near here. (The total number of passes through the data is equal to a base of $\maxDepth + 3$, plus $xVal$ times $(\maxDepth + 2)$, where $xVal$ is the number of folds for cross-validation and \maxDepth is the maximum tree depth. Thus a depth 10 tree with 4-fold cross-validation requires $13 + 48$, or 61, passes through the data.)

To prune the tree back, use the `prune.rxDTree` function:

```

airlineTree4 <- prune.rxDTree(airlineTree, cp=1e-4)
airlineTree4

Call:
rxDTTree(formula = ArrDel15 ~ CRSDepTime + DayOfWeek, data = sampleAirData,
maxDepth = 5, cp = 1e-05, blocksPerRead = 30)
File: C:\MRS\Data\AirOnTime7Pct.xdf
Number of valid observations: 10186272
Number of missing observations: 213483

Tree representation:
n= 10186272

node), split, n, deviance, yval
 * denotes terminal node

1) root 10186272 1630331.00 0.20008640
 2) CRSDepTime< 13.1745 4941190 642452.00 0.15361830
 4) CRSDepTime< 8.3415 1777685 189395.70 0.12123970
 8) CRSDepTime>=0.658 1717573 178594.90 0.11787560
16) CRSDepTime< 6.7665 599548 52711.45 0.09740671 *
17) CRSDepTime>=6.7665 1118025 125497.50 0.12885220 *
 9) CRSDepTime< 0.658 60112 10225.96 0.21736090 *
 5) CRSDepTime>=8.3415 3163505 450145.40 0.17181290
10) CRSDepTime< 11.3415 19644400 268472.40 0.16335320
20) DayOfWeek=Sun 271900 30839.16 0.13043400 *
21) DayOfWeek=Mon,Tues,Wed,Thur,Fri,Sat 1692500 237291.30 0.16864170 *
11) CRSDepTime>=11.3415 1199105 181302.00 0.18567180
22) DayOfWeek=Mon,Tues,Wed,Sat,Sun 852016 124610.90 0.17790390 *
23) DayOfWeek=Thur,Fri 347089 56513.56 0.20474000 *
3) CRSDepTime>=13.1745 5245082 967158.50 0.24386220
 6) DayOfWeek=Mon,Tues,Wed,Sat,Sun 3708992 651771.30 0.22746990
12) DayOfWeek=Sat 635207 96495.57 0.18681000 *
13) DayOfWeek=Mon,Tues,Wed,Sun 3073785 554008.60 0.23587240
26) CRSDepTime< 16.508 1214018 203375.70 0.21281150
 52) CRSDepTime< 15.1325 709846 114523.30 0.20223400 *
 53) CRSDepTime>=15.1325 504172 88661.12 0.22770400 *
27) CRSDepTime>=16.508 1859767 349565.80 0.25092610
 54) DayOfWeek=Mon,Tues 928523 168050.90 0.23729730 *
 55) DayOfWeek=Wed,Sun 931244 181170.60 0.26451500 *
 7) DayOfWeek=Thur,Fri 1536090 311984.20 0.28344240
14) CRSDepTime< 15.608 445085 82373.02 0.24519140 *
15) CRSDepTime>=15.608 1091005 228694.30 0.29904720
30) CRSDepTime>=21.9915 64127 11932.93 0.24718140 *
31) CRSDepTime< 21.9915 1026878 216578.10 0.30228620 *

```

If rpart is installed, *prune.rxDTree* acts as a method for the *prune* function, so you can call it more simply:

```

airlineTree4 <- prune(airlineTree, cp=1e-4)

```

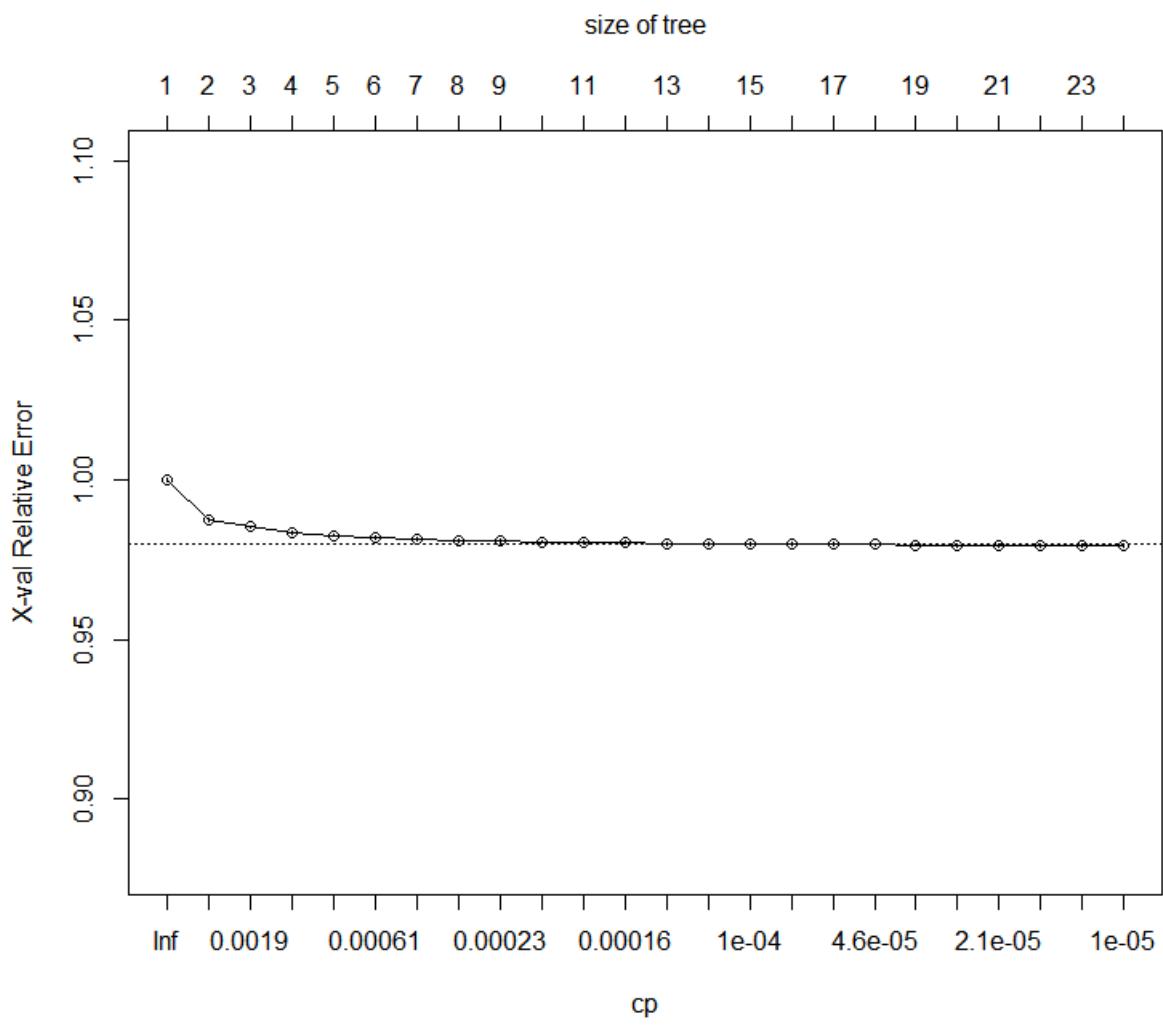
For models fit with 2-fold or greater cross-validation, it is useful to use the cross-validation standard error (part of the cptable component) as a guide to pruning. The rpart function *plotcp* can be useful for this:

```

plotcp(rxAddInheritance(airlineTree))

```

This yields the following plot:



From this plot, it appears we can prune even further, to perhaps seven or eight splits. Looking again at the cptable, a cp of 2.5e-4 seems a reasonable pruning choice:

```

airlineTreePruned <- prune.rxDTree(airlineTree, cp=2.5e-4)
airlineTreePruned

Call:
rxDTTree(formula = ArrDel15 ~ CRSDepTime + DayOfWeek, data = sampleAirData,
maxDepth = 5, cp = 1e-05, blocksPerRead = 30)
File: C:\MRS\Data\AirOnTime7Pct.xdf
Number of valid observations: 10186272
Number of missing observations: 213483

Tree representation:
n= 10186272

node), split, n, deviance, yval
* denotes terminal node

1) root 10186272 1630331.00 0.2000864
2) CRSDepTime< 13.1745 4941190 642452.00 0.1536183
4) CRSDepTime< 8.3415 1777685 189395.70 0.1212397
8) CRSDepTime>=0.658 1717573 178594.90 0.1178756 *
9) CRSDepTime< 0.658 60112 10225.96 0.2173609 *
5) CRSDepTime>=8.3415 3163505 450145.40 0.1718129 *
3) CRSDepTime>=13.1745 5245082 967158.50 0.2438622
6) DayOfWeek=Mon,Tues,Wed,Sat,Sun 3708992 651771.30 0.2274699
12) DayOfWeek=Sat 635207 96495.57 0.1868100 *
13) DayOfWeek=Mon,Tues,Wed,Sun 3073785 554008.60 0.2358724
26) CRSDepTime< 16.508 1214018 203375.70 0.2128115 *
27) CRSDepTime>=16.508 1859767 349565.80 0.2509261 *
7) DayOfWeek=Thur,Fri 1536090 311984.20 0.2834424
14) CRSDepTime< 15.608 445085 82373.02 0.2451914 *
15) CRSDepTime>=15.608 1091005 228694.30 0.2990472 *

```

Handling Missing Values

The *removeMissings* argument to *rxDTtree*, as in most RevoScaleR analysis functions, controls how the function deals with missing data in the model fit. If *TRUE*, all rows containing missing values for the response or any predictor variable are removed before model fitting. If *FALSE* (the default), only those rows for which the value of the response or all values of the predictor variables are missing are removed. Using *removeMissings=TRUE* is roughly equivalent to the effect of the *na.omit* function for *rpart*; in that if the file is written out, all rows containing NAs are removed. There is no equivalent for *rxDTtree* to the *na.exclude* function, which pads the output with NAs for observations that cannot be predicted. Using *removeMissings=FALSE* is the equivalent of using the *na.rpart* or *na.pass* functions; the data is passed through unchanged, but rows that have no data for either all predictors or the response are excluded from the model.

Prediction

As with other RevoScaleR analysis functions, prediction is performed using the *rxPredict* function, to which you supply a fitted model object and a set of new data (which may be the original data set, but in any event must contain the variables used in the original model).

The adult data set is a widely used machine learning data set, similar to the censusWorkers data we have already analyzed. The data set is available from the machine learning data repository at UC Irvine (<http://archive.ics.uci.edu/ml/datasets/Adult>) (and comes in two pieces: a training data set (adult.data) and a test data set (adult.test). This makes it ready-made for use in prediction. To run the following examples, download this data and add a .txt extension, so that you have adult.data.txt and adult.test.txt. (A third file, adult.names, gives a description of the variables; we use this in the code below as a source for the variable names, which are not part of the data files):

```

# Prediction

if (bHasAdultData){

bigDataDir <- "C:/MRS/Data"
adultDataFile <- file.path(bigDataDir, "adult.data.txt")
adultTestFile <- file.path(bigDataDir, "adult.test.txt")

newNames <- c("age", "workclass", "fnlwgt", "education",
"education_num", "marital_status", "occupation", "relationship",
"ethnicity", "sex", "capital_gain", "capital_loss", "hours_per_week",
"native_country", "income")
adultTrain <- rxImport(adultDataFile, stringsAsFactors = TRUE)
names(adultTrain) <- newNames
adultTest <- rxImport(adultTestFile, rowsToSkip = 1,
stringsAsFactors=TRUE)
names(adultTest) <- newNames
adultTree <- rxDTree(income ~ age + sex + hours_per_week, pweights = "fnlwgt",
data = adultTrain)
adultPred <- rxPredict(adultTree, data = adultTest, type="vector")
sum(adultPred == as.integer(adultTest$income))/length(adultTest$income)
} # End of bHasAdultData

[1] 0.7734169

```

The result shows that the fitted model accurately classifies about 77% of the test data.

When using *rxPredict* with *rxDTree* objects, you should keep in mind how it differs from *predict* with *rpart* objects. First, a *data* argument is always required—this can be either the original data or new data; there is no *newData* argument as in *rpart*. Prediction with the original data provides fitted values, not predictions, but the predicted variable name still defaults to *varname_Pred*.

Visualizing Trees

The RevoTreeView package can be used to plot decision trees from *rxDTree* or *rpart* in an HTML page. Both classification and regression trees are supported. By plotting the tree objects returned by RevoTreeView's *createTreeView* function in a browser, you can interact with your decision tree. The resulting tree's HTML page can also be shared with other people or displayed on different machines using the package's *zipTreeView* function.

As an example, consider a classification tree built from the *kyphosis* data that is included in the *rpart* package. It produces the following text output:

```

data("kyphosis", package="rpart")
kyphTree <- rxDTree(Kyphosis ~ Age + Start + Number,
data = kyphosis, cp=0.01)
kyphTree

Call:
rxDTree(formula = Kyphosis ~ Age + Start + Number, data = kyphosis,
cp = 0.01)
Data: kyphosis
Number of valid observations: 81
Number of missing observations: 0

Tree representation:
n= 81

node), split, n, loss, yval, (yprob)
 * denotes terminal node
1) root 81 17 absent (0.79012346 0.20987654)
  2) Start>=8.5 62 6 absent (0.90322581 0.09677419)
  4) Start>=14.5 29 0 absent (1.00000000 0.00000000) *
  5) Start< 14.5 33 6 absent (0.81818182 0.18181818)
10) Age< 55 12 0 absent (1.00000000 0.00000000) *
11) Age>=55 21 6 absent (0.71428571 0.28571429)
22) Age>=111 14 2 absent (0.85714286 0.14285714) *
23) Age< 111 7 3 present (0.42857143 0.57142857) *
  3) Start< 8.5 19 8 present (0.42105263 0.57894737) *

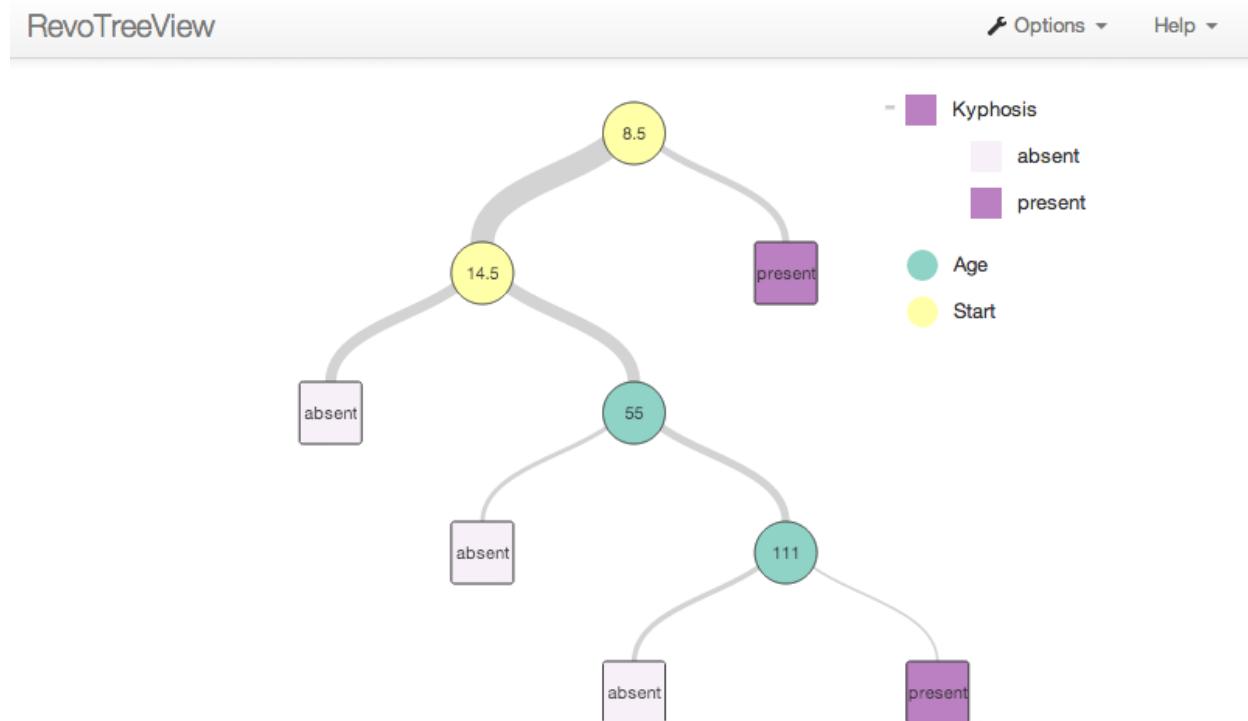
```

Now, you can display an HTML version of the tree output by plotting the object produced by the *createTreeView* function. After running the preceding R code, run the following to load the *RevoTreeView* package and display an interactive decision tree in your browser:

```

library(RevoTreeView)
plot(createTreeView(kyphTree))

```



In this interactive tree, click on the circular split nodes to expand or collapse the tree branch. Clicking a node will expand and collapse the node to the last view of that branch. If you use a *CTRL + Click*, the tree displays only the

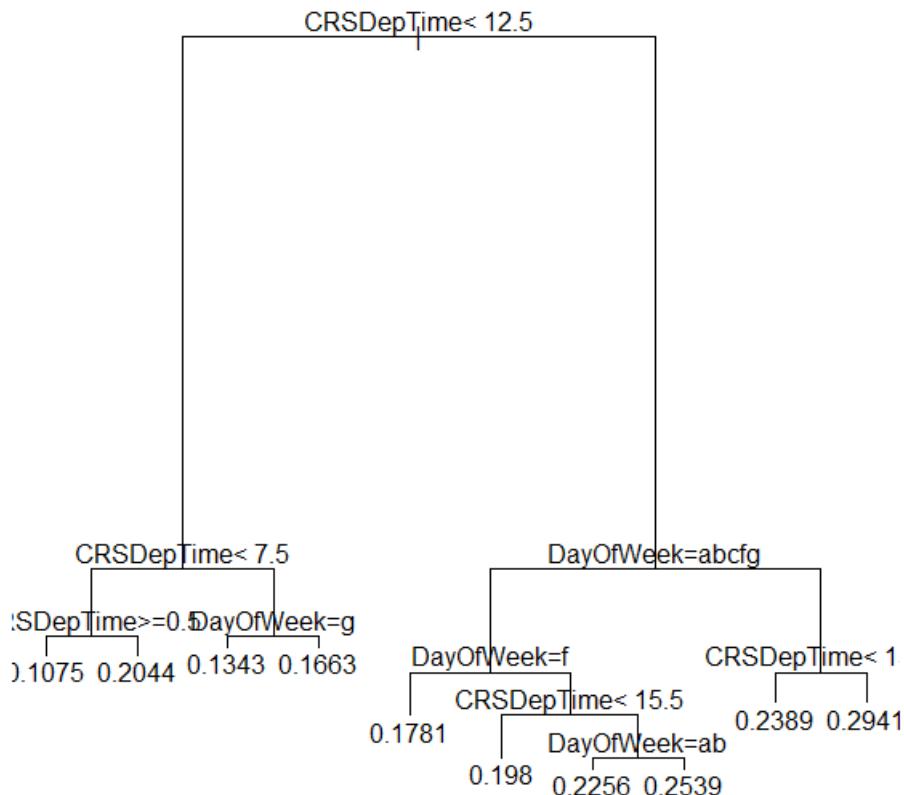
children of the selected node. If you click *ALT + Click*, the tree displays all levels below the selected node. The square-shaped nodes, called leaf, or terminal nodes, cannot be expanded.

To get additional information, hover over the node to expose the node details such as its name, the next split variable, its value, the *n*, the predicted value, and other details such as loss or deviance.

You can also use the rpart *plot* and *text* methods with *rxDTree* objects, provided you use the *rxAddInheritance* function to provide rpart inheritance:

```
# Plotting Trees  
  
plot(rxAddInheritance(airlineTreePruned))  
text(rxAddInheritance(airlineTreePruned))
```

Provides the following plot:



Estimating Decision Forest Models

7/12/2022 • 8 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

The `rxDForest` function in RevoScaleR fits a *decision forest*, which is an ensemble of decision trees. Each tree is fitted to a bootstrap sample of the original data, which leaves about 1/3 of the data unused in the fitting of each tree. Each data point in the original data is fed through each of the trees for which it was unused; the decision forest prediction for that data point is the statistical *mode* of the individual tree predictions, that is, the majority prediction (for classification; for regression problems, the prediction is the mean of the individual predictions).

Unlike individual decision trees, decision forests are not prone to overfitting, and they are consistently shown to be among the best machine learning algorithms. RevoScaleR implements decision forests in the `rxDForest` function, which uses the same basic tree-fitting algorithm as `rxDTree` (see "[The `rxDTree` Algorithm](#)"). To create the forest, you specify the number of trees using the `nTree` argument and the number of variables to consider for splitting in each tree using the `mTry` argument. In most cases, you specify the maximum depth to grow the individual trees: greater depth typically results in greater accuracy, but as with `rxDTree`, also results in longer fitting times.

A Simple Classification Forest

In [Logistic Regression Models](#), we fit a simple classification tree model to rpart's kypnosis data. That model is easily recast as a classification decision forest using `rxDForest` as follows (we set the `seed` argument to ensure reproducibility; in most cases you can omit):

```
data("kypnosis", package="rpart")
kypnForest <- rxDForest(Kyphosis ~ Age + Start + Number, seed = 10,
  data = kypnosis, cp=0.01, nTree=500, mTry=3)
kypnForest

Call:
rxDForest(formula = Kyphosis ~ Age + Start + Number, data = kypnosis,
  cp = 0.01, nTree = 500, mTry = 3, seed = 10)

Type of decision forest: class
Number of trees: 500
No. of variables tried at each split: 3

OOB estimate of error rate: 19.75%
Confusion matrix:
 Predicted
Kyphosis absent present class.error
absent      56      8  0.1250000
present      8      9  0.4705882
```

While decision forests do not produce a unified model, as logistic regression and decision trees do, they do produce reasonable predictions for each data point. In this case, we can obtain predictions using `rxPredict` as follows:

```
dfPreds <- rxPredict(kyphForest, data=kyphosis)
```

Compared to the Kyphosis variable in the original kyphosis data, we see that approximately 88 percent of cases are classified correctly:

```
sum(as.character(dfPreds[,1]) ==  
as.character(kyphosis$Kyphosis))/81  
  
[1] 0.8765432
```

A Simple Regression Forest

As a simple example of a regression forest, consider the classic *stackloss* data set, containing observations from a chemical plant producing nitric acid by the oxidation of ammonia, and let's fit the stack loss (*stack.loss*) using air flow (*Air.Flow*), water temperature (*Water.Temp*), and acid concentration (*Acid.Conc.*) as predictors:

```
# A Simple Regression Forest  
  
stackForest <- rxDForest(stack.loss ~ Air.Flow + Water.Temp + Acid.Conc.,  
  data=stackloss, nTree=200, mTry=2)  
stackForest  
  
Call:  
  rxDForest(formula = stack.loss ~ Air.Flow + Water.Temp + Acid.Conc.,  
  data = stackloss, maxDepth = 3, nTree = 200, mTry = 2)  
  
Type of decision forest: anova  
Number of trees: 200  
No. of variables tried at each split: 2  
  
Mean of squared residuals: 44.54992  
% Var explained: 65
```

A Larger Regression Forest Model

As a more complex example, we return to the *censusWorkers* data to which we earlier fit a decision tree. We create a regression forest predicting wage income from age, sex, and weeks worked, using the *perwt* variable as probability weights (note that we retain the *maxDepth* and *minBucket* parameters from our earlier decision tree example):

```

# A Larger Regression Forest Model

censusWorkers <- file.path(rxGetOption("sampleDataDir"),
  "CensusWorkers.xdf")
rxGetInfo(censusWorkers, getInfo=TRUE)
incForest <- rxDForest(incwage ~ age + sex + wkswork1, pweights = "perwt",
  maxDepth = 3, minBucket = 30000, mTry=2, nTree=200, data = censusWorkers)
incForest

Call:
rxDForest(formula = incwage ~ age + sex + wkswork1, data = censusData,
pweights = "perwt", maxDepth = 5, nTree = 200, mTry = 2)

Type of decision forest: anova
Number of trees: 200
No. of variables tried at each split: 2

Mean of squared residuals: 1458969472
% Var explained: 11

```

Large Data Decision Forest Models

As with decision trees, scaling decision forests to very large data sets should be done with caution. The wrong choice of model parameters can easily lead to models that take hours or longer to estimate, even in a distributed computing environment, or that simply cannot be fit at all. For non-binary classification problems, as with decision trees, categorical predictors should have a small to moderate number of levels.

As an example of a large data classification forest, consider the following simple model using the 7% subsample of the full airline data (this uses the variable *ArrDel15* indicating flights with an arrival delay of 15 minutes or more):

The `blocksPerRead` argument is ignored if run locally using R Client. [Learn more...](#)

```

# Large Data Tree Models

bigDataDir <- "C:/MRS/Data"
sampleAirData <- file.path(bigDataDir, "AirOnTime7Pct.xdf")
airlineForest <- rxDForest(ArrDel15 ~ CRSDepTime + DayOfWeek,
  data = sampleAirData, blocksPerRead = 30, maxDepth = 5,
  nTree=20, mTry=2, method="class", seed = 8)

```

Yields the following:

```
airlineForest

Call:
rxDForest(formula = ArrDel15 ~ CRSDepTime + DayOfWeek, data = sampleAirData,
method = "class", maxDepth = 5, nTree = 20, mTry = 2, seed = 8,
blocksPerRead = 30)

Type of decision forest: class
Number of trees: 20
No. of variables tried at each split: 2

OOB estimate of error rate: 20.01%
Confusion matrix:
 Predicted
ArrDel15 FALSE TRUE class.error
FALSE 8147274    0        0
TRUE   2037941    0        1
```

One problem with this model is that it predicts all flights to be on time. As we iterate over this model, we'll remove this limitation.

Looking at the fitted object's forest component, we see that a number of the fitted trees do not split at all:

```

airlineForest$forest

[[1]]
Number of valid observations: 6440007
Number of missing observations: 3959748

Tree representation:
n= 10186709

node), split, n, loss, yval, (yprob)
* denotes terminal node

1) root 10186709 2038302 FALSE (0.7999057 0.2000943) *

[[2]]
Number of valid observations: 6440530
Number of missing observations: 3959225

Tree representation:
n= 10186445

node), split, n, loss, yval, (yprob)
* denotes terminal node

1) root 10186445 2038249 FALSE (0.7999057 0.2000943) *

[[3]]
...
[[6]]
Number of valid observations: 6439485
Number of missing observations: 3960270

Tree representation:
n= 10186656

node), split, n, loss, yval, (yprob)
* denotes terminal node

1) root 10186656 2038291 FALSE (0.7999057 0.2000943) *

[[7]]
Number of valid observations: 6439307
Number of missing observations: 3960448

Tree representation:
n= 10186499

node), split, n, loss, yval, (yprob)
* denotes terminal node

1) root 10186499 2038260 FALSE (0.7999057 0.2000943) *
...

```

This may well be because our response is extremely unbalanced--that is, the percentage of flights that are late by 15 minutes or more is quite small. We can tune the fit by providing a *loss matrix*, which allows us to penalize certain predictions in favor of others. You specify the loss matrix using the `parms` argument, which takes a list with named components. The loss component is specified as either a matrix, or equivalently, a vector that can be coerced to a matrix. In the binary classification case, it can be useful to start with a loss matrix with a penalty roughly equivalent to the ratio of the two classes. So, in our case we know that the on-time flights outnumber the late flights approximately 4 to 1:

```

airlineForest2 <- rxDForest(ArrDel15 ~ CRSDepTime + DayOfWeek,
  data = sampleAirData, blocksPerRead = 30, maxDepth = 5, seed = 8,
  nTree=20, mTry=2, method="class", parms=list(loss=c(0,4,1,0)))

Call:
rxDForest(formula = ArrDel15 ~ CRSDepTime + DayOfWeek, data = sampleAirData,
  method = "class", parms = list(loss = c(0, 4, 1, 0)), maxDepth = 5,
  nTree = 20, mTry = 2, seed = 8, blocksPerRead = 30)

Type of decision forest: class
Number of trees: 20
No. of variables tried at each split: 2

OOB estimate of error rate: 42.27%
Confusion matrix:
 Predicted
ArrDel15 FALSE TRUE class.error
FALSE 4719374 3427900 0.420742
TRUE 877680 1160261 0.430670

```

This model no longer predicts all flights as on time, but now over-predicts late flights. Adjusting the loss matrix again, this time reducing the penalty, yields the following output:

```

Call:
rxDForest(formula = ArrDel15 ~ CRSDepTime + DayOfWeek, data = sampleAirData,
  method = "class", parms = list(loss = c(0, 3, 1, 0)), maxDepth = 5,
  nTree = 20, mTry = 2, seed = 8, blocksPerRead = 30)

Type of decision forest: class
Number of trees: 20
No. of variables tried at each split: 2

OOB estimate of error rate: 30.15%
Confusion matrix:
 Predicted
ArrDel15 FALSE TRUE class.error
FALSE 6465439 1681835 0.2064292
TRUE 1389092 648849 0.6816154

```

Controlling the Model Fit

The *rxDForest* function has a number of options for controlling the model fit. Most of these control parameters are identical to the same controls in *rxDTree*. A full listing of these options can be found in the *rxDForest* help file, but the following have been found in our testing to be the most useful at controlling the time required to fit a model with *rxDForest*.

- *maxDepth*: sets the maximum depth of any node of the tree. Computations grow rapidly more expensive as the depth increases, so we recommend a *maxDepth* of 10 to 15.
- *maxNumBins*: controls the maximum number of bins used for each variable. Managing the number of bins is important in controlling memory usage. The default is to use the larger of 101 and the square root of the number of observations for small to moderate size data sets (up to about one million observations), but for larger sets to use 1001 bins. For small data sets with continuous predictors, you may find that you need to increase the *maxNumBins* to obtain models that resemble those from *rpart*.
- *minSplit*, *minBucket*: determine how many observations must be in a node before a split is attempted (*minSplit*) and how many must remain in a terminal node (*minBucket*).

Estimate Models Using Stochastic Gradient Boosting

7/12/2022 • 4 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

The *rxBTrees* function in RevoScaleR, like *rxDForest*, fits a decision forest to your data, but the forest is generated using a stochastic gradient boosting algorithm. This is similar to the decision forest algorithm in that each tree is fitted to a subsample of the training set (sampling without replacement) and predictions are made by aggregating over the predictions of all trees. Unlike the *rxDForest* algorithm, the boosted trees are added one at a time, and at each iteration, a regression tree is fitted to the current pseudo-residuals, that is, the gradient of the loss functional being minimized.

Like the *rxDForest* function, *rxBTrees* uses the same basic tree-fitting algorithm as *rxDTree* (see "[The *rxDTree* Algorithm](#)"). To create the forest, you specify the number of trees using the *nTree* argument and the number of variables to consider for splitting at each tree node using the *mTry* argument. In most cases, you specify the maximum depth to grow the individual trees: shallow trees seem to be sufficient in many applications, but as with *rxDTree* and *rxDForest*, greater depth typically results in longer fitting times.

Different model types are supported by specifying different loss functions, as follows:

- "*gaussian*", for regression models
- "*bernoulli*", for binary classification models
- "*multinomial*", for multi-class classification models

A Simple Binary Classification Forest

In [Logistic Regression](#), we fit a simple classification tree model to rpart's kyphosis data. That model is easily recast as a classification decision forest using *rxBTrees* as follows. We set the *seed* argument to ensure reproducibility; in most cases you can omit it:

```
# A Simple Classification Forest

data("kyphosis", package="rpart")
kyphBTrees <- rxBTrees(Kyphosis ~ Age + Start + Number, seed = 10,
  data = kyphosis, cp=0.01, nTree=500, mTry=3, lossFunction="bernoulli")
kyphBTrees

Call:
rxBTrees(formula = Kyphosis ~ Age + Start + Number, data = kyphosis,
  cp = 0.01, nTree = 500, mTry = 3, seed = 10, lossFunction = "bernoulli")

Loss function of boosted trees: bernoulli
Number of iterations: 500
OOB estimate of deviance: 0.1441952
```

In this case, we don't need to explicitly specify the loss function; "*bernoulli*" is the default.

A Simple Regression Forest

As a simple example of a regression forest, consider the classic *stackloss* data set, containing observations from

a chemical plant producing nitric acid by the oxidation of ammonia, and let's fit the stack loss (*stack.loss*) using air flow (*Air.Flow*), water temperature (*Water.Temp*), and acid concentration (*Acid.Conc.*) as predictors:

```
# A Simple Regression Forest

stackBTrees <- rxDForest(stack.loss ~ Air.Flow + Water.Temp + Acid.Conc.,
  data=stackloss, nTree=200, mTry=2, lossFunction="gaussian")
stackBTrees

Call:
rxForest(formula = stack.loss ~ Air.Flow + Water.Temp + Acid.Conc.,
  data = stackloss, nTree = 200, mTry = 2, lossFunction = "gaussian")

Loss function of boosted trees: gaussian
Number of iterations: 200
OOB estimate of deviance: 1.46797
```

A Simple Multinomial Forest Model

As a simple multinomial example, we fit an *rxBTrees* model to the iris data:

```
# A Multinomial Forest Model

irisBTrees <- rxBTrees(Species ~ Sepal.Length + Sepal.Width +
  Petal.Length + Petal.Width, data=iris,
  nTree=50, seed=0, maxDepth=3, lossFunction="multinomial")
irisBTrees

Call:
rxBTrees(formula = Species ~ Sepal.Length + Sepal.Width + Petal.Length +
  Petal.Width, data = iris, maxDepth = 3, nTree = 50, seed = 0,
  lossFunction = "multinomial")

Loss function of boosted trees: multinomial
Number of boosting iterations: 50
No. of variables tried at each split: 1

OOB estimate of deviance: 0.02720805
```

Controlling the Model Fit

The principal parameter controlling the boosting algorithm itself is the *learning rate*. The learning rate (or shrinkage) is used to scale the contribution of each tree when it is added to the ensemble. The default learning rate is 0.1.

The *rxBTrees* function has a number of other options for controlling the model fit. Most of these control parameters are identical to the same controls in *rxDTree*. A full listing of these options can be found in the *rxBTrees* help file, but the following have been found in our testing to be the most useful at controlling the time required to fit a model with *rxBTrees*:

- *maxDepth*: sets the maximum depth of any node of the tree. Computations grow more expensive as the depth increases, so we recommend starting with *maxDepth*=1, that is, boosted stumps.
- *maxNumBins*: controls the maximum number of bins used for each variable. Managing the number of bins is important in controlling memory usage. The default is to use the larger of 101 and the square root of the number of observations for small to moderate size data sets (up to about one million observations), but for larger sets to use 1001 bins. For small data sets with continuous predictors, you may find that you need to increase the *maxNumBins* to obtain models that resemble *rpart*.
- *minSplit*, *minBucket*: determine how many observations must be in a node before a split is attempted

(*minSplit*) and how many must remain in a terminal node (*minBucket*).

Naïve Bayes Classifier using RevoScaleR

7/12/2022 • 6 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

In this article, we describe one simple and effective family of classification methods known as Naïve Bayes. In RevoScaleR, Naïve Bayes classifiers can be implemented using the *rxNaiveBayes* function. Classification, simply put, is the act of dividing observations into classes or categories. Some examples of this are the classification of product reviews into positive or negative categories or the detection of email spam. These classification examples can be achieved manually using a set of rules. However, this is not efficient or scalable. In Naïve Bayes and other machine learning based classification algorithms, the decision criteria for assigning class are learned from a training data set, which has classes assigned manually to each observation.

The *rxNaiveBayes* Algorithm

The Naïves Bayes classification method is simple, effective, and robust. This method can be applied to data large or small, it requires minimal training data, and is unlikely to produce a classifier that performs poorly compared to more complex algorithms. This family of classifiers utilizes Bayes Theorem to determine the probability that an observation belongs to a certain class. A training dataset is used to calculate prior probabilities of an observation occurring in a class within the predefined set of classes. In RevoScaleR this is done using the *rxNaiveBayes* function. These probabilities are then used to calculate posterior probabilities that an observation belongs to each class. The class membership is decided by choosing the class with the largest posterior probability for each observation. This is accomplished with the *rxPredict* function using the Naïve Bayes object from a call to *rxNaiveBayes*. Part of the beauty of Naïve Bayes is its simplicity due to the conditional independent assumption: that the values of each predictor are independent of each other given the class. This assumption reduces the number of parameters needed and in turn makes the algorithm extremely efficient. Naïve Bayes methods differ in their choice of distribution for any continuous independent variables. Our implementation via the *rxNaiveBayes* function assumes the distribution to be Gaussian.

A Simple Naïve Bayes Classifier

In [Logistic Regression Models](#), we fit a simple logistic regression model to rpart's kypnosis data and in [Decision Trees](#) and [Decision Forests](#) we used the kypnosis data again to create classification and regression trees. We can use the same data with our Naïve Bayes classifier to see which patients are more likely to acquire Kypnosis based on age, number, and start. We can train and test our classifier on the kypnosis data for the sake of illustration. We use the *rxNaiveBayes* function to construct a classifier for the kypnosis data:

```

# A Simple Naïve Bayes Classifier

data("kyphosis", package="rpart")
kyphNaiveBayes <- rxNaiveBayes(Kyphosis ~ Age + Start + Number, data = kyphosis)
kyphNaiveBayes

Call:
rxNaiveBayes(formula = Kyphosis ~ Age + Start + Number, data = kyphosis)

A priori probabilities:
Kyphosis
absent   present
0.7901235 0.2098765

Predictor types:
Variable Type
1      Age numeric
2     Start numeric
3    Number numeric

Conditional probabilities:
$Age
  Means StdDev
absent 79.89062 61.86111
present 97.82353 39.27505

$Start
  Means StdDev
absent 12.609375 4.427967
present 7.294118 4.283175

$Number
  Means StdDev
absent 3.750000 1.414214
present 5.176471 1.878673

```

The returned object `kyphNaiveBayes` is an object of class `rxNaiveBayes`. Objects of this class provide the following useful components: *apriori* and *tables*. The *apriori* component contains the conditional probabilities for the response variable, in this case the Kyphosis variable. The *tables* component contains a list of tables, one for each predictor variable. For a categorical variable, the table contains the conditional probabilities of the variable given the target level of the response variable. For a numeric variable, the table contains the mean and standard deviation of the variable given the target level of the response variable. These components are printed in the output above.

We can use our Naïve Bayes object with `rxPredict` to re-classify the Kyphosis variable for each child in our original dataset:

```
kyphPred <- rxPredict(kyphNaiveBayes, kyphosis)
```

When we table the results from the Naïve Bayes classifier with the original Kyphosis variable, it appears that 13 of 81 children are misclassified:

```
table(kyphPred[["Kyphosis_Pred"]], kyphosis[["Kyphosis"]])
```

absent	present
absent	59 8
present	5 9

A Larger Naïve Bayes Classifier

As a more complex example, consider the mortgage default example. For that example, there are ten input files

total and we use nine input data files to create the training data set. We then use the model built from those files to make predictions on the final dataset. In this section we will use the same strategy to build a Naïve Bayes classifier on the first nine data sets and assign the outcome variable for the tenth data set.

The mortgage default data sets are available for download [online](#). With the data downloaded we can create the training data set and test data set as follows (remember to modify the first line to match the location of the mortgage default text data files on your own system):

```
# A Larger Naïve Bayes Classifier

bigDataDir <- "C:/MRS/Data"
mortCsvDataName <- file.path(bigDataDir, "mortDefault", "mortDefault")
trainingDataFileName <- "mortDefaultTraining"
mortCsv2009 <- paste(mortCsvDataName, "2009.csv", sep = "")
targetDataFileName <- "mortDefault2009.xdf"
defaultLevels <- as.character(c(0,1))
ageLevels <- as.character(c(0:40))
yearLevels <- as.character(c(2000:2009))
colInfo <- list(list(name = "default", type = "factor",
  levels = defaultLevels), list(name = "houseAge", type = "factor",
  levels = ageLevels), list(name = "year", type = "factor",
  levels = yearLevels))
append= FALSE
for (i in 2000:2008)
{
  importFile <- paste(mortCsvDataName, i, ".csv", sep = "")
  rxImport(inData = importFile, outFile = trainingDataFileName,
  colInfo = colInfo, append = append, overwrite=TRUE)
  append = TRUE
}

rxImport(inData = mortCsv2009, outFile = targetDataFileName,
  colInfo = colInfo)
```

In the above code the response variable *default* is converted to a factor using the *colInfo* argument to *rxImport*. For the *rxNaiveBayes* function, the response variable must be a factor or you will get an error.

Now that we have training and test data sets we can fit a Naïve Bayes classifier with our training data using *rxNaiveBayes* and assign values of the *default* variable for observations within the test data using *rxPredict*.

```
mortNB <- rxNaiveBayes(default ~ year + creditScore + yearsEmploy + ccDebt,
  data = trainingDataFileName, smoothingFactor = 1)
mortNBPred <- rxPredict(mortNB, data = targetDataFileName)
```

Notice that we added an additional argument, *smoothingFactor*, to our *rxNaiveBayes* call. This is a useful argument when your data are missing levels of a certain variable that you expect to be in your test data. Based on our training data, the conditional probability for year 2009 will be 0, since it only includes data between the years of 2000 and 2008. If we try to use our classifier on the test data without specifying a smoothing factor in our call to *rxNaiveBayes* the function *rxPredict* produces no results since our test data only has data from 2009. In general, smoothing is used to avoid overfitting your model. It follows that to achieve the optimal classifier you may want to smooth the conditional probabilities even if every level of each variable is observed.

We can compare the predicted values of the *default* variable from the Naïve Bayes classifier with the actual data in the test dataset:

```
results <- table(mortNBPred[["default_Pred"]], rxDataStep(targetDataFileName,
  maxRowsByCols=6000000)[["default"]])
results

  0      1
0 877272  3792
1 97987   20949

pctMisclassified <- sum(results[2:3])/sum(results)*100
pctMisclassified

[1] 10.1779
```

These results demonstrate a 10.2% misclassification rate using our Naïve Bayes classifier.

Naïve Bayes with Missing Data

You can control the handling of missing data using the *byTerm* argument in the *rxNaiveBayes* function. By default, *byTerm* is set to *TRUE*, which means that missings are removed by variable before computing the conditional probabilities. If you prefer to remove observations with missings in any variable before computations are done, set the *byTerm* argument to *FALSE*.

Cluster classification in RevoScaleR

7/12/2022 • 4 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Clustering is the general name for any of a large number of classification techniques that involve assigning observations to membership in one of two or more clusters on the basis of some distance metric.

K-means Clustering

K-means clustering is a classification technique that groups observations of numeric data using one of several *iterative relocation* algorithms. Starting from some initial classification, which may be random, points are moved from cluster to another so as to minimize sums of squares. In RevoScaleR, the algorithm used is that of Lloyd.

To perform k-means clustering with RevoScaleR, use the *rxKmeans* function.

Clustering the Airline Data

As a first example of k-means clustering, we will cluster the arrival delay and scheduled departure time in the airline data 7% subsample. To start, we extract variables of interest into a new working data set to which we are writing additional information:

```
# K-means Clustering

# Clustering the Airline Data
bigDataDir <- "C:/MRS/Data"
sampleAirData <- file.path(bigDataDir, "AirOnTime7Pct.xdf")
rxDataStep(inData = sampleAirData, outFile = "AirlineDataClusterVars.xdf",
  varsToKeep=c("DayOfWeek", "ArrDelay", "CRSDepTime", "DepDelay"))
```

We specify the variables to cluster as a formula, and specify the number of clusters we'd like. Initial centers for these clusters are then chosen at random.

```
kclusts1 <- rxKmeans(formula= ~ArrDelay + CRSDepTime,
  data = "AirlineDataClusterVars.xdf",
  seed = 10,
  outFile = "airlineDataClusterVars.xdf", numClusters=5)
kclusts1
```

This produces the following output (because the initial centers are chosen at random, your output will probably look different):

```

Call:
rxKmeans(formula = ~ArrDelay + CRSDepTime, data = "AirlineDataClusterVars.xdf",
outFile = "AirlineDataClusterVars.xdf", numClusters = 5)

Data: "AirlineDataClusterVars.xdf"
Number of valid observations: 10186272
Number of missing observations: 213483
Clustering algorithm:

K-means clustering with 5 clusters of sizes 922985, 38192, 4772791, 261779, 4190525

Cluster means:
  ArrDelay CRSDepTime
1 45.258179 14.86596
2 275.363820 14.81432
3 -10.284426 13.08375
4 118.365205 15.52079
5  7.803893 13.53811

Within cluster sum of squares by cluster:
      1       2       3       4       5 
223220709 501736748 354763376 233533349 312403604

Available components:
[1] "centers"        "size"          "withinss"       "valid.obs"
[5] "missing.obs"    "numIterations" "tot.withinss"  "totss"
[9] "betweenss"      "cluster"       "params"        "formula"
[13] "call"

```

The value returned by `rxKmeans` is a list similar to the list returned by the standard R `kmeans` function. The printed output shows a subset of this information, including the number of valid and missing observations, the cluster sizes, the cluster centers, and the within-cluster sums of squares.

The cluster membership component is returned if the input is a data frame, but if the input is a .xdf file, cluster membership is returned only if `outFile` is specified, in which case it is returned not as part of the return object, but as a column in the specified file. In our example, we specified an `outFile`, and we see the cluster membership variable when we look at the file with `rxGetInfo`:

```

rxGetInfo("AirlineDataClusterVars.xdf", getVarInfo=TRUE)
File name: AirlineDataClusterVars.xdf
Number of observations: 10399755
Number of variables: 5
Number of blocks: 19
Compression type: zlib
Variable information:
Var 1: DayOfWeek
  7 factor levels: Mon Tues Wed Thur Fri Sat Sun
Var 2: ArrDelay, Type: integer, Low/High: (-1233, 2453)
Var 3: CRSDepTime, Type: numeric, Storage: float32, Low/High: (0.0000, 24.0000)
Var 4: DepDelay, Type: integer, Low/High: (-1199, 2467)
Var 5: .rxCluster, Type: integer, Low/High: (1, 5)

```

Using the Cluster Membership Information

A common follow-up to clustering is to use the cluster membership information to see whether a given model varies appreciably from cluster to cluster. Since we can use the `rowSelection` argument to extract a single cluster on the fly, there is no need to sort the data first. As an example, we fit our original linear model of `ArrDelay` by `DayOfWeek` for two of the clusters:

```
# Using the Cluster Membership Information

clust1Lm <- rxLinMod(ArrDelay ~ DayOfWeek, "AirlineDataClusterVars.xdf",
  rowSelection = .rxCluste r == 1 )
clust5Lm <- rxLinMod(ArrDelay ~ DayOfWeek, "AirlineDataClusterVars.xdf",
  rowSelection = .rxCluster == 5)
summary(clust1Lm)
summary(clust5Lm)
```

Looking at the summary for *clust1Lm* shows the following:

```
Call:
rxLinMod(formula = ArrDelay ~ DayOfWeek, data = "AirlineDataClusterVars.xdf",
  rowSelection = .rxCluster == 1)

Linear Regression Results for: ArrDelay ~ DayOfWeek
File name: AirlineDataClusterVars.xdf
Dependent variable(s): ArrDelay
Total independent variables: 8 (Including number dropped: 1)
Number of valid observations: 922985
Number of missing observations: 0

Coefficients: (1 not defined because of singularities)
  Estimate Std. Error t value Pr(>|t|)
(Intercept)  45.21591   0.04237 1067.199 2.22e-16 ***
DayOfWeek=Mon  0.23053   0.05893   3.912 9.16e-05 ***
DayOfWeek=Tues -0.06496   0.05968  -1.089  0.2764
DayOfWeek=Wed   0.10139   0.05869   1.727  0.0841 .
DayOfWeek=Thur   0.06098   0.05708   1.068  0.2854
DayOfWeek=Fri    0.23222   0.05660   4.103 4.08e-05 ***
DayOfWeek=Sat   -0.43444   0.06364  -6.827 8.68e-12 ***
DayOfWeek=Sun    Dropped    Dropped   Dropped   Dropped
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 14.89 on 922978 degrees of freedom
Multiple R-squared:  0.0001705
Adjusted R-squared:  0.000164
F-statistic: 26.24 on 6 and 922978 DF,  p-value: < 2.2e-16
Condition number: 12.8655
```

Similarly, the summary for *clust5Lm* shows the following:

```

Call:
rxLinMod(formula = ArrDelay ~ DayOfWeek, data = "AirlineDataClusterVars.xdf",
rowSelection = .rxCluster == 5)

Linear Regression Results for: ArrDelay ~ DayOfWeek
File name: AirlineDataClusterVars.xdf
Dependent variable(s): ArrDelay
Total independent variables: 8 (Including number dropped: 1)
Number of valid observations: 4190525
Number of missing observations: 0

Coefficients: (1 not defined because of singularities)
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 7.808093  0.009593 813.960 2.22e-16 ***
DayOfWeek=Mon -0.131001  0.013320 -9.835 2.22e-16 ***
DayOfWeek=Tues -0.228087  0.013374 -17.055 2.22e-16 ***
DayOfWeek=Wed -0.035954  0.013292 -2.705  0.00683 **
DayOfWeek=Thur  0.231958  0.013170 17.613 2.22e-16 ***
DayOfWeek=Fri   0.313961  0.013171 23.838 2.22e-16 ***
DayOfWeek=Sat  -0.257716  0.014036 -18.361 2.22e-16 ***
DayOfWeek=Sun    Dropped   Dropped Dropped Dropped
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 7.238 on 4190518 degrees of freedom
Multiple R-squared: 0.0007911
Adjusted R-squared: 0.0007897
F-statistic: 553 on 6 and 4190518 DF, p-value: < 2.2e-16
Condition number: 12.0006

```

Estimating Correlation and Variance/Covariance Matrices

7/12/2022 • 12 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

The `rxCovCor` function in RevoScaleR calculates the covariance, correlation, or sum of squares/cross-product matrix for a set of variables in a .xdf file or data frame. The size of these matrices is determined by the number of variables rather than the number of observations, so typically the results can easily fit into memory in R. A broad category of analyses can be computed from some form of a cross-product matrix, for example, factor analysis and principal components.

A cross-product matrix is a matrix of the form $X'X$, where X represents an arbitrary set of raw or standardized variables. More generally, this matrix is of the form $X'WX$, where W is a diagonal weighting matrix.

Computing Cross-Product Matrices

While `rxCovCor` is the primary tool for computing covariance, correlation, and other cross-product matrices, you will seldom call it directly. Instead, it is generally simpler to use one of the following convenience functions:

- `rxCov`: Use `rxCov` to return the covariance matrix
- `rxCor`: Use `rxCor` to return the correlation matrix
- `rxSSCP`: Use `rxSSCP` to return the augmented cross-product matrix, that is, we first add a column of 1's (if no weights are specified) or a column equaling the square root of the weights to the data matrix, and then compute the cross-product.

Computing a Correlation Matrix for Use in Factor Analysis

The 5% sample of the U.S. 2000 census has over 14 million observations. In this example, we compute the correlation matrix for 16 variables derived from variables in the data set for individuals over the age of 20. This correlation matrix is then used as input into the standard R factor analysis function, `factanal`.

First, we specify the name and location of the data set:

```
# Computing a Correlation Matrix for Use in Factor Analysis

bigDataDir <- "C:/MRS/Data"
bigCensusData <- file.path(bigDataDir, "Census5PCT2000.xdf")
```

Next, we can take a quick look at some basic demographic and socio-economic variables by calling `rxSummary`. (For more information on variables in the census data, see <http://usa.ipums.org/usa-action/variables/group>.) Throughout the analysis we use the probability weighting variable, `perwt`, and restrict the analysis to people over the age of 20.

The `blocksPerRead` argument is ignored if run locally using R Client. [Learn more...](#)

```

rxSummary(~phone + speakeng + wkswork1 + incwelfr + incss + educrec + metro +
ownershd + marst + lingisol + nfams + yrsusa1 + movedin + racwht + age,
data = bigCensusData, blocksPerRead = 5, pweights = "perwt",
rowSelection = age > 20)

```

This call provides summary information about the variables in this weighted subsample, including cell counts for factor variables:

```

Call:
rxSummary(formula = ~phone + speakeng + wkswork1 + incwelfr +
incss + educrec + metro + ownershd + marst + lingisol + nfams +
yrsusa1 + movedin + racwht + age, data = censusData, pweights = "perwt",
rowSelection = age > 20, blocksPerRead = 5)

```

```

Summary Statistics Results for: ~phone + speakeng + wkswork1 + incwelfr
+ incss + educrec + metro + ownershd + marst + lingisol + nfams +
yrsusa1 + movedin + racwht + age
File name: C:\MRS\Data\Census5PCT2000.xdf
Probability weights: perwt
Number of valid observations: 9822124

```

Name	Mean	StdDev	Min	Max	SumOfWeights	MissingWeights
wkswork1	32.068473	23.2438663	0	52	196971131	0
incwelfr	61.155293	711.0955602	0	25500	196971131	0
incss	1614.604835	3915.7717233	0	26800	196971131	0
nfams	1.163434	0.5375238	1	48	196971131	0
yrsusa1	2.868573	9.0098343	0	90	196971131	0
age	46.813005	17.1797905	21	93	196971131	0

Category Counts for phone

Number of categories: 3

phone	Counts
N/A	5611380
No, no phone available	3957030
Yes, phone available	187402721

Category Counts for speakeng

Number of categories: 10

speakeng	Counts
N/A (Blank)	0
Does not speak English	2956934
Yes, speaks English...	0
Yes, speaks only English	162425091
Yes, speaks very well	17664738
Yes, speaks well	7713303
Yes, but not well	6211065
Unknown	0
Illegible	0
Blank	0

Category Counts for educrec

Number of categories: 10

educrec	Counts
N/A or No schooling	0
None or preschool	2757363
Grade 1, 2, 3, or 4	1439820
Grade 5, 6, 7, or 8	10180870
Grade 9	4862980
Grade 10	5957922
Grade 11	5763456
Grade 12	63691961
1 to 3 years of college	55834997
4+ years of college	46481762

Category Counts for metro

Number of categories: 5

metro	Counts
Not applicable	13829398
Not in metro area	32533836
In metro area, central city	32080416
In metro, area, outside central city	60836302
Central city status unknown	57691179

Category Counts for ownershd

Number of categories: 8

ownershd	Counts
N/A	5611380
Owned or being bought	0
Check mark (owns?)	0
Owned free and clear	40546259
Owned with mortgage or loan	94626060
Rents	0
No cash rent	3169987
With cash rent	53017445

Category Counts for marst

Number of categories: 6

marst	Counts
Married, spouse present	112784037
Married, spouse absent	5896245
Separated	4686951
Divorced	21474299
Widowed	14605829
Never married/single (N/A)	37523770

Category Counts for lingisol

Number of categories: 3

lingisol	Counts
N/A (group quarters/vacant)	5611380
Not linguistically isolated	182633786
Linguistically isolated	8725965

Category Counts for movedin

Number of categories: 7

movedin	Counts
NA	92708540
This year or last year	20107246
2-5 years ago	30328210
6-10 years ago	16959897
11-20 years ago	16406155
21-30 years ago	10339278
31+ years ago	10121805

Category Counts for racwht

Number of categories: 2

racwht	Counts
No	40684944
Yes	156286187

Next we call the *rxCor* function, a convenience function for *rxCovCor* that returns just the Pearson's correlation matrix for the variables specified. We make heavy use of the *transforms* argument to create a series of logical (or dummy) variables from factor variables to be used in the creation of the correlation matrix.

```

censusCor <- rxCor(formula=~poverty + noPhone + noEnglish + onSocialSecurity +
onWelfare + working + incearn + noHighSchool + inCity + renter +
noSpouse + langIsolated + multFamilies + newArrival + recentMove +
white + sei + older,
data = bigCensusData, pweightsb= "perwt", blocksPerRead = 5,
rowSelection = age > 20,
transforms= list(
  noPhone = phone == "No, no phone available",
  noEnglish = speakeng == "Does not speak English",
  working = wkswork1 > 20,
  onWelfare = incwelfr > 0,
  onSocialSecurity = incss > 0,
  noHighSchool =
    !(educrec %in%
      c("Grade 12", "1 to 3 years of college", "4+ years of college")),
  inCity = metro == "In metro area, central city",
  renter = ownershd %in% c("No cash rent", "With cash rent"),
  noSpouse = marst != "Married, spouse present",
  langIsolated = lingisol == "Linguistically isolated",
  multFamilies = nfams > 2,
  newArrival = yrsusa2 == "0-5 years",
  recentMove = movedin == "This year or last year",
  white = racwht == "Yes",
  older = age > 64
))

```

The resulting correlation matrix is used as input into the factor analysis function provided by the stats package in R. In interpreting the results, the variable *poverty* represents family income as a percentage of a poverty threshold, so increases as family income increases. First, specify two factors:

```

censusFa <- factanal(covmat = censusCor, factors=2)
print(censusFa, digits=2, cutoff = .2, sort= TRUE)

```

Results in:

```
Call:  
factanal(factors = 2, covmat = censusCor)
```

Uniquenesses:

	poverty	noPhone	noEnglish	onSocialSecurity
onWelfare	0.53	0.96	0.93	0.23
multFamilies	0.96	working	incearn	noHighSchool
sei	0.96	0.51	0.68	0.82
inCity	0.97	renter	noSpouse	langIsolated
sei	0.97	0.79	0.90	0.90
newArrival	0.96	recentMove		white
older	0.57	0.93	0.97	0.88
		0.24		

Loadings:

	Factor1	Factor2
onSocialSecurity	0.83	-0.29
working	-0.68	
sei	-0.59	-0.29
older	0.82	-0.29
poverty	-0.28	-0.63
noPhone		
noEnglish		0.26
onWelfare		
incearn	-0.46	-0.34
noHighSchool	0.31	0.29
inCity		
renter		0.45
noSpouse		0.28
langIsolated		0.31
multFamilies		
newArrival		0.26
recentMove		
white		-0.34

	Factor1	Factor2
SS loadings	2.60	1.67
Proportion Var	0.14	0.09
Cumulative Var	0.14	0.24

The degrees of freedom for the model is 118 and the fit was 0.6019

We can use the same correlation matrix to estimate three factors:

```

censusFa <- factanal(covmat = censusCor, factors=3)
print(censusFa, digits=2, cutoff = .2, sort= TRUE)

Call:
factanal(factors = 3, covmat = censusCor)

Uniquenesses:
      poverty      noPhone      noEnglish  onSocialSecurity
      0.49          0.96          0.72          0.24
      onWelfare     working     inearn      noHighSchool
      0.96          0.50          0.62          0.80
      inCity        renter      noSpouse    langIsolated
      0.96          0.80          0.90          0.59
      multFamilies newArrival   recentMove   white
      0.96          0.79          0.97          0.88
      sei           older
      0.56          0.22

Loadings:
      Factor1 Factor2 Factor3
onSocialSecurity  0.87
working          -0.62   0.34
sei              -0.51   0.42
older             0.88
poverty           0.68
inearn            -0.36   0.51
noEnglish         0.53
langIsolated     0.64
noPhone
onWelfare         -0.21
noHighSchool     0.25   -0.26   0.26
inCity
renter            -0.36   0.24
noSpouse          -0.30
multFamilies
newArrival        0.46
recentMove
white             0.24   -0.24

      Factor1 Factor2 Factor3
SS loadings       2.41   1.50   1.18
Proportion Var    0.13   0.08   0.07
Cumulative Var   0.13   0.22   0.28

```

The degrees of freedom for the model is 102 and the fit was 0.343

Computing A Covariance Matrix for Principal Components Analysis

Principal components analysis, or PCA, is a technique closely related to factor analysis. PCA seeks to find a set of orthogonal axes such that the first axis, or *first principal component*, accounts for as much variability as possible, and subsequent axes or components are chosen to maximize variance while maintaining orthogonality with previous axes. Principal components are typically computed either by a singular value decomposition of the data matrix or an eigenvalue decomposition of a covariance or correlation matrix; the latter permits us to use *rxCor* and its relatives with the standard R function *princomp*.

As an example, we use the *rxCor* function to calculate a covariance matrix for the log of the classic iris data, and pass the matrix to the *princomp* function (reproduced from [Modern Applied Statistics with S](#)):

```
# Computing A Covariance Matrix for Principal Components Analysis

irisLog <- as.data.frame(lapply(iris[,1:4], log))
irisSpecies <- iris[,5]
irisCov <- rxCor(~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
  data=irisLog)
irisPca <- princomp(covmat=irisCov, cor=TRUE)
summary(irisPca)
```

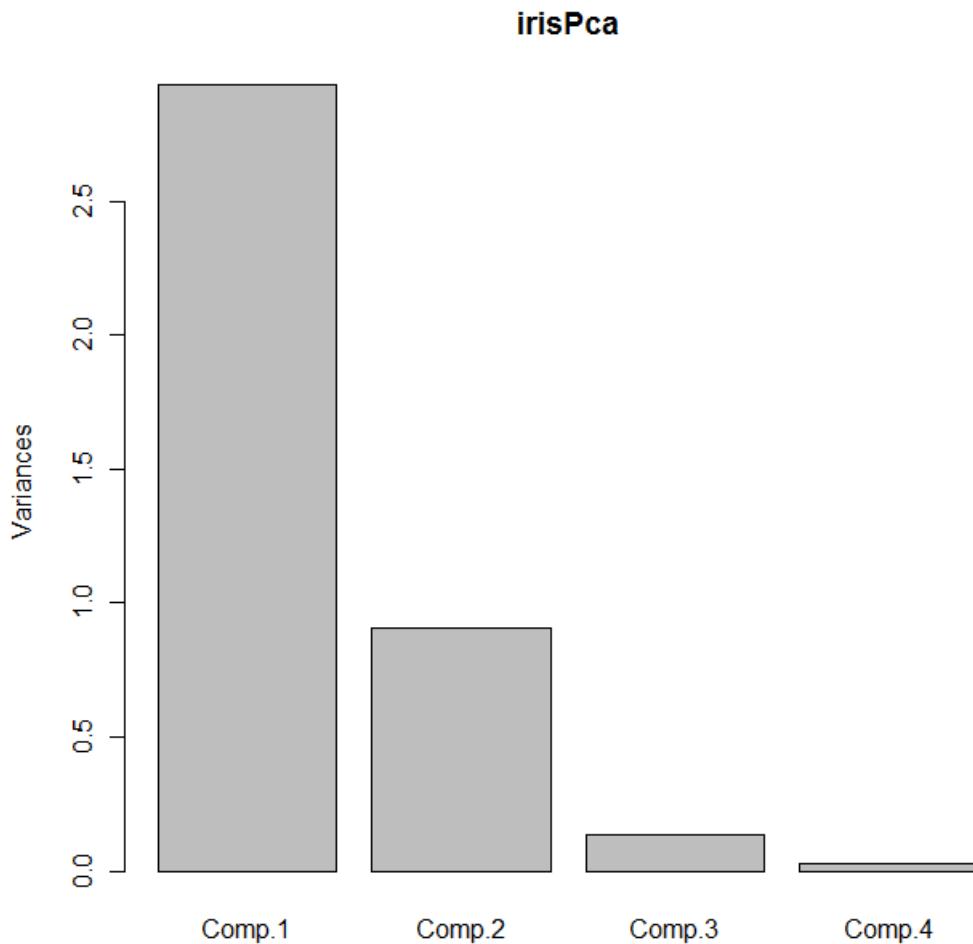
Yields the following:

```
Importance of components:
   Comp.1    Comp.2    Comp.3    Comp.4
Standard deviation     1.7124583 0.9523797 0.36470294 0.1656840
Proportion of Variance 0.7331284 0.2267568 0.03325206 0.0068628
Cumulative Proportion  0.7331284 0.9598851 0.99313720 1.0000000
```

The default plot method for objects of class `princomp` is a *scree plot*, which is a barplot of the variances of the principal components. We can obtain the plot as usual by calling `plot` with our principal components object:

```
plot(irisPca)
```

Yields the following plot:



Another useful bit of output is given by the `loadings` function, which returns a set of columns showing the linear combinations for each principal component:

```

loadings(irisPca)

Loadings:
  Comp.1 Comp.2 Comp.3 Comp.4
Sepal.Length  0.504 -0.455  0.709  0.191
Sepal.Width   -0.302 -0.889 -0.331
Petal.Length  0.577      -0.219 -0.786
Petal.Width   0.567      -0.583  0.580

  Comp.1 Comp.2 Comp.3 Comp.4
SS loadings     1.00    1.00    1.00    1.00
Proportion Var  0.25    0.25    0.25    0.25
Cumulative Var 0.25    0.50    0.75    1.00

```

You may have noticed that we supplied the flag `cor=TRUE` in the call to `princomp`; this flag tells `princomp` to use the correlation matrix rather than the covariance matrix to compute the principal components. We can obtain the same results by omitting the flag but submitting the correlation matrix as returned by `rxCor` instead:

```

irisCor <- rxCor(~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
  data=irisLog)
irisPca2 <- princomp(covmat=irisCor)
summary(irisPca2)
loadings(irisPca2)
plot(irisPca2)

```

A Large Data Principal Components Analysis

Stock market data for open, high, low, close, and adjusted close from 1962 to 2010 is available at https://github.com/thebigjc/HackReduce/blob/master/datasets/nyse/daily_prices/NYSE_daily_prices_subset.csv. The full data set includes 9.2 million observations of daily open-high-low-close data for some 2800 stocks. As you might expect, these data are highly correlated, and principal components analysis can be used for data reduction. We read the original data into a .xdf file, `NYSE_daily_prices.xdf`, using the same process we used in the [Tutorial: Analyzing loan data with RevoScaleR](#) to read our mortgage data (set `revoDataDir` to the full path to the NYSE directory containing the .csv files when you unpack the download):

```

# A Large Data Principal Components Analysis

if (bHasNYSE){

bigDataDir <- "C:/MRS/Data"
nyseCsvFiles <- file.path(bigDataDir, "NYSE_daily_prices","NYSE",
  "NYSE_daily_prices_")

nyseXdf <- "NYSE_daily_prices.xdf"
append <- "none"
for (i in LETTERS)
{
  importFile <- paste(nyseCsvFiles, i, ".csv", sep="")
  rxImport(importFile, nyseXdf, append=append)
  append <- "rows"
}

```

Once we have our .xdf file, we proceed as before:

```

stockCor <- rxCor(~ stock_price_open + stock_price_high +
  stock_price_low + stock_price_close +
  stock_price_adj_close, data="NYSE_daily_prices.xdf")
stockPca <- princomp(covmat=stockCor)
summary(stockPca)
loadings(stockPca)
plot(stockPca)

```

Yields the following output:

```

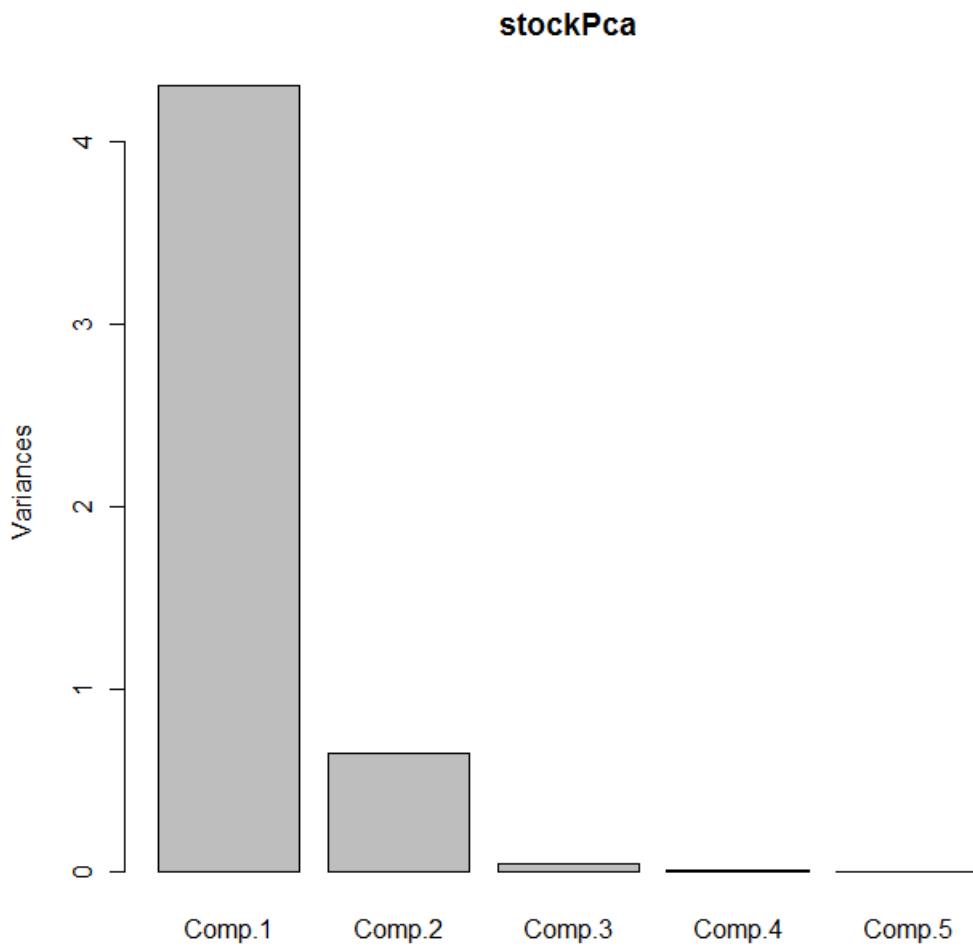
> summary(stockPca)
Importance of components:
            Comp.1    Comp.2    Comp.3    Comp.4
Standard deviation     2.0756631 0.8063270 0.197632281 0.0454173922
Proportion of Variance 0.8616755 0.1300327 0.007811704 0.0004125479
Cumulative Proportion  0.8616755 0.9917081 0.999519853 0.9999324005
            Comp.5
Standard deviation     1.838470e-02
Proportion of Variance 6.759946e-05
Cumulative Proportion  1.000000e+00
> loadings(stockPca)

Loadings:
            Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
stock_price_open      -0.470 -0.166  0.867
stock_price_high      -0.477 -0.151 -0.276  0.410 -0.711
stock_price_low       -0.477 -0.153 -0.282  0.417  0.704
stock_price_close     -0.477 -0.149 -0.305 -0.811
stock_price_adj_close -0.309  0.951

            Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
SS loadings        1.0    1.0    1.0    1.0    1.0
Proportion Var     0.2    0.2    0.2    0.2    0.2
Cumulative Var    0.2    0.4    0.6    0.8    1.0

```

The scree plot is shown as follows:



Between them, the first two principal components explain 99% of the variance; we can therefore replace the five original variables by these two principal components with no appreciable loss of information.

Ridge Regression

Another application of correlation matrices is to calculate ridge regression, a type of regression that can help deal with multicollinearity and is part of a broader class of models called Penalized Regression Models.

Where the ordinary least squares regression minimizes the sum of squared residuals

$$\sum_i (y_i - x_i^T \beta)^2$$

ridge regression minimizes the slightly modified sum

$$\sum_i (y_i - x_i^T \beta)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

The solution to the ridge regression is

$$\hat{\beta} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

where \mathbf{X} is the model matrix. This matrix is similar to the ordinary least squares regression solution with a "ridge" added along the diagonal.

Since the model matrix is embedded in the correlation matrix, the following function allows us to compute the

ridge regression solution:

```
# Ridge regression
rxRidgeReg <- function(formula, data, lambda, ...) {
  myTerms <- all.vars(formula)
  newForm <- as.formula(paste("~", paste(myTerms, collapse = "+")))
  myCor <- rxCorCor(newForm, data = data, type = "Cor", ...)
  n <- myCor$valid.obs
  k <- nrow(myCor$CovCor) - 1
  bridgeprime <- do.call(rbind, lapply(lambda,
    function(l) qr.solve(myCor$CovCor[-1,-1] + l*diag(k),
      myCor$CovCor[-1,1])))
  bridge <- myCor$StdDevs[1] * sweep(bridgeprime, 2,
    myCor$StdDevs[-1], "/")
  bridge <- cbind(t(myCor$Means[1] -
    tcrossprod(myCor$Means[-1], bridge)), bridge)
  rownames(bridge) <- format(lambda)
  return(bridge)
}
```

The following example shows how ridge regression dramatically improves the solution in a regression involving heavy multicollinearity:

```
set.seed(14)
x <- rnorm(100)
y <- rnorm(100, mean=x, sd=.01)
z <- rnorm(100, mean=2 + x + y)
data <- data.frame(x=x, y=y, z=z)
lm(z ~ x + y)

Call:
lm(formula = z ~ x + y)

Coefficients:
(Intercept)          x            y
1.827        4.359       -2.584
rxRidgeReg(z ~ x + y, data=data, lambda=0.02)
x            y
0.02 1.827674 0.8917924 0.8723334
```

You can supply a vector of lambdas as a quick way to compare various choices:

```
rxRidgeReg(z ~ x + y, data=data, lambda=c(0, 0.02, 0.2, 2, 20))
           x            y
0.00 1.827112 4.35917238 -2.58387778
0.02 1.827674 0.89179239  0.87233344
0.20 1.833899 0.81020130  0.80959911
2.00 1.865339 0.44512940  0.44575060
20.00 1.896779 0.08092296  0.08105322
```

For $\lambda = 0$, the ridge regression is identical to ordinary least squares, while as $\lambda \rightarrow \infty$, the coefficients of x and y approach 0.

How to use RevoScaleR in a Spark compute context

7/12/2022 • 29 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

This article provides a step-by-step introduction to using the [RevoScaleR functions](#) in Apache Spark running on a Hadoop cluster. You can use a small built-in sample dataset to complete the walkthrough, and then step through tasks again using a larger dataset.

- Download sample data
- Start Revo64
- Create a compute context for Spark
- Copy a data set into HDFS
- Create a data source
- Summarize your data
- Fit a linear model to the data

Fundamentals

In a Spark cluster, you typically connect to Machine Learning Server on the edge node for most of your work, writing and running script in a local [compute context](#), using client tools and the RevoScaleR engine on that machine. Your script calls the RevoScaleR functions to execute scalable and high-performance data management, analysis, and visualization functions. As with any script using RevoScaleR, you can call base R functions, as well as other functions available in packages you have installed separately. The RevoScaleR engine is built on the R interpreter, extending but not breaking any core functions you are accustomed to using.

To load data and run analyses on worker nodes, you set the compute context in your script to [RxSpark](#). In this context, RevoScaleR functions automatically distribute the workload across all the worker nodes, with no built-in requirement for managing jobs or the queue.

Internally, RevoScaleR in an RxSpark compute context uses the Hadoop infrastructure, including the Yarn scheduler, HDFS, and of course, Spark as the processing framework. Alternatively, if project requirements dictate manual overrides, RevoScaleR provides functions for manual data allocation and scheduled processing across selected nodes.

How RevoScaleR distributes jobs in Spark and Hadoop

In a Spark cluster, the RevoScaleR analysis functions go through the following steps:

- A master process is initiated to run the main thread of the algorithm.
- The master process initiates a Spark job to make a pass through the data.
- Spark worker produces an intermediate results object for each task processing a chunk of data. These are then combined and returned to master node.
- The master process examines the results. For iterative algorithms, it decides if another pass through the data is required. If so, it initiates another Spark job and repeats.
- When complete, the final results are computed and returned.

When running on Spark, the RevoScaleR analysis functions process data contained in the Hadoop Distributed File System (HDFS). HDFS data can also be accessed directly from RevoScaleR, without performing the computations within the Hadoop framework. An example can be found in [Using a Local Compute Context with HDFS Data](#).

NOTE

For a comprehensive list of functions for the Spark or Hadoop compute contexts, see [RevoScaleR Functions for Hadoop](#). For a list of which functions support distributed computing, see [Running distributed analyses using RevoScaleR](#).

1 - Download sample data

Sample data is required if you intend to follow the steps. This walkthrough starts with the AirlineDemoSmall.csv file from the local RevoScaleR SampleData directory.

Optionally, you can graduate to a second series of tasks using a larger dataset. The *Airline 2012 On-Time Data Set* consists of 12 comma-separated files containing information on flight arrival and departure details for all commercial flights within the USA, for the year 2012. This is a big data set with over six million observations.

To download the larger dataset, go to <https://packages.revolutionanalytics.com/datasets/>.

2 - Start Revo64

Machine Learning Server for Hadoop runs on Linux. On the edge node, start an R session by typing `Revo64` at the shell prompt: \$ `Revo64`

3 - Spark compute context

The Spark compute context is established through RxSpark or rxSparkConnect() to create the Spark compute context, and uses rxSparkDisconnect() to return to a local compute context.

The rxSparkConnect() function maintains a persistent Spark session and results in more efficient execution of RevoScaleR functions for this context.

In the Revo64 command line, define a Spark compute context using default values:

```
myHadoopCluster <- ()
```

The default settings include a specification of `/var/RevoShare/$USER` as the `shareDir` and `/user/RevoShare/$USER` as the `hdfsShareDir`. These are the default locations for writing files to the native local file system and HDFS file system, respectively. These directories must be writable for your cluster jobs to succeed. You must either create these directories or specify suitable writable directories for these parameters. If you are working on a node of the cluster, the default specifications for the shared directories are:

```
myShareDir = paste( "/var/RevoShare", Sys.info()[["user"]],  
sep="/" )  
myHdfsShareDir = paste( "/user/RevoShare", Sys.info()[["user"]],  
sep="/" )
```

TIP

You can have multiple compute context objects available for use, but you can only use one at a time. Specify the active compute context using the `rxSetComputeContext` function: `rxSetComputeContext(myHadoopCluster2)`

Defining a Compute Context on a High-Availability Cluster

On a Hadoop cluster configured for high-availability, you must specify the node providing the name service using the `nameNode` argument to `RxSpark`, and also specify the Hadoop port with the `port` argument:

```
myHadoopCluster <- RxSpark(nameNode = "my-name-service-server", port = 8020)
```

4 - Copy data to HDFS

Start with the built-in data set, `AirlineDemoSmall.csv` provided with RevoScaleR. You can verify that it is on your local system as follows:

```
file.exists(system.file("SampleData/AirlineDemoSmall.csv",
  package="RevoScaleR"))
[1] TRUE
```

To use this file in our distributed computations, it must first be copied to Hadoop Distributed File System (HDFS). For our examples, we make extensive use of the HDFS shared directory, `/share`:

```
bigDataDirRoot <- "/share" # HDFS location of the example data
```

First, check to see what directories and files are already in your shared file directory. You can use the `rxHadoopListFiles` function, which will automatically check your active compute context for information:

```
rxHadoopListFiles(bigDataDirRoot)
```

If the `AirlineDemoSmall` subdirectory does not exist and you have write permission, you can use the following functions to copy the data there:

```
source <- system.file("SampleData/AirlineDemoSmall.csv",
  package="RevoScaleR")
inputDir <- file.path(bigDataDirRoot, "AirlineDemoSmall")
rxHadoopMakeDir(inputDir)
rxHadoopCopyFromLocal(source, inputDir)
```

We can then verify that it exists as follows:

```
rxHadoopListFiles(inputDir)
```

5 - Create a data source

In a Spark compute context, you can create data sources using the following functions:

FUNCTION	DESCRIPTION
<code>RxTextData</code>	A comma-delimited text data source.
<code>RxDfData</code>	Data in the XDF data file format. In RevoScaleR, the XDF file format is modified for Hadoop to store data in a composite set of files rather than a single file.

FUNCTION	DESCRIPTION
RxHiveData	Generates a Hive Data Source object.
RxParquetData	Generates a Parquet Data Source object.
RxOrcData	Generates an Orc Data Source object.

In this step, create an RxTextData object using the text file you just copied to HDFS. First, create a file system object that uses the default values:

```
hdfsFS <- RxHdfsFileSystem()
```

The input .csv file uses the letter M to represent missing values, rather than the default NA, so we specify this with the *missingValueString* argument. We will explicitly set the factor levels for *DayOfWeek* in the desired order using the *colInfo* argument:

```
colInfo <- list(DayOfWeek = list(type = "factor",
  levels = c("Monday", "Tuesday", "Wednesday", "Thursday",
  "Friday", "Saturday", "Sunday")))

airDS <- RxTextData(file = inputDir, missingValueString = "M",
  colInfo = colInfo, fileSystem = hdfsFS)
```

6 - Summarize data

Use the *rxSummary* function to obtain descriptive statistics for your data. The *rxSummary* function takes a formula as its first argument, and the name of the data set as the second.

```
adsSummary <- rxSummary(~ArrDelay+CRSDepTime+DayOfWeek,
  data = airDS)
adsSummary
```

Summary statistics are computed on the variables in the formula, removing missing values for all rows in the included variables:

```

Call:
rxSummary(formula = ~ArrDelay + CRSDepTime + DayOfWeek, data = airDS)
Summary Statistics Results for: ~ArrDelay + CRSDepTime + DayOfWeek
Data: airDS (RxTextData Data Source)
File name: /share/AirlineDemoSmall
Number of valid observations: 6e+05

      Name      Mean    StdDev     Min      Max      ValidObs MissingObs
ArrDelay   11.31794 40.688536 -86.000000 1490.00000 582628    17372
CRSDepTime 13.48227  4.697566  0.016667  23.98333 600000        0

Category Counts for DayOfWeek
Number of categories: 7
Number of valid observations: 6e+05
Number of missing observations: 0

DayOfWeek Counts
Monday    97975
Tuesday   77725
Wednesday 78875
Thursday  81304
Friday    82987
Saturday  86159
Sunday    94975

```

Notice that the summary information shows cell counts for categorical variables, and appropriately does not provide summary statistics such as *Mean* and *StdDev*. Also notice that the *Call:* line shows the actual call you entered or the call provided by *summary*, so appear differently in different circumstances.

You can also compute summary information by one or more categories by using interactions of a numeric variable with a factor variable. For example, to compute summary statistics on Arrival Delay by Day of Week:

```
rxSummary(~ArrDelay:DayOfWeek, data = airDS)
```

The output shows the summary statistics for *ArrDelay* for each day of the week:

```

Call:
rxSummary(formula = ~ArrDelay:DayOfWeek, data = airDS)

Summary Statistics Results for: ~ArrDelay:DayOfWeek
Data: airDS (RxTextData Data Source)
File name: /share/AirlineDemoSmall
Number of valid observations: 6e+05

      Name      Mean    StdDev     Min Max  ValidObs MissingObs
ArrDelay:DayOfWeek 11.31794 40.68854 -86 1490 582628    17372

Statistics by category (7 categories):
Category          DayOfWeek Means    StdDev   Min Max  ValidObs
ArrDelay for DayOfWeek=Monday Monday 12.025604 40.02463 -76 1017 95298
ArrDelay for DayOfWeek=Tuesday Tuesday 11.293808 43.66269 -70 1143 74011
ArrDelay for DayOfWeek=Wednesday Wednesday 10.156539 39.58803 -81 1166 76786
ArrDelay for DayOfWeek=Thursday Thursday 8.658007 36.74724 -58 1053 79145
ArrDelay for DayOfWeek=Friday Friday 14.804335 41.79260 -78 1490 80142
ArrDelay for DayOfWeek=Saturday Saturday 11.875326 45.24540 -73 1370 83851
ArrDelay for DayOfWeek=Sunday Sunday 10.331806 37.33348 -86 1202 93395

```

7 - Fit a linear model

Use the *rxLinMod* function to fit a linear model using your *airDS* data source. Use a single dependent variable,

the factor *DayOfWeek*:

```
arrDelayLm1 <- rxLinMod(ArrDelay ~ DayOfWeek, data = airDS)
summary(arrDelayLm1)
```

The resulting output is:

```
Call:
rxLinMod(formula = ArrDelay ~ DayOfWeek, data = airDS)

Linear Regression Results for: ArrDelay ~ DayOfWeek
Data: airDS (RxTextData Data Source)
File name: /share/AirlineDemoSmall
Dependent variable(s): ArrDelay
Total independent variables: 8 (Including number dropped: 1)
Number of valid observations: 582628
Number of missing observations: 17372

Coefficients: (1 not defined because of singularities)
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   10.3318    0.1330  77.673 2.22e-16 ***
DayOfWeek=Monday 1.6938    0.1872   9.049 2.22e-16 ***
DayOfWeek=Tuesday 0.9620    0.2001   4.809 1.52e-06 ***
DayOfWeek=Wednesday -0.1753   0.1980  -0.885    0.376
DayOfWeek=Thursday -1.6738   0.1964  -8.522 2.22e-16 ***
DayOfWeek=Friday   4.4725    0.1957  22.850 2.22e-16 ***
DayOfWeek=Saturday 1.5435    0.1934   7.981 2.22e-16 ***
DayOfWeek=Sunday    Dropped   Dropped Dropped Dropped
---
Signif. codes:  0 `***` 0.001 `**` 0.01 `*` 0.05 `.` 0.1 ` ' 1

Residual standard error: 40.65 on 582621 degrees of freedom
Multiple R-squared:  0.001869
Adjusted R-squared:  0.001858
F-statistic: 181.8 on 6 and 582621 DF,  p-value: < 2.2e-16
```

More Spark scenarios

Use Machine Learning Server as a Hadoop Client

A common practice is to do exploratory analysis on your laptop, then deploy the same analytics code on a Hadoop cluster. The underlying RevoScaleR engine in Machine Learning Server handles the distribution of the computations across cores and nodes automatically.

If you are running Machine Learning Server from Linux or from a Windows computer equipped with PuTTY *and/or* both the Cygwin shell and Cygwin OpenSSH packages, you can create a compute context that runs RevoScaleR functions from your local client in a distributed fashion on your Hadoop cluster. You use *RxSpark* to create the compute context, but use additional arguments to specify your user name, the file-sharing directory where you have read and write access, the publicly facing host name, or IP address of your Hadoop cluster's name node or an edge node that run the master processes, and any additional switches to pass to the ssh command (such as the *-i* flag if you are using a pem or ppk file for authentication, or *-p* to specify a non-standard ssh port number). For example:

```

mySshUsername <- "user1"
#public facing cluster IP address
mySshHostname <- "12.345.678.90"
mySshSwitches <- "-i /home/yourName/user1.pem" #See NOTE below
myShareDir <- paste("/var/RevoShare", mySshUsername, sep ="/")
myHdfsShareDir <- paste("/user/RevoShare",mySshUsername, sep="/")

myHadoopCluster <- RxSpark(
  hdfsShareDir = myHdfsShareDir,
  shareDir     = myShareDir,
  sshUsername  = mySshUsername,
  sshHostname  = mySshHostname,
  sshSwitches  = mySshSwitches)

```

NOTE

if you are using a pem or ppk file for authentication the permissions of the file must be modified to ensure that only the owner has full read and write access (that is, chmod go-rwx user1.pem).

If you are using PuTTY, you may incorporate the publicly facing host name and any authentication requirements into a PuTTY saved session, and use the name of that saved session as the sshHostname. For example:

```

mySshUsername <- "user1"
#name of PuTTY saved session
mySshHostname <- "myCluster"
myShareDir <- paste("/var/RevoShare", mySshUsername, sep ="/")
myHdfsShareDir <- paste("/user/RevoShare",mySshUsername, sep="/")

myHadoopCluster <- RxSpark(
  hdfsShareDir = myHdfsShareDir,
  shareDir     = myShareDir,
  sshUsername  = mySshUsername,
  sshHostname  = mySshHostname)

```

The above assumes that the directory containing the ssh and scp commands (Linux/Cygwin) or plink and pscp commands (PuTTY) is in your path (or that the Cygwin installer has stored its directory in the Windows registry). If not, you can specify the location of these files using the sshClientDir argument:

```

myHadoopCluster <- RxSpark(
  hdfsShareDir = myHdfsShareDir,
  shareDir     = myShareDir,
  sshUsername  = mySshUsername,
  sshHostname  = mySshHostname,
  sshClientDir = "C:\\\\Program Files (x86)\\\\PuTTY")

```

In some cases, you may find that environment variables needed by Hadoop are not set in the remote sessions run on the sshHostname computer. This may be because a different profile or startup script is being read on ssh login. You can specify the appropriate profile script by specifying the sshProfileScript argument to RxSpark; this should be an absolute path:

```

myHadoopCluster <- RxSpark(
  hdfsShareDir = myHdfsShareDir,
  shareDir     = myShareDir,
  sshUsername  = mySshUsername,
  sshHostname  = mySshHostname,
  sshProfileScript = "/home/user1/.bash_profile",
  sshSwitches  = mySshSwitches)

```

Now that you have defined your compute context, make it the active compute context using the `rxSetComputeContext` function:

```
rxSetComputeContext(myHadoopCluster)
```

Use RxSpark Compute Context in Persistent Mode

By default, with RxSpark Compute Context, a new Spark application is launched when a job starts and is terminated when the job completes. To avoid the overhead of Spark initialization on launching time, the `persistentRun` mode (experimental) is introduced. If you set `persistentRun` as "TRUE", the Spark application (and associated processes) persists across jobs until `idleTimeout` or the `rxStopEngine` is called explicitly:

```
myHadoopCluster <- RxSpark(persistentRun = TRUE, idleTimeout = 600)
```

The `idleTimeout` is the number of seconds of being idle before the system kills the Spark process.

If `persistentRun` mode is enabled, then the RxSpark compute context cannot be a Non-Waiting Compute Context. See [Non-Waiting Compute Context](#).

Use a local compute context

At times, it may be more efficient to perform smaller computations on the local node rather than using Spark. You can easily do this, accessing the same data from the HDFS file system. When working with the local compute context, you need to specify the name of a specific data file. The same basic code is then used to run the analysis; simply change the compute context to a local context. (Note that this will not work if you are accessing the Hadoop cluster via a client.)

```
rxSetComputeContext("local")
inputFile <- file.path(bigDataDirRoot, "AirlineDemoSmall/AirlineDemoSmall.csv")
airDSLocal <- RxTextData(file = inputFile,
  missingValueString = "M",
  colInfo = colInfo, fileSystem = hdfsFS)
adsSummary <- rxSummary(~ArrDelay+CRSDepTime+DayOfWeek,
  data = airDSLocal)
adsSummary
```

The results are the same as doing the computations across the nodes with the RxSpark compute context:

```

Call:
rxSummary(formula = ~ArrDelay + CRSDepTime + DayOfWeek, data = airDS)

Summary Statistics Results for: ~ArrDelay + CRSDepTime + DayOfWeek
Data: airDS (RxTextData Data Source)
File name: /share/AirlineDemoSmall/AirlineDemoSmall.csv
Number of valid observations: 6e+05

      Name      Mean    StdDev     Min      Max      ValidObs MissingObs
ArrDelay   11.31794 40.688536 -86.000000 1490.00000 582628    17372
CRSDepTime 13.48227  4.697566  0.016667  23.98333 600000        0

Category Counts for DayOfWeek
Number of categories: 7
Number of valid observations: 6e+05
Number of missing observations: 0

DayOfWeek Counts
Monday    97975
Tuesday   77725
Wednesday 78875
Thursday  81304
Friday    82987
Saturday  86159
Sunday    94975

```

Now set the compute context back to the Hadoop cluster for further analyses:

```
rxSetComputeContext(myHadoopCluster)
```

Create a Non-Waiting Compute Context

When running all but the shortest analyses in Hadoop, it can be convenient to let Hadoop do its processing while returning control of your R session to you immediately. You can do this by specifying `wait = FALSE` in your compute context definition. By using our existing compute context as the first argument, all of the other settings will be carried over to the new compute context:

```
myHadoopNoWaitCluster <- RxSpark(myHadoopCluster, wait = FALSE)
rxSetComputeContext(myHadoopNoWaitCluster)
```

Once you have set your compute context to non-waiting, distributed **RevoScaleR** functions return relatively quickly with a `jobInfo` object, which you can use to track the progress of your job, and, when the job is complete, obtain the results of the job. For example, we can rerun our linear model in the non-waiting case as follows:

```
job1 <- rxLinMod(ArrDelay ~ DayOfWeek, data = airDS)
rxGetJobStatus(job1)
```

Right after submission, the job status will typically return `"running"`. When the job status returns `"finished"`, you can obtain the results using `rxGetJobResults` as follows:

```
arrDelayLm1 <- rxGetJobResults(job1)
summary(arrDelayLm1)
```

You should always assign the `jobInfo` object so that you can easily track your work, but if you forget, the most recent `jobInfo` object is saved in the global environment as the object `rxgLastPendingJob`. (By default, after you've retrieved your job results, the results are removed from the cluster. To have your job results remain, set

the `autoCleanup` argument to `FALSE` in `RxSpark`.)

If after submitting a job in a non-waiting compute context, you decide you don't want to complete the job, you can cancel the job using the `rxCancelJob` function:

```
job2 <- rxLinMod(ArrDelay ~ DayOfWeek, data = airDS)
rxCancelJob(job2)
```

The `rxCancelJob` function returns `TRUE` if the job is successfully canceled, `FALSE` otherwise.

For the remainder of this tutorial, we return to a waiting compute context:

```
rxSetComputeContext(myHadoopCluster)
```

Analyze a Large Data Set

Set up the sample airline dataset

We will now move to examples using a much larger version of the airline data set. The [airline on-time data](#) has been gathered by the Department of Transportation since 1987. The data through 2008 was used in the American Statistical Association Data Expo in 2009 (<http://stat-computing.org/dataexpo/2009/>). ASA describes the data set as follows:

The data consists of flight arrival and departure details for all commercial flights within the USA, from October 1987 to April 2008. This is a large dataset: there are nearly 120 million records in total, and takes up 1.6 gigabytes of space compressed and 12 gigabytes when uncompressed.

The airline on-time data set for 2012, consisting of 12 separate CSV files, is available [online](#). We assume you have uploaded them to the `/tmp` directory on your edge node or a name node (although any directory visible as a native file system directory from the name node will work.)

As before, our first step is to copy the data into HDFS. We specify the location of your Hadoop-stored data for the `airDataDir` variable:

```
airDataDir <- file.path(bigDataDirRoot, "/airOnTime12/CSV")
rxHadoopMakeDir(airDataDir)
rxHadoopCopyFromLocal("/tmp/airOT2012*.csv", airDataDir)
```

The original CSV files have rather unwieldy variable names, so we supply a `collInfo` list to make them more manageable (we won't use all of these variables in this manual, but you use the data sources created in this manual as you continue to explore distributed computing in the [RevoScaleR Distributed Computing Guide](#):

```

airlineColInfo <- list(
  MONTH = list(newName = "Month", type = "integer"),
  DAY_OF_WEEK = list(newName = "DayOfWeek", type = "factor",
    levels = as.character(1:7),
    newLevels = c("Mon", "Tues", "Wed", "Thur", "Fri", "Sat",
      "Sun")),
  UNIQUE_CARRIER = list(newName = "UniqueCarrier", type =
    "factor"),
  ORIGIN = list(newName = "Origin", type = "factor"),
  DEST = list(newName = "Dest", type = "factor"),
  CRS_DEP_TIME = list(newName = "CRSDepTime", type = "integer"),
  DEP_TIME = list(newName = "DepTime", type = "integer"),
  DEP_DELAY = list(newName = "DepDelay", type = "integer"),
  DEP_DELAY_NEW = list(newName = "DepDelayMinutes", type =
    "integer"),
  DEP_DEL15 = list(newName = "DepDel15", type = "logical"),
  DEP_DELAY_GROUP = list(newName = "DepDelayGroups", type =
    "factor",
    levels = as.character(-2:12),
    newLevels = c("< -15", "-15 to -1", "0 to 14", "15 to 29",
      "30 to 44", "45 to 59", "60 to 74",
      "75 to 89", "90 to 104", "105 to 119",
      "120 to 134", "135 to 149", "150 to 164",
      "165 to 179", ">= 180")),
  ARR_DELAY = list(newName = "ArrDelay", type = "integer"),
  ARR_DELAY_NEW = list(newName = "ArrDelayMinutes", type =
    "integer"),
  ARR_DEL15 = list(newName = "ArrDel15", type = "logical"),
  AIR_TIME = list(newName = "AirTime", type = "integer"),
  DISTANCE = list(newName = "Distance", type = "integer"),
  DISTANCE_GROUP = list(newName = "DistanceGroup", type =
    "factor",
    levels = as.character(1:11),
    newLevels = c("< 250", "250-499", "500-749", "750-999",
      "1000-1249", "1250-1499", "1500-1749", "1750-1999",
      "2000-2249", "2250-2499", ">= 2500")))
varNames <- names(airlineColInfo)

```

We create a data source using these definitions as follows:

```

hdfsFS <- RxHdfsFileSystem()
bigAirDS <- RxTextData( airDataDir,
  colInfo = airlineColInfo,
  varsToKeep = varNames,
  fileSystem = hdfsFS )

```

Estimate a linear model with a big data set

We fit our first model to the large airline data much as we created the linear model for the AirlineDemoSmall data, and we time it to see how long it takes to fit the model on this large data set:

```

system.time(
  delayArr <- rxLinMod(ArrDelay ~ DayOfWeek, data = bigAirDS,
  cube = TRUE)
)

```

To see a summary of your results:

```

print(
  summary(delayArr)
)

```

You should see the following results:

```

Call:
rxLinMod(formula = ArrDelay ~ DayOfWeek, data = bigAirDS, cube = TRUE)

Cube Linear Regression Results for: ArrDelay ~ DayOfWeek
Data: bigAirDS (RxTextData Data Source)
File name: /share/airOnTime12/CSV
Dependent variable(s): ArrDelay
Total independent variables: 7
Number of valid observations: 6005381
Number of missing observations: 91381

Coefficients:
Estimate Std. Error t value Pr(>|t|)    | Counts
DayOfWeek=Mon   3.54210   0.03736  94.80 2.22e-16 *** | 901592
DayOfWeek=Tues  1.80696   0.03835  47.12 2.22e-16 *** | 855805
DayOfWeek=Wed   2.19424   0.03807  57.64 2.22e-16 *** | 868505
DayOfWeek=Thur  4.65502   0.03757 123.90 2.22e-16 *** | 891674
DayOfWeek=Fri   5.64402   0.03747 150.62 2.22e-16 *** | 896495
DayOfWeek=Sat   0.91008   0.04144  21.96 2.22e-16 *** | 732944
DayOfWeek=Sun   2.82780   0.03829  73.84 2.22e-16 *** | 858366
---
Signif. codes:  0 `***` 0.001 `**` 0.01 `*` 0.05 `.` 0.1 ` ' 1

Residual standard error: 35.48 on 6005374 degrees of freedom
Multiple R-squared:  0.001827 (as if intercept included)
Adjusted R-squared:  0.001826
F-statistic:  1832 on 6 and 6005374 DF,  p-value: < 2.2e-16
Condition number: 1

```

Notice that the results indicate we have processed all the data, six million observations, using all the .csv files in the specified directory. Notice also that because we specified `cube = TRUE`, we have an estimated coefficient for each day of the week (and not the intercept).

Import data as composite XDF files

As we have seen, you can analyze CSV files directly with RevoScaleR on Hadoop, but the analysis can be done more quickly if the data is stored in a more efficient format. The RevoScaleR .xdf format is extremely efficient, but is modified somewhat for HDFS so that individual files remain within a single HDFS block. (The HDFS block size varies from installation to installation but is typically either 64 MB or 128 MB.) When you use `rxImport` on Hadoop, you specify an RxTextData data source such as `bigAirDS` as the `inData` and an RxXdfData data source with `fileSystem` set to an HDFS file system as the `outFile` argument to create a set of *composite .xdf files*. The RxXdfData object can then be used as the `data` argument in subsequent RevoScaleR analyses.

For our airline data, we define our RxXdfData object as follows (the second “user” should be replaced by your actual user name on the Hadoop cluster):

```

bigAirXdfName <- "/user/RevoShare/user/AirlineOnTime2012"
airData <- RxXdfData( bigAirXdfName,
  fileSystem = hdfsFS )

```

We set a block size of 250000 rows and specify that we read all the data by specifying

```
numRowsToRead = -1:  
blockSize <- 250000  
numRowsToRead = -1
```

We then import the data using rxImport:

```
rxImport(inData = bigAirDS,  
        outFile = airData,  
        rowsPerRead = blockSize,  
        overwrite = TRUE,  
        numRows = numRowsToRead )
```

Write to CSV in HDFS

If you [converted your CSV to XDF](#) to take advantage of the efficiency while running analyses, but now wish to convert your data back to CSV you can do so using *rxDataStep*.

To create a folder of CSV files, first create an RxTextData object using a directory name as the file argument; this represents the folder in which to create the CSV files. This directory is created when you run the *rxDataStep*. Then, point to this *RxTextData* object in the *outFile* argument of the *rxDataStep*. Each CSV created will be named based on the directory name and followed by a number.

Suppose we want to write out a folder of CSV in HDFS from our airData composite XDF after we performed the logistic regression and prediction, so that the new CSV files contain the predicted values and residuals. We can do this as follows:

```
airDataCsvDir <- file.path("user/RevoShare/user/airDataCsv")  
airDataCsvDS <- RxTextData(airDataCsvDir, fileSystem=hdfsFS)  
rxDataStep(inData=airData, outFile=airDataCsvDS)
```

You notice that the *rxDataStep* wrote out one CSV for every .xdfd file in the input composite XDF file. This is the default behavior for writing CSV from composite XDF to HDFS when the compute context is set to RxSpark.

Alternatively, you could switch your compute context back to "local" when you are done performing your analyses and take advantage of two arguments within *RxTextData* that give you slightly more control when writing out CSV files to HDFS: *createFileSet* and *rowsPerOutfile*. When *createFileSet* is set to TRUE a folder of CSV files is written to the directory you specify. When *createFileSet* is set to FALSE a single CSV file is written. The second argument, *rowsPerOutfile*, can be set to an integer to indicate how many rows to write to each CSV file when *createFileSet* is TRUE. Returning to the example above, suppose we wanted to write out a folder of CSVs but we wanted to write out the airData but into only six CSV files.

```
rxSetComputeContext("local")  
airDataCsvRowsDir <- file.path("/user/RevoShare/MRS/airDataCsvRows")  
rxHadoopMakeDir(airDataCsvRowsDir)  
airDataCsvRowsDS <- RxTextData(airDataCsvRowsDir, fileSystem=hdfsFS, createFileSet=TRUE,  
rowsPerOutfile=1000000)  
rxDataStep(inData=airData, outFile=airDataCsvRowsDS)
```

When using an RxSpark compute context, *createFileSet* defaults to TRUE and *rowsPerOutfile* has no effect. Thus if you wish to create a single CSV or customize the number of rows per file you must perform the *rxDataStep* in a local compute context (data can still be in HDFS).

Estimate a linear model using composite XDF files

Now we can re-estimate the same linear model, using the new, faster data source:

```

system.time(
  delayArr <- rxLinMod(ArrDelay ~ DayOfWeek, data = airData,
  cube = TRUE)
)

```

To see a summary of your results:

```

print(
  summary(delayArr)
)

```

You should see the following results (which should match the results we found for the CSV files above):

```

Call:
rxLinMod(formula = ArrDelay ~ DayOfWeek, data = airData, cube = TRUE)

Cube Linear Regression Results for: ArrDelay ~ DayOfWeek
Data: airData (RxXdfData Data Source)
File name: /user/RevoShare/v7alpha/AirlineOnTime2012
Dependent variable(s): ArrDelay
Total independent variables: 7
Number of valid observations: 6005381
Number of missing observations: 91381

Coefficients:
Estimate Std. Error t value Pr(>|t|)    | Counts
DayOfWeek=Mon   3.54210   0.03736   94.80 2.22e-16 *** | 901592
DayOfWeek=Tues  1.80696   0.03835   47.12 2.22e-16 *** | 855805
DayOfWeek=Wed   2.19424   0.03807   57.64 2.22e-16 *** | 868505
DayOfWeek=Thur  4.65502   0.03757  123.90 2.22e-16 *** | 891674
DayOfWeek=Fri   5.64402   0.03747  150.62 2.22e-16 *** | 896495
DayOfWeek=Sat   0.91008   0.04144   21.96 2.22e-16 *** | 732944
DayOfWeek=Sun   2.82780   0.03829   73.84 2.22e-16 *** | 858366
---
Signif. codes:  0 `***` 0.001 `**` 0.01 `*` 0.05 `.` 0.1 ` ' 1

Residual standard error: 35.48 on 6005374 degrees of freedom
Multiple R-squared:  0.001827 (as if intercept included)
Adjusted R-squared: 0.001826
F-statistic: 1832 on 6 and 6005374 DF,  p-value: < 2.2e-16
Condition number: 1

```

Handling larger linear models

The data set contains a variable *UniqueCarrier*, which contains airline codes for 15 carriers. Because the RevoScaleR Compute Engine handles factor variables so efficiently, we can do a linear regression looking at the Arrival Delay by Carrier. This takes a little longer than the previous analysis, because we are estimating 15 instead of 7 parameters.

```

delayCarrier <- rxLinMod(ArrDelay ~ UniqueCarrier,
  data = airData, cube = TRUE)

```

Next, sort the coefficient vector so that we can see the airlines with the highest and lowest values.

```

dcCoef <- sort(coef(delayCarrier))

```

Next, see how the airlines rank in arrival delays:

```

print(
  dcCoef
)

```

The result is:

```

UniqueCarrier=AS UniqueCarrier=US UniqueCarrier=DL UniqueCarrier=FL
-1.9022483 -1.2418484 -0.8013952 -0.2618747
UniqueCarrier=HA UniqueCarrier=YV UniqueCarrier=WN UniqueCarrier=MQ
0.3143564 1.3189086 2.8824285 2.8843612
UniqueCarrier=OO UniqueCarrier=VX UniqueCarrier=B6 UniqueCarrier=UA
3.9846305 4.0323447 4.8508215 5.0905867
UniqueCarrier=AA UniqueCarrier=EV UniqueCarrier=F9
6.3980937 7.2934991 7.8788931

```

Notice that Alaska Airlines comes in as the best in on-time arrivals, while Frontier is at the bottom of the list. We can see the difference by subtracting the coefficients:

```

print(
  sprintf("Frontier's additional delay compared with Alaska: %f",
    dcCoef["UniqueCarrier=F9"]-dcCoef["UniqueCarrier=AS"])
)

```

which results in:

```
[1] "Frontier's additional delay compared with Alaska: 9.781141"
```

A Large Data Logistic Regression

Our airline data includes a logical variable, ArrDel15, that tells whether a flight is 15 or more minutes late. We can use this as the response for a logistic regression to model the likelihood that a flight will be late given the day of the week:

```

logitObj <- rxLogit(ArrDel15 ~ DayOfWeek, data = airData)
logitObj

```

which results in:

```

Logistic Regression Results for: ArrDel15 ~ DayOfWeek
Data: airData (RxXdfData Data Source)
File name: AirOnTime2012.xdf
Dependent variable(s): ArrDel15
Total independent variables: 8 (Including number dropped: 1)
Number of valid observations: 6005381
Number of missing observations: 91381

Coefficients:
  ArrDel15
(Intercept) -1.61122563
DayOfWeek=Mon 0.04600367
DayOfWeek=Tues -0.08431814
DayOfWeek=Wed -0.07319133
DayOfWeek=Thur 0.12585721
DayOfWeek=Fri 0.18987931
DayOfWeek=Sat -0.13498591
DayOfWeek=Sun Dropped

```

Prediction on Large Data

You can predict (or score) from a fitted model on Hadoop using *rxPredict*. In this example, we compute predicted values and their residuals for the logistic regression in the previous section. These new prediction variables are output to a new composite XDF in HDFS. (As in an earlier example, the second “user” in the *airDataPredDir* path should be changed to the actual user name of your Hadoop user.)

```
airDataPredDir <- "/user/RevoShare/user/airDataPred"  
airDataPred <- RxXdfData(airDataPredDir, fileSystem=hdfsFS)  
rxPredict(modelObject=logitObj, data=airData, outData=airDataPred,  
computeResiduals=TRUE)
```

By putting in a call to *rxGetVarInfo* we see that two variables, *ArrDel15_Pred* and *ArrDel15_Resid* were written to the *airDataPred* composite XDF. If in addition to the prediction variables, we wanted to have the variables used in the model written to our *outData* we would need to add the *writeModelVars=TRUE* to our *rxPredict* call.

Alternatively, we can update the *airData* to include the predicted values and residuals by not specifying an *outData* argument, which is *NULL* by default. Since the *airData* composite XDF already exists, we would need to add *overwrite=TRUE* to the *rxPredict* call.

An *RxTextData* object can be used as the input data for *rxPredict* on Hadoop, but only XDF can be written to HDFS. When using a CSV file or directory of CSV files as the input data to *rxPredict* the *outData* argument must be set to an *RxXdfData* object.

Performing Data Operations on Large Data

To create or modify data in HDFS on Hadoop, we can use the *rxDataStep* function. Suppose we want to repeat the analyses with a “cleaner” version of the large airline dataset. To do this we keep only the flights having arrival delay information and flights that did not depart more than one hour early. We can put a call to *rxDataStep* to output a new composite XDF to HDFS. (As in an earlier example, the second “user” in the *newAirDir* path should be changed to the actual user name of your Hadoop user.)

```
newAirDir <- "/user/RevoShare/user/newAirData"  
newAirXdf <- RxXdfData(newAirDir, fileSystem=hdfsFS)  
  
rxDataStep(inData = airData, outFile = newAirXdf,  
rowSelection = !is.na(ArrDelay) & (DepDelay > -60))
```

To modify an existing composite XDF using *rxDataStep* set the *overwrite* argument to *TRUE* and either omit the *outFile* argument or set it to the same data source specified for *inData*.

Parallel Partial Decision Forests

As mentioned earlier, both *rxDForest* and *rxBTrees* are available on Hadoop--these provide two different methods for fitting classification and regression decision forests. In the default case, both algorithms generate multiple stages in the Spark job, and thus can tend to incur significant overhead, particularly with smaller data sets. However, the *scheduleOnce* argument to both functions allows the computation to be performed via *rxExec*, which generates only a single stage in the Spark job, and thus incurs significantly less overhead. When using the *scheduleOnce* argument, you can specify the number of trees to be grown within each *rxExec* task using the forest function’s *nTree* argument together with *rxExec*’s *rxElemArgs* function, as in the following regression example using the built-in claims data:

```

file.name <- "claims.xdf"
sourceFile <- system.file(file.path("SampleData", file.name),
  package="RevoScaleR")
inputDir <- "/share/claimsXdf"
rxHadoopMakeDir(inputDir)
rxHadoopCopyFromLocal(sourceFile, inputDir)
input <- RxXdfData(file.path(inputDir, file.name,
  fsep="/"), fileSystem=hdfsFS)

partial.forests <- rxDForest(formula = age ~ car.age +
  type + cost + number, data = input,
  minSplit = 5, maxDepth = 2,
  nTree = rxElemArg(rep(2,8)), seed = 0,
  maxNumBins = 200, computeOobError = -1,
  reportProgress = 2, verbose = 0,
  scheduleOnce = TRUE))

```

Equivalently, you can use `nTree` together with `rxExec`'s `timesToRun` argument:

```

partial.forests <- rxDForest(formula = age ~ car.age +
  type + cost + number, data = input,
  minSplit = 5, maxDepth = 2,
  nTree = 2, seed = 0,
  maxNumBins = 200, computeOobError = -1,
  reportProgress = 2, verbose = 0,
  scheduleOnce = TRUE, timesToRun = 8)

```

In this example, using `scheduleOnce` can be up to 45 times faster than the default, and so we recommend it for decision forests with small to moderate-sized data sets.

Similarly, the `rxPredict` methods for `rxDForest` and `rxBTrees` objects include the `scheduleOnce` argument, and should be used when using these methods on small to moderate-sized data sets.

Using Hive data

There are multiple ways to access and use data from Hive for analyses with RevoScaleR. Here are some general recommendations, assuming in each case that the data for analysis is defined by the results of a Hive query.

1. If running from a remote client or edge node, and the data is modest, then use `RxOdbcData` to stream results, or land them as XDF in the local file system, for subsequent analysis in a local compute context.
2. If the data is large, then use the Hive command-line interface (`hive` or `beeline`) from an edge node to run the Hive query with results spooled to a text file on HDFS for subsequent analysis in a distributed fashion using the HadoopMR or Spark compute contexts.

Here's how to get started with each of these approaches.

Accessing Hive data via ODBC

Start by following your Hadoop vendor's recommendations for accessing Hive via ODBC from a remote client or edge node. Once you have the prerequisite software installed and have run a smoke test to verify connectivity, then accessing data in Hive from Machine Learning Server is just like accessing data from any other data source.

```

mySQL = "SELECT * FROM CustData"
myDS <- RxOdbcData(sqlQuery = mySQL, connectionString = "DSN=HiveODBC")
xdfFile <- RxXdfData("dataFromHive.xdf")
rxImport(myDS, xdfFile, stringsAsFactors = TRUE, overwrite=TRUE)

```

Accessing Hive data via an Export to Text Files

This approach uses the Hive command-line interface to first spool the results to text files in the local or HDFS file

systems and then analyze them in a distributed fashion using local or HadoopMR/Spark compute contexts.

First, check with your Hadoop system administrator find out whether the ‘hive’ or ‘beeline’ command should be used to run a Hive query from the command line, and obtain the needed credentials for access.

Next, try out your hive/beeline command from the Linux command line to make sure it works and that you have the proper authentication and syntax.

Finally, run the command from within R using R’s system command.

Here are some sample R statements that output a Hive query to the local and HDFS file systems:

Run a Hive query and dump results to a local text file (edge node)

```
system('hive -e "select * from emp" > myFile.txt')
```

Run the same query using beeline’s csv2 output format

```
system('beeline -u "jdbc:hive2:///" --outputformat=csv2 -e "select * from emp" > myFile.txt')
```

Run the same query but dump the results to CSV in HDFS

```
system('beeline -u "jdbc:hive2:///" -e "insert overwrite directory \'/your-hadoop-dir\' row format  
delimited fields terminated by \',\' select * from emp"')
```

After you’ve exported the query results to a text file, it can be streamed directly as input to a RevoScaleR analysis routine via use of the RxTextdata data source, or imported to XDF for improved performance upon repeated access. Here’s an example assuming output was spooled as text to HDFS:

```
hiveOut <- file.path(bigDataDirRoot,"myHiveData")  
myHiveData <- RxTextData(file = hiveOut, fileSystem = hdfsFS)  
  
rxSummary(~var1 + var2+ var3+ var4, data=myHiveData)
```

Clean up Data

You can run the following commands to clean up data in this tutorial:

```
rxHadoopRemoveDir("/share/AirlineDemoSmall")  
rxHadoopRemoveDir("/share/airOnTime12")  
rxHadoopRemoveDir("/share/claimsXdf")  
rxHadoopRemoveDir("/user/RevoShare/<user>/AirlineOnTime2012")  
rxHadoopRemoveDir("/user/RevoShare/<user>/airDataCsv")  
rxHadoopRemoveDir("/user/RevoShare/<user>/airDataCsvRows")  
rxHadoopRemoveDir("/user/RevoShare/<user>/newAirData")  
rxHadoopRemoveDir("/user/RevoShare/<user>/airDataPred")
```

How to use RevoScaleR with Hadoop

7/12/2022 • 30 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

This guide is an introduction to using the [RevoScaleR functions](#) in an Apache Hadoop distributed computing environment. RevoScaleR functions offer scalable and extremely high-performance data management, analysis, and visualization. Hadoop provides a distributed file system and a MapReduce framework for distributed computation. This guide focuses on using ScaleR's big data capabilities with MapReduce.

While this guide is not a Hadoop tutorial, no prior experience in Hadoop is required to complete the tutorial. If you can connect to your Hadoop cluster, this guide walks you through the rest.

NOTE

The RxHadoopMR compute context for Hadoop MapReduce is deprecated. We recommend using [RxSpark](#) as a replacement. For guidance, see [How to use RevoScaleR in a Spark compute context](#).

Data Sources and Functions Supported in Hadoop

The **RevoScaleR** package provides a set of portable, scalable, distributable data analysis functions. Many functions are platform-agnostic; others are exclusive to the computing context, leveraging platform-specific capabilities for tasks like file management.

To perform an analysis using RevoScaleR functions, the user specifies three distinct pieces of information: where the computations should take place (the compute context), the data to use (the data source), and what analysis to perform (the analysis function).

This section briefly summarizes how functions are used. For a comprehensive list of functions for the Hadoop compute context, see [RevoScaleR Functions for Hadoop](#).

Compute Context and Supported Data Sources

The Hadoop compute context is established through *RxHadoopMR*, where the following two types of data sources can be used: a comma-delimited text data source (see *RxTextData*) and an efficient XDF data file format (see *RxXdfData*). In RevoScaleR, the XDF file format is modified for Hadoop to store data in a composite set of files rather than a single file. Both of these data sources can be used with the Hadoop Distributed File System (HDFS).

Data Manipulation and Computations

The data manipulation and computation functions in **RevoScaleR** are appropriate for small and large datasets, but are useful for these scenarios:

- Analyze data sets that are too large to fit in memory.
- Perform computations distributed over several cores, processors, or nodes in a cluster.
- Create scalable data analysis routines that can be developed locally with smaller data sets, then deployed to larger data and/or a cluster of computers.

These scenarios are ideal candidates for **RevoScaleR** because **RevoScaleR** is based on the concept of operating on chunks of data and using *updating algorithms*.

High Performance Analysis (HPA)

HPA functions in **RevoScaleR** do the heavy lifting in terms of data science. Most HPA functions are portable across multiple computing platforms, including Windows and RedHat Linux workstations and servers, and distributed computing platforms such as Hadoop. You can do exploratory analysis on your laptop, then deploy the same analytics code on a Hadoop cluster. The underlying **RevoScaleR** code handles the distribution of the computations across cores and nodes automatically.

How **RevoScaleR** distributes jobs in Hadoop

On Hadoop, the **RevoScaleR** analysis functions go through the following steps:

- A master process is initiated to run the main thread of the algorithm.
- The master process initiates a MapReduce job to make a pass through the data.
- The mapper produces “intermediate results objects” for each task processing a chunk of data. These are combined using a combiner and then a reducer.
- The master process examines the results. For iterative algorithms, it decides if another pass through the data is required. If so, it initiates another MapReduce job and repeats.
- When complete, the final results are computed and returned.

When running on Hadoop, the **RevoScaleR** analysis functions process data contained in the Hadoop Distributed File System (HDFS). HDFS data can also be accessed directly from **RevoScaleR**, without performing the computations within the Hadoop framework. An example is provided further on, in [Using a Local Compute Context with HDFS Data](#).

Tutorial steps

This tutorial introduces several high-performance analytics features of **RevoScaleR** using data stored on your Hadoop cluster and these tasks:

1. Start Machine Learning Server.
2. Create a compute context for Hadoop.
3. Copy a data set into the Hadoop Distributed File System.
4. Create a data source.
5. Summarize your data.
6. Fit a linear model to the data.

Check versions

Supported distributions of Hadoop are listed in [Supported platforms](#). For setup instructions, see [Install Machine Learning Server on Hadoop](#).

You can confirm the server version by typing `print(Revo.version)`.

Download sample data

Sample data is required when you intend to follow the steps. The tutorial uses the *Airline 2012 On-Time Data Set*, a set of 12 comma-separated files containing information on flight arrival and departure details for all commercial flights within the USA, for the year 2012. This is a big data set with over six million observations.

This tutorial also uses the `AirlineDemoSmall.csv` file from the **RevoScaleR** `SampleData` directory.

You can obtain both data sets [online](#).

Start Revo64

Machine Learning Server for Hadoop runs on Linux. On Linux hosts in a Hadoop cluster, start server session by typing `Revo64` at the shell prompt.

```
[<username>]$ cd MRSLinux90  
[<username> MRSLinux90]$ Revo64
```

You see a `>` prompt, indicating an Revo64 session, and the ability to issue RevoScaleR commands, including commands that set up the compute context.

Create a compute context

A *compute context* specifies the computing resources to be used by ScaleR's distributable computing functions. In this tutorial, we focus on using the nodes of the Hadoop cluster (internally via MapReduce) as the computing resources. In defining your compute context, you may have to specify different parameters depending on whether commands are issued from a node of your cluster or from a client accessing the cluster remotely.

Define a Compute Context on the Cluster

On a node of the Hadoop cluster (which may be an edge node), define a Hadoop MapReduce compute context using default values:

```
$ myHadoopCluster <- RxHadoopMR()
```

NOTE

The default settings include a specification of `/var/RevoShare/$USER` as the `shareDir` and `/user/RevoShare/$USER` as the `hdfsShareDir`. These are the default locations for writing various files on the cluster's local file system and HDFS file system, respectively. These directories must be writable for your cluster jobs to succeed. You must either create these directories or specify suitable writable directories for these parameters. If you are working on a node of the cluster, the default specifications for the shared directories are:

```
myShareDir = paste( "/var/RevoShare", Sys.info()[["user"]],  
sep="/" )  
myHdfsShareDir = paste( "/user/RevoShare", Sys.info()[["user"]],  
sep="/" )
```

You can have many compute context objects but only one is active at any one time. To specify the active compute context, use the `rxSetComputeContext` function:

```
> rxSetComputeContext(myHadoopCluster)
```

Define a Compute Context on a High-Availability Cluster

If you are running on a Hadoop cluster configured for high-availability, you must specify the node providing the name service using the `nameNode` argument to `RxHadoopMR`, and also specify the Hadoop port with the `port` argument:

```
myHadoopCluster <- RxHadoopMR(nameNode = "my-name-service-server",  
port = 8020)
```

Use Machine Learning Server as a Hadoop Client

If you are running Machine Learning Server from Linux or from a Windows computer equipped with PuTTY and/or both the Cygwin shell and Cygwin OpenSSH packages, you can create a compute context that runs RevoScaleR functions from your local client in a distributed fashion on your Hadoop cluster. You use

RxHadoopMR to create the compute context, but use additional arguments to specify your user name, the file-sharing directory where you have read and write access, the publicly facing host name, or IP address of your Hadoop cluster's name node or an edge node that run the master processes, and any additional switches to pass to the ssh command (such as the -i flag if you are using a pem or ppk file for authentication, or -p to specify a non-standard ssh port number). For example:

```
mySshUsername <- "user1"
#public facing cluster IP address
mySshHostname <- "12.345.678.90"
mySshSwitches <- "-i /home/yourName/user1.pem" #See NOTE below
myShareDir <- paste("/var/RevoShare", mySshUsername, sep ="/")
myHdfsShareDir <- paste("/user/RevoShare",mySshUsername, sep="/")

myHadoopCluster <- RxHadoopMR(
  hdfsShareDir = myHdfsShareDir,
  shareDir      = myShareDir,
  sshUsername   = mySshUsername,
  sshHostname   = mySshHostname,
  sshSwitches   = mySshSwitches)
```

NOTE

if you are using a pem or ppk file for authentication the permissions of the file must be modified to ensure that only the owner has full read and write access (that is, chmod go-rwx user1.pem).

If you are using PuTTY, you may incorporate the publicly facing host name and any authentication requirements into a PuTTY saved session, and use the name of that saved session as the sshHostname. For example:

```
mySshUsername <- "user1"
#name of PuTTY saved session
mySshHostname <- "myCluster"
myShareDir <- paste("/var/RevoShare", mySshUsername, sep ="/")
myHdfsShareDir <- paste("/user/RevoShare",mySshUsername, sep="/")

myHadoopCluster <- RxHadoopMR(
  hdfsShareDir = myHdfsShareDir,
  shareDir      = myShareDir,
  sshUsername   = mySshUsername,
  sshHostname   = mySshHostname)
```

The preceding assumes that the directory containing the ssh and scp commands (Linux/Cygwin) or plink and pscp commands (PuTTY) is in your path (or that the Cygwin installer has stored its directory in the Windows registry). If not, you can specify the location of these files using the sshClientDir argument:

```
myHadoopCluster <- RxHadoopMR(
  hdfsShareDir = myHdfsShareDir,
  shareDir      = myShareDir,
  sshUsername   = mySshUsername,
  sshHostname   = mySshHostname,
  sshClientDir = "C:\\\\Program Files (x86)\\\\PuTTY")
```

In some cases, you may find that environment variables needed by Hadoop are not set in the remote sessions run on the sshHostname computer. This may be because a different profile or startup script is being read on ssh login. You can specify the appropriate profile script by specifying the sshProfileScript argument to RxHadoopMR; this should be an absolute path:

```
myHadoopCluster <- RxHadoopMR(  
  hdfsShareDir = myHdfsShareDir,  
  shareDir     = myShareDir,  
  sshUsername  = mySshUsername,  
  sshHostname  = mySshHostname,  
  sshProfileScript = "/home/user1/.bash_profile",  
  sshSwitches  = mySshSwitches)
```

Now that you have defined your compute context, make it the active compute context using the `rxSetComputeContext` function:

```
rxSetComputeContext(myHadoopCluster)
```

Copy a data file

For our first explorations, we work with one of RevoScaleR's built-in data sets, `AirlineDemoSmall.csv`. This is part of the standard Machine Learning Server distribution. You can verify that it is on your local system as follows:

```
file.exists(system.file("SampleData/AirlineDemoSmall.csv",  
  package="RevoScaleR"))  
[1] TRUE
```

To use this file in our distributed computations, it must first be copied to HDFS. For our examples, we make extensive use of the HDFS shared director, `/share`:

```
bigDataDirRoot <- "/share" # HDFS location of the example data
```

First, check to see what directories and files are already in your shared file directory. You can use the `rxHadoopListFiles` function, which will automatically check your active compute context for information:

```
rxHadoopListFiles(bigDataDirRoot)
```

If the `AirlineDemoSmall` subdirectory does not exist and you have write permission, you can use the following functions to copy the data there:

```
source <- system.file("SampleData/AirlineDemoSmall.csv",  
  package="RevoScaleR")  
inputDir <- file.path(bigDataDirRoot, "AirlineDemoSmall")  
rxHadoopMakeDir(inputDir)  
rxHadoopCopyFromLocal(source, inputDir)
```

We can then verify that it exists as follows:

```
rxHadoopListFiles(inputDir)
```

Create a data source

We create a data source using this file, specifying that it is on the Hadoop Distributed File System. We first create a file system object that uses the default values:

```
hdfsFS <- RxHdfsFileSystem()
```

The input .csv file uses the letter M to represent missing values, rather than the default NA, so we specify this

with the *missingValueString* argument. We will explicitly set the factor levels for *DayOfWeek* in the desired order using the *colInfo* argument:

```
colInfo <- list(DayOfWeek = list(type = "factor",
  levels = c("Monday", "Tuesday", "Wednesday", "Thursday",
  "Friday", "Saturday", "Sunday")))

airDS <- RxTextData(file = inputDir, missingValueString = "M",
  colInfo = colInfo, fileSystem = hdfsFS)
```

Summarize data

Use the *rxSummary* function to obtain descriptive statistics for your data. The *rxSummary* function takes a formula as its first argument, and the name of the data set as the second.

```
adsSummary <- rxSummary(~ArrDelay+CRSDepTime+DayOfWeek,
  data = airDS)
adsSummary
```

Summary statistics are computed on the variables in the formula, removing missing values for all rows in the included variables:

```
Call:
rxSummary(formula = ~ArrDelay + CRSDepTime + DayOfWeek, data = airDS)
Summary Statistics Results for: ~ArrDelay + CRSDepTime + DayOfWeek
Data: airDS (RxTextData Data Source)
File name: /share/AirlineDemoSmall
Number of valid observations: 6e+05

      Name      Mean     StdDev     Min      Max      ValidObs MissingObs
ArrDelay    11.31794 40.688536 -86.000000 1490.00000 582628    17372
CRSDepTime 13.48227  4.697566   0.016667  23.98333 600000        0

Category Counts for DayOfWeek
Number of categories: 7
Number of valid observations: 6e+05
Number of missing observations: 0

      DayOfWeek  Counts
      Monday     97975
      Tuesday    77725
      Wednesday  78875
      Thursday   81304
      Friday     82987
      Saturday   86159
      Sunday     94975
```

Notice that the summary information shows cell counts for categorical variables, and appropriately does not provide summary statistics such as *Mean* and *StdDev*. Also notice that the *Call:* line shows the actual call you entered or the call provided by *summary*, so will appear differently in different circumstances.

You can also compute summary information by one or more categories by using interactions of a numeric variable with a factor variable. For example, to compute summary statistics on Arrival Delay by Day of Week:

```
rxSummary(~ArrDelay:DayOfWeek, data = airDS)
```

The output shows the summary statistics for *ArrDelay* for each day of the week:

```

Call:
rxSummary(formula = ~ArrDelay:DayOfWeek, data = airDS)

Summary Statistics Results for: ~ArrDelay:DayOfWeek
Data: airDS (RxTextData Data Source)
File name: /share/AirlineDemoSmall
Number of valid observations: 6e+05

Name          Mean      StdDev   Min Max  ValidObs MissingObs
ArrDelay:DayOfWeek 11.31794 40.68854 -86 1490 582628    17372

Statistics by category (7 categories):
Category           DayOfWeek Means      StdDev   Min Max  ValidObs
ArrDelay for DayOfWeek=Monday Monday 12.025604 40.02463 -76 1017 95298
ArrDelay for DayOfWeek=Tuesday Tuesday 11.293808 43.66269 -70 1143 74011
ArrDelay for DayOfWeek=Wednesday Wednesday 10.156539 39.58803 -81 1166 76786
ArrDelay for DayOfWeek=Thursday Thursday 8.658007 36.74724 -58 1053 79145
ArrDelay for DayOfWeek=Friday Friday 14.804335 41.79260 -78 1490 80142
ArrDelay for DayOfWeek=Saturday Saturday 11.875326 45.24540 -73 1370 83851
ArrDelay for DayOfWeek=Sunday Sunday 10.331806 37.33348 -86 1202 93395

```

Use a local compute context

At times, it may be more efficient to perform smaller computations on the local node rather than using MapReduce. You can easily do this, accessing the same data from the HDFS file system. When working with the local compute context, you need to specify the name of a specific data file. The same basic code is then used to run the analysis; change the compute context to a local context. (this will not work if you are accessing the Hadoop cluster via a client.)

```

rxSetComputeContext("local")
inputFile <- file.path(bigDataDirRoot, "AirlineDemoSmall/AirlineDemoSmall.csv")
airDSLocal <- RxTextData(file = inputFile,
  missingValueString = "M",
  colInfo = colInfo, fileSystem = hdfsFS)
adsSummary <- rxSummary(~ArrDelay+CRSDepTime+DayOfWeek,
  data = airDSLocal)
adsSummary

```

The results are the same as doing the computations across the nodes with the Hadoop MapReduce compute context:

```

Call:
rxSummary(formula = ~ArrDelay + CRSDepTime + DayOfWeek, data = airDS)

Summary Statistics Results for: ~ArrDelay + CRSDepTime + DayOfWeek
Data: airDS (RxTextData Data Source)
File name: /share/AirlineDemoSmall/AirlineDemoSmall.csv
Number of valid observations: 6e+05

      Name      Mean    StdDev     Min      Max      ValidObs MissingObs
ArrDelay   11.31794 40.688536 -86.000000 1490.00000  582628    17372
CRSDepTime 13.48227  4.697566  0.016667  23.98333  600000        0

Category Counts for DayOfWeek
Number of categories: 7
Number of valid observations: 6e+05
Number of missing observations: 0

DayOfWeek Counts
Monday    97975
Tuesday   77725
Wednesday 78875
Thursday  81304
Friday    82987
Saturday  86159
Sunday    94975

```

Now set the compute context back to the Hadoop cluster for further analyses:

```
rxSetComputeContext(myHadoopCluster)
```

Fitting a linear model

Use the *rxLinMod* function to fit a linear model using your *airDS* data source. Use a single dependent variable, the factor *DayOfWeek*.

```
arrDelayLm1 <- rxLinMod(ArrDelay ~ DayOfWeek, data = airDS)
summary(arrDelayLm1)
```

The resulting output is:

```

Call:
rxLinMod(formula = ArrDelay ~ DayOfWeek, data = airDS)

Linear Regression Results for: ArrDelay ~ DayOfWeek
Data: airDS (RxTextData Data Source)
File name: /share/AirlineDemoSmall
Dependent variable(s): ArrDelay
Total independent variables: 8 (Including number dropped: 1)
Number of valid observations: 582628
Number of missing observations: 17372

Coefficients: (1 not defined because of singularities)
Estimate Std. Error t value Pr(>|t|)
(Intercept) 10.3318 0.1330 77.673 2.22e-16 ***
DayOfWeek=Monday 1.6938 0.1872 9.049 2.22e-16 ***
DayOfWeek=Tuesday 0.9620 0.2001 4.809 1.52e-06 ***
DayOfWeek=Wednesday -0.1753 0.1980 -0.885 0.376
DayOfWeek=Thursday -1.6738 0.1964 -8.522 2.22e-16 ***
DayOfWeek=Friday 4.4725 0.1957 22.850 2.22e-16 ***
DayOfWeek=Saturday 1.5435 0.1934 7.981 2.22e-16 ***
DayOfWeek=Sunday Dropped Dropped Dropped Dropped
---
Signif. codes: 0 `***` 0.001 `**` 0.01 `*` 0.05 `.` 0.1 ' ' 1

Residual standard error: 40.65 on 582621 degrees of freedom
Multiple R-squared: 0.001869
Adjusted R-squared: 0.001858
F-statistic: 181.8 on 6 and 582621 DF, p-value: < 2.2e-16

```

Create a non-waiting compute context

When running all but the shortest analyses in Hadoop, it can be convenient to let Hadoop do its processing while returning control of your R session to you immediately. You can do this by specifying `wait = FALSE` in your compute context definition. By using our existing compute context as the first argument, all of the other settings will be carried over to the new compute context:

```

myHadoopNoWaitCluster <- RxHadoopMR(myHadoopCluster, wait = FALSE)
rxSetComputeContext(myHadoopNoWaitCluster)

```

After you have set your compute context to non-waiting, distributed **RevoScaleR** functions return relatively quickly with a `jobInfo` object, which you can use to track the progress of your job, and, when the job is complete, obtain the results of the job. For example, we can rerun our linear model in the non-waiting case as follows:

```

job1 <- rxLinMod(ArrDelay ~ DayOfWeek, data = airDS)
rxGetJobStatus(job1)

```

Right after submission, the job status will typically return *"running"*. When the job status returns *"finished"*, you can obtain the results using `rxGetJobResults` as follows:

```

arrDelayLm1 <- rxGetJobResults(job1)
summary(arrDelayLm1)

```

You should always assign the `jobInfo` object so that you can easily track your work, but if you forget, the most recent `jobInfo` object is saved in the global environment as the object `rxgLastPendingJob`. (By default, after you've retrieved your job results, the results are removed from the cluster. To have your job results remain, set the `autoCleanup` argument to `FALSE` in `RxHadoopMR`.)

If after submitting a job in a non-waiting compute context, you decide you don't want to complete the job, you

can cancel the job using the `rxCancelJob` function:

```
job2 <- rxLinMod(ArrDelay ~ DayOfWeek, data = airDS)
rxCancelJob(job2)
```

The `rxCancelJob` function returns `TRUE` if the job is successfully canceled, `FALSE` otherwise.

For the remainder of this tutorial, we return to a waiting compute context:

```
rxSetComputeContext(myHadoopCluster)
```

Analyze a Large Data Set

Set up the sample airline dataset

We will now move to examples using a more recent version of the airline data set. The [airline on-time data](#) has been gathered by the Department of Transportation since 1987. The data through 2008 was used in the American Statistical Association Data Expo in 2009 (<http://stat-computing.org/dataexpo/2009/>). ASA describes the data set as follows:

The data consists of flight arrival and departure details for all commercial flights within the USA, from October 1987 to April 2008. This is a large dataset: there are nearly 120 million records in total, and takes up 1.6 gigabytes of space compressed and 12 gigabytes when uncompressed.

The airline on-time data set for 2012, consisting of 12 separate CSV files, is available [online](#). We assume you have uploaded them to the `/tmp` directory on your name node (although any directory visible as a native file system directory from the name node works.)

As before, our first step is to copy the data into HDFS. We specify the location of your Hadoop-stored data for the `airDataDir` variable:

```
airDataDir <- file.path(bigDataDirRoot, "/airOnTime12/CSV")
rxHadoopMakeDir(airDataDir)
rxHadoopCopyFromLocal("/tmp/airOT2012*.csv", airDataDir)
```

The original CSV files have rather unwieldy variable names, so we supply a `collInfo` list to make them more manageable (we won't use all of these variables in this manual, but you use the data sources created in this manual as you continue to explore distributed computing in the [RevoScaleR Distributed Computing Guide](#):

```

airlineColInfo <- list(
  MONTH = list(newName = "Month", type = "integer"),
  DAY_OF_WEEK = list(newName = "DayOfWeek", type = "factor",
    levels = as.character(1:7),
    newLevels = c("Mon", "Tues", "Wed", "Thur", "Fri", "Sat",
      "Sun")),
  UNIQUE_CARRIER = list(newName = "UniqueCarrier", type =
    "factor"),
  ORIGIN = list(newName = "Origin", type = "factor"),
  DEST = list(newName = "Dest", type = "factor"),
  CRS_DEP_TIME = list(newName = "CRSDepTime", type = "integer"),
  DEP_TIME = list(newName = "DepTime", type = "integer"),
  DEP_DELAY = list(newName = "DepDelay", type = "integer"),
  DEP_DELAY_NEW = list(newName = "DepDelayMinutes", type =
    "integer"),
  DEP_DEL15 = list(newName = "DepDel15", type = "logical"),
  DEP_DELAY_GROUP = list(newName = "DepDelayGroups", type =
    "factor",
    levels = as.character(-2:12),
    newLevels = c("< -15", "-15 to -1", "0 to 14", "15 to 29",
      "30 to 44", "45 to 59", "60 to 74",
      "75 to 89", "90 to 104", "105 to 119",
      "120 to 134", "135 to 149", "150 to 164",
      "165 to 179", ">= 180")),
  ARR_DELAY = list(newName = "ArrDelay", type = "integer"),
  ARR_DELAY_NEW = list(newName = "ArrDelayMinutes", type =
    "integer"),
  ARR_DEL15 = list(newName = "ArrDel15", type = "logical"),
  AIR_TIME = list(newName = "AirTime", type = "integer"),
  DISTANCE = list(newName = "Distance", type = "integer"),
  DISTANCE_GROUP = list(newName = "DistanceGroup", type =
    "factor",
    levels = as.character(1:11),
    newLevels = c("< 250", "250-499", "500-749", "750-999",
      "1000-1249", "1250-1499", "1500-1749", "1750-1999",
      "2000-2249", "2250-2499", ">= 2500")))
varNames <- names(airlineColInfo)

```

We create a data source using these definitions as follows:

```

hdfsFS <- RxHdfsFileSystem()
bigAirDS <- RxTextData( airDataDir,
  colInfo = airlineColInfo,
  varsToKeep = varNames,
  fileSystem = hdfsFS )

```

Estimate a linear model with a big data set

We fit our first model to the large airline data much as we created the linear model for the AirlineDemoSmall data, and we time it to see how long it takes to fit the model on this large data set:

```

system.time(
  delayArr <- rxLinMod(ArrDelay ~ DayOfWeek, data = bigAirDS,
  cube = TRUE)
)

```

To see a summary of your results:

```

print(
  summary(delayArr)
)

```

You should see the following results:

```
Call:
rxLinMod(formula = ArrDelay ~ DayOfWeek, data = bigAirDS, cube = TRUE)

Cube Linear Regression Results for: ArrDelay ~ DayOfWeek
Data: bigAirDS (RxTextData Data Source)
File name: /share/airOnTime12/CSV
Dependent variable(s): ArrDelay
Total independent variables: 7
Number of valid observations: 6005381
Number of missing observations: 91381

Coefficients:
Estimate Std. Error t value Pr(>|t|)    | Counts
DayOfWeek=Mon   3.54210   0.03736  94.80 2.22e-16 *** | 901592
DayOfWeek=Tues  1.80696   0.03835  47.12 2.22e-16 *** | 855805
DayOfWeek=Wed   2.19424   0.03807  57.64 2.22e-16 *** | 868505
DayOfWeek=Thur  4.65502   0.03757 123.90 2.22e-16 *** | 891674
DayOfWeek=Fri   5.64402   0.03747 150.62 2.22e-16 *** | 896495
DayOfWeek=Sat   0.91008   0.04144  21.96 2.22e-16 *** | 732944
DayOfWeek=Sun   2.82780   0.03829  73.84 2.22e-16 *** | 858366
---
Signif. codes:  0 `***` 0.001 `**` 0.01 `*` 0.05 `.` 0.1 ' ' 1

Residual standard error: 35.48 on 6005374 degrees of freedom
Multiple R-squared: 0.001827 (as if intercept included)
Adjusted R-squared: 0.001826
F-statistic: 1832 on 6 and 6005374 DF,  p-value: < 2.2e-16
Condition number: 1
```

Notice that the results indicate we have processed all the data, six million observations, using all the .csv files in the specified directory. Notice also that because we specified `cube = TRUE`, we have an estimated coefficient for each day of the week (and not the intercept).

Import data as composite XDF files

As we have seen, you can analyze CSV files directly with RevoScaleR on Hadoop, but the analysis can be done more quickly if the data is stored in a more efficient format. The RevoScaleR .xdf format is efficient, but is modified somewhat for HDFS so that individual files remain within a single HDFS block. (The HDFS block size varies from installation to installation but is typically either 64 MB or 128 MB.) When you use `rxImport` on Hadoop, you specify an RxTextData data source such as `bigAirDS` as the `inData` and an RxXdfData data source with `fileSystem` set to an HDFS file system as the `outFile` argument to create a set of *composite .xdf files*. The RxXdfData object can then be used as the `data` argument in subsequent RevoScaleR analyses.

For our airline data, we define our RxXdfData object as follows (the second “user” should be replaced by your actual user name on the Hadoop cluster):

```
bigAirXdfName <- "/user/RevoShare/user/AirlineOnTime2012"
airData <- RxXdfData( bigAirXdfName,
                      fileSystem = hdfsFS )
```

We set a block size of 250000 rows and specify that we read all the data by specifying

```
numRowsToRead = -1:
blockSize <- 250000
numRowsToRead = -1
```

We then import the data using `rxImport`:

```

rxImport(inData = bigAirDS,
        outFile = airData,
        rowsPerRead = blockSize,
        overwrite = TRUE,
        numRows = numRowsToRead )

```

Estimate a linear model using composite XDF files

Now we can re-estimate the same linear model, using the new, faster data source:

```

system.time(
  delayArr <- rxLinMod(ArrDelay ~ DayOfWeek, data = airData,
  cube = TRUE)
)

```

To see a summary of your results:

```

print(
  summary(delayArr)
)

```

You should see the following results (which should match the results we found for the CSV files preceding):

```

Call:
rxLinMod(formula = ArrDelay ~ DayOfWeek, data = airData, cube = TRUE)

Cube Linear Regression Results for: ArrDelay ~ DayOfWeek
Data: airData (RxXdfData Data Source)
File name: /user/RevoShare/v7alpha/AirlineOnTime2012
Dependent variable(s): ArrDelay
Total independent variables: 7
Number of valid observations: 6005381
Number of missing observations: 91381

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    | Counts
DayOfWeek=Mon   3.54210   0.03736   94.80 2.22e-16 *** | 901592
DayOfWeek=Tues  1.80696   0.03835   47.12 2.22e-16 *** | 855805
DayOfWeek=Wed   2.19424   0.03807   57.64 2.22e-16 *** | 868505
DayOfWeek=Thur  4.65502   0.03757  123.90 2.22e-16 *** | 891674
DayOfWeek=Fri   5.64402   0.03747  150.62 2.22e-16 *** | 896495
DayOfWeek=Sat   0.91008   0.04144   21.96 2.22e-16 *** | 732944
DayOfWeek=Sun   2.82780   0.03829   73.84 2.22e-16 *** | 858366
---
Signif. codes:  0 `***` 0.001 `**` 0.01 `*` 0.05 `.` 0.1 ` ' 1

Residual standard error: 35.48 on 6005374 degrees of freedom
Multiple R-squared:  0.001827 (as if intercept included)
Adjusted R-squared: 0.001826
F-statistic: 1832 on 6 and 6005374 DF,  p-value: < 2.2e-16
Condition number: 1

```

Handling larger linear models

The data set contains a variable *UniqueCarrier*, which contains airline codes for 15 carriers. Because the RevoScaleR Compute Engine handles factor variables so efficiently, we can do a linear regression looking at the Arrival Delay by Carrier. This takes a little longer than the previous analysis, because we are estimating 15 instead of 7 parameters.

```
delayCarrier <- rxLinMod(ArrDelay ~ UniqueCarrier,
  data = airData, cube = TRUE)
```

Next, sort the coefficient vector so that we can see the airlines with the highest and lowest values.

```
dcCoef <- sort(coef(delayCarrier))
```

Next, see how the airlines rank in arrival delays:

```
print(
  dcCoef
)
```

The result is:

```
UniqueCarrier=AS UniqueCarrier=US UniqueCarrier=DL UniqueCarrier=FL
-1.9022483      -1.2418484      -0.8013952      -0.2618747
UniqueCarrier=HA UniqueCarrier=YV UniqueCarrier=WN UniqueCarrier=MQ
  0.3143564      1.3189086      2.8824285      2.8843612
UniqueCarrier=OO UniqueCarrier=VX UniqueCarrier=B6 UniqueCarrier=UA
  3.9846305      4.0323447      4.8508215      5.0905867
UniqueCarrier=AA UniqueCarrier=EV UniqueCarrier=F9
  6.3980937      7.2934991      7.8788931
```

Notice that Alaska Airlines comes in as the best in on-time arrivals, while Frontier is at the bottom of the list. We can see the difference by subtracting the coefficients:

```
print(
sprintf("Frontier's additional delay compared with Alaska: %f",
  dcCoef["UniqueCarrier=F9"]-dcCoef["UniqueCarrier=AS"])
)
```

which results in:

```
[1] "Frontier's additional delay compared with Alaska: 9.781141"
```

A Large Data Logistic Regression

Our airline data includes a logical variable, ArrDel15, that tells whether a flight is 15 or more minutes late. We can use this as the response for a logistic regression to model the likelihood that a flight will be late given the day of the week:

```
logitObj <- rxLogit(ArrDel15 ~ DayOfWeek, data = airData)
logitObj
```

which results in:

```

Logistic Regression Results for: ArrDel15 ~ DayOfWeek
Data: airData (RxXdfData Data Source)
File name: AirOnTime2012.xdf
Dependent variable(s): ArrDel15
Total independent variables: 8 (Including number dropped: 1)
Number of valid observations: 6005381
Number of missing observations: 91381

Coefficients:
  ArrDel15
(Intercept) -1.61122563
DayOfWeek=Mon  0.04600367
DayOfWeek=Tues -0.08431814
DayOfWeek=Wed  -0.07319133
DayOfWeek=Thur   0.12585721
DayOfWeek=Fri   0.18987931
DayOfWeek=Sat  -0.13498591
DayOfWeek=Sun    Dropped

```

Prediction on Large Data

You can predict (or score) from a fitted model on Hadoop using *rxPredict*. In this example, we compute predicted values and their residuals for the logistic regression in the previous section. These new prediction variables are output to a new composite XDF in HDFS. As in an earlier example, the second “user” in the *airDataPredDir* path should be changed to the actual user name of your Hadoop user.

```

airDataPredDir <- "/user/RevoShare/user/airDataPred"
airDataPred <- RxXdfData(airDataPredDir, fileSystem=hdfsFS)
rxPredict(modelObject=logitObj, data=airData, outData=airDataPred,
computeResiduals=TRUE)

```

By putting in a call to *rxGetVarInfo* we see that two variables, *ArrDel15_Pred* and *ArrDel15_Resid* were written to the *airDataPred* composite XDF. If in addition to the prediction variables, we wanted to have the variables used in the model written to our *outData* we would need to add the *writeModelVars=TRUE* to our *rxPredict* call.

Alternatively, we can update the *airData* to include the predicted values and residuals by not specifying an *outData* argument, which is *NULL* by default. Since the *airData* composite XDF already exists, we would need to add *overwrite=TRUE* to the *rxPredict* call.

An *RxTextData* object can be used as the input data for *rxPredict* on Hadoop, but only XDF can be written to HDFS. When using a CSV file or directory of CSV files as the input data to *rxPredict* the *outData* argument must be set to an *RxXdfData* object.

Performing Data Operations on Large Data

To create or modify data in HDFS on Hadoop, we can use the *rxDataStep* function. Suppose we want to repeat the analyses with a “cleaner” version of the large airline dataset. To do this we keep only the flights having arrival delay information and flights that did not depart more than one hour early. We can put a call to *rxDataStep* to output a new composite XDF to HDFS. (As in an earlier example, the second “user” in the *newAirDir* path should be changed to the actual user name of your Hadoop user.)

```

newAirDir <- "/user/RevoShare/user/newAirData"
newAirXdf <- RxXdfData(newAirDir, fileSystem=hdfsFS)

rxDataStep(inData = airData, outFile = newAirXdf,
rowSelection = !is.na(ArrDelay) & (DepDelay > -60))

```

To modify an existing composite XDF using *rxDataStep* set the *overwrite* argument to *TRUE* and either omit the *outFile* argument or set it to the same data source specified for *inData*.

Using Data from Hive for Your Analyses

There are multiple ways to access and use data from Hive for analyses with Machine Learning Server. Here are some general recommendations, assuming in each case that the data for analysis is defined by the results of a Hive query.

1. If running from a remote client or edge node, and the data is modest, then use RxOdbcData to stream results, or land them as XDF in the local file system, for subsequent analysis in a local compute context.
2. If the data is large, then use the Hive command-line interface (hive or beeline) from an edge node to run the Hive query with results spooled to a text file on HDFS for subsequent analysis in a distributed fashion using the HadoopMR or Spark compute contexts.

Here's how to get started with each of these approaches.

Accessing data via ODBC

Start by following your Hadoop vendor's recommendations for accessing Hive via ODBC from a remote client or edge node. After you have the prerequisite software installed and have run a smoke test to verify connectivity, then accessing data in Hive from Machine Learning Server is just like accessing data from any other data source.

```
mySQL = "SELECT * FROM CustData"
myDS <- RxOdbcData(sqlQuery = mySQL, connectionString = "DSN=HiveODBC")
xdfFile <- RxXdfData("dataFromHive.xdf")
rxImport(myDS, xdfFile, stringsAsFactors = TRUE, overwrite=TRUE)
```

Accessing data via an Export to Text Files

This approach uses the Hive command-line interface to first spool the results to text files in the local or HDFS file systems and then analyze them in a distributed fashion using local or HadoopMR/Spark compute contexts.

First, check with your Hadoop system administrator find out whether the 'hive' or 'beeline' command should be used to run a Hive query from the command line, and obtain the needed credentials for access.

Next, try out your hive/beeline command from the Linux command line to make sure it works and that you have the proper authentication and syntax.

Finally, run the command from within R using R's system command.

Here are some sample R statements that output a Hive query to the local and HDFS file systems:

Run a Hive query and dump results to a local text file (edge node)

```
system('hive -e "select * from emp" > myFile.txt')
```

Run the same query using beeline's csv2 output format

```
system('beeline -u "jdbc:hive2://..." --outputformat=csv2 -e "select * from emp" > myFile.txt')
```

Run the same query but dump the results to CSV in HDFS

```
system('beeline -u "jdbc:hive2://..." -e "insert overwrite directory \'/your-hadoop-dir\' row format
delimited fields terminated by \',\' select * from emp"')
```

After you've exported the query results to a text file, it can be streamed directly as input to a RevoScaleR analysis routine via use of the RxTextdata data source, or imported to XDF for improved performance upon repeated access. Here's an example assuming output was spooled as text to HDFS:

```

hiveOut <- file.path(bigDataDirRoot,"myHiveData")
myHiveData <- RxTextData(file = hiveOut, fileSystem = hdfsFS)

rxSummary(~var1 + var2+ var3+ var4, data=myHiveData)

```

Writing to CSV in HDFS

If you converted your CSV to XDF to take advantage of the efficiency while running analyses, but now wish to convert your data back to CSV you can do so using *rxDataStep*. (This feature is still experimental.) To create a folder of CSV files, first create an *RxTextData* object using a directory name as the file argument; this represents the folder in which to create the CSV files. This directory is created when you run the *rxDataStep*. Then, point to this *RxTextData* object in the *outFile* argument of the *rxDataStep*. Each CSV created will be named based on the directory name and followed by a number.

Suppose we want to write out a folder of CSV in HDFS from our *airData* composite XDF after we performed the logistic regression and prediction, so that the new CSV files contain the predicted values and residuals. We can do this as follows:

```

airDataCsvDir <- file.path("user/RevoShare/user/airDataCsv")
airDataCsvDS <- RxTextData(airDataCsvDir,fileSystem=hdfsFS)
rxDataStep(inData=airData, outFile=airDataCsvDS)

```

You notice that the *rxDataStep* wrote out one CSV for every .xdfd file in the input composite XDF file. This is the default behavior for writing CSV from composite XDF to HDFS when the compute context is set to HadoopMR.

Alternatively, you could switch your compute context back to "local" when you are done performing your analyses and take advantage of two arguments within *RxTextData* that give you slightly more control when writing out CSV files to HDFS: *createFileSet* and *rowsPerOutFile*. When *createFileSet* is set to TRUE a folder of CSV files is written to the directory, you specify. When *createFileSet* is set to FALSE a single CSV file is written. The second argument, *rowsPerOutFile*, can be set to an integer to indicate how many rows to write to each CSV file when *createFileSet* is TRUE. Returning to the example preceding, suppose we wanted to write out a folder of CSVs but we wanted to write out the *airData* but into only six CSV files.

```

rxSetComputeContext("local")
airDataCsvRowsDir <- file.path("/user/RevoShare/MRS/airDataCsvRows")
rxHadoopMakeDir(airDataCsvRowsDir)
airDataCsvRowsDS <- RxTextData(airDataCsvRowsDir, fileSystem=hdfsFS, createFileSet=TRUE,
rowsPerOutFile=1000000)
rxDataStep(inData=airData, outFile=airDataCsvRowsDS)

```

When using a HadoopMR compute context, *createFileSet* defaults to TRUE and *rowsPerOutFile* has no effect. Thus if you wish to create a single CSV or customize the number of rows per file you must perform the *rxDataStep* in a local compute context (data can still be in HDFS).

Parallel Partial Decision Forests

Both *rxDForest* and *rxBTrees* are available on Hadoop, providing two different methods for fitting classification and regression decision forests. In the default case, both algorithms generate multiple MapReduce jobs, and thus can tend to incur significant overhead, particularly with smaller data sets. However, the *scheduleOnce* argument to both functions allows the computation to be performed via *rxExec*, which generates only a single MapReduce job, and thus incurs significantly less overhead. When using the *scheduleOnce* argument, you can specify the number of trees to be grown within each *rxExec* task using the forest function's *nTree* argument together with *rxExec*'s *rxElemArgs* function, as in the following regression example using the built-in claims data:

```

file.name <- "claims.xdf"
sourceFile <- system.file(file.path("SampleData", file.name),
  package="RevoScaleR")
inputDir <- "/share/claimsXdf"
rxHadoopMakeDir(inputDir)
rxHadoopCopyFromLocal(sourceFile, inputDir)
input <- RxXdfData(file.path(inputDir, file.name,
  fsep="/"), fileSystem=hdfsFS)

partial.forests <-rxForest(formula = age ~ car.age +
  type + cost + number, data = input,
  minSplit = 5, maxDepth = 2,
  nTree = rxElemArg(rep(2,8), seed = 0,
  maxNumBins = 200, computeOobError = -1,
  reportProgress = 2, verbose = 0,
  scheduleOnce = TRUE)

```

Equivalently, you can use `nTree` together with `rxExec`'s `timesToRun` argument:

```

partial.forests <-rxForest(formula = age ~ car.age +
  type + cost + number, data = input,
  minSplit = 5, maxDepth = 2,
  nTree = 2, seed = 0,
  maxNumBins = 200, computeOobError = -1,
  reportProgress = 2, verbose = 0,
  scheduleOnce = TRUE, timesToRun = 8)

```

In this example, using `scheduleOnce` can be up to 45 times faster than the default, and so we recommend it for decision forests with small to moderate-sized data sets.

Similarly, the `rxPredict` methods for `rxForest` and `rxBTrees` objects include the `scheduleOnce` argument, and should be used when using these methods on small to moderate-sized data sets.

Cleaning up Data

You can run the following commands to clean up data in this tutorial:

```

rxHadoopRemoveDir("/share/AirlineDemoSmall")
rxHadoopRemoveDir("/share/airOnTime12")
rxHadoopRemoveDir("/share/claimsXdf")
rxHadoopRemoveDir("/user/RevoShare/<user>/AirlineOnTime2012")
rxHadoopRemoveDir("/user/RevoShare/<user>/airDataCsv")
rxHadoopRemoveDir("/user/RevoShare/<user>/airDataCsvRows")
rxHadoopRemoveDir("/user/RevoShare/<user>/newAirData")
rxHadoopRemoveDir("/user/RevoShare/<user>/airDataPred")

```

Continuing with Distributed Computing

With the linear model and logistic regression performed in the previous sections, you have seen a taste of high-performance analytics on the Hadoop platform. You are now ready to continue with the [RevoScaleR Distributed Computing Guide](#), which continues the analysis of the 2012 airline on-time data with examples for all of RevoScaleR's HPA functions.

The *Distributed Computing Guide* also provides more examples of using non-waiting compute contexts, including managing multiple jobs, and examples of using `rxExec` to perform traditional high-performance computing, including Monte Carlo simulations and other embarrassingly parallel problems.

Running distributed analyses using RevoScaleR

7/12/2022 • 13 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Many RevoScaleR functions support parallelization. On a standalone multi-core server, functions that are multithreaded run on all available cores. In an rxSparkConnect or RxHadoop remote compute context, multithreaded analyses run on all data nodes having the RevoScaleR engine.

RevoScaleR can structure an analysis for parallel execution with no additional configuration on your part, assuming you set the [compute context](#). Setting the compute context to rxSparkConnect or RxHadoopMR tells RevoScaleR to look for data nodes. In contrast, using the default local compute context tells the engine to look for available processors on the local machine.

NOTE

RevoScaleR also runs on R Client. On R Client, RevoScaleR is limited to two threads for processing and in-memory datasets. To avoid paging data to disk, R Client is engineered to ignore the `blocksPerRead` argument, which results in all data being read into memory. If your datasets exceed memory, you should push the compute context to a server instance on a supported platform (Hadoop, Linux, Windows, SQL Server).

Given a registered a distributed compute context, the following functions can perform distributed computations:

- `rxSummary`
- `rxLinMod`
- `rxLogit`
- `rxGlm`
- `rxCovCor` (and its convenience functions, `rxCov`, `rxCor`, and `rxSSCP`)
- `rxCube` and `rxCrossTabs`
- `rxKmeans`
- `rxDTree`
- `rxDForest`
- `rxBTrees`
- `rxNaiveBayes`
- `rxExec`

Except for `rxExec`, we refer to these functions as the RevoScaleR *high-performance analytics*, or HPA functions.

The exception, `rxExec`, is used to execute an arbitrary function on specified nodes (or cores) of your compute context. It can be used for traditional high-performance computing functions. The `rxExec` function offers great flexibility in how arguments are passed, so that you can specify that all nodes receive the same arguments, or provide different arguments to each node.

`rxPredict` on a cluster is only redistributed if the data file is split.

Obtain node information

You can use informational functions, such as `rxGetInfo` and `rxGetVarInfo`, to confirm data availability. Before beginning data analysis, you can use `rxGetInfo` to confirm the data set is available on the compute resources.

You can request basic information about a data set from each node using the `rxGetInfo` function. Assuming a data source named "airData", you can call `rxGetInfo` as follows:

```
rxGetInfo(data=airData)
```

NOTE

To load a dataset, use AirOnTime2012.xdf from the [data set download site](#) and make sure it is in your dataPath. You can then run `airData <- RxXdfData("AirOnTime2012.xdf")` to load the data on a cluster.

On a five-node cluster, the call to `rxGetInfo` returns the following:

```
$CLUSTER_HEAD
File name: C:\data-RevoScaleR-AcceptanceTest\AirOnTime2012.xdf
Number of observations: 6096762
Number of variables: 46
Number of blocks: 31
Compression type: zlib

$COMPUTE10
File name: C:\data-RevoScaleR-AcceptanceTest\AirOnTime2012.xdf
Number of observations: 6096762
Number of variables: 46
Number of blocks: 31
Compression type: zlib

$COMPUTE11
File name: C:\data-RevoScaleR-AcceptanceTest\AirOnTime2012.xdf
Number of observations: 6096762
Number of variables: 46
Number of blocks: 31
Compression type: zlib

$COMPUTE12
File name: C:\data-RevoScaleR-AcceptanceTest\AirOnTime2012.xdf
Number of observations: 6096762
Number of variables: 46
Number of blocks: 31
Compression type: zlib

$COMPUTE13
File name: C:\data-RevoScaleR-AcceptanceTest\AirOnTime2012.xdf
Number of observations: 6096762
Number of variables: 46
Number of blocks: 31
Compression type: zlib
```

This confirms that our data set is in fact available on all nodes of our cluster.

Obtain a Data Summary

The `rxSummary` function returns summary statistics on a data set, including datasets that run in a distributed context.

When you run one of **RevoScaleR**'s HPA functions in a distributed compute context, it automatically distributes

the computation among the available compute resources and coordinates the returned values to create the final return value.

Assuming a job is waiting (or blocking), control is not returned until a computation is complete. We assume that the airline data has been copied to the appropriate data directory on all the computing resources and its location specified by the `airData` data source object.

NOTE

The `blocksPerRead` argument is ignored if script runs locally using R Client.

For example, we start by taking a summary of three variables from the airline data:

```
rxSummary(~ ArrDelay + CRSDepTime + DayOfWeek, data=airData,  
blocksPerRead=30)
```

We get the following results (identical to what we would have gotten from the same command in a local compute context):

```
Call:  
rxSummary(formula = ~ArrDelay + CRSDepTime + DayOfWeek, data = airData,  
blocksPerRead = 30)  
  
Summary Statistics Results for: ~ArrDelay + CRSDepTime + DayOfWeek  
Data: airData (RxXdfData Data Source)  
File name: /var/RevoShare/v7alpha/aot12  
Number of valid observations: 6096762  
  
      Name      Mean     StdDev     Min          Max      ValidObs MissingObs  
ArrDelay    3.155596 35.510870 -411.0000000 1823.00000 6005381  91381  
CRSDepTime 13.457386  4.707193   0.01666667 23.98333 6096761       1  
  
Category Counts for DayOfWeek  
Number of categories: 7  
Number of valid observations: 6096762  
Number of missing observations: 0  
  
DayOfWeek Counts  
Mon      916747  
Tues     871412  
Wed      883207  
Thur     905827  
Fri      910135  
Sat      740232  
Sun      869202
```

Perform an rxCube computation

We can perform an `rxCube` computation using the same data set to compute the average arrival delay for departures for each hour of the day for each day of the week. Again, the code is identical to the code used when performing the computations on a single computer, as are the results.

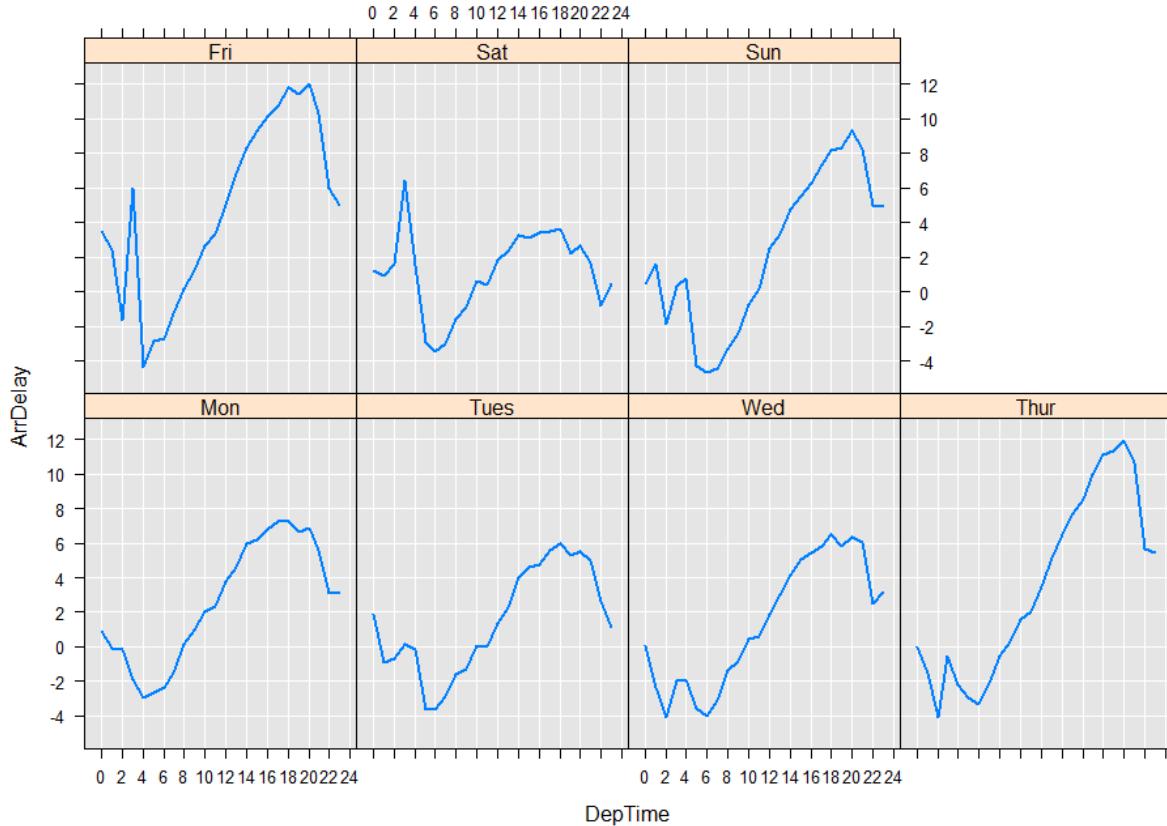
```
delayArrCube <- rxCube(ArrDelay ~ F(CRSDepTime):DayOfWeek,  
data=airData, blocksPerRead=30)
```

NOTE

The `blocksPerRead` argument is ignored if script runs locally using R Client.

Notice that in this case we have returned an `rxCube` object. We can use this object locally to, for example, extract a data frame and plot the results:

```
plotData <- rxResultsDF( delayArrCube )
names(plotData)[1] <- "DepTime"
rxLinePlot(ArrDelay~DepTime|DayOfWeek, data=plotData)
```



Perform an `rxCrossTabs` computation

The `rxCrossTabs` function provides essentially the same computations as `rxCube`, but presents the results in a more traditional cross-tabulation. Here we look at late flights (those whose arrival delay is 15 or greater) by late departure and day of week:

```
crossTabs <- rxCrossTabs(formula = ArrDel15 ~ F(DepDel15):DayOfWeek,
                         data = airData, means = TRUE)
crossTabs
```

which yields:

```

Call:
rxCrossTabs(formula = ArrDel15 ~ F(DepDel15):DayOfWeek, data = airData,
means = TRUE)

Cross Tabulation Results for: ArrDel15 ~ F(DepDel15):DayOfWeek
Data: airData (RxXdfData Data Source)
File name: /var/RevoShare/v7alpha/AirlineOnTime2012
Dependent variable(s): ArrDel15
Number of valid observations: 6005381
Number of missing observations: 91381
Statistic: means

ArrDel15 (means):
  DayOfWeek
F_DepDel15      Mon      Tues      Wed      Thur      Fri      Sat
  0 0.04722548 0.04376271 0.04291565 0.05006577 0.05152312 0.04057934
  1 0.79423651 0.78904905 0.79409615 0.80540551 0.81086142 0.76329539
  DayOfWeek
F_DepDel15      Sun
  0 0.04435956
  1 0.79111488

```

Compute a Covariance or Correlation Matrix

The `rxCovCor` function is used to compute covariance and correlation matrices; the convenience functions `rxCov`, `rxCor`, and `rxSSCP` all depend upon it and are usually used in practical situations. For examples, see [Correlation and variance/covariance matrices](#).

The following example shows how the main function can be used directly:

```

covForm <- ~ DepDelayMinutes + ArrDelayMinutes + AirTime
cov <- rxCovCor(formula = covForm, data = airData, type = "Cov")
cor <- rxCovCor(formula = covForm, data = airData, type = "Cor")
cov # covariance matrix
Call:
rxCovCor(formula = ~DepDelayMinutes + ArrDelayMinutes + AirTime,
  data = <S4 object of class structure("RxXdfData", package = "RevoScaleR")>,
  type = "Cov")

Data: <S4 object of class structure("RxXdfData", package = "RevoScaleR")> (RxXdfData Data Source)
File name: /var/RevoShare/v7alpha/AirlineOnTime2012
Number of valid observations: 6005381
Number of missing observations: 91381
Statistic: COV

  DepDelayMinutes ArrDelayMinutes   AirTime
DepDelayMinutes      1035.09355    996.88898   39.60668
ArrDelayMinutes       996.88898   1029.07742   59.77224
AirTime              39.60668     59.77224  4906.02279
cor # correlation matrix
Call:
rxCovCor(formula = ~DepDelayMinutes + ArrDelayMinutes + AirTime,
  data = <S4 object of class structure("RxXdfData", package = "RevoScaleR")>,
  type = "Cor")

Data: <S4 object of class structure("RxXdfData", package = "RevoScaleR")> (RxXdfData Data Source)
File name: /var/RevoShare/v7alpha/AirlineOnTime2012
Number of valid observations: 6005381
Number of missing observations: 91381
Statistic: COR

  DepDelayMinutes ArrDelayMinutes   AirTime
DepDelayMinutes      1.00000000  0.96590179  0.01757575
ArrDelayMinutes       0.96590179  1.00000000  0.02660178
AirTime              0.01757575  0.02660178  1.00000000

```

Compute a Linear Model

We can model the arrival delay as a function of day of the week, departure time, and flight distance as follows:

```

linModObj <- rxLinMod(ArrDelay~ DayOfWeek + F(CRSDepTime) + Distance,
  data = airData)

```

We can then view a summary of the results as follows:

```

summary(linModObj)
Call:
rxLinMod(formula = ArrDelay ~ DayOfWeek + F(CRSDepTime) + Distance,
  data = airData)

Linear Regression Results for: ArrDelay ~ DayOfWeek + F(CRSDepTime) +
  Distance
Data: airData (RxXdfData Data Source)
File name: /var/RevoShare/v7alpha/AirlineOnTime2012
Dependent variable(s): ArrDelay
Total independent variables: 33 (Including number dropped: 2)
Number of valid observations: 6005380
Number of missing observations: 91382

Coefficients: (2 not defined because of singularities)
Estimate Std. Error t value Pr(>|t|)
(Intercept) 3.570e+00 2.053e-01 17.389 2.22e-16 ***
DayOfWeek=Mon 1.014e+00 5.320e-02 19.061 2.22e-16 ***
DayOfWeek=Tues -7.077e-01 5.389e-02 -13.131 2.22e-16 ***
DayOfWeek=Wed -3.503e-01 5.369e-02 -6.524 6.85e-11 ***
DayOfWeek=Thur 2.122e+00 5.334e-02 39.782 2.22e-16 ***
DayOfWeek=Fri 3.089e+00 5.327e-02 57.976 2.22e-16 ***
DayOfWeek=Sat -1.343e+00 5.615e-02 -23.925 2.22e-16 ***
DayOfWeek=Sun Dropped Dropped Dropped Dropped
F_CRSDepTime=0 -2.283e+00 4.548e-01 -5.020 5.17e-07 ***
F_CRSDepTime=1 -3.277e+00 6.035e-01 -5.429 5.65e-08 ***
F_CRSDepTime=2 -4.926e+00 1.223e+00 -4.028 5.63e-05 ***
F_CRSDepTime=3 -2.316e+00 1.525e+00 -1.519 0.128881
F_CRSDepTime=4 -5.063e+00 1.388e+00 -3.648 0.000265 ***
F_CRSDepTime=5 -7.178e+00 2.377e-01 -30.197 2.22e-16 ***
F_CRSDepTime=6 -7.317e+00 2.065e-01 -35.441 2.22e-16 ***
F_CRSDepTime=7 -6.397e+00 2.065e-01 -30.976 2.22e-16 ***
F_CRSDepTime=8 -4.907e+00 2.061e-01 -23.812 2.22e-16 ***
F_CRSDepTime=9 -4.211e+00 2.074e-01 -20.307 2.22e-16 ***
F_CRSDepTime=10 -2.857e+00 2.070e-01 -13.803 2.22e-16 ***
F_CRSDepTime=11 -2.537e+00 2.069e-01 -12.262 2.22e-16 ***
F_CRSDepTime=12 -9.556e-01 2.073e-01 -4.609 4.05e-06 ***
F_CRSDepTime=13 1.180e-01 2.070e-01 0.570 0.568599
F_CRSDepTime=14 1.470e+00 2.073e-01 7.090 2.22e-16 ***
F_CRSDepTime=15 2.147e+00 2.076e-01 10.343 2.22e-16 ***
F_CRSDepTime=16 2.701e+00 2.074e-01 13.023 2.22e-16 ***
F_CRSDepTime=17 3.447e+00 2.065e-01 16.688 2.22e-16 ***
F_CRSDepTime=18 4.080e+00 2.080e-01 19.614 2.22e-16 ***
F_CRSDepTime=19 3.649e+00 2.079e-01 17.553 2.22e-16 ***
F_CRSDepTime=20 4.216e+00 2.119e-01 19.895 2.22e-16 ***
F_CRSDepTime=21 3.276e+00 2.151e-01 15.225 2.22e-16 ***
F_CRSDepTime=22 -1.729e-01 2.284e-01 -0.757 0.449026
F_CRSDepTime=23 Dropped Dropped Dropped Dropped
Distance -4.220e-04 2.476e-05 -17.043 2.22e-16 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 35.27 on 6005349 degrees of freedom
Multiple R-squared: 0.01372
Adjusted R-squared: 0.01372
F-statistic: 2785 on 30 and 6005349 DF, p-value: < 2.2e-16
Condition number: 442.0146

```

Compute a Logistic Regression

We can compute a similar logistic regression using the logical variable ArrDel15 as the response. This variable specifies whether a flight's arrival delay was 15 minutes or greater:

```

logitObj <- rxLogit(ArrDel15~DayOfWeek + F(CRSDepTime) + Distance,
  data = airData)
summary(logitObj)

Call:
rxLogit(formula = ArrDel15 ~ DayOfWeek + F(CRSDepTime) + Distance,
  data = airData)

Logistic Regression Results for: ArrDel15 ~ DayOfWeek + F(CRSDepTime) +
  Distance
Data: airData (RxXdfData Data Source)
File name: /var/RevoShare/v7alpha/AirlineOnTime2012
Dependent variable(s): ArrDel15
Total independent variables: 33 (Including number dropped: 2)
Number of valid observations: 6005380
-2*LogLikelihood: 5320489.0684 (Residual deviance on 6005349 degrees of freedom)

Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.740e+00 1.492e-02 -116.602 2.22e-16 ***
DayOfWeek=Mon 7.852e-02 4.060e-03 19.341 2.22e-16 ***
DayOfWeek=Tues -5.222e-02 4.202e-03 -12.428 2.22e-16 ***
DayOfWeek=Wed -4.431e-02 4.178e-03 -10.606 2.22e-16 ***
DayOfWeek=Thur 1.593e-01 4.023e-03 39.596 2.22e-16 ***
DayOfWeek=Fri 2.225e-01 3.981e-03 55.875 2.22e-16 ***
DayOfWeek=Sat -8.336e-02 4.425e-03 -18.839 2.22e-16 ***
DayOfWeek=Sun Dropped Dropped Dropped Dropped
F_CRSDepTime=0 -2.537e-01 3.555e-02 -7.138 2.22e-16 ***
F_CRSDepTime=1 -3.852e-01 4.916e-02 -7.836 2.22e-16 ***
F_CRSDepTime=2 -4.118e-01 1.032e-01 -3.989 6.63e-05 ***
F_CRSDepTime=3 -1.046e-01 1.169e-01 -0.895 0.370940
F_CRSDepTime=4 -4.402e-01 1.202e-01 -3.662 0.000251 ***
F_CRSDepTime=5 -9.115e-01 2.008e-02 -45.395 2.22e-16 ***
F_CRSDepTime=6 -8.934e-01 1.553e-02 -57.510 2.22e-16 ***
F_CRSDepTime=7 -6.559e-01 1.536e-02 -42.716 2.22e-16 ***
F_CRSDepTime=8 -4.608e-01 1.518e-02 -30.364 2.22e-16 ***
F_CRSDepTime=9 -3.657e-01 1.525e-02 -23.975 2.22e-16 ***
F_CRSDepTime=10 -2.305e-01 1.514e-02 -15.220 2.22e-16 ***
F_CRSDepTime=11 -1.868e-01 1.512e-02 -12.359 2.22e-16 ***
F_CRSDepTime=12 -6.100e-02 1.509e-02 -4.041 5.32e-05 ***
F_CRSDepTime=13 4.476e-02 1.503e-02 2.979 0.002896 ***
F_CRSDepTime=14 1.573e-01 1.501e-02 10.480 2.22e-16 ***
F_CRSDepTime=15 2.218e-01 1.500e-02 14.786 2.22e-16 ***
F_CRSDepTime=16 2.718e-01 1.498e-02 18.144 2.22e-16 ***
F_CRSDepTime=17 3.468e-01 1.489e-02 23.284 2.22e-16 ***
F_CRSDepTime=18 4.008e-01 1.498e-02 26.762 2.22e-16 ***
F_CRSDepTime=19 4.023e-01 1.497e-02 26.875 2.22e-16 ***
F_CRSDepTime=20 4.484e-01 1.520e-02 29.489 2.22e-16 ***
F_CRSDepTime=21 3.767e-01 1.543e-02 24.419 2.22e-16 ***
F_CRSDepTime=22 8.995e-02 1.656e-02 5.433 5.55e-08 ***
F_CRSDepTime=23 Dropped Dropped Dropped Dropped
Distance 1.336e-04 1.829e-06 73.057 2.22e-16 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Condition number of final variance-covariance matrix: 445.2487
Number of iterations: 5

```

View Console Output

You may notice when running distributed computations that you get virtually no feedback while running waiting jobs. Since the computations are in general not running on the same computer as your R Console, the "usual" feedback is not returned by default. However, you can set the *consoleOutput* parameter in your compute context to TRUE to enable return of console output from all the nodes. For example, here we update our compute

context `myCluster` to include `consoleOutput=TRUE`.

NOTE

The `blocksPerRead` argument is ignored if script runs locally using R Client.

```
myCluster <- RxSpark(myCluster, consoleOutput=TRUE)
rxOptions(computeContext=myCluster)
```

Then, rerunning our previous example results in much more verbose output:

```
delayArrCube <- rxCube(ArrDelay ~ F(CRSDepTime):DayOfWeek,
  data="AirlineData87to08.xdf", blocksPerRead=30)

===== CLUSTER-HEAD2 ( process 1 ) has started run at
Thu Aug 11 15:56:10 2011 =====
*****
Worker Node 'COMPUTE10' has received a task from Master Node 'CLUSTER-HEAD2'.... Thu Aug 11 15:56:10.791
2011
*****
Worker Node 'COMPUTE11' has received a task from Master Node 'CLUSTER-HEAD2'.... Thu Aug 11 15:56:10.757
2011
*****
Worker Node 'COMPUTE12' has received a task from Master Node 'CLUSTER-HEAD2'.... Thu Aug 11 15:56:10.769
2011
*****
Worker Node 'COMPUTE13' has received a task from Master Node 'CLUSTER-HEAD2'.... Thu Aug 11 15:56:10.889
2011
```

```
COMPUTE13: Rows Read: 4440596, Total Rows Processed: 4440596, Total Chunk Time: 0.031 seconds
COMPUTE11: Rows Read: 4361843, Total Rows Processed: 4361843, Total Chunk Time: 0.031 seconds
COMPUTE12: Rows Read: 4467780, Total Rows Processed: 4467780, Total Chunk Time: 0.031 seconds
COMPUTE10: Rows Read: 4492157, Total Rows Processed: 4492157, Total Chunk Time: 0.047 seconds
COMPUTE13: Rows Read: 4500000, Total Rows Processed: 8940596, Total Chunk Time: 0.062 seconds
COMPUTE12: Rows Read: 4371359, Total Rows Processed: 8839139, Total Chunk Time: 0.078 seconds
COMPUTE10: Rows Read: 4470501, Total Rows Processed: 8962658, Total Chunk Time: 0.062 seconds
COMPUTE11: Rows Read: 4500000, Total Rows Processed: 8861843, Total Chunk Time: 0.078 seconds
COMPUTE13: Rows Read: 4441922, Total Rows Processed: 13382518, Total Chunk Time: 0.078 seconds
COMPUTE10: Rows Read: 4430048, Total Rows Processed: 13392706, Total Chunk Time: 0.078 seconds
COMPUTE12: Rows Read: 4500000, Total Rows Processed: 13339139, Total Chunk Time: 0.062 seconds
COMPUTE11: Rows Read: 4484721, Total Rows Processed: 13346564, Total Chunk Time: 0.062 seconds
COMPUTE13: Rows Read: 4500000, Total Rows Processed: 17882518, Total Chunk Time: 0.063 seconds
COMPUTE12: Rows Read: 4388540, Total Rows Processed: 17727679, Total Chunk Time: 0.078 seconds
COMPUTE10: Rows Read: 4500000, Total Rows Processed: 17892706, Total Chunk Time: 0.078 seconds
COMPUTE11: Rows Read: 4477884, Total Rows Processed: 17824448, Total Chunk Time: 0.078 seconds
COMPUTE13: Rows Read: 4453215, Total Rows Processed: 22335733, Total Chunk Time: 0.078 seconds
COMPUTE12: Rows Read: 4429270, Total Rows Processed: 22156949, Total Chunk Time: 0.063 seconds
COMPUTE10: Rows Read: 4427435, Total Rows Processed: 22320141, Total Chunk Time: 0.063 seconds
COMPUTE11: Rows Read: 4483047, Total Rows Processed: 22307495, Total Chunk Time: 0.078 seconds
COMPUTE13: Rows Read: 2659728, Total Rows Processed: 24995461, Total Chunk Time: 0.062 seconds
COMPUTE12: Rows Read: 2400000, Total Rows Processed: 24556949, Total Chunk Time: 0.078 seconds
Worker Node 'COMPUTE13' has completed its task successfully. Thu Aug 11 15:56:11.341 2011
Elapsed time: 0.453 secs.
*****
```

```
Worker Node 'COMPUTE12' has completed its task successfully. Thu Aug 11 15:56:11.221 2011
Elapsed time: 0.453 secs.
*****
```

```
COMPUTE10: Rows Read: 2351983, Total Rows Processed: 24672124, Total Chunk Time: 0.078 seconds
COMPUTE11: Rows Read: 2400000, Total Rows Processed: 24707495, Total Chunk Time: 0.078 seconds
```

```
Worker Node 'COMPUTE10' has completed its task successfully. Thu Aug 11 15:56:11.244 2011  
Elapsed time: 0.453 secs.
```

```
*****  
Worker Node 'COMPUTE11' has completed its task successfully. Thu Aug 11 15:56:11.209 2011  
Elapsed time: 0.453 secs.
```

```
*****  
Master node [CLUSTER-HEAD2] is starting a task.... Thu Aug 11 15:56:10.961 2011  
CLUSTER-HEAD2: Rows Read: 4461826, Total Rows Processed: 4461826, Total Chunk Time: 0.038 seconds  
CLUSTER-HEAD2: Rows Read: 4452096, Total Rows Processed: 8913922, Total Chunk Time: 0.071 seconds  
CLUSTER-HEAD2: Rows Read: 4441200, Total Rows Processed: 13355122, Total Chunk Time: 0.075 seconds  
CLUSTER-HEAD2: Rows Read: 4370893, Total Rows Processed: 17726015, Total Chunk Time: 0.074 seconds  
CLUSTER-HEAD2: Rows Read: 4476925, Total Rows Processed: 22202940, Total Chunk Time: 0.071 seconds  
CLUSTER-HEAD2: Rows Read: 2400000, Total Rows Processed: 24602940, Total Chunk Time: 0.072 seconds  
Master node [CLUSTER-HEAD2] has completed its task successfully. Thu Aug 11 15:56:11.410 2011  
Elapsed time: 0.449 secs.
```

```
Time to compute summary on all servers: 0.461 secs.  
Processing results on client ...  
Computation time: 0.471 seconds.  
===== CLUSTER-HEAD2 ( process 1 ) has completed run at Thu Aug 11 15:56:11 2011 =====
```

See also

- [Distributed and parallel processing in Machine Learning Server](#)
- [Compute context in Machine Learning Server](#)
- [What is RevoScaleR](#)

Running background jobs using RevoScaleR

7/12/2022 • 8 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

In Machine Learning Server, you can run jobs interactively, waiting for results before continuing on to the next operation, or you can run them asynchronously in the background if a job is long-running.

Non-Waiting jobs

By default, all jobs are "waiting jobs" or "blocking jobs" (where control of the R prompt is not returned until the job is complete). As you can imagine, you might want a different interaction model if you are sending time-intensive jobs to your distributed compute context. Decoupling your current session from in-progress jobs will enable jobs to proceed in the background while you continue to work on your R Console for the duration of the computation. This can be useful if you expect the distributed computations to take a significant amount of time, and when such computations are managed by a job scheduler.

Convert a Waiting Job to a Non-Waiting Job

Suppose you submit a job as a "waiting" job, and then realize that you'd prefer to be able to work in your R session on the local computer while it is running.

In Windows, simply pressing the Esc will return the cursor to your screen. Depending on how quickly you press Esc, your job will either be canceled (if it has not yet been accepted by the job scheduler), or will continue to run on the cluster.

Similarly, on Red Hat Enterprise Linux, pressing Ctrl-C will return the cursor to your screen, and either cancel the job or convert it to a non-waiting job.

For all jobs that run on the cluster, the object `rxgLastPendingJob` is automatically created. You can use the `rxgLastPendingJob` object to retrieve your results later or to cancel the job.

Create Non-Blocking Jobs

To create non-waiting jobs, you simply set `wait=FALSE` in your compute context object:

```
myNoWaitCluster <- RxSpark(nameNode = "my-name-service-server", port = 8020), wait=FALSE)
rxOptions(computeContext=myNoWaitCluster)
```

When `wait` is set to `FALSE`, a job information object rather than a job results object is returned from the submitted job. You should always *assign* this result so that you can use it to obtain job status while the job is running and obtain the job results when the job completes. For example, returning to our initial waiting job example, calling `rxExec` to get data set information, in the non-blocking case we augment our call to `rxExec` with an assignment, and then use the assigned object as input to the `rxGetJobStatus` and `rxGetJobResults` functions:

```
airData <- "AirlineData87to08.xdf"
job1 <- rxExec(rxGetInfo, data=airData)
rxGetJobStatus(job1)
```

If you call `rxGetJobStatus` quickly, it may show us that the job is "running", but if called after a few seconds (or longer if another job of higher priority is ahead in the queue) it should report "finished", at which point we can ask for the results:

```
rxGetJobResults(job1)
```

As in the case of the waiting job, we obtain the following results from our five-node cluster (note that the name of the head node is mangled here to be an R syntactic name):

```
$CLUSTER_HEAD2
File name: C:\data\AirlineData87to08.xdf
Number of observations: 123534969
Number of variables: 30
Number of blocks: 832

$COMPUTE10
File name: C:\data\AirlineData87to08.xdf
Number of observations: 123534969
Number of variables: 30
Number of blocks: 832

$COMPUTE11
File name: C:\data\AirlineData87to08.xdf
Number of observations: 123534969
Number of variables: 30
Number of blocks: 832

$COMPUTE12
File name: C:\data\AirlineData87to08.xdf
Number of observations: 123534969
Number of variables: 30
Number of blocks: 832

$COMPUTE13
File name: C:\data\AirlineData87to08.xdf
Number of observations: 123534969
Number of variables: 30
Number of blocks: 832
```

Running a linear model on the airline data is more likely to show us the "running" status:

```
delayArrJobInfo <- rxLinMod(ArrDelay ~ DayOfWeek,
  data=airData, cube=TRUE, blocksPerRead=30)
rxGetJobStatus(delayArrJobInfo)
```

This shows us the following:

```
[1] "running"
```

Calling `rxGetJobStatus` again a few seconds later shows us that the job has completed:

```
rxGetJobStatus(delayArrJobInfo)
```

```
[1] "finished"
```

We can then call `rxGetJobResults` to obtain the actual computation results:

```
delayArr <- rxGetJobResults(delayArrJobInfo)
delayArr
```

As in the blocking case, this gives the following results:

```
Call:
rxLinMod(formula = ArrDelay ~ DayOfWeek, data = "AirlineData87to08.xdf",
  cube = TRUE, blocksPerRead = 30)

Cube Linear Regression Results for: ArrDelay ~ DayOfWeek
File name: C:\data\AirlineData87to08.xdf
Dependent variable(s): ArrDelay
Total independent variables: 7
Number of valid observations: 120947440
Number of missing observations: 2587529

Coefficients:
  ArrDelay
DayOfWeek=Monday   6.669515
DayOfWeek=Tuesday   5.960421
DayOfWeek=Wednesday 7.091502
DayOfWeek=Thursday  8.945047
DayOfWeek=Friday    9.606953
DayOfWeek=Saturday  4.187419
DayOfWeek=Sunday    6.525040
```

Capture Job Information

If you forget to assign the job information object when you first submit your job, don't panic. **RevoScaleR** saves the job information for the last pending job as the object `rsgLastPendingJob`. You can assign this value to a more specific name at any time until you submit another non-blocking job.

```
rxOptions(computeContext=myNoWaitCluster)
rxLinMod(ArrDelay ~ DayOfWeek, data="AirlineData87to08.xdf",
  cube=TRUE, blocksPerRead=30)
delayArrJobInfo <- rsgLastPendingJob
rxGetJobStatus(delayArrJobInfo)
```

NOTE

The `blocksPerRead` argument is ignored if script runs locally using R Client.

Also, as in all R sessions, the last value returned can be accessed as `.Last.value`; if you remember immediately that you forgot to assign the result, you can simply assign `.Last.value` to your desired job name and be done.

For jobs older than the last pending job, you can use `rxGetJobs` to obtain all the jobs associated with a given compute context. More details on `rxGetJobs` can be found in the next section.

Cancel a Non-Waiting Job

Suppose you submit a job and realize you've mis-specified the formula. In the non-waiting case, it is easy to cancel your job simply by calling `rxCancelJob` with the job information object you saved when you submitted the job:

```
rxCancelJob(job1)
```

Non-Waiting Logistic Regression

Logistic regression uses an iteratively re-weighted least squares algorithm, and thus in general requires multiple passes through the data for successive iterations. This makes it a logical candidate for non-waiting distributed computing. For example, we replicated the large airline data set 8 times to create a data set with about one billion observations. We also added a variable "Late" to indicate which flights were at least fifteen minutes late. To find the probability of a late flight by day of week, we perform the following logistic regression:

```
job2 <- rxLogit(Late ~ DayOfWeek, data = "AirlineData87to08Rep8.xdf")
```

This immediately returns control back to our R Console, and we can do some other things while this 1-billion observation logistic regression completes on our distributed computing resources. (Although even with one billion observations, the logistic regression completes in less than a minute.)

We verify that the job is finished and retrieve the results as follows:

```
rxGetJobStatus(job2)
logitResults <- rxGetJobResults(job2)
summary(logitResults)
```

We obtain the following results:

```
Call:
rxLogit(formula = Late ~ DayOfWeek, data = "AirlineData87to08Rep8.xdf")

Logistic Regression Results for: Late ~ DayOfWeek
File name: C:\data\AirlineData87to08Rep8.xdf
Dependent variable(s): Late
Total independent variables: 8 (Including number dropped: 1)
Number of valid observations: 967579520
Number of missing observations: 20700232
-2*LogLikelihood: 947605223.9911 (Residual deviance on 967579513 degrees of freedom)

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.4691555 0.0002209 -6651.17 2.22e-16 ***
DayOfWeek=Monday -0.0083930 0.0003088 -27.18 2.22e-16 ***
DayOfWeek=Tuesday -0.0559740 0.0003115 -179.67 2.22e-16 ***
DayOfWeek=Wednesday 0.0386048 0.0003068 125.82 2.22e-16 ***
DayOfWeek=Thursday 0.1862203 0.0003006 619.41 2.22e-16 ***
DayOfWeek=Friday 0.2388796 0.0002985 800.14 2.22e-16 ***
DayOfWeek=Saturday -0.1785315 0.0003285 -543.45 2.22e-16 ***
DayOfWeek=Sunday Dropped Dropped Dropped Dropped
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Condition number of final variance-covariance matrix: 78.6309
Number of iterations: 2
```

Clean up job artifacts using `rxGetJobs` and `rxCleanJobs` (distributed)

computing)

Normally, whenever a waiting job completes or whenever you call `rxGetJobResults` to obtain the results of a non-waiting job, any artifacts created during the distributed computation are automatically removed. (This is controlled by the `autoCleanup` flag to the compute context constructor, which defaults to `TRUE`.)

However, if a waiting job fails to complete for some reason, or you do not collect all the results from your non-waiting jobs, you may begin to accumulate artifacts on your distributed computing resources. Eventually, this could fill the storage space on these resources, causing system slowdown or malfunction. It is therefore a best practice to make sure you clean up your distributed computing resources from time to time. One way to do this is to simply use standard operating system tools to delete files from the various shared and working directories you specified in your compute context objects. But **RevoScaleR** also supplies a number of tools to help you remove any accumulated artifacts.

The first of these, `rxGetJobs`, allows you to get a list of all the jobs associated with a given compute context. By default, it matches just the head node (if available) and shared directory specified in the compute context; if you re-use these two specifications, ALL the jobs associated with that head node and shared directory are returned:

```
myJobs <- rxGetJobs(myNoWaitCluster)
```

To restrict the matching to only those jobs associated with that specific compute context, specify `exactMatch=TRUE` when calling `rxGetJobs`.

```
myJobs <- rxGetJobs(myNoWaitCluster, exactMatch=TRUE)
```

To obtain the jobs from a specified range of times, use the `startTime` and `endTime` arguments. For example, to obtain a list of jobs for a particular day, you could use something like the following:

```
myJobs <- rxGetJobs(myNoWaitCluster,  
  startTime=as.POSIXct("2013/01/16 0:00"),  
  endTime=as.POSIXct("2013/01/16 23:59"))
```

Once you've obtained the list of jobs, you can try to clean them up using `rxCleanupJobs`:

```
rxCleanupJobs(myJobs)
```

If any of the jobs is in a "finished" state, `rxCleanupJobs` will not clean up that job but instead warn you that the job is finished and that you can access the results with `rxGetJobResults`. This helps prevent data loss. You can, however, force the cleanup by specifying `force=TRUE` in the call to `rxCleanupJobs`:

```
rxCleanupJobs(myJobs, force=TRUE)
```

You can also use `rxCleanupJobs` to clean up individual jobs:

```
rxCleanupJobs(job1)
```

Running jobs in parallel using rxExec

7/12/2022 • 19 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

[RevoScaleR](#) functions are engineered to execute in parallel automatically. However, if you require a custom implementation, you can use [rxExec](#) to manually construct and manage a distributed workload. With rxExec, you can take an arbitrary function and run it in parallel on your distributed computing resources in Hadoop. This in turn allows you to tackle a wide variety of parallel computing problems.

This article provides comprehensive steps on how to use rxExec, starting with examples showcasing rxExec in a variety of use cases.

Examples using rxExec

To demonstrate rxExec usage, this article provides several examples:

- Simulate a dice-rolling game
- Determine the probability that any two persons in a given group size share a birthday
- Create a plot of the Mandelbrot set
- Perform naive k-means clustering

In general, the only required arguments to rxExec are the function to be run and any required arguments of that function. Additional optional arguments can be used to control the computation.

Before trying the examples, be sure that your [compute context](#) is set with the option `wait=TRUE`.

Playing Dice: A Simulation

A familiar casino game consists of rolling a pair of dice. If you roll a 7 or 11 on your initial roll, you win. If you roll 2, 3, or 12, you lose. Roll a 4, 5, 6, 8, 9, or 10, NS that number becomes your *point* and you continue rolling until you either roll your point again (in which case you win) or roll a 7, in which case you lose. The game is easily simulated in R using the following function:

```

playDice <- function()
{
  result <- NULL
  point <- NULL
  count <- 1
  while (is.null(result))
  {
    roll <- sum(sample(6, 2, replace=TRUE))

    if (is.null(point))
    {
      point <- roll
    }
    if (count == 1 && (roll == 7 || roll == 11))
    {
      result <- "Win"
    }
    else if (count == 1 && (roll == 2 || roll == 3 || roll == 12))
    {
      result <- "Loss"
    }
    else if (count > 1 && roll == 7 )
    {
      result <- "Loss"
    }
    else if (count > 1 && point == roll)
    {
      result <- "Win"
    }
    else
    {
      count <- count + 1
    }
  }
  result
}

```

Using rxExec, you can invoke thousands of games to help determine the probability of a win. Using a Hadoop 5-node cluster, we play the game 10000 times, 2000 times on each node:

```

z <- rxExec(playDice, timesToRun=10000, taskChunkSize=2000)
table(unlist(z))

Loss  Win
5087 4913

```

We expect approximately 4929 wins in 10000 trials, and our result of 4913 wins is close.

The Birthday Problem

The birthday problem is an old standby in introductory statistics classes because its result seems counterintuitive. In a group of about 25 people, the chances are better than 50-50 that at least two people in the room share a birthday. Put 50 people in a room and you are practically guaranteed there is a birthday-sharing pair. Since 50 is so much less than 365, most people are surprised by this result.

We can use the following function to estimate the probability of at least one birthday-sharing pair in groups of various sizes (the first line of the function is what allows us to obtain results for more than one value at a time; the remaining calculations are for a single n):

```

"pbirthday" <- function(n, ntests=5000)
{
  if (length(n) > 1L) return(sapply(n, pbirthday, ntests = ntests))

  daysInYear <- seq.int(365)
  anydup <- function(i)
  {
    any(duplicated(sample(daysInYear, size = i, replace = TRUE)))
  }

  prob <- sum(sapply(seq.int(ntests), anydup)) / ntests
  names(prob) <- n
  prob
}

```

We can test that it works in a sequential setting, estimating the probability for group sizes 3, 25, and 50 as follows:

```
pbirthday(c(3,25,50))
```

For each group size, 5000 random tests were performed. For this run, the following results were returned:

```

 3      25      50
0.0078 0.5710 0.9726

```

Make sure your compute context is set to a “waiting” context. Then distribute this computation for groups of 2 to 100 using `rxExec` as follows, using `rxElemArg` to specify a different argument for each call to `pbirthday`, and then using the `taskChunkSize` argument to pass these arguments to the nodes in chunks of 20:

```
z <- rxExec(pbirthday, n=rxElemArg(2:100), taskChunkSize=20)
```

The results are returned in a list, with one element for each node. We can use `unlist` to convert the results into a single vector:

```
probSameBD <- unlist(z)
```

We can make a colorful plot of the results by constructing variables for the party sizes and the nodes where each computation was performed:

```

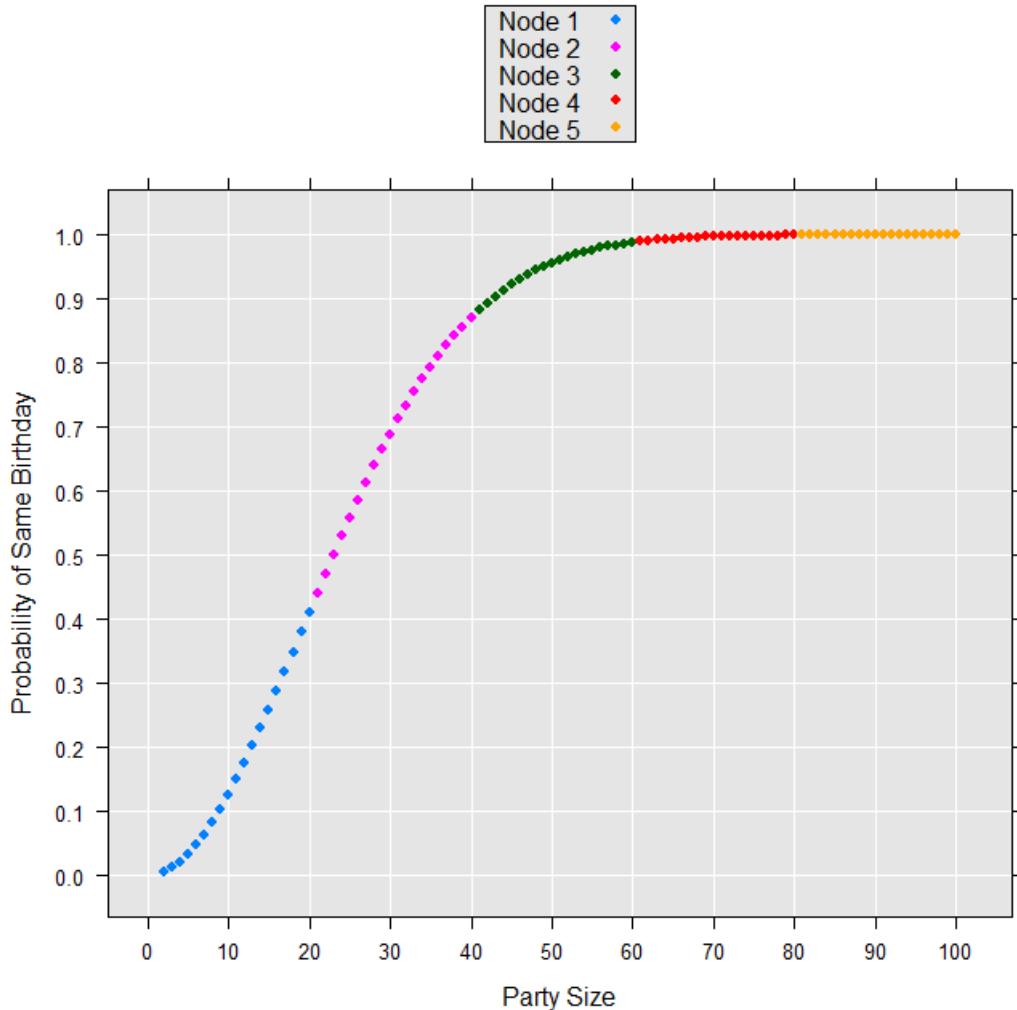
partySize <- 2:100
nodes <- as.factor(rep(1:5, each=20)[2:100])
levels(nodes) <- paste("Node", levels(nodes))
birthdayData <- data.frame(probSameBD, partySize, nodes)

rxLinePlot( probSameBD~partySize, groups = nodes, data=birthdayData,
  type = "p",
  xTitle = "Party Size",
  yTitle = "Probability of Same Birthday",
  title = "Our Rockin Soiree!")

```

The resulting plot is shown as follows:

Our Rockin Soiree!



Plotting the Mandelbrot Set

Computing the Mandelbrot set is a popular parallel computing example because it involves a simple computation performed independently on an array of points in the complex plane. For any point $z = x + yi$ in the complex plane, z belongs to the Mandelbrot set if and only if z remains bounded under the iteration $z_{(n+1)} = z_n^2 + z_n$. If we are associating a point (x_0, y_0) in the plane with a pixel on a computer screen, the following R function returns the number of iterations before the point becomes unbounded, or the maximum number of iterations. If the maximum number of iterations is returned, the point is assumed to be in the set:

```
mandelbrot <- function(x0,y0,lim)
{
  x <- x0; y <- y0
  iter <- 0
  while (x^2 + y^2 < 4 && iter < lim)
  {
    xtemp <- x^2 - y^2 + x0
    y <- 2 * x * y + y0
    x <- xtemp
    iter <- iter + 1
  }
  iter
}
```

The following function retains the basic computation but returns a vector of results for a given y value:

```

vmandelbrot <- function(xvec, y0, lim)
{
  unlist(lapply(xvec, mandelbrot, y0=y0, lim=lim))
}

```

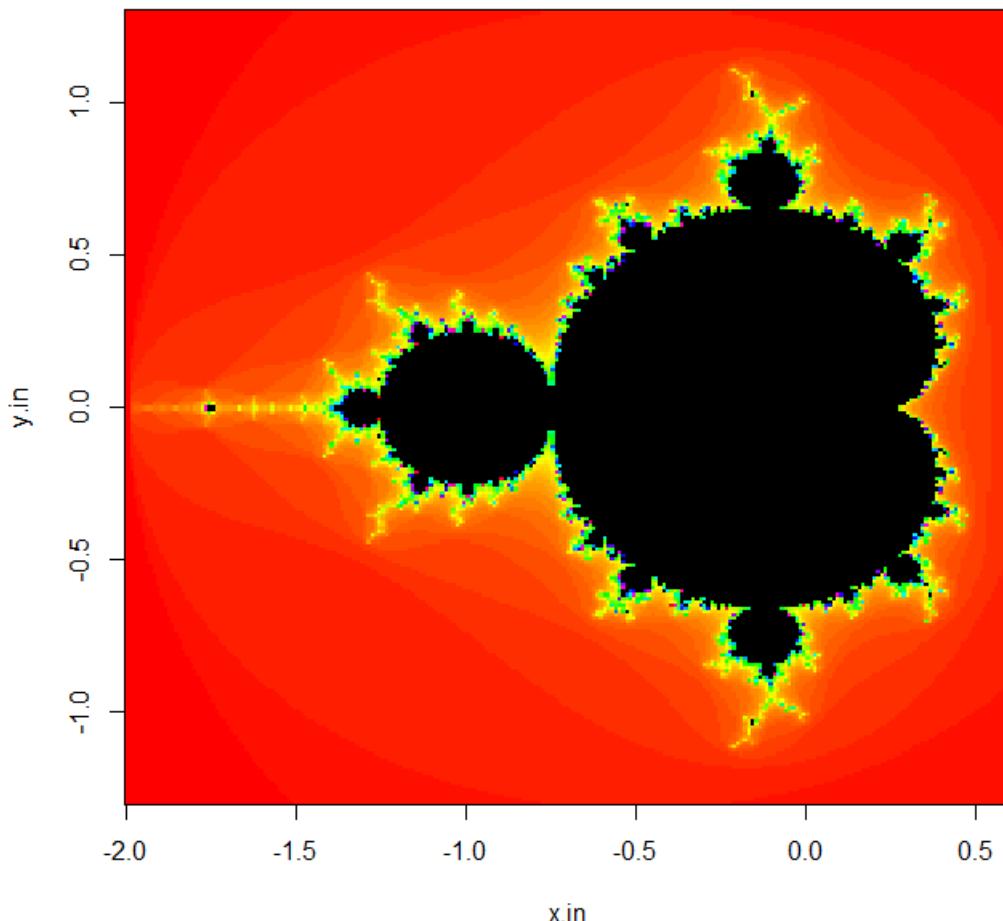
We can then distribute this computation by computing several rows at a time on each compute resource. In the following, we create an input `x` vector of length 240, a `y` vector of length 240, and specify the iteration limit as 100. We then call `rxExec` with our `vmandelbrot` function, giving 1/5 of the `y` vector to each computational node in our five node HPC Server cluster. This should be done in a compute context with `wait=TRUE`. Finally, we put the results into a 240x240 matrix and create an image plot that shows the familiar Mandelbrot set:

```

x.in <- seq(-2.0, 0.6, length.out=240)
y.in <- seq(-1.3, 1.3, length.out=240)
m <- 100
z <- rxExec(vmandelbrot, x.in,y0=rxElemArg(y.in), m, taskChunkSize=48,
  execObjects="mandelbrot")
z <- matrix(unlist(z), ncol=240)
image(x.in, y.in, z, col=c(rainbow(m), '#000000'), useRaster=TRUE)

```

The resulting plot is shown as follows (not all graphics devices support the `useRaster` argument; if your plot is empty, try omitting that argument):



Naïve Parallel k-Means Clustering

`RevoScaleR` has a built-in analysis function, `rxKmeans`, to perform distributed k-means, but in this section we see how the regular R `kmeans` function can be put to use in a distributed context.

The kmeans function implements several *iterative relocation* algorithms for clustering. An iterative relocation algorithm starts from an initial classification and then iteratively moves data points from one cluster to another to reduce sums of squares. One possible starting point is to pick cluster centers at random and then assign points to each cluster so that the sum of squares is minimized. If this procedure is repeated many times for different sets of centers, the set with the smallest error can be chosen.

We can do this with the ordinary kmeans function, which has a parameter *nstart* that tells it how many times to pick the starting centers, and also to pick the set of centers that returns a result with smallest error:

```
x <- matrix(rnorm(250000), nrow = 5000, ncol = 50)
system.time(kmeans(x, centers=10, iter.max = 35, nstart = 400))
```

On a Dell XPS laptop with 8 GB of RAM, this takes about a minute and a half.

To parallelize this computation efficiently, we should do the following:

- Pass the data to a specified number of computing resources once.
- Split the work into smaller tasks for passing to each computing resource.
- Combine the results from all the computing resources so the best result is returned.

In the case of *kmeans*, we can ask for the computations to be done by *cores*, rather than by *nodes*. Because we are redistributing the computation, we can do fewer repetitions (*nstarts*) on each compute element. We can do all of this with the following function (again, this should be run with a compute context for which *wait=TRUE*):

```
kMeansRSR <- function(x, centers=5, iter.max=10, nstart=1)
{
  numTimes <- 20
  results <- rxExec(FUN = kmeans, x=x, centers=centers, iter.max=iter.max,
                     nstart=nstart, timesToRun=numTimes)
  best <- 1
  bestSS <- sum(results[[1]]$withinss)
  for (j in 1:numTimes)
  {
    jSS <- sum(results[[j]]$withinss)
    if (bestSS > jSS)
    {
      best <- j
      bestSS <- jSS
    }
  }
  results[[best]]
}
```

Notice that in our *kMeansRSR* function we are letting the underlying *kmeans* function find *nstart* sets of centers per call and the choice of "best"

is done in our function after we have called *kmeans numTimes*. No parallelization is done to *kmeans* itself.

With our *kMeansRSR* function, we can then repeat the computation from before:

```
system.time(kMeansRSR(x, 10, 35, 20))
```

With our 5-node HPC Server cluster, this reduces the time from a minute and a half to about 15 seconds.

Share data across parallel processes

Data can be shared between rxExec parallel processes by copying it to the environment of each process through the `execObjects` option to rxExec, or by specifying the data as arguments to each function call. For small data, this works well but as the data objects get larger this can create a significant performance penalty due to the

time needed to do the copy. In such cases, it can be much more efficient to share the data by storing it in a location accessible by each of the parallel processes, such as a local or network file share.

The following example shows how this can be done when parallelizing the computation of statistics on subsets of a larger data table.

We'll start by creating some sample data using the **data.table** package.

```
library(data.table)

nrows <- 5e6
test.data <- data.table(tag=sample(LETTERS[1:7], nrows, replace=TRUE),
                         a=runif(nrows), b=runif(nrows))

# get the unique tags that we'll subset by
tags <- unique(test.data$tag)
tags
```

Next we'll setup for use of the **doParallel** backend.

```
library(doParallel)
(nCore <- parallel::detectCores(all.tests = TRUE, logical = FALSE) )
cl <- makeCluster(nCore, useXDR = F)
registerDoParallel(cl)
rxSetComputeContext(RxForeachDoPar())

# set MKL thread to 1 to avoid contention
setMKLthreads(1)
```

Now we create a function for computing statistics on a selected subset of the data table by passing in both the data table and the tag value to subset by. We'll then run the function using rxExec for each tag value.

```
filterTest <- function(d, ztag) {
  sum(subset(d, tag==ztag, select=a))
}
system.time({
  res1 <- rxExec(filterTest, test.data, rxElemArg(tags))
})
##   user   system elapsed
##  9.67   7.40  22.05
```

To see the impact of sharing the file via a common storage location rather than passing it to each parallel process we'll save the data table into an RDS file, which is an efficient way of storing individual R objects, create a new version of the function that reads the data table from the RDS file, and then rerun rxExec using the new function.

```

saveRDS(test.data, 'c:/temp/tmp.rds', compress=FALSE)
filterTest <- function(ztag) {
d <- readRDS('c:/temp/tmp.rds')
sum(subset(d, tag==ztag, select=a))
}

system.time({
res2 <- rxExec(filterTest, rxElemArg(tags))
})
##   user   system elapsed
##  0.03    0.00   11.01

# make sure the results match
identical(res2, res2)
## [1] TRUE

# close up shop
stopCluster(c1)

```

Although results may vary, in this case we've reduced the elapsed time by 50% by sharing the data table rather than passing it as an in-memory object to each parallel process.

Parallel Random Number Generation

When generating random numbers in parallel computation, a frequent problem is the possibility of highly correlated random number streams. High-quality parallel random number generators avoid this problem. RevoScaleR includes several high-quality parallel random number generators and these can be used with `rxExec` to improve the quality of your parallel simulations.

By default, a parallel version of the Mersenne-Twister random number generator is used that supports 6024 separate substreams. We can set it to work on our dice example by setting a non-null seed:

```

z <- rxExec(playDice, timesToRun=10000, taskChunkSize=2000, RNGseed=777)
table(unlist(z))

```

This makes our simulation repeatable:

```

Loss  Win
5104 4896

z <- rxExec(playDice, timesToRun=10000, taskChunkSize=2000, RNGseed=777)
table(unlist(z))

Loss  Win
5104 4896

```

This random number generator can be asked for explicitly by specifying `RNGkind="MT2203"`:

```

z <- rxExec(playDice, timesToRun=10000, taskChunkSize=2000, RNGseed=777,
RNGkind="MT2203")
table(unlist(z))

Loss  Win
5104 4896

```

We can build reproducibility into our naïve k-means example as follows:

```

kMeansRSR <- function(x, centers=5, iter.max=10, nstart=1, numTimes = 20, seed = NULL)
{
  results <- rxExec(FUN = kmeans, x=x, centers=centers, iter.max=iter.max,
    nstart=nstart, timesToRun=numTimes, RNGseed = seed)
  best <- 1
  bestSS <- sum(results[[1]]$withinss)
  for (j in 1:numTimes)
  {
    jSS <- sum(results[[j]]$withinss)
    if (bestSS > jSS)
    {
      best <- j
      bestSS <- jSS
    }
  }
  results[[best]]
}
km1 <- kMeansRSR(x, 10, 35, 20, seed=777)
km2 <- kMeansRSR(x, 10, 35, 20, seed=777)
all.equal(km1, km2)

[1] TRUE

```

To obtain the default random number generators without setting a seed, specify "auto" as the argument to either RNGseed or RNGkind:

```

x3 <- rxExec(runif, 500, timesToRun=5, RNGkind="auto")
x4 <- rxExec(runif, 500, timesToRun=5, RNGseed="auto")

```

To verify that we are actually getting uncorrelated streams, we can use runif within rxExec to generate a list of vectors of random vectors, then use the cor function to measure the correlation between vectors:

```

x <- rxExec(runif, 500, timesToRun=5, RNGkind="MT2203")
x.df <- data.frame(x)
corx <- cor(x.df)
diag(corx) <- 0
any(abs(corx) > 0.3)

```

Correlations are above 0.3; in repeated runs of the code, the maximum correlation seldom exceeded 0.1.

Because the MT2203 generator offers such a rich array of substreams, we recommend its use. You can, however, use several other generators, all from Intel's Vector Statistical Library, a component of the Intel Math Kernel Library. The available generators are as follows: "MCG31", "R250", "MRG32K3A", "MCG59", "MT19937", "MT2203", "SFMT19937" (all of which are pseudo-random number generators that can be used to generate uncorrelated random number streams) plus "SOBOL" and "NIEDERR", which are quasi-random number generators that do not generate uncorrelated random number streams. Detailed descriptions of the available generators can be found in the [Vector Statistical Library Notes](#).

About reproducibility

For distributed compute contexts, rxExec starts the random number streams on a per-worker basis; if there are more tasks than workers, you may not obtain completely reproducible results because different tasks may be performed by randomly chosen workers. If you need completely reproducible results, you can use the taskChunkSize argument to force the number of task *chunks* to be less than or equal to the number of workers. This will ensure that each chunk of tasks is performed on a single random number stream. You can also define a custom function that includes random number generation control within it; this moves the random number control into each task. See the help file for rxRngNewStream for details.

Work with results from non-blocking jobs

So far, all of our examples have required a blocking, or waiting, compute context so that we could make immediate use of the results returned by rxExec. However some computations are so time consuming that it is not practical to wait on the results. In such cases, it is probably best to divide your analysis into two or more pieces, one of which can be structured as a non-blocking job, and then use the pending job (or more usefully, the job results, when available) as input to the remaining pieces.

For example, let's return to the birthday example, and see how to restructure our analysis to use a non-blocking job for the distributed computations. The pbirthday function itself requires no changes, and our variable specifying the number of ntests can be used as is:

```
"pbirthday" <- function(n, ntests=5000)
{
  if (length(n) > 1L) return(sapply(n, pbirthday, ntests = ntests))

  daysInYear <- seq.int(365)
  anydup <- function(i)
  {
    any(duplicated(sample(daysInYear, size = n, replace = TRUE)))
  }

  prob <- sum(sapply(seq.int(ntests), anydup)) / ntests
  names(prob) <- n
  prob
}
ntests <- 2000
```

However, when we call rxExec, the return object will no longer be the results list, but a jobInfo object:

```
z <- rxExec(pbirthday, n=rxElemArg(2:100), ntests=ntests, taskChunkSize=20)
```

We check the job status:

```
rxGetJobStatus(z)

[1] "finished"
```

We can then proceed almost as before:

```
probSameBD <- unlist(rxGetJobResults(z))
partySize <- 2:100
nodes = as.factor(rep(1:5, each=20)[2:100])
levels(nodes) <- paste("Node", levels(nodes))
birthdayData <- data.frame(probSameBD, partySize, nodes)

rxLinePlot( probSameBD~partySize, groups = nodes, data=birthdayData,
  type = "p",
  xTitle = "Party Size",
  yTitle = "Probability of Same Birthday",
  title = "Our Rockin Soiree!")
```

The other examples are a bit trickier, in that the result of the calls to rxExec were embedded in functions. But again, dividing the computations into distributed and non-distributed components can help—the distributed computations can be non-blocking, and the non-distributed portions can then be applied to the results. Thus the kmeans example can be rewritten thus:

```

genKmeansClusters <- function(x, centers=5, iter.max=10, nstart=1)
{
  numTimes <- 20
  rxExec(FUN = kmeans, x=x, centers=centers, iter.max=iter.max,
  nstart=nstart, timesToRun=numTimes)
}

findKmeansBest <- function(results){
  numTimes <- length(results)
  best <- 1
  bestSS <- sum(results[[1]]$withinss)
  for (j in 1:numTimes)
  {
    jSS <- sum(results[[j]]$withinss)
    if (bestSS > jSS)
    {
      best <- j
      bestSS <- jSS
    }
  }
  results[[best]]
}

```

To run this in our non-blocking cluster context, we do the following:

```

x <- matrix(rnorm(250000), nrow = 5000, ncol = 50)
z <- genKmeansClusters(x, 10, 35, 20)

```

Once we see that z's job status is "finished", we can run findKmeansBest on the results:

```
findKmeansBest(rxGetJobResults(z))
```

Call RevoScaleR HPA functions with rxExec

To this point, none of the functions we have called with rxExec has been a RevoScaleR function, because the intent has been to show how rxExec can be used to address the large class of traditional high-performance computing problems.

However, there is no inherent reason why rxExec cannot be used with RevoScaleR's high performance analysis (HPA) functions, and many times it can be useful to do so. For example, if you are running a cluster on which every node has two or more cores, you can use rxExec to start an independent analysis on each node, and each of those analyses can take advantage of the multiple cores on its node.

The following simulation simulates data from a Poisson distribution and then fits a generalized linear model to the simulated data:

```

"SimAndEstimatePoisson" <- function(nobs, trials)
{
  "SimulatePoissonData" <- function(nobs)
  {
    x1 <- log(runif(nobs, min=.5, max=1.5))
    x2 <- log(runif(nobs, min=.5, max=1.5))

    b0 <- 0
    b1 <- 1
    b2 <- 2
    lambda <- exp(b0 + b1*x1 + b2*x2)
    count <- rpois(nobs,lambda)
    pSim <- data.frame(count=count, x1=x1, x2=x2)
  }

  cf <- NULL
  rxOptions(reportProgress = 0)
  for (i in 1:trials)
  {
    simData <- SimulatePoissonData(nobs)
    result1 <- rxGlm(count~x1+x2,data=simData,family=poisson())
    cf <- rbind(cf,as.double(coefficients(result1)))
  }
  cf
}

```

If we call the above function with `rxExec` on a five-node cluster compute context, we get five simulations running simultaneously, and can easily produce 1000 simulations as follows:

```
rxExec(SimAndEstimatePoisson, nobs=50000, trials=10, taskChunkSize=5, timesToRun=100)
```

It is important to recognize the distinction between running an HPA function with a distributed compute context, and calling an HPA function using `rxExec` with a distributed compute context. In the former case, we are fitting just one model, using the distributed compute context to farm out portions of the computations, but ultimately returning just one model object. In the latter case, we are calculating one model per task, the tasks being farmed out to the various nodes or cores as desired, and a list of models is returned.

Use `rxExec` in the local compute context

By default, if you call `rxExec` in the local compute context, your computation runs sequentially on your local machine. However, you can incorporate parallel computing on your local machine using the special compute context `RxLocalParallel` as follows:

```
rxSetComputeContext(RxLocalParallel())
```

This allows the ParallelR package `doParallel` to distribute the computation among the available cores of your computer.

If you are using random numbers in the local parallel context, be aware that `rxExec` chooses a number of workers based on the number of tasks and the current value of `rxGetOption("numCoresToUse")`, to guarantee each task runs with a separate random number stream, set `rxOptions(numCoresToUse)` equal to the number of tasks, and explicitly set `timesToRun` to the number of tasks. For example, if we want a list consisting of five sets of uniform random numbers, we could do the following to obtain reproducible results:

```
rxOptions(numCoresToUse=5)
x1 <- rxExec(runif, 500, timesToRun=5, RNGkind="MT2203", RNGseed=14)
x2 <- rxExec(runif, 500, timesToRun=5, RNGkind="MT2203", RNGseed=14)
all.equal(x1, x2)
```

numCoresToUse is a scalar integer specifying the number of cores to use. If you set this parameter to either -1 or a value in excess of available cores, ScaleR uses however many cores are available. Increasing the number of cores also increases the amount of memory required for ScaleR analysis functions.

NOTE

HPA functions are not affected by the `RxLocalParallel` compute context; they run locally and in the usual internally distributed fashion when the `RxLocalParallel` compute context is in effect.

Use `rxExec` with `foreach` back ends

If you do not have access to a Hadoop cluster or enterprise database, but do have access to a cluster via PVM, MPI, socket, or NetWorkSpaces connections or a multicore workstation, you can use `rxExec` with an arbitrary `foreach` backend (doParallel, doSNOW, doMPI, etc.) Register your parallel backend as usual and then set your RevoScaleR compute context using the special compute context `RxForeachDoPar`:

```
rxSetComputeContext(RxForeachDoPar())
```

For example, here is how you might start a SNOW-like cluster connection with the `doParallel` back end:

```
library(doParallel)
cl <- makeCluster(4)
registerDoParallel(cl)
rxSetComputeContext(RxForeachDoPar())
```

You then call `rxExec` as usual. The computations are automatically directed to the registered `foreach` back end.

WARNING

HPA functions are not usually affected by the `RxForeachDoPar` compute context; they run locally and in the usual internally distributed fashion when the `RxForeachDoPar` compute context is in effect. The one exception is when HPA functions are called within `rxExec`; in this case it is possible that the internal threading of the HPA functions can be affected by the launch mechanism of the parallel backend workers. The `doMC` backend and the multicore-like backend of `doParallel` both use forking to launch their workers; this is known to be incompatible with the HPA functions.

Control `rxExec` computations

As we have seen in these examples, there are several arguments to `rxExec` that allow you to fine-tune your `rxExec` commands. Both the birthday example and the Mandelbrot example used the `taskChunkSize` argument to specify how many tasks should go to each worker. The Mandelbrot example also used the `execObjects` argument, which can be used to pass either a character vector or an environment containing objects—the objects specified by the vector or contained in the environment are added to the environment of the function specified in the `FUN` argument, unless that environment is locked, in which case they are added to the parent frame in which `FUN` is evaluated. (If you use an environment, it should be one you create with `parent=emptyenv()`;) this allows you to pass only those objects you need to the function's environment.) These two examples also show the use of `rxElemArg` in passing arguments to the workers. In the kmeans example, we

covered the *timesToRun* argument. The *packagesToLoad* argument allows you to specify packages to load on each worker. The *consoleOutput* and *autoCleanup* flags serve the same purpose as their counterparts in the compute context constructor functions—that is, they can be used to specify whether console output should be displayed or the associated task files should be cleaned up on job completion for an individual call to 'rxExec'.

Two additional arguments remain to be introduced: *oncePerElem* and *continueOnFailure*. The *oncePerElem* argument restricts the called function to be run once per allotted node; this is frequently used with the *timesToRun* argument to ensure that each occurrence is run on a separate node. The *oncePerElem* argument, however, can only be set to *TRUE* if *elemType*= "nodes". It must be set to *FALSE* if *elemType*= "cores".

If *oncePerElem* is *TRUE* and *elemType*= "nodes", *rxExec*'s results are returned in a list with components named by node. If a given node does not have a valid R syntactic name, its name is mangled to become a valid R syntactic name for use in the return list.

The *continueOnFailure* argument is used to say that a computation should continue even if one or more of the compute elements fails for some reason; this is useful, for example, if you are running several thousand independent simulations and it doesn't matter if you get results for all of them. Using *continueOnFailure*=*TRUE* (the default), you get results for all compute elements that finish the simulation and error messages for the compute elements that fail.

NOTE

The arguments *elemType*, *consoleOutput*, *autoCleanup*, *continueOnFailure*, and *oncePerElem* are ignored by the special compute contexts 'RxLocalParallel' and 'RxForeachDoPar'.

Conclusion

This article provides comprehensive documentation on *rxExec* for writing custom script that executes jobs in parallel. Several use cases were used to show how *rxExec* is used in context, with additional sections providing guidance on relevant tasks associated with custom execution, such as sharing data, structuring jobs, and controlling calculations. To learn more, see these related links:

- [Distributed and parallel computing in Machine Learning Server](#)
- [Using foreach and iterators for manual parallel execution](#)
- [Parallel execution using doRSR for script containing RevoScaleR and foreach constructs](#)

Enforcing YARN queue usage

7/12/2022 • 4 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

R Server tasks running on Spark or MapReduce can be managed through use of YARN job queues. To direct a job to a specific queue, the end user must include the queue name in the MapReduce or Spark Compute Context.

MapReduce

Use the "hadoopSwitches" option to direct jobs to a specific YARN queue.

```
RxHadoopMR(..., hadoopSwitches=' -Dmapreduce.job.queuename=mrsjobs')
```

Spark

Use the "extraSparkConfig" option to direct jobs to a specific YARN queue.

```
RxSpark(..., extraSparkConfig=' --conf spark.yarn.queue=mrsjobs')  
  
RxSparkConnect(...,  
    extraSparkConfig=' --conf spark.yarn.queue=mrsjobs')
```

Overrides

Use of a specific queue can be enforced by the Hadoop system administrator by providing an installation override to the `RxSpark()`, `RxSparkConnect()`, and `RxHadoopMR()` compute context functions. A benefit is that you no longer have to explicitly specify the queue.

This procedure involves creating a custom R package that contains the function overrides, installing that package on the nodes in use by end users, and adding the package to the default search path on these nodes. The following code block provides an example. If you use this code as a template, remember to change the 'mrsjobs' YARN queue name to the queue name that's valid for your system.

1. Create a new R package, such as "abcMods" if your company abbreviation is 'abc', by installing the "devtools" package and running the following command, or use the `package.skeleton()` function in base R. To learn more about creating R packages, see [Writing R Extensions](#) on the CRAN website, or [online version of R Packages](#) from Hadley Wickham.

```
> library(devtools)  
> create('/dev/abcMods', rstudio=FALSE)
```

2. This creates the essential package files in the requested directory, in this case '/dev/abcMods'. Edit each of the following to fill in the relevant info.

DESCRIPTION – text file containing the description of the R package:

```
Package: abcMods
Date: 2016-09-01
Title: Company ABC Modified Functions
Depends: RevoScaleR
Description: R Server functions modified for use by Company ABC
License: file LICENSE
```

NAMESPACE – text file containing the list of overridden functions to be exported:

```
export("RxHadoopMR", "RxSpark", "RxSparkConnect")
```

LICENSE – create a text file named 'LICENSE' in the package directory with a single line for the license associated with the R package:

```
This package is for internal Company ABC use only -- not for redistribution.
```

3. In the package's R directory add one or more `*.R` files with the code for the functions to be overridden. The following sample code provides for overriding `RxHadoopMR`, `RxSpark`, and `RxSparkConnect` that you might save to a file called "ccOverrides.r" in that directory. The `RxSparkConnect` function is only available in V9 and later releases.

```

# sample code to enforce use of YARN queues for RxHadoopMR, RxSpark,
# and RxSparkConnect

RxHadoopMR <- function(...) {
  dotargs <- list(...)
  hswitch <- dotargs$hadoopSwitches

  # remove any queue info that may already present
  y <- hswitch
  if (isTRUE(grep('queue',hswitch) > 0)) {
    hswitch <- gsub(' *= *','=',hswitch,perl=TRUE)
    hswitch <- gsub(' +',' ',hswitch)
    x <- unlist(strsplit(hswitch," "))
    i <- grep('queue',x)
    y <- paste(x[- c(i-1,i)],collapse=' ')
  }

  # add in the required queue info
  dotargs$hadoopSwitches <- paste(y,'-Dmapreduce.job.queuename=mrsjobs')
  do.call( RevoScaleR::RxHadoopMR, dotargs )
}

RxSpark <- function(...) {
  dotargs <- list(...)
  hswitch <- dotargs$extraSparkConfig

  # remove any queue info that may already present
  y <- hswitch
  if (isTRUE(grep('queue',hswitch) > 0)) {
    hswitch <- gsub(' *= *','=',hswitch,perl=TRUE)
    hswitch <- gsub(' +',' ',hswitch)
    x <- unlist(strsplit(hswitch," "))
    i <- grep('queue',x)
    y <- paste(x[- c(i-1,i)],collapse=' ')
  }

  # add in the required queue info
  dotargs$extraSparkConfig <- paste(y,'--conf spark.yarn.queue=mrsjobs')
  do.call( RevoScaleR::RxSpark, dotargs )
} }

RxSparkConnect <- function(...) {
  dotargs <- list(...)
  hswitch <- dotargs$extraSparkConfig

  # remove any queue info that may already present
  y <- hswitch
  if (isTRUE(grep('queue',hswitch) > 0)) {
    hswitch <- gsub(' *= *','=',hswitch,perl=TRUE)
    hswitch <- gsub(' +',' ',hswitch)
    x <- unlist(strsplit(hswitch," "))
    i <- grep('queue',x)
    y <- paste(x[- c(i-1,i)],collapse=' ')
  }

  # add in the required queue info
  dotargs$extraSparkConfig <- paste(y,'--conf spark.yarn.queue=mrsjobs')
  do.call( RevoScaleR::RxSparkConnect, dotargs )
}

```

4. After editing the previous components of the package, run the following Linux commands to build the package from the directory containing the **abcMods** directory:

```
R CMD build abcMods  
R CMD INSTALL abcmods_0.1-0      (if needed run this using sudo)
```

If you need to build the package for users on Windows, then the equivalent commands would be as follows where 'rpath' is defined to point to the version of R Server to which you'd like to add the library.

```
set rpath="C:\Program Files\Microsoft\R Client\R_SERVER\bin\x64\R.exe"  
%rpath% CMD build abcMods  
%rpath% CMD INSTALL abcMods_0.1-0.tar.gz
```

5. To test it, start R, load the library, and make a call to `RxHadoopMR()` :

```
> library(abcMods)  
> RxHadoopMR(hadoopSwitches="-Dmapreduce.job.queuename=XYZ")
```

You should see the result come back with the queue name set to your override value (for example, `Dmapreduce.job.queuename=mrsjobs`) .

6. To automate the loading of the package so that users don't need to specify "library(abcMods)", edit `Rprofile.site` and modify the line specifying the default packages to include `abcMods` as the last item:

```
options(defaultPackages=cgetOption("defaultPackages"), "rpart", "lattice", "RevoScaleR", "RevoMods",  
"RevoUtils", "RevoUtilsMath", "abcMods"))
```

7. Once everything tests out to your satisfaction, install the package on all the edge nodes that your users are logging in to. To do this, copy "`Rprofile.site`" and R's library/`abcMods` directory to each of these nodes, or install the package from the `abcmods_0.1-0` tar file on each node and manually edit the "`Rprofile.site`" file on each node.

Cheat sheet: How to choose a MicrosoftML algorithm

7/12/2022 • 5 minutes to read • [Edit Online](#)

IMPORTANT

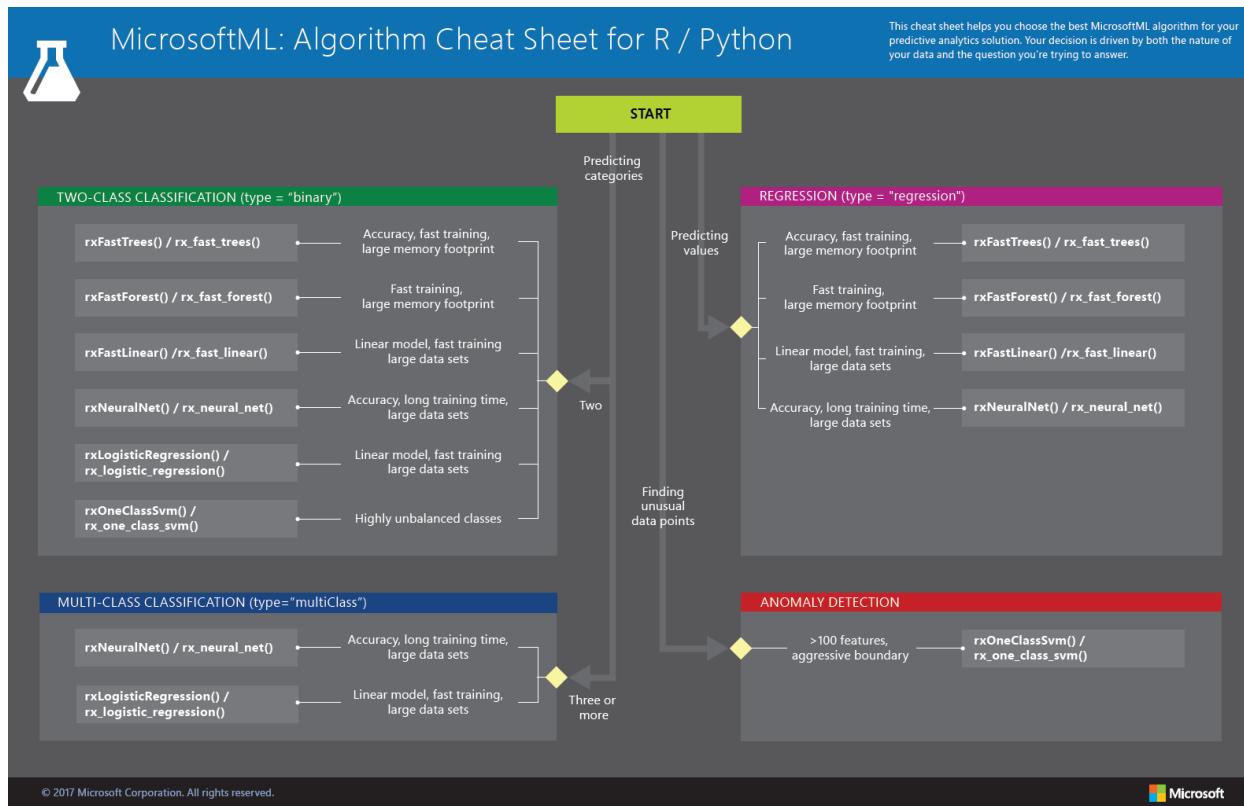
This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

The **MicrosoftML: Algorithm Cheat Sheet** helps you choose the right machine learning algorithm for a predictive analytics model when using Machine Learning Server. The algorithms are available in R or Python.

MicrosoftML provides a library of algorithms from the **regression**, **classification (two-class and multi-class)**, and **anomaly detection** families. Each is designed to address a different type of machine learning problem.

Download the MicrosoftML Algorithm Cheat Sheet

Download the cheat sheet here: [MicrosoftML Package: Algorithm Cheat Sheet v2 \(11x17 in.\)](#)



Download and print the **MicrosoftML: Algorithm Cheat Sheet** in tabloid size to keep it handy for guidance when choosing a machine learning algorithm.

MicrosoftML machine learning algorithms

This section contains descriptions of the machine learning algorithms contained in the Algorithm Cheat Sheet. The algorithms are available in R or Python. The R And Python names are provided in the format:

****R name/Python name** .**

Fast Linear model (SDCA)

The `rxFastTrees() / rx_fast_trees()` algorithm is based on the Stochastic Dual Coordinate Ascent (SDCA) method, a state-of-the-art optimization technique for convex objective functions. The algorithm can be scaled for use on large out-of-memory data sets due to a semi-asynchronized implementation that supports multithreaded processing. Several choices of loss functions are also provided and elastic net regularization is supported. The SDCA method combines several of the best properties and capabilities of logistic regression and SVM algorithms.

Tasks supported: binary classification, linear regression

OneClass SVM

The `rxOneClassSvm() / rx_one_class_svm()` algorithm is used for one-class anomaly detection. This is a type of unsupervised learning as its training set contains only examples from the target class and not any anomalous instances. It infers what properties are normal for the objects in the target class and from these properties predicts which examples are unlike these normal examples. This is useful as typically there are very few examples of network intrusion, fraud, or other types of anomalous behavior in training data sets.

Tasks supported: anomaly detection

Fast Tree

The `rxFastTrees() / rx_fast_trees()` algorithm is a high performing, state of the art scalable boosted decision tree that implements FastRank, an efficient implementation of the MART gradient boosting algorithm. MART learns an ensemble of regression trees, which is a decision tree with scalar values in its leaves. For binary classification, the output is converted to a probability by using some form of calibration.

Tasks supported: binary classification, regression

Fast Forest

The `rxFastForest() / rx_fast_forest()` algorithm is a random forest that provides a learning method for classification that constructs an ensemble of decision trees at training time, outputting the class that is the mode of the classes of the individual trees. Random decision forests can correct for the overfitting to training data sets to which decision trees are prone.

Tasks supported: binary classification, regression

Neural Network

The `rxNeuralNet() / rx_neural_net()` algorithm supports a user-defined multilayer network topology with GPU acceleration. A neural network is a class of prediction models inspired by the human brain. It can be represented as a weighted directed graph. Each node in the graph is called a neuron. The neural network algorithm tries to learn the optimal weights on the edges based on the training data. Any class of statistical models can be considered a neural network if they use adaptive weights and can approximate non-linear functions of their inputs. Neural network regression is especially suited to problems where a more traditional regression model cannot fit a solution.

Tasks supported: binary and multiclass classification, regression

Logistic regression

The `rxLogisticRegression() / rx_logistic_regression()` algorithm is used to predict the value of a categorical dependent variable from its relationship to one or more independent variables assumed to have a logistic distribution. If the dependent variable has only two possible values (success/failure), then the logistic regression is binary. If the dependent variable has more than two possible values (blood type given diagnostic test results), then the logistic regression is multinomial.

Tasks supported: binary and multiclass classification

Ensemble methods

The `rxEnsemble()` / `rx_ensemble()` algorithm uses a combination of learning algorithms to provide better predictive performance than the algorithms could individually. The approach is used primarily in the Hadoop/Spark environment for training across a multi-node cluster. But it can also be used in a single-node/local context.

Tasks supported: binary and multiclass classification, regression

More help with algorithms

For a list by category of all the machine learning algorithms available in the MicrosoftML package, see:

- [MicrosoftML R functions](#)
- [MicrosoftML Python functions](#)

Notes and terminology definitions for the machine learning algorithm cheat sheet

- The suggestions offered in this algorithm cheat sheet are approximate rules-of-thumb. Some can be bent and some can be flagrantly violated. This sheet is only intended to suggest a starting point. Don't be afraid to run a head-to-head competition between several algorithms on your data. There is simply no substitute for understanding the principles of each algorithm and understanding the system that generated your data.
- Every machine learning algorithm has its own style or *inductive bias*. For a specific problem, several algorithms may be appropriate and one algorithm may be a better fit than others. But anticipating which will be the best fit beforehand is not always possible. In cases like these, several algorithms are listed together in the cheat sheet. An appropriate strategy would be to try one algorithm, and if the results are not yet satisfactory, try the others.
- Two categories of machine learning are supported by MicrosoftML: **supervised learning** and **unsupervised learning**.
 - In **supervised learning**, each data point is labeled or associated with a category or value of interest. The goal of supervised learning is to study many labeled examples like these, and then to be able to make predictions about future data points. All of the algorithms in MicrosoftML are supervised learners except `rxOneClassSvm()` used for anomaly detection.
 - In **unsupervised learning**, data points have no labels associated with them. Instead, the goal of an unsupervised learning algorithm is to organize the data in some way or to describe its structure. Only the `rxOneClassSvm()` algorithm used for anomaly detection is an unsupervised learner.

What's next?

[Quickstarts for MicrosoftML](#) shows how to use pre-trained models for sentiment analysis and image featurization.

Authenticate with Machine Learning Server in Python with azureml-model-management-sdk

7/12/2022 • 4 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Applies to: Machine Learning Server

The [azureml-model-management-sdk](#) package, delivered with Machine Learning Server, provides functions for publishing and managing a Python web service that is backed by the Python code block or script you provided.

This section describes how to authenticate with Machine Learning Server on-premises or in the cloud using `azureml-model-management-sdk`. Every API call between the client and the Web node must be authenticated. The `azureml-model-management-sdk` functions, which place API calls on your behalf, are no different. If the user does not provide a valid login, an `Unauthorized` HTTP `401` status code is returned.

Overview

`azureml-model-management-sdk` provides the client that supports several ways of authenticating against the Machine Learning Server. Authentication of user identity is handled via Active Directory. Machine Learning Server never stores or manages any usernames and passwords. Ask your administrator for [authentication type configured](#) for Machine Learning Server and the connection details.

By default, all web services operations are available to authenticated users. Tasks, such as deleting or updating a web service, are available only to the user who initially created the service. However, your administrator can also [assign role-based authorization controls](#) to further restrict the permissions around web services.

On premises authentication

Use this approach if you are:

- authenticating using Active Directory server on your network
- using the [default administrator account](#)

Pass the username and password as a Python tuple for on premises authentication. If you do not know your connection settings, contact your administrator.

```

# Import the DeployClient and MLServer classes from
# the azureml-model-management-sdk package so you can
# connect to Machine Learning Server (use=MLServer).

from azureml.deploy import DeployClient
from azureml.deploy.server import MLServer

# Define the location of the Machine Learning Server
HOST = '{{https://YOUR_HOST_ENDPOINT}}'
# And provide your username and password as a Python tuple
# for local admin with password Pass123!
# context = ('admin', 'Pass123!')
context = ('{{YOUR_USERNAME}}', '{{YOUR_PASSWORD}}')
client = DeployClient(HOST, use=MLServer, auth=context)

```

This code calls the `/user/login` API.

ARGUMENT	DESCRIPTION
host endpoint	Required. The Machine Learning Server HTTP/HTTPS endpoint, including the port number.
username	Required. Enter your AD username or 'admin' if default administrator account defined.
password	Required. Enter the password.

Cloud authentication (AAD)

Use this approach if you are authenticating using Azure Active Directory in the cloud.

Pass the credentials as a Python dictionary {} for AAD authentication.

```

# First, import the DeployClient and MLServer classes from
# the azureml-model-management-sdk package so you can
# connect to Machine Learning Server (use=MLServer).

from azureml.deploy import DeployClient
from azureml.deploy.server import MLServer

# Define the endpoint of the host Machine Learning Server.
HOST = '{{YOUR_HOST_ENDPOINT}}'

# Pass in credentials for the AAD context as a dictionary.
# Omit username & password to use ADAL to authenticate.
context = {
    'authuri': 'https://login.windows.net',
    'tenant': '{{AAD_DOMAIN}}',
    'clientid': '{{NATIVE_APP_CLIENT_ID}}',
    'resource': '{{WEB_APP_CLIENT_ID}}',
    'username': '{{YOUR_USERNAME}}',
    'password': '{{YOUR_PASSWORD}}'
}

client = DeployClient(HOST, use=MLServer, auth=context)

```

If you do not know your tenant ID, clientid, or other details, contact your administrator. Or, if you have access to the Azure portal for the relevant Azure subscription, you can find [these authentication details](#).

ARGUMENT	DESCRIPTION
host endpoint	Required. The Machine Learning Server HTTP/HTTPS endpoint, including the port number. This endpoint is the SIGN-ON URL value from the web application
authuri	Required. The URI of the authentication service for Azure Active Directory.
tenant	Required. The tenant ID of the Azure Active Directory account being used to authenticate is the domain of AAD account.
clientid	Required. The numeric CLIENT ID of the AAD "native" application for the Azure Active Directory account.
resource	Required. The numeric CLIENT ID from the AAD "Web" application for the Azure Active Directory account, also known by the Audience in the configuration file.
username	Optional. If NULL, user is prompted. See following section.
password	Optional. If NULL, user is prompted. See following section.

Alternatives to putting the username and password in the script

If you omit the username and password from the dictionary for the AAD context, then you can either:

- Get a device code to complete authentication and token creation via Azure Active Directory Authentication Libraries (ADAL). Enter that code at <https://aka.ms/devicelogin> to complete the authentication.
- Programmatically authenticate using a call back, such as:

```
def callback_fn(code):
    print(code)

context = {
    'authuri': 'https://login.windows.net',
    'tenant': '{{AAD_DOMAIN}}',
    'clientid': '{{NATIVE_APP_CLIENT_ID}}',
    'resource': '{{WEB_APP_CLIENT_ID}}',
    'user_code_callback': callback_fn
}
```

A note about access tokens

Keep in mind that all APIs require authentication; therefore, all users must authenticate when making an API call using the `POST /login` API or through Azure Active Directory.

To simplify this process, bearer access tokens are issued so that users need not provide their credentials for every single call. This bearer token is a lightweight security token that grants the "bearer" access to a protected resource, in this case, Machine Learning Server's APIs. After authentication, the user does not need to provide credentials again as long as the token is still valid, and a header is submitted with every request. The application must validate the user's bearer token to ensure that authentication was successful for the intended parties.

[Learn more about managing these tokens.](#)

Deploy and manage web services in Python

7/12/2022 • 4 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Applies to: Machine Learning Server

This article is for data scientists who wants to learn how to deploy and manage Python code/models as [analytic web services](#) hosted in Machine Learning Server. This article assumes you are proficient in Python.

Using the [azureml-model-management-sdk](#) Python package, which ships with Machine Learning Server, you can develop, test, and [deploy these Python analytics](#) as web services in your production environment. This package can also be [installed locally on a Windows machine](#), but requires a connection to a Machine Learning Server instance at runtime. [RESTful APIs](#) are also available to provide direct programmatic access to a service's lifecycle.

By default, web service operations are available to authenticated users. However, your administrator can also [assign role-based authorization \(RBAC\)](#) to further control the permissions around web services.

IMPORTANT

Web service definition: In Machine Learning Server, your R and Python code and models can be deployed as web services. Exposed as web services hosted in Machine Learning Server, these models and code can be accessed and consumed in R, Python, programmatically using REST APIs, or using Swagger generated client libraries. Web services can be deployed from one platform and consumed on another. [Learn more...](#)

Overview

To deploy your analytics, you must publish them as web services in Machine Learning Server. Web services offer fast execution and scoring of arbitrary Python or R code and models. [Learn more about web services](#).

When you deploy Python code or a model as a web service, a [service object](#) containing the client stub for consuming that service is returned.

Once hosted on Machine Learning Server, you can update and manage them. Authenticated users can [consume web services in Python](#) or in a [preferred language via Swagger](#).

Requirements

Before you can use the web service management functions in the [azureml-model-management-sdk](#) Python package, you must:

- Have access to a Python-enabled instance of Machine Learning Server that was [properly configured](#) to host web services.
- Authenticate with Machine Learning Server in Python as described in "[Connecting to Machine Learning Server](#)."

Web service definitions

The list of available parameters that can be defined for a web service depends on whether it is a standard or real-time web service.

Parameters for standard web services

Standard web services, like all web services, are identified by their name and version. Additional parameters can be defined for the service, including:

- A description
- Arbitrary Python code as a function or a string, such as `code_fn=(run, init)` or `code_str=('run', 'init')`
- Models
- Any model assets
- Inputs
- Outputs for integrators
- Artifacts

For a full example, try out the [quickstart guide for deploying web services in Python](#).

This table presents the supported data types for the input and output schemas of Python web services. Write these as a reference `bool` or as a string `'bool'`.

I/O DATA TYPES	DESCRIPTION
Float → float	Array → numpy.array
Integer → int	Matrix → numpy.matrix
Boolean → bool	Dataframe → numpy.DataFrame
String → str	

Parameters for real-time web services

[Real-time web services](#) are also identified by their name and version. The following additional parameters can be defined for the real-time service:

- A description
- A model created with certain [supported functions](#) and serialized with [revoscalepy.rx_serialize_model](#)

Dataframes are assumed for inputs and outputs. Code is not supported.

For a full example of real-time web services in Python, try out [this Jupyter notebook](#).

Deploy and update services

Publish a service

To publish a service, specify the name, version, and any other needed parameters. Be sure to end with '.deploy'.

Here is a standard web service example:

```
service = client.service('myService')\
    .version('v1.0')\
    .description('cars model')\
    .code_fn(manualTransmission, init)\
    .inputs(hp=float, wt=float)\\
{{...}}\
.deploy()
```

Here is a real-time web service example:

```
# Serialize the model using revoscalepy.rx_serialize_model
from revoscalepy import rx_serialize_model
s_model = rx_serialize_model(model, realtime_scoring_only=True)

# Publish the real-time service
webserv = client.realtime_service('myService') \
    .version('1.0') \
    .serialized_model(s_model) \
    .description('this is a realtime model') \
    .deploy()
```

For a full example of real-time web services in Python, try out [this Jupyter notebook](#).

Update an existing service version

To update an existing web service, specify the name and version of that service and the parameters that need changed. When you use '.redeploy', the service version is overwritten and a new [service object](#) containing the client stub for consuming that service is returned.

Update the service with a '.redeploy' such as:

```
# Redeploy this standard service 'myService' version 'v1.0'
# Only the description has changed.
service = client.service('myService')\
    .version('v1.0')\
    .description('The updated description for cars model')\
    .redeploy()
```

Publish a new service version

You can deploy different versions of a web service. To publish a [new version of the service](#), specify the same name, a new version, and end with '.deploy', such as:

```
service = client.service('myService')\
    .version('v2.0')\
    .description('cars model')\
    .code_fn(manualTransmission, init)\
    .inputs(hp=float, wt=float)\\
{{...}}\
.deploy()
```

Delete web services

When you no longer want to keep a web service that you have published, you can delete it. If you are [assigned to "Owner" role](#), you can also delete the services of others.

You can call 'delete_service' on the 'DeployClient' object to delete a specific service on Machine Learning Server.

```
# -- List all services to find the one to delete--
client.list_services()
# -- Now delete myService v1.0
client.delete_service('myService', version='v1.0')
```

If successful, the status '*True*' is returned. If it fails, then the execution stops and an error message is returned.

See also

[What are web services?](#)

[Quickstart: Deploying web services in Python.](#)

[How to authenticate in Python](#)

[How to find, get, and consume web services](#)

[How to consume web services in Python synchronously \(request/response\)](#)

[How to consume web services in Python asynchronously \(batch\)](#)

List, get, and consume web services in Python

7/12/2022 • 4 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Applies to: Machine Learning Server

This article is for data scientists who wants to learn how to find, examine, and consume the [analytic web services](#) hosted in Machine Learning Server using Python. Web services offer fast execution and scoring of arbitrary Python or R code and models. [Learn more about web services](#). This article assumes that you are proficient in Python.

After a web service has been published, any authenticated user can list, examine, and consume that web service. You can do so directly in Python using the functions in the [azureml-model-management-sdk package](#). The `azureml-model-management-sdk` package is installed with Machine Learning Server. To list, examine, or consume the web service *outside* of Python, use the [RESTful APIs](#) that provide direct programmatic access to a service's lifecycle or in a [preferred language via Swagger](#).

By default, web service operations are available to authenticated users. However, your administrator can also assign [roles](#) (RBAC) to further control the permissions around web services.

IMPORTANT

Web service definition: In Machine Learning Server, your R and Python code and models can be deployed as web services. Exposed as web services hosted in Machine Learning Server, these models and code can be accessed and consumed in R, Python, programmatically using REST APIs, or using Swagger generated client libraries. Web services can be deployed from one platform and consumed on another. [Learn more...](#)

Requirements

Before you can use the web service management functions in the [azureml-model-management-sdk](#) Python package, you must:

- Have access to a Python-enabled instance of Machine Learning Server that was [properly configured](#) to host web services.
- Authenticate with Machine Learning Server in Python as described in "[Connecting to Machine Learning Server in Python](#)."

Find and list web services

Any authenticated user can retrieve a list of web services using the `list_services` function on the `DeployClient` object. You can use arguments to return a specific web service or all labeled versions of a given web service.

```
## -- Return metadata for all services hosted on this server
client.list_services()

## -- Return metadata for all versions of service "myService"
client.list_services('myService')

## -- Return metadata for a specific version "v1.0" of service "myService"
client.list_services('myService', version='v1.0')
```

Once you find the service you want, use the [get_service](#) function to retrieve the service object for consumption.

Retrieve and examine service objects

Authenticated users can retrieve the [web service object](#) in order to get the client stub for consuming that service. Use the `get_service` function from the `azureml-model-management-sdk` package to retrieve the object.

After the object is returned, you can use a help function to explore the published service, such as

```
print(help(myServiceObject))
```

You can call the help function on any `azureml-model-management-sdk` functions, even those that are dynamically generated to learn more about them.

You can also print the capabilities that define the service holdings to see what the service can do and how it should be consumed. Service holdings include the service name, version, descriptions, inputs, outputs, and the name of the function to be consumed. [Learn more about capabilities...](#)

You can use [supported public functions](#) to interact with service object.

Example code:

```
# -- List all versions of the service 'myService'--
client.list_services('myService')

# -- Retrieve the service object for myService v2.0
svc = client.get_service('myService', version='v2.0')

# -- Learn more about that service.
print(help(svc))

# -- View the service object's capabilities/schema
svc.capabilities()
```

NOTE

You can only see the code stored within a web service if you have published the web service or are assigned to the "Owner" role. To learn more [about roles](#) in your organization, contact your Machine Learning Server administrator.

Consume web services

Web services are published to facilitate the consumption and integration of the operationalized models and code into systems and applications. Whenever the web service is deployed or updated, a Swagger-based JSON file, which defines the service, is automatically generated.

When you publish a service, you can let people know that it is ready for consumption by sharing its name and version with them. Web services are [consumed by data scientists, quality engineers, and application developer](#). They can be consumed in Python, R, or via the API.

Users can consume the service directly using a single consumption call, which is referred to as a "Request

Response" approach. [Learn about the various approaches to consuming web services.](#)

Consuming in Python

After authenticating with Machine Learning Server, users can also interact with and consume services using other functions in the `azureml-model-management-sdk` Python package.

[For a full example of a request-response consume, see this Jupyter notebook](#)

```
# Let's call the function `manualTransmission` in this service
res = svc.manualTransmission(120, 2.8)

# Pluck out the named output `answer`.
print(res.output('answer'))

# Get `swagger.json` that defines the service.
cap = svc.capabilities()
swagger_URL = cap['swagger']
print(swagger_URL)

# Print the contents of the swagger file.
print(svc.swagger())
```

Sharing the Swagger with application developers

Application developers can call and integrate a web service into their applications using the service-specific Swagger-based JSON file and by providing any required inputs to that service. The Swagger-based JSON file is used to generate client libraries for integration. Read "[How to integrate web services and authentication into your application](#)" for more details.

The easiest way to share the Swagger file with an application developer is to use the code shown [in this Jupyter notebook](#). Alternately, the application developer can request the file as an authenticated user with an [active bearer token](#) in the request header using this API:

```
GET /api/{{service-name}}/{{service-version}}/swagger.json
```

See also

- [What are web services?](#)
- [Jupyter notebook: how explore and consume a web service](#)
- [Package overview: azureml-model-management-sdk](#)
- [Quickstart: Deploying an Python model as a web service](#)
- [How to authenticate with Machine Learning Server in Python.](#)
- [How to publish and manage web services in Python](#)
- [How to integrate web services and authentication into your application](#)

Asynchronous web service consumption via batch processing in Python

7/12/2022 • 7 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Applies to: Machine Learning Server

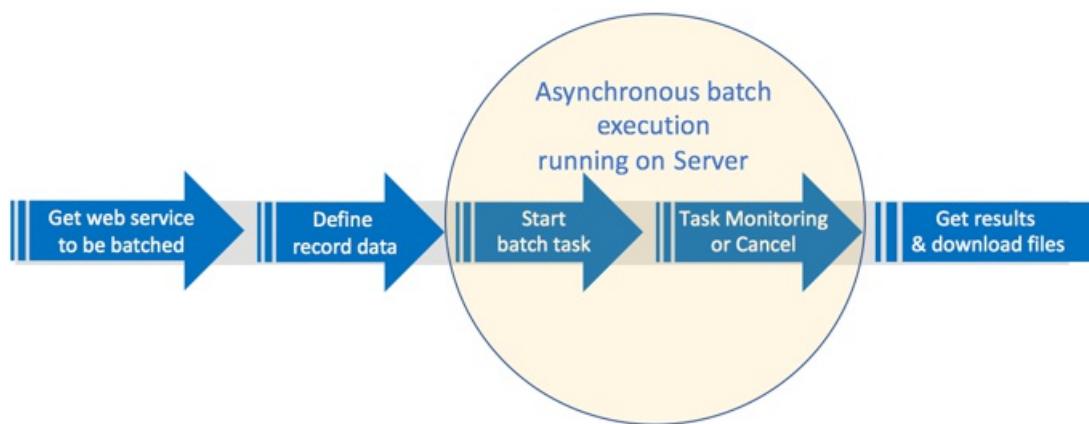
In this article, you can learn how to consume a web service asynchronously, which is especially useful with large input data sets and long-running computations. The typical approach to consuming web services in Python, "[Request Response" consumption](#)", involves a single API call to execute the code in that web service once. The "Asynchronous Batch" approach involves the execution of code without manual intervention using multiple asynchronous API calls on a specific web service sent as a single request to Machine Learning Server. Then, Machine Learning Server immediately executes those operations once for every row of data provided.

Asynchronous batch workflow

Generally speaking, the process for asynchronous batch consumption of a web service involves the following:

1. Call the web service on which the batch execution should be run
2. Define the data records for the batch execution task
3. Start (or cancel) the batch execution task
4. Monitor task and interact with results

Use these following [public API functions](#) to define, start, and interact with your batch executions.



Example code

The batch example code is stored in a Jupyter notebook. This notebook format allows you to not only see the code alongside detailed explanations, but also allows you to try out the code.

This example walks through the asynchronous consumption of a Python model as a web service hosted in Machine Learning Server. Consume a simple linear model built using the [rx_lin_mod](#) function from the [revoscalepy package](#).

► [Download the Jupyter notebook to try it out.](#)

Public functions for batch

You can use the following supported public functions to consume a service asynchronously.

Batch functions performed on the service object

Once you get the service object, use these public functions on that object.

FUNCTION	USAGE	DESCRIPTION
<code>batch</code>	view	Define the data records to be batched and the thread count
<code>get_batch_executions</code>	view	Get the list of batch execution identifiers
<code>get_batch</code>	view	Get batch object using its unique execution identifier

Batch functions performed on the batch object

Once you have the batch object, use these public functions to interact with it.

FUNCTION	DESCRIPTION	USAGE
<code>start</code>	view	Starts the execution of a batch scoring operation
<code>cancel</code>	view	Cancel the named batch execution
<code>execution_id</code>	view	Get the execution identifier for the named batch process
<code>state</code>	view	Poll for the state of the batch execution (failed, complete, ...)
<code>results</code>	view	Poll for batch execution results, partial or full results as defined
<code>execution</code>		Get results for a given index row returned as an array
<code>list_artifacts</code>	view	List of every artifact files that was generated by this execution index
<code>artifact</code>	view	Print the contents of the named artifact file generated by the batch execution
<code>download</code>	view	Download one or all artifact files from execution index

1. Get the web service

Once you have authenticated, retrieve the web service from the server, assign it to a variable, and define the inputs to it as record data in a data frame, CSV, or TSV.

Batching begins by retrieving the web service containing the code against which you score the data records you define next. You can get a service using its name and version with the `get_service` function from `azureml-model-management-sdk`. The result is a service object.

The `get_service` function is covered in detail in the article "[How to interact with and consume web services in R](#)."

2. Define the data records to be batched

Next, use the public api function `batch` to define the input record data for the batch and set the number of concurrent threads for processing.

Syntax: `batch(inputs, parallelCount = 10)`

ARGUMENT	DESCRIPTION
<code>inputs</code>	Specify the data.frame name directly
<code>parallelCount</code>	Default value is 10. Specify the number of concurrent threads that can be dedicated to processing records in the batch. Take care not to set a number so high that it negatively impacts performance.

Returns: The batch object

Example:

```
# -- Import the dataset from the microsoftml package
from microsoftml.datasets.datasets import get_dataset
mtcars = get_dataset('mtcars')

# -- Represent the dataset as a dataframe.
mtcars = mtcars.as_df()

# -- Define the data for the execution.
records = mtcars[['hp', 'wt']]

# -- Assign the record data to the batch service and set the thread count.
batch = svc.batch(records, parallelCount = 10)
```

3. Start, find, or cancel the batch execution

Next, use the public api functions to start the asynchronous batch execution on the batch object, monitor the execution, or even cancel it.

Start batch execution task

Start the batch task with `start()`. Machine Learning Server starts the batch execution and assigns an ID to the execution and returns the batch object.

Syntax: `start()`

No arguments.

Returns: The batch object

Example: `batch = batch.start()`

NOTE

We recommend you always use the `id` function after you start the execution so that you can find it easily with this id later such as: `id = batch.execution_id print("Batch execution_id is: {}".format(id))`

Get batch ID

Get the batch task's identifier from the service object so you can reference it during or after its execution using `id()`.

Syntax: `execution_id`

No arguments.

Returns: The ID for the named batch object.

Example: `id = batch.execution_id`

Get batch by ID

Retrieve the `Batch` based on an `execution_id`.

Syntax: `get_batch(execution_id)`

ARGUMENT	DESCRIPTION
<code>execution_id</code>	The batch execution identifiers

Returns: The `Batch` object

List the Batch execution identifiers

Gets all batch executions currently queued for this service.

Syntax: `list_batch_executions()`

No arguments

Returns: List of batch execution identifiers

Example: `list_batch_executions()`

Cancel execution

Cancel the batch execution using `cancel()`.

Syntax: `cancel()`

No arguments.

Returns: The batch object

Example: `batch = batch.cancel()`

4. Monitor, retrieve, and interact with results

While the batch task is running, you can monitor and poll the results. Once the batch task has completed, you can get the web service consumption output by index from the batch results object, including:

- Monitor execution results and status
- Get results for a given index row returned as an array
- Get a list of every file that was generated by this execution index
- Print the contents of a specific artifact or all artifacts returned
- Download artifacts from execution index

Monitor execution results and status

There are several public functions you can use to get the results and status of a batch execution.

– Monitor or get the batch execution results

- **Syntax:** `results(showPartialResults = TRUE)`

ARGUMENT	DESCRIPTION
<code>showPartialResults</code>	This argument returns the already processed results of the batch execution even if it has not been fully completed. If <code>showPartialResults = FALSE</code> , then returns only the results if the execution has completed.

- **Returns:** A batch result object is returned, which in our example is called `batchRes`.

– Get the status of the batch execution.

- **Syntax:** `state`
no arguments
- **Returns:** The status of the batch execution.

Example: In this example, we return partial results every three seconds until the batch execution fails or completes. Then, we return results for a given index row returned as an array.

```
batchRes = None
while(True):
    batchRes = batch.results()
    print(batchRes)
    if batchRes.state == "Failed":
        print("Batch execution failed")
        break
    if batchRes.state == "Complete":
        print("Batch execution succeeded")
        break
    print("Polling for asynchronous batch to complete...")
    time.sleep(1)
```

Get list of generated artifacts

Retrieve a list of every artifact that was generated during the batch execution for a given data record, or index, with `listArtifacts()`. This function can be made part of a loop to get the list of the artifacts for every data record (see workflow example for a loop).

Syntax: `list_artifacts(index)`

ARGUMENT	DESCRIPTION
<code>index</code>	Index value for a given batch data record

Returns: A list of every artifact that was generated during the batch execution for a given data record

Example:

```
# List every artifact generated by this execution index for a specific row.  
lst_artifact = batch.list_artifacts(1)  
print(lst_artifact)
```

Display artifact contents

Display the contents of a named artifact returned in the preceding list with `artifact()`. Machine Learning Server returns the ID for the named batch object.

Syntax: `artifact(index, fileName)`

ARGUMENT	DESCRIPTION
<code>index</code>	Index value for a given batch data record
<code>fileName</code>	Name of file artifact created during batch execution

Returns: The ID for the named batch object

Example:

```
# Then, get the contents of each artifact returned in the previous list.  
# The result is a byte string of the corresponding object.  
for obj in lst_artifact:  
    content = batch.artifact(1, obj)
```

Download generated artifacts

Download any artifacts from a specific execution index using `download()`. By default, artifacts are downloaded to the current working directory `getwd()` unless a different `dest = "<path>"` is specified. You can choose to download a specific artifact or all artifacts.

Syntax: `download(index, fileName, dest = "<path>")`

ARGUMENT	DESCRIPTION
<code>index</code>	Index value for a given batch data record
<code>fileName</code>	Name of specific artifact generated during batch execution. If omitted, all artifacts are downloaded for that index.

ARGUMENT	DESCRIPTION
dest	The download directory on your local machine. The default is the current working directory. The directory must already exist on the local machine.

Returns: The path to each downloaded artifact.

Example:

In this example, we download a named artifact for the first index to the current working directory.

```
batch.download(1, "answer.csv")
```

See also

- [What are web services](#)
- [Functions in azureml-model-management-sdk package](#)
- [Connecting to Machine Learning Server in Python](#)
- [Working with web services in Python](#)
- [How to consume web services in Python synchronously \(request/response\)](#)
- [How to consume web services in Python asynchronously \(batch\)](#)
- [How to integrate web services and authentication into your application](#)
- [Get started for administrators](#)
- [User Forum](#)

How to create and manage session pools for fast web service connections in Python

7/12/2022 • 2 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

**Applies to: Machine Learning Server 9.3 (Python) | Machine Learning Server 9.4 (Python) **

Fast connections to a web service are possible when you create sessions and load dependencies in advance. Sessions are available in a pool dedicated to a specific web service, where each session includes an instance of the Python interpreter and a copy of dependencies required by the web service.

For example, if you create ten sessions in advance for a web service that uses numpy, pandas, scikit, revoscalepy, microsoftml, and azureml-model-management-sdk, each session would have its own instance of the Python interpreter plus a copy of each module loaded into memory.

A web service having a dedicated session pool never requests connections from the [generic session pool](#) shared resource, not even when maximum sessions are reached. The generic session pool services only those web services that do not have dedicated resources.

For Python script, the [MLServer](#) class in the [azureml-model-management-sdk](#) function library provides three functions for creating and managing sessions:

- [create_or_update_service_pool](#)
- [get_service_pool_status](#)
- [delete_service_pool](#)

Create or modify a dedicated session pool

You can use a [Python interactive window](#) to run the following commands on the local server if you configured Machine Learning Server for one-box, or on a compute node if you have a distributed topology.

```

# load modules and classes
from azureml.deploy import DeployClient
from azureml.deploy.server import MLServer

# Set up the connection
# Be sure to replace the password placeholder
host = "http://localhost:12800"
ctx = ("admin", "password-placeholder")
client = DeployClient(host, use=MLServer, auth=ctx)

# Return a list of web services to get the service and version
# Both service name and version number are required
svc = client.list_services()
print(svc[0]['name'])
print(svc[0]['version'])

# Create the session pool using a case-sensitive web service name
# A status code of 200 is returned upon success
svc = client.create_or_update_service_pool(name = "myWebservice1234", version = "v1.0.0", initial_pool_size = 5, max_pool_size = 10 )

# Return status
# Pending indicates session creation is in progress. Success indicates sessions are ready.
svc = client.get_service_pool_status(name = "myWebService1234", version = "v1.0.0")
print(svc[0])

```

Currently, there are no commands or functions that return actual session pool usage. The log file is your best resource for analyzing connection and service activity. For more information, see [Trace user actions](#).

Delete a session pool

On the compute node, run the following command to delete the session pool for a given service.

```

# Return a list of web services to get the service and version information
svc = client.list_services()
print(svc[0]['name'])
print(svc[0]['version'])

# Deletes the dedicated session pool and releases resources
svc = client.delete_service_pool(name = "myWebService1234", version = "v1.0.0")

```

This feature is still under development. In rare cases, the [delete_service_pool](#) command may fail to actually delete the pool on the computeNode. If you encounter this situation, issue a new [delete_service_pool](#) request. Use the [get_service_pool_status](#) command to monitor the status of dedicated pools on the computeNode.

```

# Deletes the dedicated session pool and releases resources
client.delete_service_pool(name = "myWebService1234", version = "v1.0.0")

# Check the real-time status of dedicated pool
client.get_service_pool_status(name = "myWebService1234", version = "v1.0.0")

# make sure the returned status is NotFound on all computeNodes
# if not, issues another delete_service_pool command again

```

See also

- [What are web services in Machine Learning Server?](#)
- [Evaluate web service capacity: generic session pools](#)

How to publish and manage R web services in Machine Learning Server with mrsdeploy

7/12/2022 • 19 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Applies to: Machine Learning Server, Microsoft R Server 9.x

This article details how you can publish and manage your [analytic web services](#) directly in R. You can deploy your R models, scripts, and code as web services using the functions in the [mrsdeploy R package](#). The mrsdeploy R package containing these functions is installed with both Machine Learning Server (and Microsoft R Server) and Microsoft R Client.

These web services can be [consumed in R](#) by other authenticated users or in the [language of their choice via Swagger](#).

Using the mrsdeploy R package, you can [publish](#), [update](#), and [delete](#) two kinds of R web services: [standard R web services](#) and [real-time R web services](#).

Additionally, you can get a [list of all services](#), retrieve a [web service object](#) for consumption, and [share services](#) with others. You can also publish or interact with a web service outside of R using the [RESTful APIs](#), which provide direct programmatic access to a service's lifecycle.

Requirements

Before you can use the web service management functions in the mrsdeploy R package, you must:

- Have access to a Machine Learning Server instance that was [properly configured](#) to host web services.
- Authenticate with Machine Learning Server using the `remoteLogin()` or `remoteLoginAAD()` functions in the mrsdeploy package as described in "[Connecting to Machine Learning Server to use mrsdeploy](#)."

Permissions for managing web services

Any authenticated user can:

- Update and delete web services they have published
- Retrieve any web service object for consumption
- Retrieve a list of any or all web services

While any authenticated user can also publish a web service by default, roles can be used to further control permissions. Beginning in version 9.1, your administrator can also [assign role-based authorization](#) to further restrict the permissions around web services to give some users more control over web services than others. Ask your administrator for details on your role.

Publish and update web services

To deploy your analytics, you must publish them as web services in Machine Learning Server. Once hosted on Machine Learning Server, you can update and manage them. They can also be consumed by other users.

Versioning

Every time a web service is published, a version is assigned to the web service. Versioning enables users to better manage the release of their web services. Versions help the users consuming your service to easily identify it.

At publish time, you can specify an alphanumeric string that is meaningful to the users who consume the service in your organization. For example, you could use '2.0', 'v1.0.0', 'v1.0.0-alpha', or 'test-1'. Meaningful versions are helpful when you intend to share services with others. We highly recommend a **consistent and meaningful versioning convention** across your organization or team such as [semantic versioning](#).

If you do not specify a version, a globally unique identifier (GUID) is automatically assigned by Machine Learning Server. These GUID version numbers are harder to remember by the users consuming your services and are therefore less desirable.

Publish service

To deploy your analytics, you must publish them as web services in Machine Learning Server. Once hosted on Machine Learning Server, you can update and manage them. They can also be consumed by other users.

After you've authenticated, use the `publishService()` function in the `mrsdeploy` package to publish a web service. See the [package reference for `publishService\(\)`](#) for the full description of all arguments.

FUNCTION	RESPONSE	R HELP
<code>publishService(...)</code>	Returns an API instance client stub for consuming that service and viewing its service holdings) as an R6 class.	View

You can publish web services to a local Machine Learning Server from your command line. If you [create a remote session](#), you can also publish a web service to a remote Machine Learning Server from your local command line.

Standard web services

[Standard web services](#), like all web services, are identified by their name and version. Additionally, a standard web service is also defined by any code, models, and any necessary model assets. When deploying, you should also define the required inputs and any output the application developers use to integrate the service in their applications. Additional arguments are possible -- many of which are shown in the following example.

Example of standard web service:

```
# Publish a standard service 'mtService' version 'v1.0.0'  
# Assign service to 'api' variable  
api <- publishService(  
  "mtService",  
  code = manualTransmission,  
  model = carsModel,  
  inputs = list(hp = "numeric", wt = "numeric"),  
  outputs = list(answer = "numeric"),  
  v = "v1.0.0"  
)
```

For a full example, you can also follow the quickstart article "[Deploying an R model as a web service](#)." You can also see the Workflow examples at the end of this article.

R SOURCE	CAN COME FROM
R code	<ul style="list-style-type: none"> - A filepath to a local R script, such as: <code>code = "/path/to/R/script.R"</code> - A block of R code as a character string, such as: <code>code = "result <- x + y"</code> - A function handle, such as: <code>code = function(hp, wt) { newdata <- data.frame(hp = hp, wt = wt) predict(model, newdata, type = "response") }</code>
R model	<p>The R model can come from an object or a file-path to an external representation of R objects to be loaded and used with the code, including:</p> <ul style="list-style-type: none"> - A filepath to an RData file holding the external R objects to be loaded and used with the code, such as: <code>model = "/path/to/glm-model.RData"</code> - A filepath to an R file that is evaluated into an environment and loaded, such as: <code>model = "/path/to/glm-model.R"</code> - A model object, such as: <code>model = am.glm</code>

Real-time web services

Real-time web services offer even lower latency to produce results faster and score more models in parallel.

[Learn more...](#)

Real-time web services are also identified by their name and version. However, unlike standard web services, you cannot specify the following for real-time web services:

- inputs and outputs (dataframes are assumed)
- code (only [certain models are supported](#))

Real-time web services only accept model objects created with the [supported functions](#) from packages installed with the product.

Example of real-time service (supported since R Server 9.1):

```
# Publish a real-time service 'kyphosisService' version 'v1.0'  
# Assign service to 'realtimeApi' variable  
realtimeApi <- publishService(  
  serviceType = "Realtime",  
  name = "kyphosisService",  
  code = NULL,  
  model = kyphosisModel,  
  v = "v1.0",  
  alias = "kyphosisService"  
)
```

For [full examples](#), see the end of this article.

Learn how to get a [list of all services](#), retrieve a [web service object](#) for consumption, and [share services](#) with others.

Update service

To change a web service after you've published it, while retaining the same name and version, use the `updateService()` function. For arguments, specify what needs to change, such as the R code, model, and inputs.

When you update a service, it overwrites that named version.

After you've authenticated, use the `updateService()` function in the `mrsdeploy` package to update a web service.

See the [package reference help page for `updateService\(\)`](#) for the full description of all arguments.

FUNCTION	RESPONSE	R HELP
<code>updateService(...)</code>	Returns an API instance client stub for consuming that service and viewing its service holdings) as an R6 class.	View

NOTE

If you want to change the name or version number, use the `publishService()` function instead and specify the new name or version number.

Example:

```
# For web service called mtService with version number v1.0.0,  
# update the model carsModel, code, inputs, and description.  
# Assign it to a variable called api.  
api <- updateService(  
  "mtService",  
  "v1.0.0",  
  code = manualTransmission,  
  mode = carsModel,  
  inputs = list(carData = "data.frame"),  
  outputs = list(answer = "data.frame"),  
  descr = "Updated after March data refresh."  
)
```

Supported I/O data types

The following table lists the supported data types for the `publishService` and `updateService` function input and output schemas.

I/O DATA TYPES	FULL SUPPORT?
numeric	Yes
integer	Yes
logical	Yes
character	Yes
vector	Yes
matrix	Partial (Not for logical & character matrices)
data.frame	Yes Note: Coercing an object during I/O is a user-defined task

Delete web services

When you no longer want to keep a web service, you can delete it. Only the user who initially created the web service can use this function.

After you've authenticated, use the `deleteService()` function in the `mrsdeploy` package to delete a web service.

Each web service is uniquely defined by a name and version. See the [package reference help page for `deleteService\(\)`](#) for the full description of all arguments.

FUNCTION	RESPONSE	R HELP
<code>deleteService(...)</code>	If it is successful, it returns a success status and message such as " <i>Service mtService version v1.0.0 deleted.</i> " If it fails for any reason, then it stops execution with error message.	View

Example:

```
result <- deleteService("mtService", "v1.0.0")
print(result)
```

Standard workflow examples

The following workflow examples demonstrate how to publish a web service, interact with it, and then consume it.

Each standard web service example uses the same code and models and returns the same results. However the code and model are represented in different formats each time, such as R scripts, objects, and files.

To learn more about standard web services, [see here](#).

Before you begin

IMPORTANT

Replace the connection details in the `remoteLogin()` function in each example with the details for your configuration. Connecting to R Server using the `mrsdeploy` package is covered [in this article](#).

The base path for files is set to your working directory, but you can change that using `ServiceOption` as follows:

- Specify a different base path for code and model arguments:

```
opts <- serviceOption()
opts$set("data-dir", "/base/path/to/some-other/location")
```

- Clear the path and expect the code and model files to be in the current working directory during `publishService()`:

```
opts <- serviceOption() s
opts$set("data-dir", NULL)
```

- Clear the path and require a FULLY QUALIFIED path for code and model parameters in `publishService()`:

```
opts <- serviceOption() s
opts$set("data-dir", "")
```

1. R code and model are objects

In this example, the code comes from an object (`code = manualTransmission`) and the model comes from a model object (`model = carsModel`). For an example of inputs/outputs as dataframes, see "[Example 5](#)".

```
#####
#      Create & Test a Logistic Regression Model      #
#####

# For R Server 9.0, load mrsdeploy package
library(mrsdeploy)

# Use logistic regression equation of vehicle transmission
# in the data set mtcars to estimate the probability of
# a vehicle being fitted with a manual transmission
# based on horsepower (hp) and weight (wt)

# Create glm model with `mtcars` dataset
carsModel <- glm(formula = am ~ hp + wt, data = mtcars, family = binomial)

# Produce a prediction function that can use the model
manualTransmission <- function(hp, wt) {
  newdata <- data.frame(hp = hp, wt = wt)
  predict(carsModel, newdata, type = "response")
}

# test function locally by printing results
print(manualTransmission(120, 2.8)) # 0.6418125

#####
#      Log into Microsoft R Server      #
#####

# Use `remoteLogin` to authenticate with R Server using
# the local admin account. Use session = false so no
# remote R session started
# REMEMBER: Replace with your login details
remoteLogin("http://localhost:12800",
            username = "admin",
            password = "{{YOUR_PASSWORD}}",
            session = FALSE)

#####
#      Publish Model as a Service      #
#####

# Generate a unique serviceName for demos
# and assign to variable serviceName
serviceName <- paste0("mtService", round(as.numeric(Sys.time()), 0))

# Publish as service using publishService() function from
# mrsdeploy package. Use the service name variable and provide
# unique version number. Assign service to the variable `api`
api <- publishService(
  serviceName,
  code = manualTransmission,
  model = carsModel,
  inputs = list(hp = "numeric", wt = "numeric"),
  outputs = list(answer = "numeric"),
  v = "v1.0.0"
)
```

```

#####
#           Consume Service in R           #
#####

# Print capabilities that define the service holdings: service
# name, version, descriptions, inputs, outputs, and the
# name of the function to be consumed
print(api$capabilities())

# Consume service by calling function, `manualTransmission`
# contained in this service
result <- api$manualTransmission(120, 2.8)

# Print response output named `answer`
print(result$output("answer")) # 0.6418125

#####
#      Get Swagger File for this Service in R Now      #
#####

# During this authenticated session, download the
# Swagger-based JSON file that defines this service
swagger <- api$swagger()
cat(swagger, file = "swagger.json", append = FALSE)

# Now you can share Swagger-based JSON so others can consume it

#####
#      Delete service version when finished      #
#####

# User who published service or user with owner role can
# remove the service when it is no longer needed
status <- deleteService(serviceName, "v1.0.0")
status

#####
#           Log off of R Server           #
#####

# Log off of R Server
remoteLogout()

```

2. R code as object and RData as file

In this example, the code is still an object (`code = manualTransmission`), but the model now comes from a Rdata file (`model = "transmission.RData"`). The result is still the same as in the first example.

```

# For R Server 9.0, load mrsdeploy package on R Server
library(mrsdeploy)

# --- AAD login -----

# Use `remoteLogin` to authenticate with R Server using
# the local admin account. Use session = false so no
# remote R session started
# REMEMBER: Replace with your login details
remoteLogin("http://localhost:12800",
            username = "admin",
            password = "{{YOUR_PASSWORD}}",
            session = FALSE)

model <- glm(formula = am ~ hp + wt, data = mtcars, family = binomial)
save(model, file = "transmission.RData")

manualTransmission <- function(hp, wt) {
  newdata <- data.frame(hp = hp, wt = wt)

```

```

predict(model, newdata, type = "response")
}

# test locally: 0.6418125
print(manualTransmission(120, 2.8))

# Generate a unique serviceName for demos
# and assign to variable serviceName
serviceName <- paste0("mtService", round(as.numeric(Sys.time()), 0))

api <- publishService(
  serviceName,
  code = manualTransmission,
  model = "transmission.RData",
  inputs = list(hp = "numeric", wt = "numeric"),
  outputs = list(answer = "numeric"),
  v = "v1.0.2"
)

api

result <- api$manualTransmission(120, 2.8)
print(result$output("answer")) # 0.6418125

swagger <- api$swagger()
cat(swagger)

swagger <- api$swagger(json = FALSE)
swagger

services <- listServices(serviceName)
services

serviceName <- services[[1]]
serviceName

api <- getService(serviceName$name, serviceName$version)
api
result <- api$manualTransmission(120, 2.8)
print(result$output("answer")) # 0.6418125

cap <- api$capabilities()
cap
cap$swagger

status <- deleteService(cap$name, cap$version)
status

remoteLogout()

```

3. Code and model as R scripts

In this example, the code (`code = transmission-code.R,`) and the model comes from R scripts (`model = "transmission.R"`). The result is still the same as in the first example.

```

# For R Server 9.0, load mrsdeploy package on R Server
library(mrsdeploy)

# --- AAD login ----

# Use `remoteLogin` to authenticate with R Server using
# the local admin account. Use session = false so no
# remote R session started
# REMEMBER: Replace with your login details
remoteLogin("http://localhost:12800",
            username = "admin",
            password = "{{YOUR_PASSWORD}}",

```

```

        session = FALSE)

# Information can come from a file
model <- "model <- glm(formula = am ~ hp + wt, data = mtcars, family = binomial)"
code <- "newdata <- data.frame(hp = hp, wt = wt)\n"
       answer <- predict(model, newdata, type = "response")"

cat(model, file = "transmission.R", append = FALSE)
cat(code, file = "transmission-code.R", append = FALSE)

# Generate a unique serviceName for demos
# and assign to variable serviceName
serviceName <- paste0("mtService", round(as.numeric(Sys.time()), 0))

api <- publishService(
  serviceName,
  code = "transmission-code.R",
  model = "transmission.R",
  inputs = list(hp = "numeric", wt = "numeric"),
  outputs = list(answer = "numeric"),
  v = "v1.0.3",
  alias = "manualTransmission"
)

api

result <- api$manualTransmission(120, 2.8)
result
print(result$output("answer")) # 0.6418125

swagger <- api$swagger()
cat(swagger)

swagger <- api$swagger(json = FALSE)
swagger

services <- listServices(serviceName)
services

serviceName <- services[[1]]
serviceName

api <- getService(serviceName$name, serviceName$version)
api
result <- api$manualTransmission(120, 2.8)
print(result$output("answer")) # 0.6418125

cap <- api$capabilities()
cap
cap$swagger

status <- deleteService(cap$name, cap$version)
status

remoteLogout()

```

4. Code as script and model as a RData file

In this example, the code (`code = transmission-code.R,`) comes from an R script, and the model from an RData file (`model = "transmission.RData"`). The result is still the same as in the first example.

```

# For R Server 9.0, load mrsdeploy package on R Server
library(mrsdeploy)

# AAD login

# Use `remoteLogin` to authenticate with R Server using

```

```

# the local admin account. Use session = false so no
# remote R session started
# REMEMBER: Replace with your login details
remoteLogin("http://localhost:12800",
            username = "admin",
            password = "{{YOUR_PASSWORD}}",
            session = FALSE)

# model
model <- glm(formula = am ~ hp + wt, data = mtcars, family = binomial)
save(model, file = "transmission.RData")

# R code
code <- "newdata <- data.frame(hp = hp, wt = wt)\n"
        answer <- predict(model, newdata, type = "response)"
cat(code, file = "transmission-code.R", sep="\n", append = TRUE)

# Generate a unique serviceName for demos
# and assign to variable serviceName
serviceName <- paste0("mtService", round(as.numeric(Sys.time()), 0))

api <- publishService(
    serviceName,
    code = "transmission-code.R",
    model = "transmission.RData",
    inputs = list(hp = "numeric", wt = "numeric"),
    outputs = list(answer = "numeric"),
    v = "v1.0.4",
    alias = "manualTransmission"
)

api

result <- api$manualTransmission(120, 2.8)
print(result$output("answer")) # 0.6418125

swagger <- api$swagger()
cat(swagger)

swagger <- api$swagger(json = FALSE)
swagger

services <- listServices(serviceName)
services

serviceName <- services[[1]]
serviceName

api <- getService(serviceName$name, serviceName$version)
api
result <- api$manualTransmission(120, 2.8)
print(result$output("answer")) # 0.6418125

cap <- api$capabilities()
cap
cap$swagger

status <- deleteService(cap$name, cap$version)
status

remoteLogout()

```

5. R code & model as objects, inputs/outputs as dataframes

In this example, the code comes from an object (`code = manualTransmission`) and the model comes from a model object (`model = carsModel`) as it was in example 1. However, in this example, the inputs and outputs are

provided in the form of dataframes.

```
##### Create/Test Logistic Regression Model #####
# R Server 9.0, load mrsdeploy. Later versions can skip.
library(mrsdeploy)

# Estimate the probability of a vehicle being fitted with
# a manual transmission based on horsepower (hp) and weight (wt)

# load the mtcars dataset
data(mtcars)

# Split the mtcars dataset into 75% train and 25% test dataset
train_ind <- sample(seq_len(nrow(mtcars)), size = floor(0.75 * nrow(mtcars)))
train <- mtcars[train_ind,]
test <- mtcars[-train_ind,]

# Create glm model with training `mtcars` dataset
carsModel <- rxLogit(formula = am ~ hp + wt, data = train)

# Create a list to pass the data column info together with the model object
carsModelInfo <- list(predictiveModel = carsModel, colInfo = rxCreatColInfo(train))

# Produce a prediction function that can use the model and column info
manualTransmission <- function(carData) {
  newdata <- rxImport(carData, colInfo = carsModelInfo$colInfo)
  rxPredict(carsModelInfo$predictiveModel, newdata, type = "response")
}

# test function locally by printing results
print(manualTransmission(test))

#####
# Log into Microsoft R Server #####
# Use `remoteLogin` to authenticate with R Server.
# REMEMBER: Replace with your login details
remoteLogin("http://localhost:12800",
            username = "admin",
            password = "{{YOUR_PASSWORD}}",
            session = FALSE)

#####
# Publish Model as a Service #####
# Generate a unique serviceName for demos and assign to variable serviceName
serviceName <- paste0("mtService", round(as.numeric(Sys.time()), 0))

# Publish as service using publishService() function from
# mrsdeploy package. Use the service name variable and provide
# unique version number. Assign service to the variable `api`
api <- publishService(
  serviceName,
  code = manualTransmission,
  model = carsModelInfo,
  inputs = list(carData = "data.frame"),
  outputs = list(answer = "data.frame"),
  v = "v1.0.0"
)

#####
# Consume Service in R #####
# Print capabilities that define the service holdings: service
# name, version, descriptions, inputs, outputs, and the
# name of the function to be consumed
print(api$capabilities())
```

```

# Consume service by calling function, `manualTransmission` contained in this service

# consume service using existing data frame `test`
result <- api$manualTransmission(test)
print(result$output("answer"))

# consume service by constructing data frames with single row and multiple rows
emptyDataFrame <- data.frame(mpg = numeric(),
                               cyl = numeric(),
                               disp = numeric(),
                               hp = numeric(),
                               drat = numeric(),
                               wt = numeric(),
                               qsec = numeric(),
                               vs = numeric(),
                               am = numeric(),
                               gear = numeric(),
                               carb = numeric())

singleRowDataFrame <- rbind(emptyDataFrame, data.frame(mpg = 21.0,
                                                       cyl = 6,
                                                       disp = 160,
                                                       hp = 110,
                                                       drat = 3.90,
                                                       wt = 2.620,
                                                       qsec = 16.46,
                                                       vs = 0,
                                                       am = 1,
                                                       gear = 4,
                                                       carb = 4))

result <- api$manualTransmission(singleRowDataFrame)
print(result$output("answer"))

multipleRowsDataFrame <- rbind(emptyDataFrame, data.frame(mpg = c(21.0, 20.1),
                                                          cyl = c(6, 5),
                                                          disp = c(160, 159),
                                                          hp = c(110, 109),
                                                          drat = c(3.90, 2.94),
                                                          wt = c(2.620, 2.678),
                                                          qsec = c(16.46, 15.67),
                                                          vs = c(0, 0),
                                                          am = c(1, 1),
                                                          gear = c(4, 3),
                                                          carb = c(4, 2)))

result <- api$manualTransmission(multipleRowsDataFrame)
print(result$output("answer"))

##### Get Swagger File for Service in R Now #####
# During this authenticated session, download the
# Swagger-based JSON file that defines this service
swagger <- api$swagger()
cat(swagger, file = "swagger.json", append = FALSE)

# Now you can share Swagger-based JSON so others can consume it

##### Delete service version when finished #####
# User who published service or user with owner role can
# remove the service when it is no longer needed
status <- deleteService(serviceName, "v1.0.0")
status

##### Log off R Server #####
remoteLogout()

```

Real-time workflow example

In this example, the local model object (`model = kyphosisModel`) is generated using the `rxLogit` modeling function in the RevoScaleR package.

Real-time web services were introduced in R Server 9.1. To learn more about the supported model formats, supported product versions, and supported platforms for real-time web services, [see here](#).

```
##          REAL-TIME WEB SERVICE EXAMPLE          ##

#####
# Create/Test Logistic Regression Model with rxLogit  #
#####

# Create logistic regression model
# using rxLogit modeling function from RevoScaleR package
# and the Rpart `kyphosis` dataset available to all R users
kyphosisModel <- rxLogit(Kyphosis ~ Age, data=kyphosis)

# Test the model locally
testData <- data.frame(Kyphosis=c("absent"), Age=c(71), Number=c(3), Start=c(5))
rxPredict(kyphosisModel, data = testData) # Kyphosis_Pred: 0.1941938

#####
#      Log into Microsoft R Server           #
#####

# Use `remoteLogin` to authenticate with R Server using
# the local admin account. Use session = false so no
# remote R session started
# REMEMBER: replace with the login info for your organization
remoteLogin("http://localhost:12800",
            username = "admin",
            password = "{{YOUR_PASSWORD}}",
            session = FALSE)

#####
# Publish Kyphosis Model as a real-time Service      #
#####

# Generate a unique serviceName for demos
# and assign to variable serviceName
serviceName <- paste0("kyphosis", round(as.numeric(Sys.time()), 0))

# Publish as service using publishService() function.
# Use the variable name for the service and version `v1.0`
# Assign service to the variable `realtimeApi`.
realtimeApi <- publishService(
    serviceType = "Realtime",
    name = serviceName,
    code = NULL,
    model = kyphosisModel,
    v = "v1.0",
    alias = "kyphosisService"
)

#####
#      Consume real-time Service in R           #
#####

# Print capabilities that define the service holdings: service
# name, version, descriptions, inputs, outputs, and the
# name of the function to be consumed
```

```
print(realtimeApi$capabilities())

# Consume service by calling function contained in this service
realtimeResult <- realtimeApi$kyphosisService(testData)

# Print response output
print(realtimeResult$outputParameters) # 0.1941938

#####
#      Get Service-specific Swagger File in R      #
#####

# During this authenticated session, download the
# Swagger-based JSON file that defines this service
rtSwagger <- realtimeApi$swagger()
cat(rtSwagger, file = "realtimeSwagger.json", append = FALSE)

# Share Swagger-based JSON with those who need to consume it
```

See also

- [What is operationalization?](#)
- [What are web services?](#)
- [mrsdeploy function overview](#)
- [Quickstart: Deploying an R model as a web service](#)
- [Connecting to R Server from mrsdeploy.](#)
- [How to interact with and consume web services in R](#)
- [How to integrate web services and authentication into your application](#)
- [Asynchronous batch execution of web services in R](#)
- [Execute on a remote Microsoft R Server](#)

How to interact with and consume web services in R with mrsdeploy

7/12/2022 • 7 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Applies to: Machine Learning Server, Microsoft R Server 9.x

After a web service has been published or updated, any authenticated user can list, examine, and consume that web service. You can do so directly in R using the functions in the [mrsdeploy R package](#). The mrsdeploy R package is installed with both Machine Learning Server and Microsoft R Client. Also note that application developers can also consume a web service in the [language of their choice via Swagger](#).

If you do not want to list, examine, or consume the web service in R, a set of [RESTful APIs](#) are also available to provide direct programmatic access to a service's lifecycle directly.

To list, examine, or consume the web service outside of R, use the [RESTful APIs](#), which provide direct programmatic access to a service's lifecycle.

Requirements

Before you can use the functions in the mrsdeploy R package to manage your web services, you must:

- Have access to a Machine Learning Server instance that was [properly configured](#) to host web services.
- Authenticate with Machine Learning Server using the `remoteLogin()` or `remoteLoginAAD()` functions in the mrsdeploy package as described in the article "[Connecting to Machine Learning Server to use mrsdeploy](#)."

Find and list web services

Any authenticated user can retrieve a list of web services using the `listServices()` function in the mrsdeploy package.

Use function arguments to return a specific web service or all labeled versions of a given web service. See the [package reference help page for listServices\(\)](#) for the full description of all arguments.

Your ability to see the code inside the web service depends on your permissions. Did you publish the web service or do you have the "Owner" role?

- If yes, then the code in the service is returned along with other metadata.
- If no, then the code in the service is never returned in the metadata.

To learn more about the roles in your organization, contact your Machine Learning Server administrator.

FUNCTION	RESPONSE	R FUNCTION HELP
listServices(...)	R list containing service metadata.	View

Once you find the service you want, use [the getService\(\) function](#) to retrieve the service object for consumption.

Example code:

```
# Return metadata for all services hosted on this server
serviceAll <- listServices()

# Return metadata for all versions of service "mtService"
mtServiceAll <- listServices("mtService")

# Return metadata for version "v1" of service "mtService"
mtService <- listServices("mtService", "v1")

# View service capabilities/schema for mtService v1.
# For example, the input schema:
#   list(name = "wt", type = "numeric")
#   list(name = "dist", type = "numeric")
print(mtService)
```

For a detailed example with listServices, check out these "[Workflow examples](#)".

Example output:

R SERVER 9.1+	R SERVER 9.0
<pre>\$creationTime [1] "2017-02-13T19:44:26.2611422" \$name [1] "mtService" \$version [1] "v1" \$description NULL \$snapshotId [1] "05053e85-c9d0-43cb-9be8-8dccf2b5da54" \$versionPublishedBy [1] "rserver-user" \$inputParameterDefinitions \$inputParameterDefinitions[[1]] \$inputParameterDefinitions[[1]]\$name [1] "hp" \$inputParameterDefinitions[[1]]\$type [1] "numeric" \$inputParameterDefinitions[[2]] \$inputParameterDefinitions[[2]]\$name [1] "wt" \$inputParameterDefinitions[[2]]\$type [1] "numeric" \$outputParameterDefinitions \$outputParameterDefinitions[[1]] \$outputParameterDefinitions[[1]]\$name [1] "answer"</pre> <hr/> <pre>\$outputParameterDefinitions[[1]]\$type [1] "numeric" \$operationId manualTransmission \$myPermissionsOnService [1] "read/write"</pre>	<pre>\$creationTime [1] "2017-02-13T19:44:26.2611422" \$name [1] "mtService" \$version [1] "v1" \$description NULL \$snapshotId [1] "05053e85-c9d0-43cb-9be8-8dccf2b5da54" \$owner [1] "rserver-user" \$inputParameterDefinitions \$inputParameterDefinitions[[1]] \$inputParameterDefinitions[[1]]\$name [1] "hp" \$inputParameterDefinitions[[1]]\$type [1] "numeric" \$inputParameterDefinitions[[2]] <inputparameterdefinitions[[2]]\$name "answer"="" "numeric"="" "wt"="" \$operationid="" \$outputparameterdefinitions="" \$outputparameterdefinitions[[1]]="" \$outputparameterdefinitions[[1]]\$name="" \$outputparameterdefinitions[[1]]\$type="" <inputparameterdefinitions[[2]]\$type="" [1]="" manualtransmission<="" pre=""> </inputparameterdefinitions[[2]]\$name></pre>

Retrieve and examine service objects

Any authenticated user can retrieve a web service object for consumption. After the object is returned, you can look at its capabilities to see what the service can do and how it should be consumed.

After you've authenticated, use the `getService()` function in the `mrsdeploy` package to retrieve a service object. See the [package reference help page for `getService\(\)`](#) for the full description of all arguments.

FUNCTION	RESPONSE	R FUNCTION HELP
<code>getService(...)</code>	Returns an API instance client stub for consuming that service and viewing its service holdings) as an R6 class.	View

Example:

```

# Get service using getService() function from `mrsdeploy` package.
# Assign service to the variable `api`.
api <- getService("mtService", "v1.0.0")

# Print capabilities to see what service can do.
print(api$capabilities())

# Start interacting with the service, for example:
# Calling the function, `manualTransmission`
# contained in this service.
result <- api$manualTransmission(120, 2.8)

```

For a detailed example with `getService`, check out these "[Workflow](#)" examples.

Interact with API clients

When you publish, update, or get a web service, an API instance is returned as an [R6](#) class. This instance is a client stub you can use to consume that service and view its service holdings.

You can use the following supported public functions to interact with the API client instance.

FUNCTION	DESCRIPTION
<code>print</code>	Print method that lists all members of the object
<code>capabilities</code>	Report on the service features such as I/O schema, name, version
<code>consume</code>	Consume the service based on I/O schema
<code>consume <i>alias</i></code>	Alias to the <code>consume</code> function for convenience (see <code>alias</code> argument for the publishService function).
<code>swagger</code>	Displays the service's swagger specification
<code>batch</code>	Define the data records to be batched. There are additional publish functions used to consume a service asynchronously via batch execution .

Example:

```

# Get service using getService() function from `mrsdeploy`.
# Assign service to the variable `api`
api <- getService("mtService", "v1.0.0")

# Print capabilities to see what service can do.
print(api)

# Print the service name, version, inputs, outputs, and the
# Swagger-based JSON file used to consume the service
cap <- api$capabilities()
print(cap$name)
print(cap$version)
print(cap$inputs)
print(cap$outputs)
print(cap$swagger)

# Start interacting with the service by calling it with the
# generic name `consume` based on I/O schema
result <- api$consume(120, 2.8)

# Or, start interacting with the service using the alias argument
# that was defined at publication time.
result <- api$manualTransmission(120, 2.8)

# Since you're authenticated, get this service's `swagger.json`.
swagger <- api$swagger(json = FALSE)
cat(swagger)

```

For a detailed example, check out these "[Workflow](#)" examples.

Consume web services

Web services are published to facilitate the consumption and integration of the operationalized models and code. Whenever the web service is published or updated, a Swagger-based JSON file is generated automatically that define the service.

When you publish a service, you can let people know that it is ready for consumption. Users can get the Swagger file they need to consume the service directly in R or via the API. To make it easy for others to find your service, provide them with the service name and version number (or they can use [the listServices\(\) function](#)).

Users can consume the service directly using a single consumption call. This approach is referred to as a "Request Response" approach and is described in the following section. Another approach is the [asynchronous "Batch" consumption approach](#), where users send a single request to Machine Learning Server, which then makes multiple asynchronous API calls on your behalf.

Collaborate with data scientists

Other data scientists may want to explore, test, and consume Web services directly in R using the functions in the mrsdeploy package. Quality engineers might want to bring the models in these web services into validation and monitoring cycles.

You can share the name and version of a web service with fellow data scientists so they can call that service in R using the functions in the mrsdeploy package. After authenticating, data scientists can use the getService() function in R to call the service. Then, they can get details about the service and start consuming it.

You can also [build a client library directly in R using the httr package](#).

NOTE

It is also possible to perform batch consumption as [described here](#).

In this example, replace the following `remoteLogin()` function with the correct login details for your configuration. Connecting to Machine Learning Server using the `mrsdeploy` package is covered [in this article](#).

```
#####
#      Perform Request-Response Consumption & Get Swagger Back in R      #
#####

# Use `remoteLogin` to authenticate with the server using the local admin
# account. Use session = false so no remote R session started
remoteLogin("http://localhost:12800",
            username = "admin",
            password = "{{YOUR_PASSWORD}}",
            session = FALSE)

# Get service using getService() function from `mrsdeploy`
# Assign service to the variable `api`.
api <- getService("mtService", "v1.0.0")

# Print capabilities to see what service can do.
print(api$capabilities())

# Start interacting with the service, for example:
# Calling the function, `manualTransmission` contained in this service.
result <- api$manualTransmission(120, 2.8)

# Print response output named `answer`
print(result$output("answer")) # 0.6418125

# Since you're authenticated now, get `swagger.json` .
swagger <- api$swagger()
cat(swagger, file = "swagger.json", append = FALSE)
```

Collaborate with application developers

Application developers can call and integrate a web service into their applications using the service-specific Swagger-based JSON file and by providing any required inputs to that service.

Using the Swagger-based JSON file, application developers can generate client libraries for integration. Read "[How to integrate web services and authentication into your application](#)" for more details.

Application developers can get the Swagger-based JSON file in one of these ways:

- A data scientist, probably the one who published the service, can send you the Swagger-based JSON file. This approach is often faster than the following one. They can get the file in R with the following code and send it to the application developer:

```
api <- getService("<name>", "<version>")
swagger <- api$swagger()
cat(swagger, file = "swagger.json", append = FALSE)
```

- Or, the application developer can request the file as **an authenticated user with an active bearer token in the request header** (since all API calls must be authenticated). The URL is formed as follows:

```
GET /api/<service-name>/<service-version>/swagger.json
```

See also

- [What is operationalization?](#)
- [What are web services?](#)
- [mrsdeploy function overview](#)
- [How to publish and manage web services in R](#)
- [Quickstart: Deploying an R model as a web service](#)
- [Connecting to Machine Learning Server from mrsdeploy.](#)
- [How to integrate web services and authentication into your application](#)
- [Asynchronous batch execution of web services in R](#)
- [Execute on a remote Machine Learning Server](#)

Asynchronous web service consumption via batch processing with mrsdeploy

7/12/2022 • 12 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Applies to: Machine Learning Server, Microsoft R Server 9.1

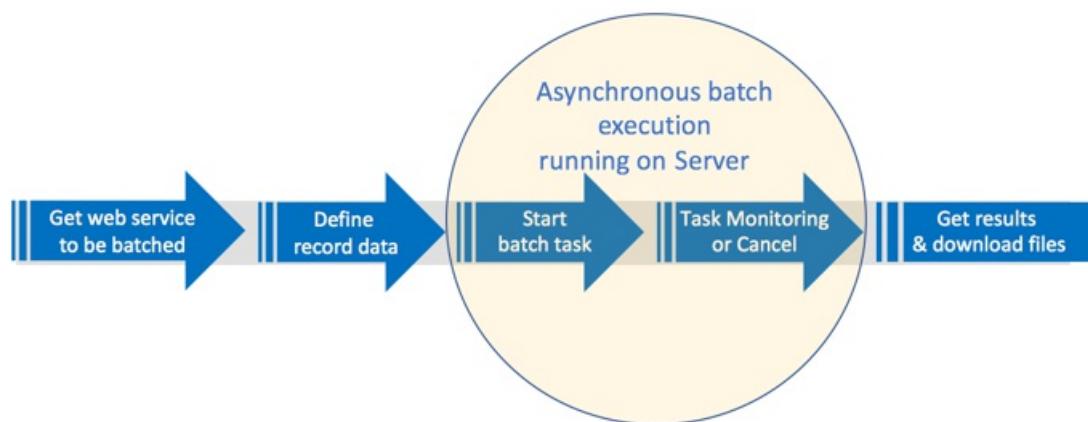
In this article, you can learn how to consume a web service asynchronously, which is especially useful with large input data sets and long-running computations. The typical approach to consuming web services, "[Request Response" consumption](#)", involves a single API call to execute the code in that web service once. The "Asynchronous Batch" approach involves the execution of code without manual intervention using multiple asynchronous API calls on a specific web service sent as a single request to Machine Learning Server. Then, Machine Learning Server immediately executes those operations once for every row of data provided.

Asynchronous batch workflow

Generally speaking, the process for asynchronous batch consumption of a web service involves the following:

1. Call the web service on which the batch execution should be run
2. Define the data records for the batch execution task
3. Start (or cancel) the batch execution task
4. Monitor task and interact with results

Use these following [public API functions](#) to define, start, and interact with your batch executions.



End-to-end workflow example

Use this sample code to follow along with the workflow described in greater detail in the following sections.

IMPORTANT

Be sure to replace the `remoteLogin()` function with the correct login details for your configuration. Connecting to Machine Learning Server using the `mrsdeploy` package is covered [in this article](#).

```

## EXAMPLE: DEPLOY MODEL & BATCH CONSUME SERVICE ##

#####
# Create & Test a Logistic Regression Model #
#####

# Use logistic regression equation of vehicle transmission in the data set
# 'mtcars' to estimate the probability of a vehicle being fitted with a
# manual transmission based on horsepower (hp) and weight (wt)

# Create glm model with `mtcars` dataset
carsModel <- glm(formula = am ~ hp + wt, data = mtcars, family = binomial)

# Produce a prediction function that can use the model
manualTransmission <- function(hp, wt) {
  # --- Build a plot to demonstrate files in results ---
  png(file = "Histogram.png", bg = "transparent")
  hist(mtcars$hp, breaks = 10, col = "red", xlab = "Horsepower",
       main="Histogram of Horsepower")
  dev.off()

  # --- Predict and return answer ---
  newdata <- data.frame(hp = hp, wt = wt)
  predict(carsModel, newdata, type = "response")
}

# test function locally by printing results
print(manualTransmission(120, 2.8)) # 0.6418125

#####
# Log into Server #
#####

# Use `remoteLogin` to authenticate local admin account.
# Use session = false so that no remote R session started
remoteLogin("http://localhost:12800",
            username = "admin",
            password = "{{YOUR_PASSWORD}}",
            session = FALSE)

#####
# Publish Model as a Service #
#####

# Generate a unique serviceName for demos and assign to variable serviceName
serviceName <- paste0("mtService", round(as.numeric(Sys.time()), 0))

# Publish as service using publishService() function from `mrsdeploy`
# package. Use the name variable and provide unique version number.
# Assign service to the variable `api`
api <- publishService(
  serviceName,
  code = manualTransmission,
  model = carsModel,
  inputs = list(hp = "numeric", wt = "numeric"),
  outputs = list(answer = "numeric"),
  artifacts = c("Histogram.png"),
  v = "v1.0.0"
)

# Consume service by calling function, `manualTransmission` contained
# in this service
result <- api$manualTransmission(120, 2.8)

# Print response output named `answer`
print(result$output("answer")) # 0.6418125

#####

```

```

#####
#           Perform Batch Consumption & Get Swagger in R           #
#####

# Get the service using getService() function from `mrsdeploy`#
# Assign service to the variable `txService`.#
txService <- getService(serviceName, "v1.0.0")

# Define the record data for the batch execution task. Record data comes
# from a data.frame called mtcars. Note: mtcars is a data.frame of
# 11 cols with names (mpg, cyl, ... , carb) and 32 rows of numerics.
# Assign to the batch object called 'txBatch'.
records <- head(mtcars[, c(4, 6)], 2) # hp & wt
txBatch <- txService$batch(records)

# Set thread count using parallelCount. Default is 10.
# txBatch <- txService$batch(records, parallelCount = 5)

# Start the batch task
txBatch <- txBatch$start()

# Get the task execution id to reference during or after its execution:
id <- txBatch$id()

# If you need to cancel the batch execution, try this:
# txBatch$cancel()

# Monitor batch execution results with results().
# Check results every 3 seconds until task finishes or fails.
# Assign returned results to batch result object we called 'batchRes'.
batchRes <- NULL
while(TRUE) {
  batchRes <- txBatch$results(showPartialResult = TRUE) #Default is true

  if (batchRes$state == txBatch$STATE$failed) { stop("Batch execution failed") }
  if (batchRes$state == txBatch$STATE$complete) { break }

  message("Polling for asynchronous batch to complete...")
  Sys.sleep(3)
}

# Once the batch task is complete, get the execution records by index from
# the batch results object, 'batchRes'. This object is the service output.

# For every record, return these results (totalItemCount = # of data records)
for(i in seq(batchRes$totalItemCount)) {

  #1. List of every artifact that was generated by this execution index.
  files <- txBatch$listArtifacts(i)

  #2. Get the contents of each artifact returned in the previous list.
  for (fileName in files) {
    content <- txBatch$artifact(i, fileName)
    if (is.null(content)) { stop("Unable to get file") }
  }

  #3. Download artifacts from execution index to the current working directory
  # unless a dest = "<path>" is specified.

  # Download of a single named artifact
  txBatch$download(i, "Histogram.png")

  # Download of all artifacts
  txBatch$download(i, dest = getwd())

  #4. Get results for a given index row
  answer <- batchRes$execution(1)$outputParameters$answer
  answer
}

```

```
        }
```

Public functions for batch

You can use the following supported public functions to consume a service asynchronously.

Batch functions performed on the service object

Once you get the service object, use these public functions on that object.

FUNCTION	USAGE	DESCRIPTION
<code>batch</code>	view	Define the data records to be batched and the thread count
<code>getBatchExecutions</code>	view	Get the list of batch execution identifiers
<code>getBatch</code>	view	Get batch object using its unique execution identifier

Batch functions performed on the batch object

Once you have the batch object, use these public functions to interact with it.

FUNCTION	DESCRIPTION	USAGE
<code>start</code>	view	Starts the execution of a batch scoring operation
<code>cancel</code>	view	Cancel the named batch execution
<code>id</code>	view	Get the execution identifier for the named batch process
<code>STATE</code>	view	Poll for the state of the batch execution (failed, complete, ...)
<code>results</code>	view	Poll for batch execution results, partial or full results as defined
<code>execution</code>	view	Get results for a given index row returned as an array
<code>listArtifacts</code>	view	List of every artifact files that was generated by this execution index
<code>artifact</code>	view	Print the contents of the named artifact file generated by the batch execution
<code>download</code>	view	Download one or all artifact files from execution index

1. Get the web service

Once you have authenticated, retrieve the web service from the server, assign it to a variable, and define the inputs to it as record data in a data frame, CSV, or TSV.

Batching begins by retrieving the web service containing the code against which you score the data records you define next. You can get a service using its name and version with the `getService()` function from `mrsdeploy`. The result is a service object, which in our example is called `txService`.

The `getService` function is covered in detail in the article "[How to interact with and consume web services in R](#)."

Syntax: `getService("<serviceName>", "<version>")`

Example:

```
# Get the service using getService() function from `mrsdeploy`
# Assign service to the variable `txService`.
txService <- getService("mtService", "v1.0.0")
```

2. Define the data records to be batched

Next, use the public api function `batch` to define the input record data for the batch and set the number of concurrent threads for processing.

Syntax: `batch(inputs, parallelCount = 10)`

ARGUMENT	DESCRIPTION
<code>inputs</code>	Specify the R data.frame name directly, or specify a flat list filename and convert it to a data.frame using the base R function, <code>read.csv</code> .
<code>parallelCount</code>	Default value is 10. Specify the number of concurrent threads that can be dedicated to processing records in the batch. Take care not to set a number so high that it negatively impacts performance.

Returns: The batch object

Example:

```
# INPUTS = data.frame
# Use mtcars data.frame as input. Assign to batch object 'txBatch'.
# Reduce thread count to 5.
records <- head(mtcars[, c(4, 6)], 2) # hp & wt
txBatch <- txService$batch(records, parallelCount = 5)

# INPUT = Flat CSV converted to a data.frame using read.csv
# Assign data.frame to 'records' variable. Then, use 'records' as input.
records <- read.csv("mtcars.csv")
txBatch <- myService$batch(records, parallelCount = 15)

# INPUT = TSV file converted to a data.frame using read.csv
# Declare the separator
records <- read.csv("mtcars.tsv", sep = "\t")
txBatch <- myService$batch(records)
```

3. Start, find, or cancel the batch execution

Next, use the public api functions to start the asynchronous batch execution on the batch object, monitor the execution, or even cancel it.

Start batch execution task

Start the batch task with `start()`. Machine Learning Server starts the batch execution and assigns an ID to the execution and returns the batch object.

Syntax: `start()`

No arguments.

Returns: The batch object

Example: `txBatch <- txBatch$start()`

NOTE

We recommend you always use the `id()` function after you start the execution so that you can find it easily with this id later such as: `txBatch$start()$id()`

Get batch ID

Get the batch task's identifier from the service object so you can reference it during or after its execution using `id()`.

Syntax: `id()`

No arguments.

Returns: The ID for the named batch object.

Example: `id <- txBatch$id()`

Get batch by ID

Syntax: `getBatch(id)`

ARGUMENT	DESCRIPTION
<code>id</code>	The batch execution identifiers

Returns: The `Batch` object

Example:

```
service <- getService("name", "version")
# Get a Services existing batch by execution Id
txBatch <- service$getBatch("my-executionId")
print(txBatch)
```

List the Batch execution identifiers

Syntax: `getBatchExecutions()`

No arguments

Returns: List of batch execution identifiers

Example: `getBatchExecutions()`

Cancel execution

Cancel the batch execution using `cancel()`.

Syntax: `cancel()`

No arguments.

Returns: The batch object

Example: `txBatch$cancel()`

4. Monitor, retrieve, and interact with results

While the batch task is running, you can monitor and poll the results. Once the batch task has completed, you can get the web service consumption output by index from the batch results object, including:

- Monitor execution results and status
- Get results for a given index row returned as an array
- Get a list of every file that was generated by this execution index
- Print the contents of a specific artifact or all artifacts returned
- Download artifacts from execution index

Monitor execution results and status

There are several public functions you can use to get the results and status of a batch execution.

– Monitor or get the batch execution results

- **Syntax:** `results(showPartialResults = TRUE)`

ARGUMENT	DESCRIPTION
<code>showPartialResults</code>	This argument returns the already processed results of the batch execution even if it has not been fully completed. If <code>showPartialResults = FALSE</code> , then returns only the results if the execution has completed.

- **Returns:** A batch result object is returned, which in our example is called `batchRes`.

– Get the status of the batch execution.

- **Syntax:** `STATE`

no arguments

- **Returns:** The status of the batch execution.

Example: In this example, we return partial results every three seconds until the batch execution fails or completes. Then, we return results for a given index row returned as an array.

```
batchRes <- NULL
while(TRUE) {
  # Send any results, including partial results, for txBatch task
  # Assign it to the batch result variable batchRes:
  batchRes <- txBatch$results(showPartialResult = TRUE)

  # Check STATUS of the task
  if (batchRes$state == txBatch$STATE$failed) { stop("Batch execution failed") }
  if (batchRes$state == txBatch$STATE$complete) { break }

  # Repeat check every 3 seconds
  message("Polling for asynchronous batch to complete...")
  Sys.sleep(3)
}
```

Get execution results as an array

Get the execution results for a given index row.

Syntax: `execution(index)`

ARGUMENT	DESCRIPTION
<code>index</code>	Index value for a given batch data record

Returns: The execution results for a given index row as an array.

Example: In this example, we return partial results every three seconds until the batch execution fails or completes. Then, we return results for a given index row returned as an array.

```
# In a loop, get results for a given index row returned as an array in a loop
# assign it to 'exe' and output a data frame for each row.
for(i in seq(batchRes$totalItemCount)) {
  answer <- batchRes$execution(1)$outputParameters$answer
  answer
}
```

Get list of generated artifacts

Retrieve a list of every artifact that was generated during the batch execution for a given data record, or index, with `listArtifacts()`. This function can be made part of a loop to get the list of the artifacts for every data record (see workflow example for a loop).

Syntax: `listArtifacts(index)`

ARGUMENT	DESCRIPTION
<code>index</code>	Index value for a given batch data record

Returns: A list of every artifact that was generated during the batch execution for a given data record

Example: `files <- txBatch$listArtifacts(i)`

Display artifact contents

Display the contents of a named artifact returned in the preceding list with `artifact()`. Machine Learning Server returns the ID for the named batch object.

Syntax: `artifact(index, fileName)`

ARGUMENT	DESCRIPTION
<code>index</code>	Index value for a given batch data record
<code>fileName</code>	Name of file artifact created during batch execution

Returns: The ID for the named batch object

Example:

```
for(i in seq(batchRes$totalItemCount)) {
  for (fileName in files) {
    content <- txBatch$artifact(i, fileName)
    if (is.null(content)) { stop("Unable to get artifact") }
  }
}
```

Download generated artifacts

Download any artifacts from a specific execution index using `download()`. By default, artifacts are downloaded to the current working directory `getwd()` unless a different `dest = "<path>"` is specified. You can choose to download a specific artifact or all artifacts.

Syntax: `download(index, fileName, dest = "<path>")`

ARGUMENT	DESCRIPTION
<code>index</code>	Index value for a given batch data record
<code>fileName</code>	Name of specific artifact generated during batch execution. If omitted, all artifacts are downloaded for that index.
<code>dest</code>	The download directory on your local machine. The default is the current R working directory. The directory must already exist on the local machine.

Returns: The path to each downloaded artifact.

Example:

In this example, we download a named artifact for the fifth index to a specified directory, and then download all artifacts to the default working directory.

```
#Download a named file for 5th index to the specified directory  
txBatch$download(5, "Histogram.png", dest = "C:/bgates/batchfiles/")  
  
# Download all files for a given index to the default current R working directory in a loop  
for(i in seq(batchRes$totalItemCount)) {  
  txBatch$download(i)  
}
```

See also

- [mrsdeploy function overview](#)
- [Connecting to Machine Learning Server with mrsdeploy](#)
- [What is operationalization?](#)
- [What are web services?](#)
- [Working with web services in R](#)
- [Execute on a remote Machine Learning Server](#)
- [How to integrate web services and authentication into your application](#)

How to create and manage session pools for fast web service connections in R

7/12/2022 • 2 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Applies to: Machine Learning Server 9.3 (R) | Machine Learning Server 9.4 (R)

Fast connections to a web service are possible when you create sessions and load dependencies in advance. Sessions are available in a pool dedicated to a specific web service, where each session includes an instance of the R interpreter and a copy of dependencies required by the web service. For example, if you create ten sessions in advance for a web service that uses Matplotlib, dplyr, cluster, RevoScaleR, MicrosoftML, and mrsdeploy, each session would have its own instance of the R interpreter plus a copy of each library loaded in memory.

A web service having a dedicated session pool never requests connections from the [generic session pool](#) shared resource, not even when maximum sessions are reached. The generic session pool services only those web services that do not have dedicated resources.

For R script, the [mrsdeploy](#) function library provides three functions for creating and managing sessions:

- [configureServicePool](#)
- [getPoolStatus](#)
- [deleteServicePool](#)

Create or modify a dedicated session pool

Given an existing connection to a Machine Learning Server with [operationalization](#) enabled, you can use [mrsdeploy](#) functions and an R console application such as Rgui.exe to run the following commands:

```
# load mrsdeploy and print the function list
library(mrsdeploy)
ls("package:mrsdeploy")

# Return a list of web services to get the service and version
# Both service name and version number are required
listServices()

# Create the session pool using a case-sensitive web service name
# A status code of 200 is returned upon success
configureServicePool(name = "myWebservice1234", version = "v1.0.0", initialPoolSize = 5, maxPoolSize = 10 )

# Return status
# Pending indicates session creation is in progress. Success indicates sessions are ready.
getPoolStatus(name = "myWebService1234", version = "v1.0.0")
```

Currently, there are no commands or functions that return actual session pool usage. The log file is your best resource for analyzing connection and service activity. For more information, see [Trace user actions](#).

Delete a session pool

At the R console, on the compute node, run the following command to delete the session pool for a given service.

```
# Return a list of web services to get the service and version information
listServices()

# Deletes the dedicated session pool and releases resources
deleteServicePool(name = "myWebService1234", version = "v1.0.0")
```

This feature is still under development. In rare cases, the [deleteServicePool](#) command may fail to actually delete the pool on the computeNode. If you encounter this situation, issue a new [deleteServicePool](#) request. Use the [getPoolStatus](#) command to monitor the status of dedicated pools on the computeNode.

```
# Deletes the dedicated session pool and releases resources
deleteServicePool(name = "myWebService1234", version = "v1.0.0")

# Check the real-time status of dedicated pool
getPoolStatus(name = "myWebService1234", version = "v1.0.0")

# make sure the return status is NotFound on all computeNodes
# if not, issue another deleteServicePool command again
```

See also

- [What are web services in Machine Learning Server?](#)
- [Evaluate web service capacity: generic session pools](#)

APIs for operationalizing your models and analytics with Machine Learning Server

7/12/2022 • 3 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Applies to: Machine Learning Server, Microsoft R Server 9.x

The Machine Learning Server (and Microsoft R Server) [REST APIs](#) are exposed by the operationalization server, a standards-based server technology [capable of scaling to meet the needs of enterprise-grade deployments](#). With the Machine Learning Server configured to operationalize, the full statistics, analytics, and visualization capabilities of R and Python can now be directly leveraged inside Web, desktop and mobile applications.

The APIs available with Machine Learning Server can be categorized into two groups: Core APIs and the Service Consumption APIs.

Looking for a specific core API call? [Look in this online API reference](#).

Core APIs for Operationalization

These core REST APIs expose the R or Python platform as a service allowing the integration of Python models and R statistics, analytics, and visualizations inside Web, desktop and mobile applications. These APIs enable you to publish Machine Learning Server-hosted R [analytics web services](#), making the full capabilities of R available to application developers on a simple yet powerful REST API. These core operationalization APIs can be grouped into several categories as shown in this table.

CORE API TYPE	DESCRIPTION	REFERENCE HELP
Authentication	These APIs provide authentication-related operations and access workflow features.	Help
Web Services	These APIs facilitate the publishing and management of user-defined analytic web services (create, delete, update, list, discover). Each web service is uniquely defined by a <code>name</code> and <code>version</code> for easy service consumption and meaningful machine-readable discovery approaches. When a service is published (<code>POST /services/{name}/{version}</code>), an endpoint is registered and a custom Swagger-based JSON file is generated .	Help

CORE API TYPE	DESCRIPTION	REFERENCE HELP
Session	These APIs provide functionality for R session management (create, delete, update, list, console output, history, and workspace and working directory files)	Help
Snapshot	These APIs provide different operations to access and manage workspace snapshots. A snapshot is a prepared environment image of an R or Python session saved to Machine Learning Server, which includes the session's packages, objects and data files. This snapshot can be loaded into any subsequent remote session for the user who created it. Learn more about R session snapshots	Help
Status	This API returns a health report of the configuration, including the number of nodes, pool size , and other details. A similar diagnostic report is available on the server.	Help

The core APIs are accessible from and described in `mlserver-swagger-<version>.json`, a **Swagger-based JSON document**. Download this file from

`https://microsoft.github.io/deployr-api-docs/swagger/mlserver-swagger-\<version>.json`, where `\<version>` is the 3-digit product version number. Swagger is a popular specification for a JSON file that describes REST APIs. For R Server users, replace `mlserver-swagger` with `rserver-swagger`.

You can access all of these core APIs using a client library built from `rserver-swagger-<version>.json` using [these instructions and example](#).

Note: For client applications written in R, you can side-step the Swagger approach altogether and exploit [the mrsdeploy package](#) directly to list, discover, and consume services. [Learn more in this article](#).

Service Consumption APIs

The service consumption REST APIs expose a wide range of Python and R analytics services to client application developers. After Python or R code is published and exposed by the server as a web service, an application can make API calls to pass inputs to the service, execute the service, and retrieve outputs from the service.

Whenever a web service is published (`POST /services/{name}/{version}`), an endpoint is registered (`/api/{name}/{version}`), which in turn triggers the generation of a custom Swagger-based JSON file. This swagger file describes each API needed to interact with that service. While the service consumption Swagger files are all named `swagger.json`, you can find them each in a unique location by calling:

```
GET /api/{service-name}/{service-version}/swagger.json
```

To make it easier to integrate R and Python web services using these APIs, build and use an API client library stub from the `swagger.json` file for each web service using [these instructions and example](#).

How to add web services and authentication to applications

7/12/2022 • 9 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Applies to: Machine Learning Server, Microsoft R Server 9.x

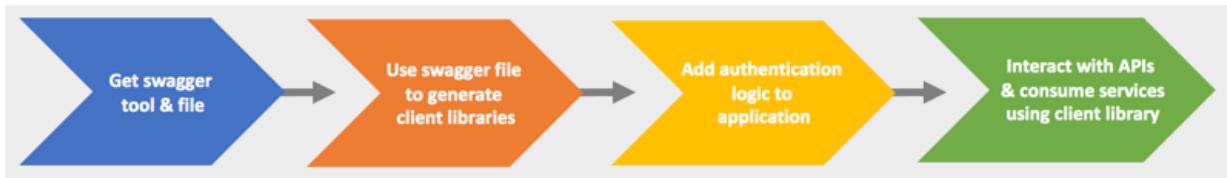
Learn how to build and use API Client libraries from Swagger to integrate into your applications. Swagger is a machine-readable representation of a RESTful API that enables support for interactive documentation, client SDK generation, and discoverability.

While data scientists can work with R / Python directly in a console window or IDE, application developers often need a different set of tools to leverage R inside applications. As an application developer integrating with these web services, typically your interest is in executing R / Python code, not writing it. Data scientists with the R programming skills write the R / Python code. Then, using some core APIs, this code can be published as a Machine Learning Server-hosted analytics Web service.

To simplify the integration of your R / Python analytics web services, Machine Learning Server (formerly R Server) provides [Swagger templates](#) for operationalization. These Swagger-based JSON files define the list of calls and resources available in the REST APIs.

To access these RESTful APIs outside of R / Python, generate an API client library in your preferred programming language, such as .NET, C#, Java, JS, Python, or node.js. This library is built with a Swagger tool. The resulting API client library simplifies the making of calls, encoding of data, and markup response handling on the API.

Swagger workflow



Get the Swagger file

Machine Learning Server provides a Swagger template to simplify the integration. This template defines the available resources in the REST API and defines the operations you can call on those resources. A standard set of core operationalization APIs is [available and defined](#) in `mlserver-swagger-<version>.json`, where `<version>` is the 3-digit product version number. Additionally, another Swagger-based JSON file is generated for each web service version. For R Server users, replace `mlserver-swagger` with `rserver-swagger` in the filename.

API TYPES

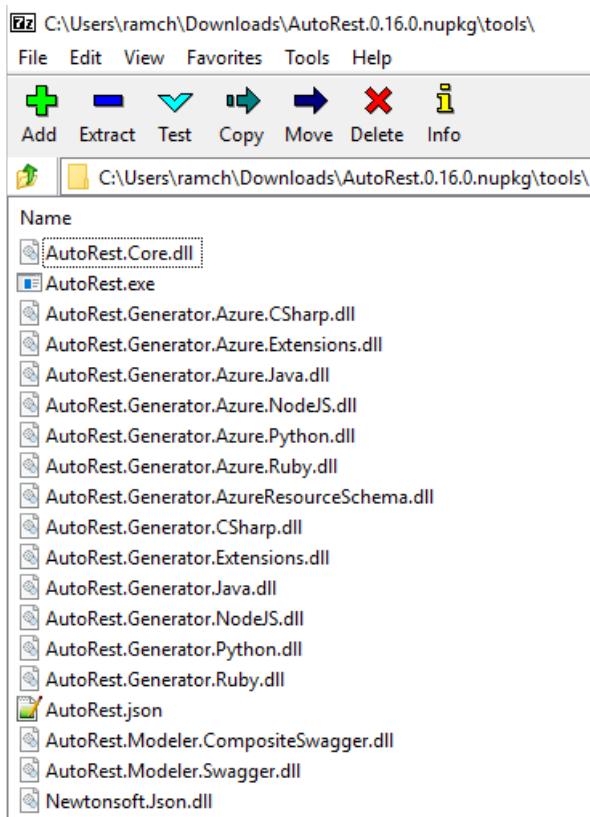
CORRESPONDING SWAGGER-BASED JSON FILE

API TYPES	CORRESPONDING SWAGGER-BASED JSON FILE
Core APIs	<p>Download Swagger file containing the set of core operationalization APIs from <a href="https://microsoft.github.io/deployr-api-docs/<version>/swagger/mlserver-swagger-<version>.json">https://microsoft.github.io/deployr-api-docs/<version>/swagger/mlserver-swagger-<version>.json, where <version> is the 3-digit product version number.</p> <ul style="list-style-type: none"> For 9.2.1: https://microsoft.github.io/deployr-api-docs/9.2.1/swagger/mlserver-swagger-9.2.1.json For 9.1.0 https://microsoft.github.io/deployr-api-docs/9.1.0/swagger/rserver-swagger-9.1.0.json
Service-specific APIs	<p>Get the service-specific APIs defined in <code>swagger.json</code> so you can consume that service. Obtain it directly from the user that published the service or retrieve yourself using 'GET /api/{service}/{version}/swagger.json'. Learn more...</p>

Build the core client library

Option 1. Build using a Swagger code generator

To build a client library, run the file through the Swagger code generator, and specify the language you want. Popular Swagger code generation tools include [Azure AutoRest](#) (requires node.js) and [Swagger Codegen](#).



Familiarize yourself with the tool so you can generate the API client libraries in your preferred programming language.

If you use AutoRest to generate a C# client library, it might look like the following command:

```
AutoRest.exe -CodeGenerator CSharp -Modeler Swagger -Input mlserver-swagger-<version>.json -Namespace MyNamespace
```

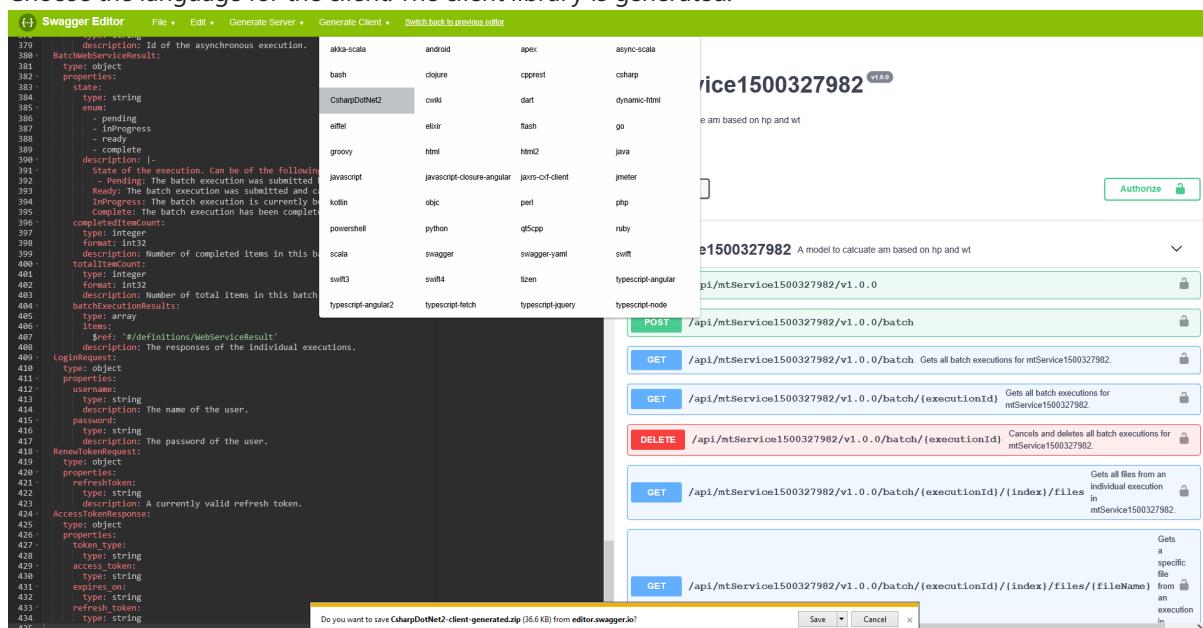
where <version> is the 3-digit product version number.

You can now provide some custom headers and make other changes before using the generated client library stub. See the [Command Line Interface](#) documentation for details regarding different configuration options and preferences.

Option 2. Build using an online Swagger editor

You are not required to install a Swagger code generator on your machine. Instead, you can build the client library in your preferred language using an online Swagger editor.

1. Go to <http://editor.swagger.io/>. The editor appears with default contents.
2. Delete the default contents so the editor is empty.
3. Open the Swagger file on your local machine in a text editor of your choice.
4. Copy the Swagger contents to your clipboard.
5. Switch back to the Swagger site and paste the contents into the online editor.
6. Click the **Generate Client** button on the toolbar.
7. Choose the language for the client. The client library is generated.



Option 3. Build using httr package in R

You can also build a client library directly in R using the httr package. Like option 2, this option does not require you to install a Swagger code generator on your machine. This is very convenient when you want to publish a web service and immediately generate a client library from the resulting Swagger file. Learn more in [this blog post](#).

Add Authentication Workflow Logic

Keep in mind that all APIs require authentication. Therefore, all users must authenticate when making an API call using the `POST /login` API or through Azure Active Directory (AAD).

To simplify this process, bearer access tokens are issued so that users need not provide their credentials for every single call. This bearer token is a lightweight security token that grants the "bearer" access to a protected resource, in this case, Machine Learning Server's operationalization APIs. After a user has provided authentication credentials, the application must validate the user's bearer token to ensure that authentication was successful. [Learn more about managing these tokens](#).

Before you interact with the core APIs, you must authenticate, get the bearer access token, and then include the token in each header for each subsequent request.

- **Azure Active Directory (AAD)**

Add code to pass the AAD credentials, authority, and client ID. In turn, AAD issues the token.

Here is an example of Azure Active Directory authentication in CSharp:

```
<swagger-client-title> client = new <swagger-client-title>(new Uri("https://<host>:<port>"));

// -----
// Note - Update these sections with your appropriate values
// -----


// Address of the authority to issue token.
const string tenantId = "microsoft.com";
const string authority = "https://login.windows.net/" + tenantId;

// Identifier of the client requesting the token
const string clientId = "00000000-0000-0000-0000-000000000000";

// Secret of the client requesting the token
const string clientKey = "00000000-0000-0000-0000-000000000000";

var authenticationContext = new AuthenticationContext(authority);
var authenticationResult = await authenticationContext.AcquireTokenAsync(
    clientId, new ClientCredential(clientId, clientKey));

// Set Authorization header with `Bearer` and access-token
var headers = client.HttpClient.DefaultRequestHeaders;
var accessToken = authenticationResult.AccessToken;

headers.Remove("Authorization");
headers.Add("Authorization", $"Bearer {accessToken}");
```

- **Active Directory LDAP or Local Admin**

For these authentication methods, you must call the `POST /login` API in order to authenticate. Now you can pass in the `username` and `password` for the local administrator, or if Active Directory is enabled, pass the LDAP account information. In turn, Machine Learning Server issues you a `bearer/access token`. After authenticated, the user does not need to provide credentials again as long as the token is still valid.

Here is an example of Active Directory/LDAP authentication in CSharp:

```
<swagger-client-title> client = new <swagger-client-title>(new Uri("https://<host>:<port>"));

// Authenticate using username/password
var loginRequest = new LoginRequest("LDAP_USER_NAME", "LDAP_PASSWORD");
var loginResponse = await client.LoginAsync(loginRequest);

// Set Authorization header with `Bearer` and access-token
var headers = <swagger-client-title>.HttpClient.DefaultRequestHeaders;
var accessToken = loginResponse.AccessToken;

headers.Remove("Authorization");
headers.Add("Authorization", $"Bearer {accessToken}");
```

Interact with the APIs

Now that you have generated the client library and added authentication logic to your application, you can interact with the core operationalization APIs.

```
<swagger-client-title> client = new <swagger-client-title>(new Uri("https://<host>:<port>"));
```

Example: Core Client Library from Swagger (in CSharp)

This example shows how you can use the `mlserver-swagger-9.2.1.json` swagger file to build a client library to interact with the core operationalization APIs from your application. For other versions, get the file from <https://microsoft.github.io/deployr-api-docs/<version>/swagger/mlserver-swagger-<version>.json> where is the server product version.

Build and use a core Machine Learning Server 9.2.1 client library from swagger in CSharp and Azure Active Directory authentication:

1. Download `mlserver-swagger-9.2.1.json` from <https://microsoft.github.io/deployr-api-docs/9.2.1/swagger/mlserver-swagger-9.2.1.json>.

2. Build the statically generated client library files for CSharp from the `mlserver-swagger-9.2.1.json` swagger. Notice the language is `CSharp` and the namespace is `IO.Swagger.Client`.

```
AutoRest.exe -CodeGenerator CSharp -Modeler Swagger -Input mlserver-swagger-9.2.1.json -Namespace IO.Swagger.Client
```

3. In Visual Studio, add the following `NuGet` package dependencies to your VS project.

- `Microsoft.Rest.ClientRuntime`
- `Microsoft.IdentityModel.Clients.ActiveDirectory`

Open the Package Manager Console for NuGet and add them with this command:

```
PM> Install-Package Microsoft.Rest.ClientRuntime
PM> Install-Package Microsoft.IdentityModel.Clients.ActiveDirectory
```

4. Use the statically generated client library files to call the operationalization APIs. In your application code, import the required namespace types and create an API client to manage the API calls:

```
// --- IMPORT NAMESPACE TYPES -----
// Use the namespace provided with `AutoRest.exe -Namespace IO.Swagger.Client`
using IO.Swagger.Client;
using IO.Swagger.Client.Models;

using Microsoft.IdentityModel.Clients.ActiveDirectory;
using Microsoft.Rest

// --- CREATE API CLIENT -----
SwaggerClientTitle client = new SwaggerClientTitle(new Uri("https://rserver.contoso.com:12800"));
```

5. Add the authentication workflow to your application. In this example, the organization has Azure Active Directory.

Since all APIs require authentication, we first need to obtain our `Bearer` access token such that it can be included in every request header like this:

```
GET /resource HTTP/1.1
Host: rserver.contoso.com
Authorization: Bearer mFfl_978_.G5p-4.94gM-
```

In your application code, insert the following:

```

// --- AUTHENTICATE WITH AAD -----
// Note - Update these sections with your appropriate values
// Once authenticated, user do not provide credentials again until token is invalid.
// You can now begin to interact with the core Op APIs
// -----


//
// ADDRESS OF AUTHORITY ISSUING TOKEN
//
const string tenantId = "microsoft.com";
const string authority = "https://login.windows.net/" + tenantId;

//
// ID OF CLIENT REQUESTING TOKEN
//
const string clientId = "00000000-0000-0000-0000-000000000000";

//
// SECRET OF CLIENT REQUESTING TOKEN
//
const string clientKey = "00000000-0000-0000-0000-000000000000";

var authenticationContext = new AuthenticationContext(authority);
var authenticationResult = await authenticationContext.AcquireTokenAsync(
    clientId, new ClientCredential(clientId, clientKey));

//
// SET AUTHORIZATION HEADER WITH BEARER ACCESS TOKEN FOR FUTURE CALLS
//
var headers = client.HttpClient.DefaultRequestHeaders;
var accessToken = authenticationResult.AccessToken;

headers.Remove("Authorization");
headers.Add("Authorization", $"Bearer {accessToken}");

```

6. Begin consuming the core operationalization APIs.

```

// --- INVOKE API -----


// Try creating an R Session `POST /sessions`
var createSessionResponse = client.CreateSession(
    new CreateSessionRequest("Session One"));

Console.WriteLine("Session ID: " + createSessionResponse.SessionId);

```

Example: Service Consumption Client Library from Swagger (in CSharp)

This example shows how you can use the `swagger.json` swagger file for version 1.0.0 of a service named `transmission` to build a client library to interact with published service from your application.

Build and use a service consumption client library from swagger in CSharp and Active Directory LDAP authentication:

1. Get the `swagger.json` for the service you want to consume named `transmission`:

```
GET /api/transmission/1.0.0/swagger.json
```

2. Build the statically generated client library files for CSharp from the `swagger.json` swagger. Notice the

language is `CSharp` and the namespace is `Transmission`.

```
AutoRest.exe -CodeGenerator CSharp -Modeler Swagger -Input swagger.json -Namespace Transmission
```

3. In Visual Studio, add the following `NuGet` package dependencies to your VS project.

- `Microsoft.Rest.ClientRuntime`
- `Microsoft.IdentityModel.Clients.ActiveDirectory`

Open the Package Manager Console for NuGet and add them with this command:

```
PM> Install-Package Microsoft.Rest.ClientRuntime
PM> Install-Package Microsoft.IdentityModel.Clients.ActiveDirectory
```

4. Use the statically-generated client library files to call the operationalization APIs. In your application code, import the required namespace types and create an API client to manage the API calls:

```
// --- IMPORT NAMESPACE TYPES -----
// Use the namespace provided with `AutoRest.exe -Namespace Transmission`
using System;

using Transmission;
using Transmission.Models;

using Microsoft.IdentityModel.Clients.ActiveDirectory;
using Microsoft.Rest

// --- CREATE API CLIENT -----
Transmission client = new Transmission(new Uri("https://rserver.contoso.com:12800"));
```

5. Add the authentication workflow to your application. In this example, the organization has Active Directory/LDAP.

Since all APIs require authentication, first get the `Bearer` access token so it can be included in every request header. For example:

```
GET /resource HTTP/1.1
Host: rserver.contoso.com
Authorization: Bearer mFfl_978_.G5p-4.94gM-
```

In your application code, insert the following code:

```
// --- AUTHENTICATE WITH ACTIVE DIRECTORY -----
// Note - Update these with your appropriate values
// Once authenticated, user won't provide credentials again until token is invalid.
// You can now begin to interact with the operationalization APIs
// -----
var loginRequest = new LoginRequest("LDAP_USERNAME", "LDAP_PASSWORD");
var loginResponse = client.Login(loginRequest);

//
// SET AUTHORIZATION HEADER WITH BEARER ACCESS TOKEN FOR FUTURE CALLS
//
var headers = client.HttpClient.DefaultRequestHeaders;
var accessToken = loginResponse.AccessToken;
headers.Remove("Authorization");
headers.Add("Authorization", $"Bearer {accessToken}");
```

6. Begin consuming the service consumption APIs.

```
// --- INVOKE API -----
InputParameters inputs = new InputParameters() { Hp = 120, Wt = 2.8 };
var serviceResult = client.ManualTransmission(inputs);

Console.Out.WriteLine(serviceResult.OutputParameters.Answer);
```

Example in Java

The following blog article shows how to create a Java client.

<https://blogs.msdn.microsoft.com/mlserver/2017/10/04/enterprise-friendly-java-client-for-microsoft-machine-learning-server/>

See also

- [Blog article: REST Calls using PostMan](#)

Manage access tokens for API requests

7/12/2022 • 4 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Applies to: Machine Learning Server, Microsoft R Server 9.x

Machine Learning Server, formerly known as Microsoft R Server, uses tokens to identify and authenticate the user who is sending the API call within your application. Users must authenticate when making an API call. They can do so with the 'POST /login HTTP/1.1' API call, after which Machine Learning Server issues a bearer token to your application for this user. Alternately, if the organization is using Azure Active Directory (AAD), users receive a bearer token from AAD when they authenticate.

This bearer token is a lightweight security token that grants the "bearer" access to a protected resource, in this case, Machine Learning Server's core APIs for operationalizing analytics. After a user has been authenticated, the application must validate the user's bearer token to ensure that authentication was successful.

IMPORTANT

For proper access token signing and verification across your configuration, ensure that the JWT settings are exactly the same for every web node. These JWT settings are defined on each web node in the configuration file, appsetting.json. Check with your administrator. [Learn more...](#)

Security Concerns

Despite the fact that a party must first authenticate to receive the token, tokens can be intercepted by an unintended party if the token is not secured in transmission and storage. While some security tokens have a built-in mechanism to protect against unauthorized parties, these tokens do not and must be [transported in a secure channel such as transport layer security \(HTTPS\)](#).

If a token is transmitted in the clear, a man-in-the-middle attack can be used by a malicious party to acquire the token to make an unauthorized access to a protected resource. The same security principles apply when storing or caching tokens for later use. Always ensure that your application transmits and stores tokens in a secure manner.

You can [revoke a token](#) if a user is no longer permitted to make requests on the API or if the token has been compromised.

Create tokens

The API bearer token's properties include an access_token / refresh_token pair and expiration dates.

Tokens can be generated in one of two ways:

- If Active Directory LDAP or a local administrator account is enabled, then send a 'POST /login HTTP/1.1'

API request to retrieve the bearer token.

- If Azure Active Directory (AAD) is enabled, then [the token comes from AAD](#).

[Learn more about these authentication methods.](#)

Example: Token creation request

- Request

```
POST /login HTTP/1.1
{
  "username": "my-user-name",
  "password": "$ecRetPas$1"
}
```

- Response

```
{
  "token_type": "Bearer",
  "access_token": "eyJhbGci....",
  "expires_in": 3600,
  "expires_on": 1479937454,
  "refresh_token": "0/LTo...."
}
```

Token Lifecycle

The bearer token is made of an access_token property and a refresh_token property.

	THE "ACCESS_TOKEN" LIFECYCLE	THE "REFRESH_TOKEN" LIFECYCLE
Gets Created	Whenever the user logs in, or a refreshToken api is called	Whenever the user logs in
Expires	After 1 hour (3660 seconds) of inactivity	After 336 hours (14 days) of inactivity
Becomes Invalid	If the refresh_token was revoked, or If not used for 336 hours (14 days), or When a new pair of access_token/refresh_token has been created	If not used for 336 hours (14 days), or When the refresh_token expires, or When a new access_token/refresh_token pair was created, or If the refresh_token was revoked

Use tokens

As defined by HTTP/1.1 [RFC2617], the application should send the access_token directly in the Authorization request header.

You can do so by including the bearer token's access_token value in the HTTP request body as 'Authorization: Bearer {access_token_value}'.

When the API call is sent with the token, Machine Learning Server attempts to validate that the user is

successfully authenticated and that the token itself is not expired.

- If an authenticated user has a bearer token's access_token or refresh_token that is expired, then a '401 - Unauthorized (invalid or expired refresh token)' error is returned.
- If the user is not successfully authenticated, a '401 - Unauthorized (invalid credentials)' error is returned.

Examples

Example HTTP header for session creation:

```
POST /sessions HTTP/1.1
Host: mrs.contoso.com
Authorization: Bearer eyJhbGci.....
...
```

Example HTTP header for publishing web service:

```
POST /api/{service}/{version} HTTP/1.1
Host: mrs.contoso.com
Authorization: Bearer eyJhbGci.....
...
```

Renew tokens

A valid bearer token (with active access_token or refresh_token properties) keeps the user's authentication alive without requiring him or her to re-enter their credentials frequently.

The access_token can be used for as long as it's active, which is up to one hour after login or renewal. The refresh_token is active for 336 hours (14 days). After the access_token expires, an active refresh_token can be used to get a new access_token / refresh_token pair as shown in the following example. This cycle can continue for up to 90 days after which the user must log in again. If the refresh_token expires, the tokens cannot be renewed and the user must log in again.

To refresh a token, use [the 'POST /login/refreshToken HTTP/1.1' API call](#).

Example: Refresh access_token

- Example request:

```
POST /login/refreshToken HTTP/1.1
Connection: Keep-Alive
Content-Type: application/json; charset=utf-8
Accept-Encoding: gzip, deflate
Content-Length: 370
Host: mrs.contoso.com

{
    "refreshToken": "0/LTo...."
}
```

- Example response:

```
{  
  "token_type": "Bearer",  
  "access_token": "eyJhbGci....",  
  "expires_in": 3600,  
  "expires_on": 1479937523,  
  "refresh_token": "ScW2t...."  
}
```

Revoke refresh tokens

A refresh_token should be revoked:

- If a user is no longer permitted to make requests on the API, or
- If the access_token or refresh_token have been compromised.

Use [the 'DELETE /login/refreshToken?refreshToken={refresh_token_value}' HTTP/1.1' API call](#) to revoke a token.

Example: Revoke token

- Example request:

```
DELETE https://mrs.contoso.com/login/refreshToken?refreshToken=ScW2t HTTP/1.1  
Connection: Keep-Alive  
Accept-Encoding: gzip, deflate  
Host: mrs.contoso.com
```

- Example response:

```
HTTP 200 Success
```

Log in to Machine Learning Server or R Server with mrsdeploy and open a remote session

7/12/2022 • 12 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Applies to: Machine Learning Server, Microsoft R Server 9.x

The `mrsdeploy` package, delivered with Microsoft R Client and Machine Learning Server (formerly known as R Server), provides functions for:

- Establishing a remote session in an R console application for the purposes of executing code on that server
- Publishing and managing an R web service that is backed by the R code block or script you provided.

Each feature can be used independently, but the greatest value is achieved when you use both.

This article explains the authentication functions, the arguments they accept, and how to switch between remote and local sessions.

Authentication

This section describes how to authenticate to Machine Learning Server on-premises or in the cloud using the functions in `mrsdeploy`.

In Machine Learning Server, every API call between the Web server and client must be authenticated. The `mrsdeploy` functions, which place API calls on your behalf, are no different. Authentication of user identity is handled via Active Directory. Machine Learning Server never stores or manages usernames and passwords.

By default, all `mrsdeploy` operations are available to authenticated users. Destructive tasks, such as deleting a web service from a remote execution command line, are available only to the user who initially created the service. However, your administrator can also [assign role-based authorization](#) to further control the permissions around web services.

`mrsdeploy` provides two functions for authentication against Machine Learning Server: `remoteLogin()` and `remoteLoginAAD()`. These functions support not just authentication, but creation of a remote R session on the Machine Learning Server. By default, the `remoteLogin()` and `remoteLoginAAD()` functions log you in, create a remote R session on the Machine Learning Server instance, and open a remote command prompt. The function you use depends on the [type of authentication and deployment in your organization](#).

On premises authentication

Use the `remoteLogin` function in these scenarios:

- you are authenticating using Active Directory server on your network
- you are using the [default administrator account](#) for an on-premises instance of Machine Learning Server

This function calls `/user/login` API, which requires a username and password. For example:

```
> remoteLogin(
  https://YourHostEndpoint,
  session = TRUE,
  diff = TRUE,
  commandline = TRUE
  username = NULL,
  password = NULL,
)
```

For example, here is an AD authentication that creates a remote R session, but prompts for a username and password. That username can be the Active Directory/LDAP username and password, or the local admin account and its password.

```
> remoteLogin("http://localhost:12800",
  session = FALSE)
```

In another example, we authenticate using the local 'admin' account and password and create a remote R session. Then, upon login, we are placed at the remote session's command prompt. A report of the differences between local and remote environments is returned.

```
> remoteLogin("http://localhost:12800",
  username = "admin",
  password = "{{YOUR_PASSWORD}}",
  diff = TRUE,
  session = TRUE,
  commandline = TRUE)
```

NOTE

Unless you specify otherwise using the arguments below, this function not only logs you in, but also creates a remote R session the server instance and put you on the remote command line. If you don't want to be in a remote session, either set session = FALSE or [switch back to the local session](#) after login and logout.

REMOTELGIN ARGUMENT	DESCRIPTION
endpoint	The Machine Learning Server HTTP/HTTPS endpoint, including the port number. You can find this on the first screen when you launch the administration utility .
session	If TRUE, create a remote session. If omitted, creates a remote session.
diff	If TRUE, creates a 'diff' report showing differences between the local and remote sessions. Parameter is only valid if session parameter is TRUE.
commandline	If TRUE, creates a "REMOTE" command line in the R console. Parameter is only valid if session parameter is TRUE. If omitted, it is the same as = TRUE .
prompt	The command prompt to be used for the remote session. By default, <code>REMOTE></code> is used.

REMOTELOGIN ARGUMENT	DESCRIPTION
username	If NULL, user is prompted to enter your AD or local Machine Learning Server username.
password	If NULL, user is prompted to enter password.

IMPORTANT

If you do not specify a username and password as arguments to the login function, you are prompted for your AD or [local Machine Learning Server](#) username and password when you run this command.

Cloud authentication

To authenticate with Azure Active Directory, use the `remoteLoginAAD` function.

This function takes several arguments as follows:

```
> remoteLoginAAD(
  endpoint,
  authuri = "https://login.windows.net",
  tenantid = "<AAD_DOMAIN>",
  clientid = "<NATIVE_APP_CLIENT_ID>",
  resource = "<WEB_APP_CLIENT_ID>",
  username = "NameOfUser",
  password = "UserPassword",
  session = TRUE,
  diff = TRUE,
  commandline = TRUE
)
```

Unless you specify otherwise, this function:

1. Logs the user in
2. Creates a remote R session on the server instance
3. Puts the user on the remote command line in that remote session

If you do not want to be in a remote session, either set `session = FALSE` or [switch back to the local session](#) after login and logout.

For example, here is another AAD authentication that **does not create a remote R session**. It also prompts for a username and password at runtime.

If you do not specify the username and password as arguments to the login function, you are prompted for your AAD username and password at runtime.

```
> remoteLoginAAD(
  "https://mlserver.contoso.com:12800",
  authuri = "https://login.windows.net",
  tenantid = "microsoft.com",
  clientid = "00000000-0000-0000-0000-000000000000",
  resource = "00000000-0000-0000-0000-000000000000",
  session = FALSE
)
```

WARNING

Whenever you omit the username or password, you are prompted for your credentials at runtime. If you have issues with the AAD login pop-up, you may need to include the username and password as command arguments directly.

REMOTELOGINAAD ARGUMENT	DESCRIPTION
endpoint	The Machine Learning Server HTTP/HTTPS endpoint, including the port number. This endpoint is the SIGN-ON URL value from the web application
authuri	The URI of the authentication service for Azure Active Directory.
tenantid	The tenant ID of the Azure Active Directory account being used to authenticate is the domain of AAD account.
clientid	The numeric CLIENT ID of the AAD "native" application for the Azure Active Directory account.
resource	The numeric CLIENT ID from the AAD "Web" application for the Azure Active Directory account, also known by the <code>Audience</code> in the configuration file.
session	If TRUE, create a remote session. If omitted, creates a remote session.
diff	If TRUE, creates a 'diff' report showing differences between the local and remote sessions. Parameter is only valid if session parameter is TRUE.
commandline	If TRUE, creates a "REMOTE" command line in the R console. Parameter is only valid if session parameter is TRUE.
prompt	The command prompt to be used for the remote session. By default, <code>REMOTE></code> is used. If omitted, it is the same as <code>= TRUE</code> .
username	If NULL, user is prompted to enter username <code><username>@<AAD-account-domain></code> . If you have issues with the AAD login pop-up, try including the username and password as command arguments directly.
password	If NULL, user is prompted to enter password.

If you do not know your `tenantid`, `clientid`, or other details, contact your administrator. Or, if you have access to the Azure portal for the relevant Azure subscription, you can find [these authentication details](#).

Arguments for remote execution

If you plan to use `mrsdeploy` to start a remote session on the server and execute code remotely, there are two key parts to the command:

1. Create that remote session

2. Indicate whether you'd like to open a local or remote command prompt upon login.

Take special note of the arguments `session` and `commandline` as these influence the state of your command line.

ARGUMENT	DESCRIPTION
<code>session</code>	If TRUE, create a remote session in the server. If omitted, it still creates a remote session. If FALSE, does not create any remote R sessions.
<code>commandline</code>	If TRUE, creates a REMOTE command line in the R console so you can interact with the remote R session. After the authenticated connection is made, the user is executing R commands remotely until they switch back to the local command line or logout. Parameter is only valid if session parameter is TRUE.

For more information on remote execution, see [this article](#).

WARNING

In the case where you are working with a [remote R session](#), there are several approaches to session management when publishing.

Access tokens

After you authenticate with Active Directory or Azure Active Directory, an [access token](#) is returned. This access token is then passed in the request header of every subsequent `mrsdeploy` request.

Keep in mind that each API call and every `mrsdeploy` function requires authentication with Machine Learning Server. If the user does not provide a valid login, an `Unauthorized` HTTP `401` status code is returned.

Remote connection states

Depending on how you configure the `session` and `commandline` login parameters are subtle, your execution context can switch between local and remote contexts.

Create remote R session and go to remote command line (1)

In this state, we authenticate using one of the two aforementioned login functions with the default argument `session = TRUE` to create a remote R session, and the default argument `commandline = TRUE` to transition to the remote R command line.

NOTE

Unless you specify `session = FALSE`, this function not only logs you in, but also creates a remote R session on the Machine Learning Server instance. And, unless you specify `commandline = FALSE`, you are on the remote command line upon login. If you don't want to be in a remote session, either set `session = FALSE` or [switch back to the local session](#) after login and logout.



When you see the default prompt `REMOTE>` in the command pane, you know that you are now interacting with your remote R session and are no longer in your local R environment:

In this example, we define an interactive authentication workflow that spans both our local and remote environments.

```
> # EXAMPLE: LOGIN, CREATE REMOTE R SESSION, GO TO REMOTE PROMPT  
  
> remoteLogin("http://localhost:12800")  
  
REMOTE> x <- 10    # Assign 10 to "x" in remote session  
  
REMOTE> ls()    # List objects in remote session  
[1] "x"  
  
REMOTE> pause()  # Pause remote interaction. Switch to local  
  
> y <- 10      # Assign 10 to "y" in local session  
  
> ls()    # List objects in local session  
[1] "y"  
  
> putLocalObject(c("y"))  # Loads local "y" into remote R session's workspace  
  
> resume()    # Resume remote interaction and move to remote command line  
  
REMOTE> ls()    # List the objects now in the remote session  
[1] "x" "y"  
  
REMOTE> exit  # Destroy remote session and logout  
  
>
```

IMPORTANT

You can only manage web services from your local session. Attempting to use the service APIs during a remote interaction results in an error.

Create remote R session and remain with local command line (2)

In this state, you can authenticate using `remoteLogin`, which is one of the two aforementioned login functions with the argument `session = TRUE` to create a remote R session, and the argument `commandline = FALSE` to remain in your local R environment and command line.

COMMAND	STATE
<pre>> remoteLogin("http://localhost:12800", session = TRUE, commandline = FALSE) ></pre>	

In this example, we define an interactive authentication workflow that spans both our local and remote environments (just like state 1), but starts out in the local R session, and only then moves to the remote R session.

<pre>> # EXAMPLE: LOGIN, CREATE REMOTE R SESSION, STAY LOCAL > remoteLogin("http://localhost:12800", session = TRUE, commandline = FALSE) > y <- 10 # Assign 10 to "y" in local session > ls() # List the objects in the local session [1] "y" > putLocalObject(c("y")) # Loads local "y" into remote R session's workspace > resume() # Switch to remote command line for remote interaction REMOTE> x <- 10 # Assign 10 to "x" in remote session REMOTE> ls() # List the objects now in the remote session [1] "x" "y" REMOTE> exit # Destroy remote session and logout ></pre>

Remain local without creating a remote R session (3)

In this state, you can authenticate with `remoteLogin()` and its argument `session = FALSE` so that no remote R session is started. Without a remote R session, you only have the local R environment and command line.

COMMAND	STATE
<pre>> remoteLogin("http://localhost:12800", session = FALSE) ></pre>	

In this example, we define an interactive authentication workflow without a remote R session (`session = FALSE`). This is useful when working only with the web service functionality of the `mrsdeploy` package. After authentication, we remain confined within the local R session in order to publish and consume a service.

```

> # EXAMPLE OF LOGIN WITHOUT REMOTE R SESSION

> remoteLogin("http://localhost:12800", session = FALSE)

> addOne <- function(x) x + 1

> api <- publishService(
  "add-one",
  code = addOne,
  inputs = list(x = "numeric"),
  outputs = list(answer = "numeric")
)

> res <- api$addOne(100)

> print(res$output("answer"))
[1] 101

```

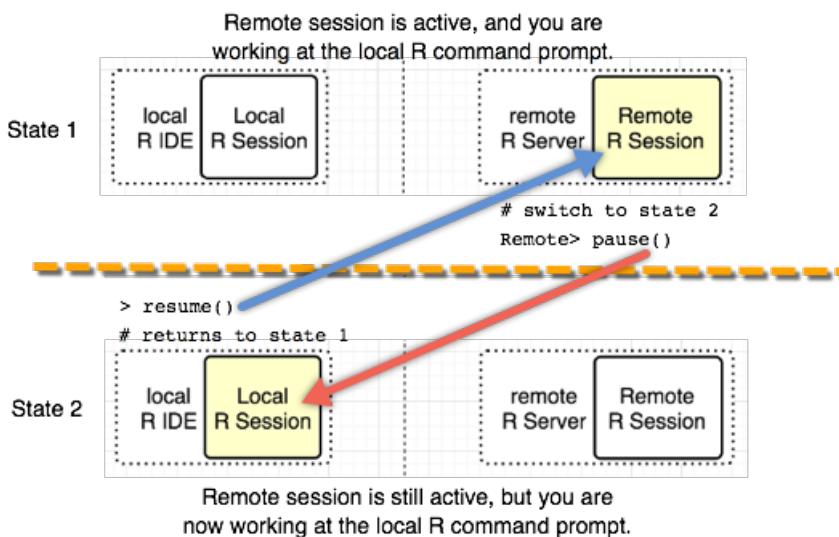
IMPORTANT

You can only manage web services from your local session. Attempting to use the service APIs during a remote interaction results in an error.

Switch between remote and local sessions

After you [log in to the remote Machine Learning Server](#) with the argument `session = TRUE`, a remote R session is created. You can switch between the remote R session and the local R session directly from the command line. The remote command line allows you to directly interact with a Machine Learning Server 9.x instance on another machine.

When the `REMOTE>` command line is displayed in the R console, any R commands entered are executed on the remote R session.



Switching between the local command line and the remote command line is done using these functions: `pause()` and `resume()`. To switch back to the local R session, type `'pause()'`. If you have switched to the local R session, you can go back to the remote R session by typing `'resume()'`.

To terminate the remote R session, type `'exit'` at the `REMOTE>` prompt. Also, to terminate the remote session from the local R session, type `'remoteLogout()'`.

CONVENIENCE FUNCTIONS	DESCRIPTION
pause()	When executed from the remote R session, returns the user to the local '>' command prompt.
resume()	When executed from the local R session, returns the user to the 'REMOTE>' command prompt, and sets a remote execution context.
exit	Logs you out of the session.

Example

```
#execute some R commands on the remote session
REMOTE>x<-rnorm(1000)
REMOTE>hist(x)

REMOTE>pause() #switches the user to the local R session
>resume()

REMOTE>exit #logout and terminate the remote R session
>
```

Logout of a remote session

To terminate the remote R session while you are on the remote command line, type 'exit' at the REMOTE> prompt.

To terminate the remote session from the local R session, type '[remoteLogout\(\)](#)'.

See also

- [What is operationalization?](#)
- [What are web services?](#)
- [mrsdeploy function overview](#)
- [Working with web services in R](#)
- [Asynchronous batch execution of web services in R](#)
- [Execute on a remote Machine Learning Server](#)
- [How to integrate web services and authentication into your application](#)

Execute on a remote server using the mrsdeploy package

7/12/2022 • 11 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Applies to: Machine Learning Server, Microsoft R Server 9.x, Microsoft R Client 3.x

Remote execution is the ability to issue R commands from either Machine Learning Server (or R Server) or R Client to a remote session running on another Machine Learning Server instance. You can use remote execution to offload heavy processing on server and test your work. It is especially useful while developing and testing your analytics.

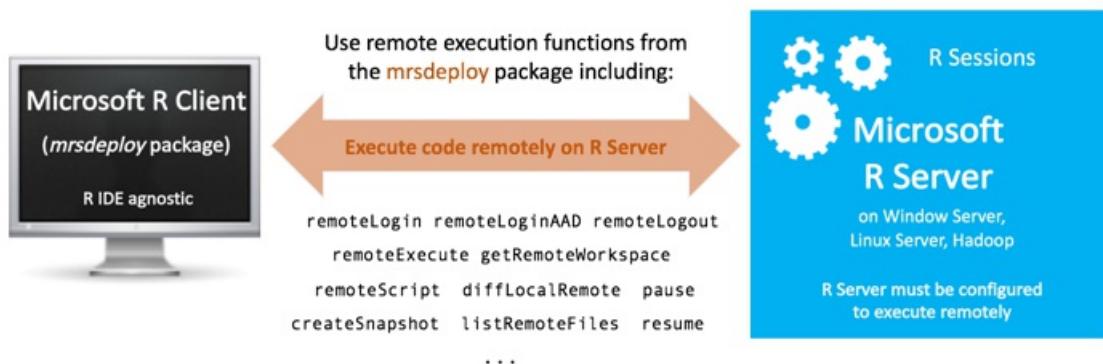
Remote execution is supported in several ways:

- From the command line in console applications
- In R scripts that call [functions from the mrsdeploy package](#)
- From code that calls the APIs.

You can enter 'R' code just as you would in a local R console. R code entered at the remote command line executes on the remote server.

With remote execution, you can:

- [Log in and out of Machine Learning Server](#)
- [Generate diff reports of the local and remote environments](#) and reconcile any differences
- [Execute R scripts and code remotely](#)
- [Work with R objects/files remotely](#)
- [Create and manage snapshots of the remote environment for reuse](#)



Supported configurations and mrsdeploy usage

The R functions used for remote execution are provided in the mrsdeploy package, which is installed with Machine Learning Server on almost every platform. Your administrator must [configure the server for the deployment and consumption of analytics](#)** before you can use the functions in the package. Read the article "

`mrsdeploy` functions" for the list of [remote execution functions](#) in that package.

How to create a remote session

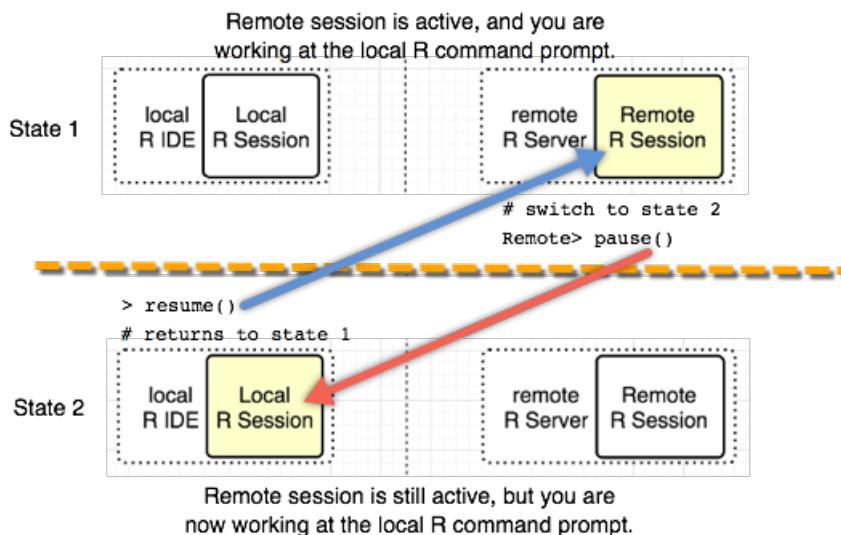
To create a remote session, you must first authenticate with Machine Learning Server using one of the `mrsdeploy` login functions: `remoteLogin()` and `remoteLoginAAD()`. With these functions, you can authenticate, set the arguments to create a remote R session on the Machine Learning Server (`session = TRUE`) and even place yourself in the remote command line upon login `commandline = TRUE`.

Read the article "[Connecting to Machine Learning Server with mrsdeploy](#)" for more on authentication with `mrsdeploy` and syntax.

How to switch between sessions or logout

After you [log in to the remote R server](#) with the argument `session = TRUE`, a remote R session is created. You can switch between the remote R session and the local R session directly from the command line. The remote command line allows you to directly interact with an R Server 9.x instance on another machine.

When the `REMOTE>` command line is displayed in the R console, any R commands entered are executed on the remote R session.



Switching between the local command line and the remote command line is done using these functions: `pause()` and `resume()`. To switch back to the local R session, type '`pause()`'. If you have switched to the local R session, you can go back to the remote R session by typing '`resume()`'.

To terminate the remote R session, type '`exit`' at the `REMOTE>` prompt. Also, to terminate the remote session from the local R session, type '`remoteLogout()`'.

CONVENIENCE FUNCTIONS	DESCRIPTION
<code>pause()</code>	When executed from the remote R session, returns the user to the local <code>></code> command prompt.
<code>resume()</code>	When executed from the local R session, returns the user to the <code>REMOTE></code> command prompt, and sets a remote execution context.

```
#EXAMPLE: SESSION SWITCHING

#execute some R commands on the remote session
REMOTE>x<-rnorm(1000)
REMOTE>hist(x)

REMOTE>pause() #switches the user to the local R session
>resume()

REMOTE>exit #logout and terminate the remote R session
>
```

Create a diff report

A `diff` report is available so you can see and manage differences between the local and remote R environments. The diff report contains details on:

- The R versions running on the server and locally
- The R packages installed locally, but not on the remote session
- The differences between R package versions.

This report appears whenever you log in, and you can also get the report using the `diffLocalRemote()` function.

Execute an R script remotely

If you have R scripts on your local machine, you can execute them remotely by using the function `remoteScript()`. This function takes a path to an R script to be executed remotely. You also have options to save or display any plots that might have been generated during script execution. The function returns a list containing the status of the execution (success/failure), the console output generated, and a list of files created.

NOTE

If you need more granular control of a remote execution scenario, use the `remoteExecute()` function.

Package dependencies

If your R script has R package dependencies, those packages must be installed on the Microsoft R Server. Your administrator can install them globally on the server or you can install them yourself for the duration of your remote session using the `install.packages()` function. Leave the `lib` parameter empty.

Limitations in a remote context

Certain functions are masked from execution, such as '`help`', '`browser`', '`q`' and '`quit`'.

In a remote context, you cannot display vignettes or get help at your command-line prompt.

In most cases, "system" commands work. However, system commands that write to `stdout/stderr` may not display their output nor wait until the entire system command has completed before displaying output.

`install.packages` is the only exception for which we explicitly handle `stdout` and `stderr` in a remote context.

Asynchronous remote execution

To continue working in your development environment during the remote script execution, you can execute your R script asynchronously. Asynchronous script execution is useful when you are running scripts that have long execution times.

To execute an R script asynchronously, set the `async` parameter for `remoteScript()` to `TRUE`. When `remoteScript()`

is executed, the script is run asynchronously in a new remote R console window. All R console output and any plots from that execution are returned to the same window.

WARNING

R Server 9.0 users! When loading a library for the REMOTE session, set lib.loc=getwd() as such:

```
library("<packagename>", lib.loc=getwd())
```

Example

```
#EXAMPLE: REMOTE SCRIPT EXECUTION

#install a package for the life of the session
REMOTE>install.packages("bitops")

#switch to the local R session
REMOTE>pause()

#execute an R script remotely
>remoteScript("C:/myScript.R")

#execute that script again in another window asynchronously
>remoteScript("C:/myScript.R", async=TRUE)
```

Work with R objects and files remotely

After you have executed an R code remotely, you may want to retrieve certain R objects and load them into your local R session. For example, if you have an R script that creates a linear model `m<-lm(x~y)`, use the function `getRemoteObject()` to retrieve the object `m` in your local R session.

Conversely, if you have a local R object that you want to make available to your remote R session, you can use the function `putLocalObject()`. If you want to sync your local and remote workspaces, the functions `putLocalWorkspace()` and `getRemoteWorkspace()` can be used.

Similar capabilities are available for files that need to be moved between the local and remote R sessions.

The following functions are available for working with files: `putLocalFile()`, `getRemoteFile()`, `listRemoteFiles()` and `deleteRemoteFile()`.

```
#EXAMPLE: REMOTE R OBJECTS AND FILES

#execute a script remotely that generated 2 R objects we are interested in retrieving
>remoteExecute("C:/myScript.R")
#retrieve the R objects from the remote R session and load them into our local R session
>getRemoteObject(c("model","out"))

#an R script depends upon an R object named `data` to be available. Move the local
#instance of `data` to the remote R session
>putLocalObject("data")
#execute an R script remotely
>remoteScript("C:/myScript2.R")

#push a data file to the remote R session
>putLocalFile("C:/data/survey.csv")
#execute an R script remotely
>remoteScript("C:/myScript2.R")
```

A word on plots

When you plot remotely, the default plot size is 400 x 400 pixels. If you desire higher-resolution output, you must tell the remote session the size of plot to create.

On a local session, you might change the width and height as follows:

```
> png(filename="myplot.png", width=1440, height=900)
> ggplot(aes(x=value, group=am, colour=factor(am)), data=mtcarsmelt) + geom_density() +
  facet_wrap(~variable, scales="free")
> dev.off()
```

When working on the REMOTE command line, you need to combine these three statements together:

```
REMOTE> png(filename="myplot.png", width=1440, height=900);ggplot(aes(x=value, group=am, colour=factor(am)),
  data=mtcarsmelt) + geom_density() + facet_wrap(~variable, scales="free");dev.off()
```

As an alternative, you can use the remoteScript() function as follows:

```
#Open a new script window in your IDE
#Enter the commands on separate lines

png(filename="myplot.png", width=1440, height=900)
ggplot(aes(x=value, group=am, colour=factor(am)), data=mtcarsmelt) + geom_density() + facet_wrap(~variable,
  scales="free")
dev.off()
```

```
#Save the script to a file such as myscript.R
#Switch from the remote session to the local session by typing pause() on the REMOTE command line
REMOTE> pause()
>
```

```
#From the local command prompt, execute your remote script
> remoteScript("myscript.R")
```

R session snapshots

Session snapshot functions are useful for remote execution scenarios. It can save the whole workspace and working directory so that you can pick up from exactly where you left last time. Think of it as similar to saving and loading a game.

What's in a session snapshot

If you need a prepared environment for remote script execution that includes R packages, R objects, or data files, consider creating a **snapshot**. A snapshot is an image of a remote R session saved to Microsoft R Server, which includes:

- The session's workspace along with the installed R packages
- Any files and artifacts in the working directory

A session snapshot can be loaded into any subsequent remote R session for the user who created it. For example, suppose you want to execute a script that needs three R packages, a reference data file, and a model object. Instead of loading the data file and object each time you want to execute the script, create a session snapshot of an R session containing them. Then, you can save time later by retrieving this snapshot using its ID

to get the session contents exactly as they were at the time the snapshot was created. However, any necessary packages must be reloaded as described in the next section.

Snapshots are only accessible to the user who creates them and cannot be shared across users.

The following functions are available for working with snapshots:

`listSnapshots()`, `createSnapshot()`, `loadSnapshot()`, `downloadSnapshot()`, and `deleteSnapshot()`.

Snapshot guidance and warnings

Take note of the following tips and recommendations around using session snapshots:

- One caveat is that while the workspace is saved inside the session snapshot, it does not save loaded packages. If your code depends on certain R packages, use the `require()` function to include those packages directly in the R code that is part of the web service. The `require()` function loads packages from within other functions. For example, you can write the following code to load the RevoScaleR package:

```
delayPrediction <- function(depTime, dayOfWeek) {  
  require(RevoScaleR)  
  test <- data.frame(CRSDepTime=depTime, DayOfWeek=factor(dayOfWeek, levels = c("Monday", "Tuesday",  
  "Wednesday", "Thursday", "Friday", "Saturday", "Sunday")))  
  rxPredict(modelObject = modelInfo$predictiveModel, data = test, outData = test, verbose = 1, type =  
  "response")  
}
```

- While you can use snapshots when publishing a web service for environment dependencies, it can have an impact on performance at consumption time. For optimal performance, consider the size of the snapshot carefully especially when publishing a service. Before creating a snapshot, ensure that you keep only those workspace objects you need and purge the rest. And, if you only need a single object, consider passing that object alone itself instead of using a snapshot.

- R Server 9.0 users When loading a library for the REMOTE session, set `lib.loc=getwd()` as such:

```
library("<packagename>", lib.loc=getwd())
```

Example of snapshot usage

```
#EXAMPLE: USING SNAPSHOTS  
  
#configure our remote session  
REMOTE>install.packages(c("arules","bitops","caTools"))  
>putLocalFile("C:/data/survey_reference.csv")  
>putLocalObject("model")  
>snapshot_id<-<-createSnapshot("my modeling environment")  
  
#whenever I need the modeling environment, reload the snapshot  
>loadSnapshot(snapshot_id)  
#execute an R script remotely  
>remoteScript("C:/myScript2.R")
```

Publishing web services in a remote session

After you understand the mechanics of remote execution, consider incorporating web service capabilities. You can publish an R web service composed of arbitrary R code block that runs on the remote R Server. For more information on publishing services, begin with the [Working with web services in R](#) guide.

To publish a web service after you create a remote session (argument `session = TRUE` with `remoteLogin()` or `remoteLoginAAD()`), you have two approaches:

- Publish from your local session: At the `REMOTE>` prompt, use `pause()` to return the R command line in

your local session. Then, publish your service. Use `resume()` from your local prompt to return to the command line in the remote R session.

- Authenticate again from within the remote session to enable connections from that remote session to the web node API. At the `REMOTE>` prompt, authenticate with `remoteLogin()` or `remoteLoginAAD()`. However, explicitly set the argument `session = FALSE` this time so that a second remote session is NOT created and provide your username and password directly in the function. When attempting to log in from a remote session, you are not prompted for user credentials. Instead, pass valid values for `username` and `password` to this function. Then, you are authenticated and able to publish from the `REMOTE>` prompt.

WARNING

If you try to publish a web service from the remote R session without authenticating from that session, you get a message such as

```
Error in curl::curl_fetch_memory(uri, handle = h) : URL using bad/illegal format or missing URL .
```

Learn more about authenticating with `remoteLogin()` or `remoteLoginAAD()` in this article "[Logging in to R Server with mrsdeploy](#)."

```
> ## AAD AUTHENTICATION TO PUBLISH FROM REMOTE SESSION ##

> remoteLoginAAD(
  "https://rserver.contoso.com:12800",
  authuri = "https://login.windows.net",
  tenantid = "microsoft.com",
  clientid = "00000000-0000-0000-0000-000000000000",
  resource = "00000000-0000-0000-0000-000000000000",
  session = TRUE
)

Your REMOTE R session is now active.
Commands:
  - pause() to switch to local session & leave remote session on hold.
  - resume() to return to remote session.
  - exit to leave (and terminate) remote session.

REMOTE> remoteLoginAAD(
  "https://rserver.contoso.com:12800",
  authuri = "https://login.windows.net",
  tenantid = "microsoft.com",
  clientid = "00000000-0000-0000-0000-000000000000",
  resource = "00000000-0000-0000-0000-000000000000",
  session = FALSE
  username = "{{YOUR_USERNAME}}"
  password = "{{YOUR_PASSWORD}}"
)

REMOTE>api <- publishService(
  serviceName,
  code = manualTransmission,
  model = carsModel,
  inputs = list(hp = "numeric", wt = "numeric"),
  outputs = list(answer = "numeric"),
  v = "v1.0.0"
)
```

See also

- [mrsdeploy function overview](#)
- [Connecting to R Server from mrsdeploy](#)

- [Get started guide for Data scientists](#)
- [Working with web services in R](#)

Write custom chunking algorithms using rxDataStep in RevoScaleR

7/12/2022 • 4 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Scalability in RevoScaleR is based on chunking or external memory algorithms that can analyze chunks of data in parallel and then combine intermediate results into a single analysis. Because of the chunking algorithms, it's possible to analyze huge datasets that vastly exceed the memory capacity of any one machine.

All of the main analysis functions in RevoScaleR (*rxSummary*, *rxLinMod*, *rxLogit*, *rxGlm*, *rxCube*, *rxCrossTabs*, *rxCovCor*, *rxKmeans*, *rxDTree*, *rxBTrees*, *rxNaiveBayes*, and *rxDForest*) use chunking or external memory algorithms. However, for scenarios where specialized behaviors are needed, you can create a custom chunking algorithms using the *rxDataStep* function to automatically chunk through your data set and apply arbitrary R functions to process your data.

In this article, you'll step through an example that teaches a simple chunking algorithm for tabulating data implemented using *rxDataStep*. A more realistic tabulation approach is to use the *rxCrossTabs* or *rxCube* functions, but since *rxDataStep* is simpler, it's a better choice for instruction.

NOTE

If *rxDataStep* does not provide sufficient customization, you can build a custom parallel external memory algorithm from the ground up using functions in the **RevoPemaR** package. For more information, see [How to use the RevoPemaR library in Machine Learning Server](#).

Prerequisites

Sample data for this example is the *AirlineDemoSmall.xdff* file with a local compute context. For instructions on how to import this data set, see the tutorial in [Practice data import and exploration](#).

Chunking is supported on Machine Learning Server, but not on the free R Client. Because the dataset is small enough to reside in memory on most computers, most systems succeed in running this example locally. However, if the data does not fit in memory, you will need to use Machine Learning Server instead.

About chunking algorithms

An updating algorithm takes a given set of values and a chunk of data, and then outputs a revised set of values cumulative for all chunks. The simplest example is an updating sum: sum is computed for the first chunk, followed by a second chunk, which each successive chunk contributing to a revised value until reaching the cumulative sum.

Updating algorithms perform four main tasks:

- Initialization: declare and initialize variables needed in the computation.
- ProcessData: for each block, perform calculations on the data in the block.

- `UpdateResults`: combine all of the results of the `ProcessData` step.
- `ProcessResults`: when results from all blocks have been combined, do any final computations.

In this example, no initialization is required.

Example: `rxDataStep` and sample airline data

The `ProcessData` step is performed within a `transformFunc` called by `rxDataStep` for each chunk of data. In this case we begin with a simple call to the `table` function after converting the chunk to a data frame. The results are then converted back to a data frame with a single row, which will be appended to a data set on disk. So, the call to `rxDataStep` reads in the data chunk-by-chunk and creates a new summary data set where each row represents the “intermediate results” of a chunk.

The `AggregateResults` function shown below combines the `UpdateResults` and `ProcessResults` tasks. The summary data set is simply read into memory and the columns are summed.

To try this out, create a new script `chunkTable.R` with the following contents:

```
chunkTable <- function(inDataSource, iroDataSource, varsToKeep = NULL,
  blocksPerRead = 1 )
{
  ProcessChunk <- function( dataList)
  {
    # Process Data
    chunkTable <- table(as.data.frame(dataList))
    # Convert table to data frame with single row
    varNames <- names(chunkTable)
    varValues <- as.vector(chunkTable)
    dim(varValues) <- c(1, length(varNames))
    chunkDF <- as.data.frame(varValues)
    names(chunkDF) <- varNames
    # Return the data frame
    return( chunkDF )
  }

  rxDataStep( inData = inDataSource, outFile = iroDataSource,
    varsToKeep = varsToKeep,
    blocksPerRead = blocksPerRead,
    transformFunc = ProcessChunk,
    reportProgress = 0, overwrite = TRUE)

  AggregateResults <- function()
  {
    iroResults <- rxDataStep(iroDataSource)
    return(colSums(iroResults))
  }

  return(AggregateResults())
}
```

Note that the `blocksPerRead` argument is ignored if this script runs locally using R Client. Since Microsoft R Client can only process datasets that fit into the available memory, chunking is not supported in R Client. When run locally with R Client, all data must be read into memory. You can work around this limitation when you push the compute context to a [Machine Learning Server instance](#).

To test the function, use the sample data `AirlineDemoSmall.xddf` file with a local compute context. For more information, see the tutorial in [Practice data import and exploration](#).

We'll call our new `chunkTable` function, processing 1 block at a time so we can take a look at the intermediate results:

```
inDataSource <- file.path(rxGetOption("sampleDataDir"),
  "AirlineDemoSmall.xdf")
iroDataSource <- "iroFile.xdf"
chunkOut <- chunkTable(inDataSource = inDataSource,
  iroDataSource = iroDataSource, varsToKeep="DayOfWeek")
chunkOut
```

You will see the following results:

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
97975	77725	78875	81304	82987	86159	94975

To see the intermediate results, we can read the data set into memory:

```
rxDataStep(iroDataSource)

Monday Tuesday Wednesday Thursday Friday Saturday Sunday
1 33137 27267 27942 28141 28184 25646 29683
2 32407 25607 25915 26106 26211 29950 33804
3 32431 24851 25018 27057 28592 30563 31488
```

And to delete the intermediate results file:

```
file.remove(iroDataSource)
```

Next steps

This article provides an example of a simple chunking algorithm for tabulating data using `rxDataStep`.

In practice, you might want to use the `rxCrossTabs` or `rxCube` functions. For more information, see [rxCrossTabs](#) and [rxCube](#), respectively.

See Also

- [Machine Learning Server](#)
- [How-to guides in Machine Learning Server](#)
- [RevoScaleR Functions](#)
- [Analyze large data with ScaleR](#)
- [Census data example for analyzing large data](#)
- [Loan data example for analyzing large data](#)

Writing custom analyses for large data sets in RevoScaleR

7/12/2022 • 2 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

This article explains how to use RevoScaleR to get data one chunk at a time, and then perform a custom analysis. There are four basic steps in the analysis:

1. Initialize results
2. Process data, a chunk at a time
3. Update results, after processing each chunk
4. Final processing of results

Prerequisites

Sample data is the airline flight delay dataset. For information on how to obtain this data, see [Sample data for RevoScaleR and revoscalepy](#).

Tutorial walkthrough: Custom chunk operations

For illustrative purposes, suppose that we want to compute the average arrival delay for flights that leave in the morning, afternoon, and evening. For each chunk of data, we need to compute the sum of arrival delay for each of the three time intervals, as well as the counts for each interval. We will accumulate these results in a list of “transformObjects” containing the six values. At the end after processing all the data, we will divide the accumulated totals by the accumulated counts to compute the averages.

Most of the work takes place within a transformation function, which processes the data and updates the results for each chunk of data that is read in. We use the .rxGet and .rxSet functions to store information from one pass of the data to the next. Because we are processing data and not creating newly transformed variables, we return NULL from the function:

```

# Writing Your Own Analyses for Large Data Sets

ProcessAndUpdateData <- function( data )
{
  # Process Data
  notMissing <- !is.na(data$ArrDelay)
  morning <- data$CRSDepTime >= 6 & data$CRSDepTime < 12 & notMissing
  afternoon <- data$CRSDepTime >= 12 & data$CRSDepTime < 17 & notMissing
  evening <- data$CRSDepTime >= 17 & data$CRSDepTime < 23 & notMissing
  mornArr <- sum(data$ArrDelay[morning], na.rm = TRUE)
  mornCounts <- sum(morning, na.rm = TRUE)
  afterArr <- sum(data$ArrDelay[afternoon], na.rm = TRUE)
  afterCounts <- sum(afternoon, na.rm = TRUE)
  evenArr <- sum(data$ArrDelay[evening], na.rm = TRUE)
  evenCounts <- sum(evening, na.rm = TRUE)

  # Update Results
  .rxSet("toMornArr", mornArr + .rxGet("toMornArr"))
  .rxSet("toMornCounts", mornCounts + .rxGet("toMornCounts"))
  .rxSet("toAfterArr", afterArr + .rxGet("toAfterArr"))
  .rxSet("toAfterCounts", afterCounts + .rxGet("toAfterCounts"))
  .rxSet("toEvenArr", evenArr + .rxGet("toEvenArr"))
  .rxSet("toEvenCounts", evenCounts + .rxGet("toEvenCounts"))

  return( NULL )
}

```

Our transformation object values are initialized in a list passed into rxDataStep. We also use the argument *returnTransformObjects* to indicate that we want updated values of the *transformObjects* returned rather than a transformed data set:

```

totalRes <- rxDataStep( inData = airData , returnTransformObjects = TRUE,
  transformObjects =
  list(toMornArr = 0, toAfterArr = 0, toEvenArr = 0,
       toMornCounts = 0, toAfterCounts = 0, toEvenCounts = 0),
  transformFunc = ProcessAndUpdateData,
  transformVars = c("ArrDelay", "CRSDepTime"))

```

The rxDataStep function will automatically chunk through the data for us. All we need to do is process the final results:

```

FinalizeResults <- function(totalRes)
{
  return(data.frame(
    AveMorningDelay = totalRes$toMornArr / totalRes$toMornCounts,
    AveAfternoonDelay = totalRes$toAfterArr / totalRes$toAfterCounts,
    AveEveningDelay = totalRes$toEvenArr / totalRes$toEvenCounts))
}
FinalizeResults(totalRes)

```

The calculated results are:

	AveMorningDelay	AveAfternoonDelay	AveEveningDelay
1	6.146039	13.66912	16.71271

In this case we can check our results by using the *rowSelection* argument in *rxSummary*:

```
rxSummary(~ArrDelay, data = "airExample.xdf",
rowSelection = CRSDepTime >= 6 & CRSDepTime < 12 & !is.na(ArrDelay))
Call:
rxSummary(formula = ~ArrDelay, data = "airExample.xdf",
rowSelection = CRSDepTime >= 6 & CRSDepTime < 12 & !is.na(ArrDelay))

Summary Statistics Results for: ~ArrDelay
File name:
C:\\YourOutputPath\\airExample.xdf
Number of valid observations: 234403

Name      Mean      StdDev  Min Max  ValidObs MissingObs
ArrDelay  6.146039 35.4734 -85 1490 234403    0
```

See Also

- [Machine Learning Server](#)
- [How-to guides in Machine Learning Server](#)
- [RevoScaleR Functions](#)

Converting RevoScaleR Model Objects for Use in PMML

7/12/2022 • 3 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

The objects returned by RevoScaleR predictive analytics functions have similarities to objects returned by the predictive analytics functions provided by base and recommended packages in R. A key difference between these two types of model objects is that RevoScaleR model objects typically do not contain any components that have the same length as the number of rows in the original data set. For example, if you use base R's `lm` function to estimate a linear model, the result object contains not only all of the data used to estimate the model, but components such as residuals and `fitted.values` that contain values corresponding to every observation in the data set. For estimating models using big data, this is not appropriate.

However, there is overlap in the model object components that can be very useful when working with other packages. For example, the `pmm` package generates PMML (Predictive Model Markup Language) code for a number of R model types. PMML is an XML-base language that allows users to share models between PMML-compliant applications. For more information about PMML, visit <http://www.dmg.org>.

RevoScaleR provides a set of coercion methods to convert a RevoScaleR model object to a standard R model object: `as.lm`, `as.glm`, `as.rpart`, and `as.kmeans`. As suggested above, these coerced model objects do not have all of the information available in a standard R model object, but do contain information about the fitted model that is similar to standard R.

For example, let's create an `rxLinMod` object by estimating a linear regression on a very large dataset: the airline data with almost 150 million observations.

```
# Use in PMML

bigDataDir <- "C:/MRS/Data"
bigAirData <- file.path(bigDataDir, "AirOnTime87to12/AirOnTime87to12.xdf")

# About 150 million observations
rxLinModObj <- rxLinMod(ArrDelay~Year + DayOfWeek, data = bigAirData,
  blocksPerRead = 10)
```

The `blocksPerRead` argument is ignored if run locally using R Client. [Learn more...](#)

Next, the R `pmm` package should be downloaded and installed from CRAN. After you have done so, generate the RevoScaleR `rxLinMod` object, and use the `as.lm` method when generating the PMML output:

```
library(pmm)
pmm(as.lm(rxLinModObj))
```

The generated output looks as follows:

```

> pmml(as.lm(rxLinModObj))
<PMML version="4.1" xmlns="http://www.dmg.org/PMML-4_1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.dmg.org/PMML-4_1 http://www.dmg.org/v4-1/pmmml-4-1.xsd">
  <Header copyright="Copyright (c) 2013 sue" description="Linear Regression Model">
    <Extension name="user" value="yourname" extender="Rattle/PMML"/>
  <Application name="Rattle/PMML" version="1.3"/>
  <Timestamp>2013-09-05 11:39:28</Timestamp>
  </Header>
  <DataDictionary numberOfFields="3">
    <DataField name="ArrDelay" optype="continuous" dataType="double"/>
    <DataField name="Year" optype="continuous" dataType="double"/>
    <DataField name="DayOfWeek" optype="categorical" dataType="string">
      <Value value="Mon"/>
      <Value value="Tues"/>
      <Value value="Wed"/>
      <Value value="Thur"/>
      <Value value="Fri"/>
      <Value value="Sat"/>
      <Value value="Sun"/>
    </DataField>
  </DataDictionary>
  <RegressionModel modelName="Linear_Regression_Model" functionName="regression" algorithmName="least squares" targetFieldName="ArrDelay">
    <MiningSchema>
      <MiningField name="ArrDelay" usageType="predicted"/>
      <MiningField name="Year" usageType="active"/>
      <MiningField name="DayOfWeek" usageType="active"/>
    </MiningSchema>
    <Output>
      <OutputField name="Predicted_ArrDelay" feature="predictedValue"/>
    </Output>
    <RegressionTable intercept="112.856953212332">
      <NumericPredictor name="Year" exponent="1" coefficient="-0.0533799688606163"/>
      <CategoricalPredictor name="DayOfWeek" value="Mon" coefficient="0.303666227836942"/>
      <CategoricalPredictor name="DayOfWeek" value="Tues" coefficient="-0.593700918634743"/>
      <CategoricalPredictor name="DayOfWeek" value="Wed" coefficient="0.473693749859764"/>
      <CategoricalPredictor name="DayOfWeek" value="Thur" coefficient="2.3393087933048"/>
      <CategoricalPredictor name="DayOfWeek" value="Fri" coefficient="2.91671831592945"/>
      <CategoricalPredictor name="DayOfWeek" value="Sat" coefficient="-2.31671307739631"/>
      <CategoricalPredictor name="DayOfWeek" value="Sun" coefficient="0"/>
    </RegressionTable>
  </RegressionModel>
</PMML>
```

Similarly, rxLogit and rxGlm functions have as.glm methods:

```

form <- case ~ age + parity + spontaneous + induced
rxLogitObj <- rxLogit(form, data = infert)
pmml(as.glm(rxLogitObj))

rxGlmObj <- rxGlm(form, data=infert, family = binomial())
pmml(as.glm(rxGlmObj))
```

RevoScaleR's *rxKmeans* objects can be coerced to *kmeans* objects:

```

set.seed(17)
irow <- unique(sample.int(nrow(women), 4L,
  replace = TRUE))[seq(2)]
centers <- women[irow,, drop = FALSE]
rxKmeansObj <- rxKmeans(~height + weight, data = women,
  centers = centers)
pmml(as.kmeans(rxKmeansObj))
```

And RevoScaleR's *rxDTTree* objects can be coerced to *rpart* objects. (Note that *rpart* is a recommended package that you may need to load.)

```
library(rpart)
method <- "class"
form <- Kyphosis ~ Number + Start
parms <- list(prior = c(0.8, 0.2), loss = c(0, 2, 3, 0),
  split = "gini")
control <- rpart.control(minsplit = 5, minbucket = 2, cp = 0.01,
  maxdepth = 10, maxcompete = 4, maxsurrogate = 5,
  usesurrogate = 2, surrogatestyle = 0, xval = 0)
cost <- 1 + seq(length(attr(terms(form)), "term.labels")))

rxDTTreeObj <- rxDTTree(formula = Kyphosis ~ Number + Start,
  data = kyphosis, method = method, parms = parms,
  control = control, cost = cost)
pmml(as.rpart(rxDTTreeObj))
```

Managing threads in RevoScaleR

7/12/2022 • 2 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

RevoScaleR provides functions for managing the thread pool used for parallel execution.

On Windows, thread pool management is enabled and should not be turned off.

On Linux, thread pool management is turned off by default to avoid interfering with how Unix systems fork processes. Process forking is only available on Unix-based systems. You can turn the thread pool on if you do not fork your R process. RevoScaleR provides an interface to activate the thread pool:

```
rxSetEnableThreadPool(TRUE)
```

Similarly, the thread pool may be disabled as follows:

```
rxSetEnableThreadPool(FALSE)
```

If you want to ensure that the RevoScaleR thread pool is always enabled on Linux, you can add the preceding command to a *.First* function defined in either your own Rprofile startup file, or the system Rprofile.site file. For example, you can add the following lines after the closing right parenthesis of the existing Rprofile.site file:

```
.First <- function()
{
  .First.sys()
  invisible(rxSetEnableThreadPool(TRUE))
}
```

The *.First.sys* function is normally run after all other initialization is complete, including the evaluation of the *.First* function. We need the call to *rxSetEnableThreadPool* to occur after RevoScaleR is loaded. That is done by *.First.sys*, so we call *.First.sys* first.

See Also

- [Machine Learning Server](#)
- [How-to guides in Machine Learning Server](#)
- [RevoScaleR Functions](#)

Using foreach and iterators for manual parallel execution

7/12/2022 • 15 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Although RevoScaleR performs parallel execution automatically, you can manage parallel execution yourself using the open-source [foreach package](#). This package provides a looping structure for R script. When you need to loop through repeated operations, and you have multiple processors or nodes to work with, you can use `foreach` in your script to execute a `for loop` in parallel. Developed by Microsoft, `foreach` is an open-source package that is bundled with Machine Learning Server but is also available on the Comprehensive R Archive Network, CRAN.

TIP

One common approach to parallelization is to see if the iterations within a loop can be performed independently, and if so, you can try to run the iterations concurrently rather than sequentially.

About the foreach package

The `foreach` package is a set of tools that allow you to run virtually anything that can be expressed as a for-loop as a set of parallel tasks. One scenario is to run multiple simulations in parallel. As a simple example, consider the case of simulating 10000 coin flips, which can be done by sampling with replacement from the vector `c(H, T)`. To run this simulation 10 times sequentially, use `foreach` with the `%do%` operator:

```
> library(foreach)
> foreach(i = 1:10) %do% sample(c("H", "T"), 10000, replace = TRUE)
```

Comparing the `foreach` output with that of a similar `for` loop shows one obvious difference: `foreach` returns a list containing the value returned by each computation. A `for` loop, by contrast, returns only the value of its last computation, and relies on user-defined side effects to do its work.

We can parallelize the operation immediately by replacing `%do%` with `%dopar%`:

```
> foreach(i = 1:10) %dopar% sample(c("H", "T"), 10000, replace = TRUE)
```

However, if we run this example, we see the following warning:

```
Warning message:
executing %dopar% sequentially: no parallel backend registered
```

To actually run in parallel, we need to have a “parallel backend” for `foreach`. Parallel backends are discussed in the next section.

Parallel Backends

In order for loops coded with `foreach` to run in parallel, you must register a parallel backend to manage the execution of the loop. Any type of mechanism for running code in parallel could potentially have a parallel backend written for it. Currently, Machine Learning Server includes the `doParallel` backend; this uses the `parallel` package of R 2.14.0 or later to run jobs in parallel, using either of the component parallelization methods incorporated into the `parallel` package: SNOW-like functionality using socket connections, or multicore-like functionality using forking (on Linux only).

The `doParallel` package is a parallel backend for `foreach` that is intended for parallel processing on a single computer with multiple cores or processors.

Additional parallel backends are available from CRAN:

- `doMPI` for use with the `Rmpi` package
- `doRedis` for use with the `rredis` package
- `doMC` provides access to the multicore functionality of the `parallel` package
- `doSNOW` for use with the now superseded `SNOW` package.

To use a parallel backend, you must first register it. Once a parallel backend is registered, calls to `%dopar%` run in parallel using the mechanisms provided by the parallel backend. However, the details of registering the parallel backends differ, so we consider them separately.

Using the `doParallel` parallel backend

The `parallel` package of R 2.14.0 and later combines elements of `snow` and `multicore`; `doParallel` similarly combines elements of both `doSNOW` and `doMC`. You can register `doParallel` with a cluster, as with `doSNOW`, or with a number of cores, as with `doMC`. For example, here we create a cluster and register it:

```
> library(doParallel)
> cl <- makeCluster(4)
> registerDoParallel(cl)
```

Once you've registered the parallel backend, you're ready to run `foreach` code in parallel. For example, to see how long it takes to run 10,000 bootstrap iterations in parallel on all available cores, you can run the following code:

```
> x <- iris[which(iris[,5] != "setosa"), c(1, 5)]
> trials <- 10000
> ptime <- system.time({
+   r <- foreach(icount(trials), .combine = cbind) %dopar%
+     ind <- sample(100, 100, replace = TRUE)
+     result1 <- glm(x[ind, 2] ~ x[ind, 1], family = binomial(logit))
+     coefficients(result1)
+ }
+ )[3]
> ptime
```

Getting information about the parallel backend

To find out how many workers `foreach` is going to use, you can use the `getDoParWorkers` function:

```
> getDoParWorkers()
```

This is a useful sanity check that you're actually running in parallel. If you haven't registered a parallel backend,

or if your machine only has one core, `getDoParWorkers` will return 1. In either case, don't expect a speed improvement.

The `getDoParWorkers` function is also useful when you want the number of tasks to be equal to the number of workers. You may want to pass this value to an iterator constructor, for example.

You can also get the name and version of the currently registered backend:

```
> getDoParName()
> getDoParVersion()
```

Nesting Calls to `foreach`

An important feature of `foreach` is nesting operator `%::%`. Like the `%do%` and `%dopar%` operators, it is a binary operator, but it operates on two `foreach` objects. It also returns a `foreach` object, which is essentially a special merger of its operands.

Let's say that we want to perform a Monte Carlo simulation using a function called `sim`. The `sim` function takes two arguments, and we want to call it with all combinations of the values that are stored in the vectors `avec` and `bvec`. The following doubly-nested `for` loop does that. For testing purposes, the `sim` function is defined to return $\$10 a + b\$$ (although an operation this trivial is not worth executing in parallel):

```
sim <- function(a, b) 10 * a + b
avec <- 1:2
bvec <- 1:4

x <- matrix(0, length(avec), length(bvec))
for (j in 1:length(bvec)) {
    for (i in 1:length(avec)) {
        x[i, j] <- sim(avec[i], bvec[j])
    }
}
x
```

In this case, it makes sense to store the results in a matrix, so we create one of the proper size called `x`, and assign the return value of `sim` to the appropriate element of `x` each time through the inner loop.

When using `foreach`, we don't create a matrix and assign values into it. Instead, the inner loop returns the columns of the result matrix as vectors, which are combined in the outer loop into a matrix. Here's how to do that using the `%::%` operator:

```
x <-
  foreach(b = bvec, .combine = 'cbind') %::%
    foreach(a = avec, .combine = 'c') %do% {
      sim(a, b)
    }
x
```

This is structured very much like the nested `for` loop. The outer `foreach` is iterating over the values in "bvec", passing them to the inner `foreach`, which iterates over the values in "avec" for each value of "bvec". Thus, the "sim" function is called in the same way in both cases. The code is slightly cleaner in this version, and has the advantage of being easily parallelized.

When parallelizing nested `for` loops, there is always a question of which loop to parallelize. The standard advice is to parallelize the outer loop. This results in larger individual tasks, and larger tasks can often be performed more efficiently than smaller tasks. However, if the outer loop doesn't have many iterations and the

tasks are already large, parallelizing the outer loop results in a small number of huge tasks, which may not allow you to use all of your processors, and can also result in load-balancing problems. You could parallelize an inner loop instead, but that could be inefficient because you're repeatedly waiting for all the results to be returned every time through the outer loop. And if the tasks and number of iterations vary in size, then it's really hard to know which loop to parallelize.

But in our Monte Carlo example, all of the tasks are completely independent of each other, and so they can all be executed in parallel. You really want to think of the loops as specifying a single stream of tasks. You just need to be careful to process all of the results correctly, depending on which iteration of the inner loop they came from.

That is exactly what the `%:%` operator does: it turns multiple **foreach** loops into a single loop. That is why there is only one `%do%` operator in the example above. And when we parallelize that nested **foreach** loop by changing the `%do%` into a `%dopar%`, we are creating a single stream of tasks that can all be executed in parallel:

```
x <-  
  foreach(b = bvec, .combine = 'cbind') %:%  
    foreach(a = avec, .combine = 'c') %dopar% {  
      sim(a, b)  
    }  
x
```

Of course, we'll actually only run as many tasks in parallel as we have processors, but the parallel backend takes care of all that. The point is that the `%:%` operator makes it easy to specify the stream of tasks to be executed, and the `.combine` argument to **foreach** allows us to specify how the results should be processed. The backend handles executing the tasks in parallel.

For more on nested **foreach** calls, see the vignette "Nesting **foreach** Loops" in the **foreach** package.

Using Iterators

An *iterator* is a special type of object that generalizes the notion of a looping variable. When passed as an argument to a function that knows what to do with it, the iterator supplies a sequence of values. The iterator also maintains information about its state, in particular its current index.

The `iterators` package includes a number of functions for creating iterators, the simplest of which is `iter`, which takes virtually any R object and turns it into an iterator object. The simplest function that operates on iterators is the `nextElem` function, which when given an iterator, returns the next value of the iterator. For example, here we create an iterator object from the sequence 1 to 10, and then use `nextElem` to iterate through the values:

```
> i1 <- iter(1:10)  
> nextElem(i1)  
[1] 1  
> nextElem(i1)  
[2] 2
```

You can create iterators from matrices and data frames, using the `by` argument to specify whether to iterate by row or column:

```

> istate <- iter(state.x77, by = 'row')
> nextElem(istate)
    Population Income Illiteracy Life Exp Murder HS Grad Frost Area
  Alabama      3615     3624       2.1    69.05   15.1    41.3    20 50708
> nextElem(istate)
    Population Income Illiteracy Life Exp Murder HS Grad Frost Area
  Alaska       365     6315       1.5    69.31   11.3    66.7    152 566432

```

Iterators can also be created from functions, in which case the iterator can be an endless source of values:

```

> ifun <- iter(function() sample(0:9, 4, replace=TRUE))
> nextElem(ifun)
[1] 9 5 2 8
> nextElem(ifun)
[1] 3 4 2 2

```

For practical applications, iterators can be paired with **foreach** to obtain parallel results quite easily:

```

> x <- matrix(rnorm(1000000), ncol = 1000)
> itx <- iter(x, by = 'row')
> foreach(i = itx, .combine = c) %dopar% mean(i)

```

Some Special Iterators

The notion of an iterator is new to R, but should be familiar to users of languages such as Python. The `iterators` package includes a number of special functions that generate iterators for some common scenarios. For example, the `irnorm` function creates an iterator for which each value is drawn from a specified random normal distribution:

```

> library(iterators)
> itrn <- irnorm(1, count = 10)
> nextElem(itrn)
[1] 0.6300053
> nextElem(itrn)
[1] 1.242886

```

Similarly, the `irunif`, `irbinom`, and `irpois` functions create iterators which draw their values from uniform, binomial, and Poisson distributions, respectively. (These functions use the standard R distribution functions to generate random numbers, and these are not necessarily useful in a distributed or parallel environment. When using random numbers with **foreach**, we recommend using the `doRNG` package to ensure independent random number streams on each worker.)

We can then use these functions just as we used `irnorm`:

```

> itru <- irunif(1, count = 10)
> nextElem(itru)
[1] 0.4960539
> nextElem(itru)
[1] 0.4071111

```

The `icount` function returns an iterator that counts starting from one:

```
> it <- icount(3)
> nextElem(it)
[1] 1
> nextElem(it)
[1] 2
> nextElem(it)
[1] 3
```

Writing Iterators

There will be times when you need an iterator that isn't provided by the `iterators` package. That is when you need to write your own custom iterator.

Basically, an iterator is an S3 object whose base class is `iter`, and has `iter` and `nextElem` methods. The purpose of the `iter` method is to return an iterator for the specified object. For iterators, that usually just means returning itself, which seems odd at first. But the `iter` method can be defined for other objects that don't define a `nextElem` method. We call those objects *iterables*, meaning that you can iterate over them. The `iterators` package defines `iter` methods for vectors, lists, matrices, and data frames, making those objects iterables. By defining an `iter` method for iterators, they can be used in the same context as an iterable, which can be convenient. For example, the `foreach` function takes iterables as arguments. It calls the `iter` method on those arguments in order to create iterators for them. By defining the `iter` method for all iterators, we can pass iterators to `foreach` that we created using any method we choose. Thus, we can pass vectors, lists, or iterators to `foreach`, and they are all processed by `foreach` in exactly the same way.

The `iterators` package comes with an `iter` method defined for the `iter` class that simply returns itself. That is usually all that is needed for an iterator. However, if you want to create an iterator for some existing class, you can do that by writing an `iter` method that returns an appropriate iterator. That will allow you to pass an instance of your class to `foreach`, which will automatically convert it into an iterator. The alternative is to write your own function that takes arbitrary arguments, and returns an iterator. You can choose whichever method is most natural.

The most important method required for iterators is `nextElem`. This simply returns the next value, or throws an error. Calling the `stop` function with the string `StopIteration` indicates that there are no more values available in the iterator.

In most cases, you don't actually need to write the `iter` and `nextElem` methods; you can inherit them. By inheriting from the class `abstractiter`, you can use the following methods as the basis of your own iterators:

```
> iterators:::iter.iter
function (obj, ...)
{
  obj
}
<environment: namespace:iterators>
> iterators:::nextElem.abstractiter
function (obj, ...)
{
  obj$nextElem()
}
<environment: namespace:iterators>
```

The following function creates a simple iterator that uses these two methods:

```

iforever <- function(x) {
  nextEl <- function() x
  obj <- list(nextElem = nextEl)
  class(obj) <- c('iforever', 'abstractiter', 'iter')
  obj
}

```

Note that we called the internal function `nextEl` rather than `nextElem` to avoid masking the standard `nextElem` generic function. That causes problems when you want your iterator to call the `nextElem` method of another iterator, which can be quite useful.

We create an instance of this iterator by calling the `iforever` function, and then use it by calling the `nextElem` method on the resulting object:

```

it <- iforever(42)
nextElem(it)
nextElem(it)

```

Notice that it doesn't make sense to implement this iterator by defining a new `iter` method, since there is no natural iterable on which to dispatch. The only argument that we need is the object for the iterator to return, which can be of any type. Instead, we implement this iterator by defining a normal function that returns the iterator.

This iterator is quite simple to implement, and possibly even useful, but exercise caution if you use it. Passing it to `foreach` will result in an infinite loop unless you pair it with a finite iterator. Similarly, never pass this iterator to `as.list` without the `n` argument.

The iterator returned by `iforever` is a list that has a single element named `nextElem`, whose value is a function that returns the value of `x`. Because we are subclassing `abstractiter`, we inherit a `nextElem` method that will call this function, and because we are subclassing `iter`, we inherit an `iter` method that will return itself.

Of course, the reason this iterator is so simple is because it doesn't contain any state. Most iterators need to contain some state, or it will be difficult to make it return different values and eventually stop. Managing the state is usually the real trick to writing iterators.

As an example of writing a stateful iterator, let's modify the previous iterator to put a limit on the number of values that it returns. We'll call the new function `irep`, and give it another argument called `times`:

```

irep <- function(x, times) {
  nextEl <- function() {
    if (times > 0) {
      times <- times - 1
    }
    else {
      stop('StopIteration')
    }
    x
  }
  obj <- list(nextElem = nextEl)
  class(obj) <- c('irep', 'abstractiter', 'iter')
  obj
}

```

Now let's try it out:

```
it <- irep(7, 6)
unlist(as.list(it))
```

The real difference between `iforever` and `irep` is in the function that gets called by the `nextElem` method. This function not only accesses the values of the variables `x` and `times`, but it also modifies the value of `times`. This is accomplished by means of the `<--` operator, and the magic of lexical scoping. Technically, this kind of function is called a *closure*, and is a somewhat advanced feature of R. The important thing to remember is that `nextEl` is able to get the value of variables that were passed as arguments to `irep`, and it can modify those values using the `<--` operator. These are *not* global variables: they are defined in the enclosing environment of the `nextEl` function. You can create as many iterators as you want using the `irep` function, and they will all work as expected without conflicts.

Note that this iterator only uses the arguments to `irep` to store its state. If any other state variables are needed, they can be defined anywhere inside the `irep` function.

More examples of writing iterators can be found in the vignette "Writing Custom Iterators" in the `iterators` package.

Parallel execution using doRSR for script containing RevoScaleR and foreach constructs

7/12/2022 • 2 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Machine Learning Server includes the open-source [foreach package](#) in case you need to substitute the built-in parallel execution methodology of RevoScaleR with a custom implementation.

To execute code that leverages `foreach`, you will need a parallel backend engine, similar to [NetWorkSpaces](#), [snow](#), and [rmpi](#). For integration with script leveraging RevoScaleR, you can use the [doRSR](#) package. The [doRSR](#) package is a parallel backend for RevoScaleR, built on top of [rxExec](#), and included with all RevoScaleR distributions.

Example script using doRSR

To get started with [doRSR](#), load the package and register the backend:

```
library(doRSR)
registerDoRSR()
```

The [doRSR](#) package uses your current compute context to determine how to run your job. In most cases, the job is run via `rxExec`, sequentially in the local compute context and in parallel in a distributed compute context. In the special case where you are in the local compute context and have set `rxOptions(useDoParallel=TRUE)`, [doRSR](#) will pass your `foreach` jobs to the [doParallel](#) package for execution in parallel using multiple cores on your machine.

A simple example is this one from the `foreach` help file:

```
foreach(i=1:3) %dopar% sqrt(i)
```

This returns, as expected, a list containing the square roots of 1, 2, and 3:

```
[[1]]
[1] 1

[[2]]
[1] 1.414214

[[3]]
[1] 1.732051
```

Another example is what the help file reports as a “simple (and inefficient) parallel matrix multiply”:

```

a <- matrix(1:16, 4, 4)
b <- t(a)
foreach(b=iter(b, by='col'), .combine=cbind) %dopar%
  (a %*% b)

```

This returns the multiplied matrix:

```

[,1] [,2] [,3] [,4]
[1,] 276 304 332 360
[2,] 304 336 368 400
[3,] 332 368 404 440
[4,] 360 400 440 480

```

Use case: simulation

In [Running jobs in parallel](#), we introduced the simulation function `playDice` to simulate a single game of dice rolling, using `rxExec` to play 10000 games. You can do the same thing with `foreach`:

```

z1 <- foreach(i=1:10000, .options.rsr=list(chunkSize=2000)) %dopar% playDice()
table(unlist(z1))

Loss  Win
5079 4921

```

Although outcomes are equivalent in both approaches, the `rxExec` approach is several times faster because you can call it directly.

Use case: naïve parallelization of the standard R kmeans function

Also in the [previous article](#), we created a function `kmeansRSR` to perform a naïve parallelization of the standard R `kmeans` function. We can do the same thing with `foreach` directly as follows:

```

kMeansForeach <- function(x, centers=5, iter.max=10, nstart=1)
{
  numTimes <- 20
  results <- foreach(i=1:numTimes) %dopar% kmeans(x=x, centers=centers, iter.max=iter.max,
    nstart=nstart)
  best <- 1
  bestSS <- sum(results[[1]]$withinss)
  for (j in 1:numTimes)
  {
    jSS <- sum(results[[j]]$withinss)
    if (bestSS > jSS)
    {
      best <- j
      bestSS <- jSS
    }
  }
  results[[best]]
}

```

Recall that the idea was to run a specified number of `kmeans` fits, then find the best set of results, where "best" is the result with the lowest within-group sum of squares. We can run this function as follows:

```

x <- matrix(rnorm(250000), nrow = 5000, ncol = 50)
kMeansForeach(x, 10, 35, 20)

```

How to use the RevoPemaR library in Machine Learning Server

7/12/2022 • 13 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

The **RevoPemaR** package provides a framework for writing custom parallel external memory algorithms in R, making use of the R reference classes introduced by John Chambers in R 2.12.

Parallel External Memory Algorithms (PEMA) are algorithms that eliminate in-memory data storage requirements, allowing data to be processed in chunks, in parallel, possibly on different nodes of a cluster. The results are then combined and processed at the end (or at the end of each iteration). The **RevoPemaR** package is used for writing custom PEMA algorithms in R.

The custom PEMA functions created using the **RevoPemaR** framework are appropriate for small and large datasets, but are particularly useful in three common situations:

1. To analyze data sets that are too large to fit in memory
2. To create scalable data analysis routines that can be developed locally with smaller data sets, then deployed to larger data
3. To perform computations distributed over nodes in a cluster

Installation

The **RevoPemaR** package is included with Microsoft Machine Learning Server and R Client (which also contains the **RevoScaleR** package).

The **RevoPemaR** package can also be installed with a standard download of [Microsoft R Open \(MRO\)](#).

About R Reference Classes

The PEMA classes used in **RevoPemaR** are based on R Reference Classes. We include a brief overview of some tips for using R Reference Classes here, before moving to the specifics of the PEMA classes.

R Reference Class objects are created using a generator function. This function has four important pieces of information:

- *Name of the class.*
- Inheritance or *superclasses* of the class. Fields and methods of parent reference classes are inherited.
- *Fields* or member variables. These fields are accessed by reference (as in C++ or Java), so values of the fields for an object of this class can change.
- *Methods* that can be invoked by objects of this class.

When working with reference class, keep these tips in mind:

- Reference class generators are created using *setRefClass*. For the PEMA classes, we use a wrapper for that function, *setPemaClass*.

- Field values are changed within methods using the non-local assignment operator (`<<-`)
- The reference class object can be accessed in the methods using `.self`
- The parent method can be accessed using `.callSuper`
- Use the `usingMethods` call to declare that a method is used by another method.
- The code for a method can be displayed using an instantiated reference class object, for example, `myRefClassObj$initialize`.
- Methods are documented internally with an initial line of text, rather than in a .Rd file. This information is accessed using the `$help` method for the generator function.

Tutorial introduction to RevoPemaR

This section contains an overview of a simple example of estimating the mean of a variable using the **RevoPemaR** framework. The key step is in creating a *PemaMean* reference class generator function that provides the fields and methods for computing the mean using a parallel external memory algorithm. This includes creating methods to compute the sum and number of observations for each chunk of data, to update these intermediate results, and at the end to use the intermediate results to compute the mean.

Using `setPemaClass` to Create a Class Generator

Start by making sure that the **RevoPemaR** package is loaded:

```
library(RevoPemaR)
```

To create a PEMA class generator function, use the `setPemaClass` function. It is a wrapper function for `setRefClass`. As with `setRefClass`, we specify four basic pieces of information when using `setPemaClass`: the class name, the superclasses, the fields, and the methods. The structure looks something like this:

```
PemaMean <- setPemaClass(
  Class = "PemaMean",
  contains = "PemaBaseClass",
  fields = list( # To be written
    ),
  methods = list( # To be written
    ))
```

The *Class* is the class name of your choice. The *contains* argument must specify *PemaBaseClass* or a child class of *PemaBaseClass*. The specification of fields and methods follows.

Specifying the fields for *PemaMean*

The fields or member variables of our class represent all of the variables we need in order to compute and store our intermediate and final results. Here are the fields we use for our “means” computation:

```
fields = list(
  sum = "numeric",
  totalObs = "numeric",
  totalValidObs = "numeric",
  mean = "numeric",
  varName = "character"
),
```

An Overview of the *methods* for *PemaMean*

There are five methods we specify for *PemaMean*. These methods are all in the *PemaBaseClass*, and need to be overloaded for any custom analysis.

- *initialize*: initializes field values.

- *processData*: processes a chunk of data and updates field values
- *updateResults*: updates the field values of a PEMA class object from another
- *processResults*: computes the final results from the final intermediate results
- *getVarsToUse*: the names of the variables in the dataset used for analysis

The *initialize* method

The primary use of the *initialize* method is to initialize field values. The one field that is initialized with user input in this example is the name of the variable to use in the computations, *varName*. Use of the ellipses in the function signature allows for initialization values to be passed up to the parent class using *.callSuper*, the first action in the *initialize* method after the documentation. Here is the beginning of our methods listing:

```
methods = list(
  initialize = function(varName = "", ...)
{
  'sum, totalValidObs, and mean are all initialized to 0'
  # callSuper calls the initialize method of the parent class
  callSuper(...)
```

The *pemaSetClass* function also provides additional functionality used in the *initialize* method to ensure that all of the methods of the class and its parent classes are included when an object is serialized. This is critical for distributed computing. To use this functionality, add the following to the *initialize* method:

```
usingMethods(.pemaMethods)
```

(If you do not want to use this functionality you can omit this line and set *includeMethods* to FALSE in *setPemaClass*.)

Now we finish the field initialization, setting the *varName* field to the input value and setting the starting values for our computations to 0, remembering to use the double-arrow non-local assignment operator to set field values:

```
varName <- varName
sum <- 0
totalObs <- 0
  totalValidObs <- 0
  mean <- 0
},
```

The *processData* method

The *processData* method is the core of an external memory algorithm. It processes a chunk of data and computes intermediate results, updating the field value(s). It takes as an argument a rectangular list of data vectors; typically only the variable(s) of interest is included. In our example code we do not compute the mean within this method; that occurs after we have processed all of the data. Here we compute and update the intermediate results: the sum and number of observations:

```

processData = function(dataList)
{
  'Updates the sum and total observations from
  the current chunk of data.'
  sum <- sum + sum(as.numeric(dataList[[varName]])),
  na.rm = TRUE)

  totalObs <- totalObs + length(dataList[[varName]])

  totalValidObs <- totalValidObs +
    sum(!is.na(dataList[[varName]])))
  invisible(NULL)
},

```

The *updateResults* method

The *updateResults* is the key method used when computations are done in parallel. Consider the following scenario:

1. The master node on a cluster assigns each worker node the task of processing a series of chunks of data.
2. The workers do so in parallel, each with their own instantiation of a reference class object. Each worker calls *processData* for each chunk of data it needs to process. In each call, the values of the fields of its reference class object are updated.
3. Now the master process must collect the information from each of the nodes, and update all of the information in a single reference class object. This is done using the *updateResults* method, which takes as an argument another instance of the reference class. The reference class object from each of the nodes is processed by the master node, resulting in the final intermediate results in the master node's reference class object's fields.

Here is the *updateResults* method for our *PemaMean*:

```

updateResults = function(pemaMeanObj)
{
  'Updates the sum and total observations from
  another PemaMean object.'

  sum <- sum + pemaMeanObj$sum
  totalObs <- totalObs + pemaMeanObj$totalObs
  totalValidObs <- totalValidObs + pemaMeanObj$totalValidObs

  invisible(NULL)
},

```

The *processResults* method

The *processResults* performs any necessary computations to produce the final result from the accumulated intermediate results. In this case, it is simple; we divide the sum by the number of valid observations (assuming we have some):

```

processResults = function()
{
  'Returns the sum divided by the totalValidObs.'
  if (totalValidObs > 0)
  {
    mean <- sum/totalValidObs
  }
  else
  {
    mean <- as.numeric(NA)
  }
  return( mean )
},

```

The `getVarsToUse` method

The `getVarsToUse` method specifies the names of the variables in the dataset that are used in the analysis. Specifying this information can improve performance if reading data from disk.

```

getVarsToUse = function()
{
  'Returns the varName.'
  varName
}
) # End of methods
) # End of class generator

```

Creating and Using a `PemaMean` Reference Class Object

Instantiating and Exploring a `PemaMean` Object

A version of the code in the previous section is contained within the `RevoPemaR` package and exported, so we can directly work with the `PemaMean` generator without first running the code. We can show the names of all the methods, including those that are explicitly overridden by the `PemaMean` class:

```

PemaMean$methods()

[1] ".pemaMethods"                 ".pemaMethods#PemaBaseClass"
[3] "callSuper"                   "compute"
[5] "copy"                         "copyFields"
[7] "createReturnObject"           "export"
[9] "field"                        "finalizeNode"
[11] "getClass"                    "getFieldList"
[13] "getRefClass"                  "getVarsToRead"
[15] "getVarsToUse"                 "getVarsToUse#PemaBaseClass"
[17] "hasConverged"                "import"
[19] "initFields"                  "initialize"
[21] "initialize#PemaBaseClass"    "initIteration"
[23] "outputTrace"                 "processAllData"
[25] "processData"                  "processData#PemaBaseClass"
[27] "processResults"               "processResults#PemaBaseClass"
[29] "setFieldList"                 "show"
[31] "trace"                        "untrace"
[33] "updateResults"                "updateResults#PemaBaseClass"
[35] "usingMethods"

```

Some of the methods (for example, `initIteration`, `getFieldList`) are inherited from the `PemaBaseClass`. Others (for example, `callSuper`, `methods`) are inherited from the base reference class generator.

We can use the `help` method with the generator function to get help on specific methods:

```

PemaMean$help(initialize)

Call:
$initialize(varName = , ...)

sum, totalValidObs, and mean are all initialized to 0

```

Next we generate a default *PemaMean* object, and print out the values of its fields (including those inherited):

```

meanPemaObj <- PemaMean()
meanPemaObj

Reference class object of class "PemaMean"
Reference class object of class "PemaMean" (from the global environment)
Field ".isPemaObject":
[1] TRUE
Field ".isDistributedContext":
[1] FALSE
Field ".hasOutFile":
[1] FALSE
Field ".outFile":
NULL
Field ".append":
[1] "none"
Field ".overwrite":
[1] FALSE
Field ".onlyKeepTransformedData":
[1] FALSE
Field "traceLevel":
[1] 0
Field "iter":
[1] 0
Field "maxIters":
[1] 2000
Field "useRevoScaleR":
[1] TRUE
Field ".dataInMemory":
[1] FALSE
Field ".dataInMemoryPrepared":
[1] FALSE
Field "reportProgress":
[1] 2
Field "sum":
[1] 0
Field "totalObs":
[1] 0
Field "totalValidObs":
[1] 0
Field "mean":
[1] 0
Field "varName":
[1] ""

```

We can also print out the code for a specific method using an instantiated object. For example, the initialize method of the *PemaMean* object in the **RevoPemaR** package is:

```

meanPemaObj$initialize

Class method definition for method initialize()
function (varName = "", ...)
{
  "sum, totalValidObs, and mean are all initialized to 0"
  callSuper(...)
  usingMethods(.pemaMethods)
  varName <- varName
  sum <- 0
  totalObs <- 0
  totalValidObs <- 0
  mean <- 0
}
<environment: 0x000000003148ea40>

Methods used:
".pemaMethods", "callSuper", "usingMethods"

```

Using a *PemaMean* Object with the *pemaCompute* Function

The *pemaCompute* function takes two required arguments: an “analysis” object and a data source object. The analysis object must be generated by *setPemaClass* and inherit (directly or indirectly) from *PemaBaseClass*. The data source object must be either a data frame or a data source object supported by the **RevoScaleR** package if it is available. The ellipses take any additional information used in the *initialize* method.

Let’s compute a mean of some random numbers:

```

set.seed(67)
pemaCompute(pemaObj = meanPemaObj,
            data = data.frame(x = rnorm(1000)), varName = "x")

[1] 0.00504128

```

If we again print the values of the fields of our *meanPemaObj*, we see the updated values:

```

meanPemaObj

Reference class object of class "PemaMean" (from the global environment)
Field ".isPemaObject":
[1] TRUE
Field ".isDistributedContext":
[1] FALSE
Field ".hasOutFile":
[1] FALSE
Field ".outFile":
NULL
Field ".append":
[1] "none"
Field ".overwrite":
[1] FALSE
Field ".onlyKeepTransformedData":
[1] FALSE
Field "traceLevel":
[1] 0
Field "iter":
[1] 1
Field "maxIters":
[1] 2000
Field "useRevoScaleR":
[1] TRUE
Field ".dataInMemory":
[1] FALSE
Field ".dataInMemoryPrepared":
[1] FALSE
Field "reportProgress":
[1] 2
Field "sum":
[1] 5.04128
Field "totalObs":
[1] 1000
Field "totalValidObs":
[1] 1000
Field "mean":
[1] 0.00504128
Field "varName":
[1] "x"

```

By default the *pemaCompute* method reinitializes the *pemaObj*. By setting the *initPema* flag to *FALSE*, we can add more data to our analysis:

```

pemaCompute(pemaObj = meanPemaObj,
  data = data.frame(x = rnorm(1000)), varName = "x",
  initPema = FALSE)
[1] 0.001516969

meanPemaObj$totalValidObs
[1] 2000

```

The number of total valid observations is now 2000.

Using a *RevoScaleR* Data Source with the *pemaCompute* Function

In the previous section, we analyzed data in memory. The **RevoScaleR** package provides a data source framework that allows data to be automatically extracted in chunks from data on disk or in a database. It also provides the *.xdf* file format that can efficiently extract chunks of data.

We can use a sample *.xdf* file provided with the package. First we create a data source for this file:

```
airXdf <- RxXdfData(file.path(rxGetOption("sampleDataDir"),
  "AirlineDemoSmall.xdf"))
```

Using the *meanPemaObj* created above, we compute the mean of the variable *ArrDelay* (the arrival delay in minutes). The data in this file is stored in three blocks, with 200,000 rows in each block. The *pemaCompute* function processes these chunks one at a time:

```
pemaCompute(meanPemaObj, data = airXdf, varName = "ArrDelay")

Rows Read: 200000, Total Rows Processed: 200000, Total Chunk Time: 0.009 seconds
Rows Read: 200000, Total Rows Processed: 400000, Total Chunk Time: 0.007 seconds
Rows Read: 200000, Total Rows Processed: 600000, Total Chunk Time: 0.041 seconds
[1] 11.31794
```

You can control the amount of progress reported to the console using the *reportProgress* field of *PemaBaseClass*.

Using *pemaCompute* in a Distributed Compute Context

RevoScaleR provides a number of distributed compute contexts, such as Hadoop clusters (Cloudera and Hortonworks). Use of the same PEMA reference class object on these platforms is experimental. It can be tried with data on those platforms by specifying the *computeContext* in the *pemaCompute* function.

Additional Examples Using RevoPemaR

A number of examples are provided in the *demoScripts* directory of the *RevoPemaR* package. You can find the location of this directory by entering:

```
path.package("RevoPemaR")
```

Basic Text Mining Examples

Two PEMA text mining analyses are provided as examples.

- *PemaPopularWords* accumulates the words used in a variable containing character data. The *initialize* method provides a variety of arguments to fine-tune the processing. The code for the reference class generator is provided in *PemaPopularWords.R*, and examples using it in *PemaPopularWordsEx.R*.
- *PemaWordCount* counts instances of specified words in a variable containing character data. The code for the reference class generator is provided in *PemaWordCount.R*, and examples using it in *PemaWordCountEx.R*.

If you are using **RevoScaleR** and are interested in exploring text mining with a large dataset, instructions for downloading and code for importing Amazon reviews of fine foods is contained in the script *finefoodsImport.R*

Performing By-Group Computations

A *PemaByGroup* class is included in *RevoPemaR* to facilitate by-group computations. Examples of using this class are provided in the *PemaByGroupEx.R* demo script. It is assumed that the relevant variables for each group can fit into memory, and are then processed by arbitrary R functions. It requires that data be pre-sorted by group before processing, so generally cannot be used in distributed compute contexts such as Hadoop.

An Iterative PEMA Algorithm: Logistic Gradient Descent

A simple logistic gradient descent algorithm is provided as an example of an iterative algorithm that inherits from a parent class.

The *PemaGradDescent* class generator (in *PemaGradDescent.R*) specifies a number of important methods for iterative algorithms, for example:

- *initIteration*: initializes the appropriate field values at the beginning of each iteration
- *fn*: a placeholder for the computation of the objective (loss) function for gradient descent
- *gradientFn*: a placeholder for the computation of the gradient function for gradient descent
- *hasConverged*: checks convergence criteria

This class generator cannot be used directly. A child class generator must be created that at a minimum specifies the objective function (*fn*) and gradient function (*gradientFn*). An example is provided in *PemaLogitGDR*, showing a logistic gradient descent. A simple example of its use is in *PemaLogitGDE.R*.

Debugging *RevoPemaR* Code

The R Reference Classes provide standard R debugging tools, and *trace* and *untrace* methods are provided in the base reference class.

The *PemaBaseClass* provides a simple way of printing trace output that is particularly useful in debugging code in a distributed environment. Calls to the *outputTrace* method within other methods print the specified text if the *traceLevel* field value exceeds or is equal to the *outTraceLevel* argument:

```
meanPemaObj$outputTrace

Class method definition for method outputTrace()
function (text, outTraceLevel = 1)
{
  "Prints text if the traceLevel >= outTraceLevel"
  if (length(traceLevel) == 0) {
    warning("traceLevel has not been initialized.")
  }
  else if (traceLevel >= outTraceLevel) {
    cat(text)
  }
}
```

Python Function Library Reference

7/12/2022 • 3 minutes to read • [Edit Online](#)

This section contains the Python reference documentation for three proprietary packages from Microsoft used for data science and machine learning on premises and at scale.

You can use these libraries and functions in combination with other open source or third-party packages, but to use the proprietary packages, your Python code must run against a service or on a computer that provides the interpreters.

LIBRARY DETAILS	DESCRIPTION
Supported platforms	Machine Learning Server 9.2.1, 9.3 and 9.4 SQL Server 2017 (Windows only)
Built on:	Anaconda 4.2 distribution of Python 3.5 (included when you add Python support during installation).

Python modules

MODULE	VERSION	DESCRIPTION
azureml-model-management-sdk	1.0.1	Classes and functions to authenticate, deploy, manage, and consume analytic web services in Python.
microsoftml	9.4	A collection of Python functions used for machine learning operations, including training and transformations, scoring, text and image analysis, and feature extraction for deriving values from existing data.
revoscalepy	9.4	Data access, manipulation and transformations, visualization, and statistical analysis. The revoscalepy functions support a broad spectrum of statistical and analytical tasks that operate at scale, bringing analytical operations to local data or remote on a Spark cluster or data residing in SQL Server.

NOTE

Developers who are familiar with Microsoft R packages might notice similarities in the functions provided in revoscalepy and microsoftml. Conceptually, revoscalepy and microsoftml are the Python equivalents of the RevoScaleR R package and the MicrosoftML R package, respectively.

How to get packages

You can get the packages when you install Machine Learning Server, or SQL Server 2017, and choose the option for Python support. In addition to Python packages, Machine Learning Server setup and SQL Server setup both install the Python interpreters and base modules required to run any script or code that calls functions from proprietary package.

For SQL Server, packages are installed by default in the \Program files\Microsoft SQL Server*instance name*\PYTHON_SERVICES

Ships in:

- [Machine Learning Server](#)
- [SQL Server 2017 Machine Learning Services](#)
- [SQL Server Machine Learning Server \(Standalone\)](#)

How to list modules and versions

To get the version of a Python module installed on your computer, start Python and execute the following commands:

1. Double-click **Python.exe** in \Program Files\Microsoft\ML Server\PYTHON_SERVER.
2. Open interactive help: `help()`
3. Type the name of a module at the help prompt: `help> revoscalepy`. Help returns the name, package contents, version, and file location.
4. List all installed modules: `modules`
5. Import a module: `import revoscalepy`

How to list functions and get function help

To view the embedded help for each class, use the `help()` command, specifying the base class of the object of interest.

1. Double-click **Python.exe** in \Program Files\Microsoft\ML Server\PYTHON_SERVER.
2. Open interactive help: `help()`
3. Type the fully-qualified class name within the brackets.
 - For `azureml-azureml-model-management-sdk`, include the class in the path. For example, for **MLServer** help, type `help(azureml.deploy.server.MLServer)`.
 - For `revoscalepy`, type `help(revoscalepy)` to get package contents, and then include one of the packages on the next iteration. For example, `help(revoscalepy.computecontext)` returns all the functions related to compute context.

Note to R Users: Python naming conventions

`revoscalepy`, `microsoftml`, and `azureml-model-management-sdk` correspond to the Microsoft R packages, `RevoScaleR`, `MicrosoftML`, and `mrsdeploy`. If you have a background in these libraries, you might notice similarities in function names and operations, with Python versions adhering to the naming conventions of that language:

- lowercase package names (`microsoftml` contrasted with `MicrosoftML`) and most function names
- underscore in function names (`rx_import` in `revoscalepy` contrasted with `rxImport` in `RevoScaleR`)

Next steps

First, read the introduction to each package to learn about common use case scenarios:

- [azureml-model-management-sdk package for Python](#)
- [microsoftml package for Python](#)
- [revoscalepy package for Python](#)

Next, follow these tutorials for hands-on experience:

- [Use revoscalepy to create a model \(in SQL Server\)](#)
- [Run Python in T-SQL](#)
- [Deploy a model as a web service in Python](#)

See also

[How-to guides in Machine Learning Server](#)
[Machine Learning Server](#)
[SQL Server Machine Learning Services with Python](#)
[SQL Server Machine Learning Server \(Standalone\)](#)
[Additional learning resources and sample datasets](#)

revoscalepy package

7/12/2022 • 7 minutes to read • [Edit Online](#)

The **revoscalepy** module is a collection of portable, scalable and distributable Python functions used for importing, transforming, and analyzing data at scale. You can use it for descriptive statistics, generalized linear models, logistic regression, classification and regression trees, and decision forests.

Functions run on the **revoscalepy** interpreter, built on open-source Python, engineered to leverage the multithreaded and multinode architecture of the host platform.

PACKAGE DETAILS	INFORMATION
Current version:	9.4
Built on:	Anaconda 4.2 distribution of Python 3.5
Package distribution:	Machine Learning Server 9.x SQL Server 2017 Machine Learning Services SQL Server 2017 Machine Learning Server (Standalone)

How to use revoscalepy

The **revoscalepy** module is found in Machine Learning Server or SQL Server Machine Learning when you add Python to your installation. You get the full collection of proprietary packages plus a Python distribution with its modules and interpreter.

You can use any Python IDE to write Python script calling functions in **revoscalepy**, but the script must run on a computer having our proprietary modules. For a review of common tasks, see [How to use revoscalepy with Spark](#).

Run it locally

This is the default. The **revoscalepy** library runs locally on all platforms. On a standalone Linux or Windows system, data and operations are local to the machine. On Spark, a local compute context means that data and operations are local to current execution environment (typically, an edge node).

Run in a remote compute context

In a remote compute context, the script running on a local Machine Learning Server shifts execution to a remote Machine Learning Server on Spark or SQL Server. For example, script running on Windows might shift execution to a Spark cluster to process data there.

On Spark, set the compute context to **RxSpark** cluster and give the cluster name. In this context, if you call a function that can run in parallel, the task is distributed across data nodes in the cluster, where the operation is co-located with the data.

On SQL Server, set the compute context to **RxInSqlServer**. There are two primary use cases for remote compute context:

- Call Python functions in T-SQL script or stored procedures running on SQL Server.
- Call **revoscalepy** functions in Python script executing in a SQL Server **compute context**. In your script, you can set a compute context to shift execution of **revoscalepy** operations to a remote SQL Server instance that has the **revoscalepy** interpreter.

Functions by category

The library includes data transformation and manipulation, visualization, predictions, and statistical analysis functions. It also includes functions for controlling jobs, serializing data, and performing common utility tasks.

This section lists the functions by category to give you an idea of how each one is used. The table of contents to lists functions in alphabetical order.

NOTE

Some function names begin with `rx-` and others with `Rx`. The `Rx` function name prefix is used for class constructors for data sources and compute contexts.

1-Compute context functions

FUNCTION	DESCRIPTION
<code>RxInSqlServer</code>	Creates a compute context for running revoscalepy analyses inside a remote Microsoft SQL Server.
<code>RxLocalSeq</code>	This is the default but you can call it switch back to a local compute context if your script runs in multiple. Computations using <code>rx_exec</code> will be processed sequentially.
<code>rx_get_compute_context</code>	Returns the current compute context.
<code>rx_set_compute_context</code>	Change the compute context to a different one.
<code>RxSpark</code>	Creates a compute context for running revoscalepy analyses in a remote Spark cluster.
<code>rx_get_pyspark_connection</code>	Gets a connection to a PySpark data set, in support of revoscalepy and PySpark interoperability.
<code>rx_spark_connect</code>	Creates a persistent Spark Connection.
<code>rx_spark_disconnect</code>	Closes the connection.

2-Data source functions

Data sources are used by [microsoftml functions](#) as well as [revoscalepy](#).

FUNCTION	COMPUTE CONTEXT	DESCRIPTION
<code>RxDataSource</code>	All	Base class for all revoscalepy data sources.
<code>RxHdfsFileSystem</code>	Local, <code>RxSpark</code>	Data source is accessed through HDFS instead of Linux.
<code>RxNativeFileSystem</code>	Local, <code>RxSpark</code>	Data source is accessed through Linux instead of HDFS.

FUNCTION	COMPUTE CONTEXT	DESCRIPTION
RxHiveData	Local, RxSpark	Generates a data source object from a Hive data file.
RxTextData	Local, RxSpark	Generates a data source object from a text data file.
RxXdfData	All	Generates a data source object from an XDF data source.
RxOdbcData	All	Generates a data source object from an ODBC data source.
RxOrcData	Local, RxSpark	Generates a data source object from an Orc data file.
RxParquetData	Local, RxSpark	Generates a data source object from a Parquet data file.
RxSparkData	Local, RxSpark	Generates a data source object from a Spark data source.
RxSparkDataFrame	Local, RxSpark	Generates a data source object from a Spark data frame.
rx_get_partitions	Local, RxSpark	Get partitions of a partitioned Xdf data source.
rx_partition	Local, RxSpark	Partition input data sources by key values and save the results to a partitioned .xdf on disk.
rx_spark_cache_data	Local, RxSpark	Generates a data source object from cached data.
rx_spark_list_data	Local, RxSpark	Generates a data source object from a list.
rx_spark_remove_data	Local, RxSpark	Deletes the Spark cached data source object.
RxSqlServerData	Local, RxInSqlServer	Generates a data source object from a SQL table or query.

3-Data manipulation (ETL) functions

FUNCTION	COMPUTE CONTEXT	DESCRIPTION
rx_import	All	Import data into an .xdf file or data frame.
rx_data_step	All	Transform data from an input data set to an output data set.

4-Analytic functions

FUNCTION	COMPUTE CONTEXT	DESCRIPTION
rx_exec_by	Local, RxSpark	Execute an arbitrary function in parallel on multiple data nodes.
rx_summary	All	Produce univariate summaries of objects in revoscalepy.
rx_lin_mod	All	Fit linear models on small or large data.
rx_logit	All	Use rx_logit to fit logistic regression models for small or large data.
rx_dtree	All	Fit classification and regression trees on an '.xdf' file or data frame for small or large data using parallel external memory algorithm.
rx_dforest	All	Fit classification and regression decision forests on an '.xdf' file or data frame for small or large data using parallel external memory algorithm.
rx_btrees	All	Fit stochastic gradient boosted decision trees on an '.xdf' file or data frame for small or large data using parallel external memory algorithm.
rx_predict_default	All	Compute predicted values and residuals using rx_lin_mod and rx_logit objects.
rx_predict_rx_dforest	All	Calculate predicted or fitted values for a data set from an rx_dforest or rx_btrees object.
rx_predict_rx_dtree	All	Calculate predicted or fitted values for a data set from an rx_dtree object.

5-Job functions

In an RxSpark context, job management is built in. You only need job functions if you want to manually control the Yarn queue.

FUNCTION	COMPUTE CONTEXT	DESCRIPTION
rx_exec	All	Allows distributed execution of a function in parallel across nodes (computers) or cores of a "compute context" such as a cluster.

FUNCTION	COMPUTE CONTEXT	DESCRIPTION
<code>rx_cancel_job</code>	All	Removes all job-related artifacts from the distributed computing resources, including any job results.
<code>rx_cleanup_jobs</code>	All	Removes the artifacts for a specific job.
<code>RxRemoteJob</code> class	All	Closes the remote job, purging all associated job-related data.
<code>RxRemoteJobStatus</code>	All	Represents the execution status of a remote Python job.
<code>rx_get_job_info</code>	All	Contains complete information on the job's compute context as well as other information needed by the distributed computing resources.
<code>rx_get_job_output</code>	All	Returns console output for the nodes participating in a distributed computing job.
<code>rx_get_job_results</code>	All	Returns results of the run or a message stating why results are not available.
<code>rx_get_job_status</code>	All	Obtain distributed computing processing status for the specified job.
<code>rx_get_jobs</code>	All	Returns a list of job objects associated with the given compute context and matching the specified parameters.
<code>rx_wait_for_job</code>	All	Block on an existing distributed job until completion, effectively turning a non-blocking job into a blocking job.

6-Serialization functions

FUNCTION	COMPUTE CONTEXT	DESCRIPTION
<code>rx_serialize_model</code>	All	Serialize a given python model.
<code>rx_read_object</code>	All	Retrieves an ODBC data source object.
<code>rx_read_xdf</code>	All	Read data from an .xdf file into a data frame.
<code>rx_write_object</code>	All	Stores an ODBC data source object.
<code>rx_delete_object</code>	All	Deletes an object from the ODBC data source.

FUNCTION	COMPUTE CONTEXT	DESCRIPTION
rx_list_keys	All	Enumerates all keys or versions for a given key, depending on the parameters.

7-Utility functions

FUNCTION	COMPUTE CONTEXT	DESCRIPTION
RxOptions	All	Specify and retrieve options needed for revoscalepy computations.
rx_get_info	All	Get basic information about a revoscalepy data source or data frame.
rx_get_var_info	All	Get variable information for a revoscalepy data source or data frame, including variable names, descriptions, and value labels.
rx_get_var_names	All	Read the variable names for data source or data frame.
rx_set_var_info	All	Set the variable information for an .xdf file, including variable names, descriptions, and value labels, or set attributes for variables in a data frame.
RxMissingValues	All	Provides missing values for various NumPy data types which you can use to mark missing values in a sequence of data in ndarray .
rx_privacy_control	All	Opt out of usage data collection .
rx_hadoop_command	Local, RxSpark	Execute arbitrary Hadoop commands and perform standard file operations in Hadoop.
rx_hadoop_copy_from_local	Local, RxSpark	Wraps the Hadoop <code>fs -copyFromLocal</code> command.
rx_hadoop_copy_to_local	Local, RxSpark	Wraps the Hadoop <code>fs -copyToLocal</code> command.
rx_hadoop_copy	Local, RxSpark	Wraps the Hadoop <code>fs -cp</code> command.
rx_hadoop_file_exists	Local, RxSpark	Wraps the Hadoop <code>fs -test -e</code> command.
rx_hadoop_list_files	Local, RxSpark	Wraps the Hadoop <code>fs -ls or -lsr</code> command.

FUNCTION	COMPUTE CONTEXT	DESCRIPTION
<code>rx_hadoop_make_dir</code>	Local, RxSpark	Wraps the Hadoop <code>fs -mkdir -p</code> command.
<code>rx_hadoop_move</code>	Local, RxSpark	wraps the Hadoop <code>fs -mv</code> command.
<code>rx_hadoop_remove_dir</code>	Local, RxSpark	Wraps the Hadoop <code>fs -rm -r</code> or <code>fs -rm -r -skipTrash</code> command.
<code>rx_hadoop_remove</code>	Local, RxSpark	Wraps the Hadoop <code>fs -rm</code> or <code>fs -rm -skipTrash</code> command.

Next steps

For Machine Learning Server, try a quickstart as an introduction to [revoscalepy](#):

- [revoscalepy and PySpark interoperability](#)

For SQL Server, add both Python modules to your computer by running setup:

- [Set up Python Machine Learning Services.](#)

Follow these SQL Server tutorials for hands-on experience:

- [Use revoscalepy to create a model](#)
- [Run Python in T-SQL](#)

See also

[Machine Learning Server](#)

[SQL Server Machine Learning Services with Python](#)

[SQL Server Machine Learning Server \(Standalone\)](#)

rx_btrees

7/12/2022 • 7 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_btrees(formula, data, output_file=None, write_model_vars=False,
                      overwrite=False, pweights=None, fweights=None, cost=None, min_split=None,
                      min_bucket=None, max_depth=1, cp=0, max_compete=0, max_surrogate=0,
                      use_surrogate=2, surrogate_style=0, n_tree=10, m_try=None, replace=False,
                      strata=None, sample_rate=None, importance=False, seed=None,
                      loss_function='bernoulli', learning_rate=0.1, max_num_bins=None,
                      max_unordered_levels=32, remove_missings=False, compute_obs_node_id=None,
                      use_sparse_cube=False, find_splits_in_parallel=True, row_selection=None,
                      transforms=None, transform_objects=None, transform_function=None,
                      transform_variables=None, transform_packages=None, blocks_per_read=1,
                      report_progress=2, verbose=0, compute_context=None, xdf_compression_level=1,
                      **kwargs)
```

Description

Fit stochastic gradient boosted decision trees on an .xdf file or data frame for small or large data using parallel external memory algorithm.

Arguments

formula

Statistical model using symbolic formulas.

data

Either a data source object, a character string specifying a '.xdf' file, or a data frame object. If a Spark compute context is being used, this argument may also be an RxHiveData, RxOrcData, RxParquetData or RxSparkDataFrame object or a Spark data frame object from pyspark.sql.DataFrame.

output_file

Either an RxXdfData data source object or a character string specifying the '.xdf' file for storing the resulting node indices. If None, then no node indices are stored to disk. If the input data is a data frame, the node indices are returned automatically.

write_model_vars

Bool value. If True, and the output file is different from the input file, variables in the model will be written to the output file in addition to the node numbers. If variables from the input data set are transformed in the model, the transformed variables will also be written out.

overwrite

Bool value. If True, an existing output_file with an existing column named outColName will be overwritten.

pweights

Character string specifying the variable of numeric values to use as probability weights for the observations.

fweights

Character string specifying the variable of integer values to use as frequency weights for the observations.

method

Character string specifying the splitting method. Currently, only "class" or "anova" are supported. The default is "class" if the response is a factor, otherwise "anova".

parms

Optional list with components specifying additional parameters for the "class" splitting method, as follows:

prior: A vector of prior probabilities. The priors must be positive and sum to 1. The default priors are proportional to the data counts.

loss: A loss matrix, which must have zeros on the diagonal and positive off-diagonal elements. By default, the off-diagonal elements are set to 1.

split: The splitting index, either gini (the default) or information.

If parms is specified, any of the components can be specified or omitted. The defaults will be used for missing components.

cost

A vector of non-negative costs, containing one element for each variable in the model. Defaults to one for all variables. When deciding which split: to choose, the improvement on splitting on a variable is divided by its cost.

min_split

The minimum number of observations that must exist in a node before a split is attempted. By default, this is $\sqrt{\text{num of obs}}$. For non-XDF data sources, as (num of obs) is unknown in advance, it is wisest to specify this argument directly.

min_bucket

The minimum number of observations in a terminal node (or leaf). By default, this is min_split/3.

cp

Numeric scalar specifying the complexity parameter. Any split that does not decrease overall lack-of-fit by at least cp is not attempted.

max_compete

The maximum number of competitor splits retained in the output. These are useful model diagnostics, as they allow you to compare splits in the output with the alternatives.

max_surrogate

The maximum number of surrogate splits retained in the output. See the Details for a description of how surrogate splits are used in the model fitting. Setting this to 0 can greatly improve the performance of the algorithm; in some cases almost half the computation time is spent in computing surrogate splits.

use_surrogate

An integer specifying how surrogates are to be used in the splitting process: 0: Display-only; observations with a missing value for the primary split variable are not sent further down the tree.

1: Use surrogates, in order, to split observations missing the primary split variable. If all surrogates are missing, the observation is not split.

2: Use surrogates, in order, to split observations missing the primary split variable. If all surrogates are missing or max_surrogate=0, send the observation in the majority direction.

The 0 value corresponds to the behavior of the tree function, and 2 (the default) corresponds to the recommendations of Breiman et al.

surrogate_style

An integer controlling selection of a best surrogate. The default, 0, instructs the program to use the total number of correct classifications for a potential surrogate, while 1 instructs the program to use the percentage of correct classification over the non-missing values of the surrogate. Thus, 0 penalizes potential surrogates with a large number of missing values.

n_tree

A positive integer specifying the number of trees to grow.

m_try

A positive integer specifying the number of variables to sample as split candidates at each tree node. The default values are $\text{sqrt}(\text{num of vars})$ for classification and $(\text{num of vars})/3$ for regression.

replace

A bool value specifying if the sampling of observations should be done with or without replacement.

cutoff

(Classification only) A vector of length equal to the number of classes specifying the dividing factors for the class votes. The default is $1/(\text{num of classes})$.

strata

A character string specifying the (factor) variable to use for stratified sampling.

sample_rate

A scalar or a vector of positive values specifying the percentage(s) of observations to sample for each tree: for unstratified sampling: A scalar of positive value specifying the percentage of observations to sample for each tree. The default is 1.0 for sampling with replacement (i.e., replace=True) and 0.632 for sampling without replacement (i.e., replace=False).

for stratified sampling: A vector of positive values of length equal to the number of strata specifying the percentages of observations to sample from the strata for each tree.

importance

A bool value specifying if the importance of predictors should be assessed.

seed

An integer that will be used to initialize the random number generator. The default is random. For reproducibility, you can specify the random seed either using set.seed or by setting this seed argument as part of your call.

compute_oob_error

An integer specifying whether and how to compute the prediction error for out-of-bag samples: <0: never. This option may reduce the computation time. =0: only once for the entire forest.

0: once for each addition of a tree. This is the default.

loss_function

Character string specifying the name of the loss function to use. The following options are currently supported: "gaussian": Regression: for numeric responses. "bernoulli": Regression: for 0-1 responses. "multinomial": Classification: for categorical responses with two or more levels.

learning_rate

Numeric scalar specifying the learning rate of the boosting procedure.

max_num_bins

The maximum number of bins to use to cut numeric data. The default is $\min(1001, \max(101, \text{sqrt}(\text{num of obs})))$.

For non-XDF data sources, as (num of obs) is unknown in advance, it is wisest to specify this argument directly. If set to 0, unit binning will be used instead of cutting. See the 'Details' section for more information.

max_unordered_levels

The maximum number of levels allowed for an unordered factor predictor for multiclass (>2) classification.

remove_missings

Bool value. If True, rows with missing values are removed and will not be included in the output data.

use_sparse_cube

Bool value. If True, sparse cube is used.

find_splits_in_parallel

bool value. If True, optimal splits for each node are determined using parallelization methods; this will typically speed up computation as the number of nodes on the same level is increased.

row_selection

None. Not currently supported, reserved for future use.

transforms

None. Not currently supported, reserved for future use.

transform_objects

None. Not currently supported, reserved for future use.

transform_function

Variable transformation function. The variables used in the transformation function must be specified in transform_variables if they are not variables used in the model.

transform_variables

List of strings of input data set variables needed for the transformation function.

transform_packages

None. Not currently supported, reserved for future use.

blocks_per_read

Number of blocks to read for each chunk of data read from the data source.

report_progress

integer value with options: 0: No progress is reported. 1: The number of processed rows is printed and updated. 2: Rows processed and timings are reported. 3: Rows processed and all timings are reported.

verbose

Integer value. If 0, no additional output is printed. If 1, additional summary information is printed.

compute_context

A RxComputeContext object for prediction.

kwargs

Additional parameters

Returns

A RxDForestResults object of dtree model.

See also

[rx_predict](#), [rx_predict_rx_dforest](#).

Example

```
import os
from revoscalepy import rx_btrees, rx_import, RxOptions, RxXdfData
sample_data_path = RxOptions.get_option("sampleDataDir")
ds = RxXdfData(os.path.join(sample_data_path, "kyphosis.xdf"))
kyphosis = rx_import(input_data = ds)

# classification
form = "Kyphosis ~ Age + Number + Start"
seed = 0
n_trees = 50
interaction_depth = 4
n_min_obs_in_node = 1
shrinkage = 0.1
bag_fraction = 0.5
distribution = "bernoulli"

btree = rx_btrees(form, kyphosis, loss_function=distribution, n_tree=n_trees, learning_rate=shrinkage,
                  sample_rate=bag_fraction, max_depth=interaction_depth, min_bucket=n_min_obs_in_node,
                  seed=seed,
                  replace=False, max_num_bins=200)

# regression
ds = RxXdfData(os.path.join(sample_data_path, "airquality.xdf"))
df = rx_import(input_data = ds)
df = df[df['Ozone'] != -2147483648]

form = "Ozone ~ Solar.R + Wind + Temp + Month + Day"
ntrees = 50
interaction_depth = 3
min_obs_in_node = 1
shrinkage = 0.1
bag_fraction = 0.5
distribution = "gaussian"

btree = rx_btrees(form, df, loss_function=distribution, n_tree=ntrees, learning_rate=shrinkage,
                  sample_rate=bag_fraction, max_depth=interaction_depth, min_bucket=min_obs_in_node, seed=0,
                  replace=False, max_num_bins=200, verbose=2)
```

rx_cancel_job

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_cancel_job(job_info: revoscalepy.computecontext.RxRemoteJob.RxRemoteJob,  
                           console_output: bool = False) -> bool
```

Description

This function does not attempt to retrieve any output objects; if the output is desired the *console_output* flag can be used to display it. This function does, however, remove all job-related artifacts from the distributed computing resources including any job results.

Arguments

job_info

A job to cancel

console_output

If *None* use the console option that was present when the job was created, if *True* output any console output that was present when the job was canceled will be displayed. If *False* then all console output will not be displayed.

Returns

True if the job is successfully canceled; *False* otherwise.

See also

[rx_get_job_status](#)

Example

```

import time
from revoscalepy import RxInSqlServer
from revoscalepy import RxSqlServerData
from revoscalepy import rx_dforest
from revoscalepy import rx_cancel_job
from revoscalepy import RxRemoteJobStatus
from revoscalepy import rx_get_job_status
from revoscalepy import rx_cleanup_jobs

connection_string = 'Driver=SQL Server;Server=.;Database=RevoTestDb;Trusted_Connection=True;'
airline_data_source = RxSqlServerData(sql_query='select * from airlinedemosmall',
                                       connection_string=connection_string,
                                       column_info={
                                           'ArrDelay': {'type': 'integer'},
                                           'CRSDepTime': {'type': 'float'},
                                           'DayOfWeek': {
                                               'type': 'factor',
                                               'levels': ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
                                                          'Friday',
                                                          'Saturday', 'Sunday']
                                           }
                                       })
# Setting wait to False allows the job to be run asynchronously
compute_context = RxInSqlServer(connection_string=connection_string,
                                 num_tasks=1,
                                 auto_cleanup=False,
                                 wait=False)

job = rx_dforest(formula='ArrDelay ~ DayOfWeek',
                 seed=1,
                 data=airline_data_source,
                 compute_context=compute_context)

# Poll until status is RUNNING
status = rx_get_job_status(job)
while status == RxRemoteJobStatus.QUEUED:
    time.sleep(1)
    status = rx_get_job_status(job)

# Only running or queued jobs can be canceled
if status == RxRemoteJobStatus.RUNNING :
    # Cancel the job
    rx_cancel_job(job)

# Only canceled or failed jobs can be cleaned up
if status == RxRemoteJobStatus.CANCELED or status == RxRemoteJobStatus.FAILED :
    # Cleanup after the job
    rx_cleanup_jobs(job)

```

rx_cleanup_jobs

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_cleanup_jobs(job_info_list: typing.Union[revoscalepy.computecontext.RxRemoteJob.RxRemoteJob,
    list], force: bool = False, verbose: bool = True)
```

Description

Removes the artifacts for the specified jobs.

If *job_info_list* is a *RxRemoteJob* object, *rx_cleanup_jobs* attempts to remove the artifacts. However, if the job has successfully completed and *force* is *False*, *rx_cleanup_jobs* issues a warning saying to either set *force=True* or use *rx_get_job_results* to get the results and delete the artifacts.

If *job_info_list* is a list of jobs, *rx_cleanup_jobs* attempts to apply the cleanup rules for a single job to each element in the list.

Arguments

job_info_list

The jobs for which to remove the artifacts, this can be a list of jobs or a single job

force

True indicates the cleanup should happen regardless of whether or not the job status can be determined and job results have already been retrieved. *False* indicates that the should be cleaned up by calling *rx_get_job_results* or the job should have completed unsuccessfully (such as by using *rx_cancel_job*).

verbose

If *True*, will print the directories/records being deleted.

Returns

This function does not return a value

See also

Example

```

import time
from revoscalepy import RxInSqlServer
from revoscalepy import RxSqlServerData
from revoscalepy import rx_dforest
from revoscalepy import rx_cancel_job
from revoscalepy import RxRemoteJobStatus
from revoscalepy import rx_get_job_status
from revoscalepy import rx_cleanup_jobs

connection_string = 'Driver=SQL Server;Server=.;Database=RevoTestDb;Trusted_Connection=True;'
airline_data_source = RxSqlServerData(sql_query='select * from airlinedemosmall',
                                       connection_string=connection_string,
                                       column_info={
                                           'ArrDelay': {'type': 'integer'},
                                           'CRSDepTime': {'type': 'float'},
                                           'DayOfWeek': {
                                               'type': 'factor',
                                               'levels': ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
                                                          'Friday',
                                                          'Saturday', 'Sunday']
                                           }
                                       })
# Setting wait to False allows the job to be run asynchronously
compute_context = RxInSqlServer(connection_string=connection_string,
                                 num_tasks=1,
                                 auto_cleanup=False,
                                 wait=False)

job = rx_dforest(formula='ArrDelay ~ DayOfWeek',
                 seed=1,
                 data=airline_data_source,
                 compute_context=compute_context)

# Poll until status is RUNNING
status = rx_get_job_status(job)
while status == RxRemoteJobStatus.QUEUED:
    time.sleep(1)
    status = rx_get_job_status(job)

if status == RxRemoteJobStatus.RUNNING :
    # Cancel the job
    rx_cancel_job(job)

# Only canceled or failed jobs can be cleaned up
if status == RxRemoteJobStatus.CANCELED or status == RxRemoteJobStatus.FAILED :
    # Cleanup after the job
    rx_cleanup_jobs(job)

```

rx_create_col_info

7/12/2022 • 3 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_create_col_info(data: typing.Union[revoscalepy.datasource.RxDataSource.RxDataSource,
    str, pandas.core.frame.DataFrame,
    revoscalepy.functions.RxGetInfoXdf.GetVarInfoResults],
    include_low_high: bool = False, factors_only: bool = False,
    vars_to_keep: list = None, sort_levels: bool = False,
    compute_info: bool = True,
    use_factor_index: bool = False) -> collections.OrderedDict
```

Description

From the data source, generates an ordered dictionary of dictionaries with variable names as keys and column information dictionaries as values. It can be used in `rx_import` or an `RxDatasource` constructor. This function can be used to ensure consistent categorical (factor) levels when importing a series of text files to xdf. It is also useful for repeated analysis on non-xdf data sources.

:param data : An `RxDatasource` object, a character string containing an '`.xdf`' file name, or a data frame. An object returned from `rx_get_var_info` is also supported.

:param include_low_high : Bool value. If True, the low/high values will be included in the `column_info` object. Note that this will override any actual low/high values in the data set if the `column_info` object is applied to a different data source.

:param factors_only : Bool value. If True, only column information for categorical (factor) variables will be included in the output.

:param vars_to_keep : None to include all variables, or list of strings of variable names to include :param sort_levels : Bool value. If True, categorical (factor) levels will be sorted.

If categorical levels represent integers, they will be put in numeric order.

:param compute_info : Bool value. If True, a pass through the data will be taken for non-xdf data sources to compute categorical (factor) levels and low/high values.

:param use_factor_index : Bool value. If True, the `factor_index` variable type will be used instead of factor

:return : Ordered dict `column_info` of named variable information dicts. It can be used as input for `rx_import` and in data sources such as `RxTextData` and `RxSqlServerData` Each variable information dict contains one or more of the named elements given below..

```
type: Character string specifying the data type for the column.  
levels: List of strings containing the levels when type = "factor".  
low: The minimum data value in the variable (used in computations using the F() function).
```

For factors, it is the index of the lowest level.

```
high: The maximum data value in the variable (used in computations using the F() function).  
For factors, it is the index of the highest level.
```

See also

[RxDataSource](#) , [RxTextData](#) , [RxSqlServerData](#) , [RxOdbcData](#) , [RxXdfData](#) , [rx_import](#) .

Example

```
:execute:  
:print_output:  
  
import os  
from revoscalepy import RxOptions, RxTextData, rx_create_col_info, rx_import, rx_logit, rx_predict  
# Get the low/high values and factor levels before using a data source  
# for rx_import or analysis  
  
# Create a text data source, specifying that 'yearsEmploy' should be a factor  
mort = os.path.join(RxOptions.get_option("sampleDataDir"), "mortDefaultSmall2000.csv")  
mort_ds = RxTextData(file = mort, column_classes = {'yearsEmploy' : "factor", 'default' : "logical"})  
  
# By default, rx_create_col_info will make a pass through the data to compute factor levels  
# and low/high values. We'll also request that the levels be sorted  
mort_column_info = rx_create_col_info(data = mort_ds, include_low_high = True, sort_levels = True)  
  
# Re-create the data source, now using the computed colInfo  
mort_ds = RxTextData(file = mort, column_info = mort_column_info)  
# Import the data  
mort_df = rx_import(mort_ds)  
print(mort_df['yearsEmploy'])  
  
# Or use the text data source directly in an analysis  
# (not needing a pass through the data to compute the factor levels)  
logit_obj = rx_logit('default~yearsEmploy', data = mort_ds)  
  
#####  
  
# Train a model on one imported data set, then score using another  
# Train a model on the first year of the data, importing it from text to a data frame  
  
mort1 = os.path.join(RxOptions.get_option("sampleDataDir"), "mortDefaultSmall2000.csv")  
mort1_ds = RxTextData(file = mort1, column_classes = {'yearsEmploy' : "factor", 'default' : "logical"})  
# Since we haven't specified factor levels, they will be created 'first come, first serve'  
mort1_df = rx_import(mort1_ds)  
mort1_df['yearsEmploy'].cat.categories  
# Estimate a logit model  
logit_obj = rx_logit('default~yearsEmploy', data = mort1_df)  
  
# Now import the second year of data  
mort2 = os.path.join(RxOptions.get_option("sampleDataDir"), "mortDefaultSmall2001.csv")  
mort2_ds = RxTextData(file = mort2, column_classes = {'yearsEmploy' : "factor", 'default' : "logical"})  
mort2_df = rx_import(mort2_ds)  
# The levels are in a different order  
mort2_df['yearsEmploy'].cat.categories  
  
# If we try to use the model estimated from the first data set to predict on the second,  
# pred_out = rx_predict(logit_obj, data = mort2_df)  
# We will get an error  
#ERROR: order of factor levels in the data are inconsistent with  
#the order of the model coefficients  
  
# Instead, we can extract the col_info from the first data set  
mort_col_info = rx_create_col_info(data = mort1_df)  
  
# And use it when importing the second  
mort2_ds = RxTextData(file = mort2, column_info = mort_col_info)  
mort2_df = rx_import(mort2_ds)  
pred_out = rx_predict(logit_obj, data = mort2_df)  
pred_out.head()
```

```
#####
# Generating Column Information for a SQL Server Data Source
#####
from revoscalepy import RxSqlServerData, rx_create_col_info, rx_set_temp_compute_context, RxInSqlServer
connection_string="Driver=SQL Server;Server=.;Database=RevoTestDb;Trusted_Connection=TRUE;"
sql_source = RxSqlServerData(connection_string = connection_string,
                             table="AirlineDemoSmall",
                             rows_per_read=500000,
                             string_as_factors = True)

with rx_set_temp_compute_context(RxInSqlServer(connection_string = connection_string)):
    results = rx_create_col_info(sql_source)
```

RxDataSource

7/12/2022 • 2 minutes to read • [Edit Online](#)

```
revoscalepy.RxDataSource(column_info: dict = None)
```

Description

Base class for all revoscalepy data sources. Can be used with head() and tail() to display the first and last rows of the data set.

Arguments

num_rows

Integer value specifying the number of rows to display starting from the beginning of the dataset. If not specified, the default of 6 will be used.

report_progress

Integer value with options:

- 0: no progress is reported.
- 1: the number of processed rows is printed and updated.
- 2: rows processed and timings are reported.
- 3: rows processed and all timings are reported.

Example

```
# Return the first 4 rows
import os
from revoscalepy import RxOptions, RxXdfData
sample_data_path = RxOptions.get_option("sampleDataDir")
ds = RxXdfData(os.path.join(sample_data_path, "AirlineDemoSmall.xdf"))
ds.head(num_rows=4)

# Return the last 4 rows
import os
from revoscalepy import RxOptions, RxXdfData
sample_data_path = RxOptions.get_option("sampleDataDir")
ds = RxXdfData(os.path.join(sample_data_path, "AirlineDemoSmall.xdf"))
ds.tail(num_rows=4)
```

rx_data_step

7/12/2022 • 5 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_data_step(input_data: typing.Union[revoscalepy.datasource.RxDataSource.RxDataSource,
    pandas.core.frame.DataFrame, str] = None,
    output_file: typing.Union[str, revoscalepy.datasource.RxXdfData.RxXdfData,
    revoscalepy.datasource.RxTextData.RxTextData] = None,
    vars_to_keep: list = None, vars_to_drop: list = None,
    row_selection: str = None, transforms: dict = None,
    transform_objects: dict = None, transform_function=None,
    transform_variables: list = None,
    transform_packages: list = None, append: list = None,
    overwrite: bool = False, row_variable_name: str = None,
    remove_missings_on_read: bool = False,
    remove_missings: bool = False, compute_low_high: bool = True,
    max_rows_by_cols: int = 3000000, rows_per_read: int = -1,
    start_row: int = 1, number_rows_read: int = -1,
    return_transform_objects: bool = False,
    blocks_per_read: int = None, report_progress: int = None,
    xdf_compression_level: int = 0, strings_as_factors: bool = None,
    **kwargs) -> typing.Union[revoscalepy.datasource.RxXdfData.RxXdfData,
    pandas.core.frame.DataFrame]
```

Description

Inline data transformations of an existing data set to an output data set

Arguments

input_data

A character string with the path for the data to import (delimited, fixed format, ODBC, or XDF). Alternatively, a data source object representing the input data source can be specified. If a Spark compute context is being used, this argument may also be an RxHiveData, RxOrcData, RxParquetData or RxSparkDataFrame object or a Spark data frame object from pyspark.sql.DataFrame.

output_file

A character string representing the output '.xdf' file, or a RxXdfData object. If None, a data frame will be returned in memory. If a Spark compute context is being used, this argument may also be an RxHiveData, RxOrcData, RxParquetData or RxSparkDataFrame object.

vars_to_keep

list of strings of variable names to include when reading from the input data file. If None, argument is ignored. Cannot be used with vars_to_drop. Not supported for ODBC or fixed format text files.

vars_to_drop

List of strings of variable names to exclude when reading from the input data file. If None, argument is ignored. Cannot be used with vars_to_keep. Not supported for ODBC or fixed format text files.

row_selection

None. Not currently supported, reserved for future use.

transforms

None. Not currently supported, reserved for future use.

transform_objects

A dictionary of variables besides the data that are used in the transform function. See the example.

transform_function

Name of the function that will be used to modify the data. The variables used in the transformation function must be specified in transform_variables. See the example.

transform_variables

List of strings of the column names needed for the transform function.

transform_packages

None. Not currently supported, reserved for future use.

append

Either "none" to create a new '.xdf' file or "rows" to append rows to an existing '.xdf' file. If output_file exists and append is "none", the overwrite argument must be set to True. Ignored if a data frame is returned.

overwrite

Bool value. If True, the existing outData will be overwritten. Ignored if a dataframe is returned.

row_variable_name

Character string or None. If inData is a data.frame: If None, the data frame's row names will be dropped. If a character string, an additional variable of that name will be added to the data set containing the data frame's row names. If a data.frame is being returned, a variable with the name row_variable_name will be removed as a column from the data frame and will be used as the row names.

remove_missings_on_read

Bool value. If True, rows with missing values will be removed on read.

remove_missings

Bool value. If True, rows with missing values will not be included in the output data.

compute_low_high

Bool value. If False, low and high values will not automatically be computed. This should only be set to False in special circumstances, such as when append is being used repeatedly. Ignored for data frames.

max_rows_by_cols

The maximum size of a data frame that will be returned if output_file is set to None and inData is an '.xdf' file, measured by the number of rows times the number of columns. If the number of rows times the number of columns being created from the '.xdf' file exceeds this, a warning will be reported and the number of rows in the returned data frame will be truncated. If max_rows_by_cols is set to be too large, you may experience problems from loading a huge data frame into memory.

rows_per_read

Number of rows to read for each chunk of data read from the input data source. Use this argument for finer control of the number of rows per block in the output data source. If greater than 0, blocks_per_read is ignored. Cannot be used if input_data is the same as output_file. The default value of -1 specifies that data should be read by blocks according to the blocks_per_read argument.

start_row

The starting row to read from the input data source. Cannot be used if input_data is the same as output_file.

number_rows_read

Number of rows to read from the input data source. If remove_missings are used, the output data set may have fewer rows than specified by number_rows_read. Cannot be used if input_data is the same as output_file.

return_transform_objects

Bool value. If True, the list of transformObjects will be returned instead of a data frame or data source object. If the input transformObjects have been modified, by using .rxSet or .rxModify in the transformFunc, the updated values will be returned. Any data returned from the transformFunc is ignored. If no transformObjects are used, None is returned. This argument allows for user-defined computations within a transform_function without creating new data.

blocks_per_read

Number of blocks to read for each chunk of data read from the data source. Ignored for data frames or if rows_per_read is positive.

report_progress

Integer value with options: 0: no progress is reported. 1: the number of processed rows is printed and updated. 2: rows processed and timings are reported. 3: rows processed and all timings are reported.

xdf_compression_level

Integer in the range of -1 to 9. The higher the value, the greater the amount of compression - resulting in smaller files but a longer time to create them. If xdf_compression_level is set to 0, there will be no compression and files will be compatible with the 6.0 release of Revolution R Enterprise. If set to -1, a default level of compression will be used.

strings_as_factors

Bool indicating whether or not to automatically convert strings to factors on import. This can be overridden by specifying "character" in column_classes and column_info. If True, the factor levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use column_info with specified "levels".

kwargs

Additional arguments to be passed to the input data source.

Returns

If an output_file is not specified, an output data frame is returned. If an output_file is specified, an RxXdfData data source is returned that can be used in subsequent revoscalepy analysis.

Example

```
import os
from revoscalepy import RxOptions, RxXdfData, rx_data_step
sample_data_path = RxOptions.get_option("sampleDataDir")
kyphosis = RxXdfData(os.path.join(sample_data_path, "kyphosis.xdf"))

# Function to label whether the age in month is over 120 months.
month_limit = 120
new_col_name = "Over10Yr"
def transformFunc(data, cutoff, new_col_name):
    ret = data
    ret[new_col_name] = data.apply(lambda row: True if row.Age > cutoff else False, axis=1)
    return ret

kyphosis_df = rx_data_step(input_data=kyphosis, transform_function=transformFunc,
                           transform_objects={"cutoff": month_limit, "new_col_name": new_col_name})
kyphosis_df.head()
```

rx_delete_object

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_delete_object(src: revoscalepy.datasource.RxOdbcData.RxOdbcData,  
    key: str = None, version: str = None,  
    key_name: str = 'id', version_name: str = 'version',  
    all: bool = False)
```

Description

Deletes an object from an ODBC data source. The APIs are modeled after a simple key value store.

Details

Deletes an object from the ODBC data source. If there are multiple objects identified by the key/version combination, all are deleted.

The key and the version column should be of some SQL character type (CHAR, VARCHAR, NVARCHAR, etc.) supported by the data source. The value column should be a binary type (VARBINARY for instance). Some conversions to other types might work, however, they are dependent on the ODBC driver and on the underlying package functions.

Arguments

src

The object being stored into the data source.

key

A character string identifying the object. The intended use is for the key+version to be unique.

version

None or a character string which carries the version of the object. Combined with key identifies the object.

key_name

Character string specifying the column name for the key in the underlying table.

version_name

Character string specifying the column name for the version in the underlying table.

all

Bool value. *True* to remove all objects from the data source. If *True*, the 'key' parameter is ignored.

Returns

`rx_read_object` returns an object. `rx_write_object` and `rx_delete_object` return `bool`, `True` on success. `rx_list_keys` returns a single column data frame containing strings.

Example

```
from pandas import DataFrame
from numpy import random
from revoscalepy import RxOdbcData, rx_write_object, rx_read_object, rx_list_keys, rx_delete_object

connection_string = 'Driver=SQL Server;Server=.;Database=RevoTestDb;Trusted_Connection=True;'
dest = RxOdbcData(connection_string, table = "dataframe")

df = DataFrame(random.randn(10, 5))

status = rx_write_object(dest, key = "myDf", value = df)

read_obj = rx_read_object(dest, key = "myDf")

keys = rx_list_keys(dest)

rx_delete_object(dest, key = "myDf")
```

rx_dforest

7/12/2022 • 6 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_dforest(formula, data, output_file=None, write_model_vars=False,
    overwrite=False, pweights=None, fweights=None, cost=None, min_split=None,
    min_bucket=None, max_depth=10, cp=0, max_compete=0, max_surrogate=0,
    use_surrogate=2, surrogate_style=0, n_tree=10, m_try=None, replace=True,
    cutoff=None, strata=None, sample_rate=None, importance=False, seed=None,
    compute_cp_table=False, compute_oob_error=1, max_num_bins=None,
    max_unordered_levels=32, remove_missings=False, use_sparse_cube=False,
    find_splits_in_parallel=True, row_selection=None, transforms=None,
    transform_objects=None, transform_function=None, transform_variables=None,
    transform_packages=None, blocks_per_read=1, report_progress=2, verbose=0,
    compute_context=None, xdf_compression_level=1, **kwargs)
```

Description

Fits classification and regression decision forests on an .xdf file or data frame for small or large data using parallel external memory algorithm.

Arguments

formula

Statistical model using symbolic formulas.

data

Either a data source object, a character string specifying a '.xdf' file, or a data frame object. If a Spark compute context is being used, this argument may also be an RxHiveData, RxOrcData, RxParquetData or RxSparkDataFrame object or a Spark data frame object from pyspark.sql.DataFrame.

output_file

Either an RxXdfData data source object or a character string specifying the '.xdf' file for storing the resulting node indices. If None, then no node indices are stored to disk. If the input data is a data frame, the node indices are returned automatically.

write_model_vars

Bool value. If True, and the output file is different from the input file, variables in the model will be written to the output file in addition to the node numbers. If variables from the input data set are transformed in the model, the transformed variables will also be written out.

overwrite

Bool value. If True, an existing output_file with an existing column named outColName will be overwritten.

pweights

Character string specifying the variable of numeric values to use as probability weights for the observations.

fweights

Character string specifying the variable of integer values to use as frequency weights for the observations.

cost

A vector of non-negative costs, containing one element for each variable in the model. Defaults to one for all variables. When deciding which split to choose, the improvement on splitting on a variable is divided by its cost.

min_split

The minimum number of observations that must exist in a node before a split is attempted. By default, this is $\sqrt{\text{num of obs}}$. For non-XDF data sources, as (num of obs) is unknown in advance, it is wisest to specify this argument directly.

min_bucket

The minimum number of observations in a terminal node (or leaf). By default, this is $\text{min_split}/3$.

cp

Numeric scalar specifying the complexity parameter. Any split that does not decrease overall lack-of-fit by at least cp is not attempted.

max_compete

The maximum number of competitor splits retained in the output. These are useful model diagnostics, as they allow you to compare splits in the output with the alternatives.

max_surrogate

The maximum number of surrogate splits retained in the output. See the Details for a description of how surrogate splits are used in the model fitting. Setting this to 0 can greatly improve the performance of the algorithm; in some cases almost half the computation time is spent in computing surrogate splits.

use_surrogate

An integer specifying how surrogates are to be used in the splitting process: 0: Display-only; observations with a missing value for the primary split variable are not sent further down the tree.

1: Use surrogates, in order, to split observations missing the primary split variable. If all surrogates are missing, the observation is not split.

2: Use surrogates, in order, to split observations missing the primary split variable. If all surrogates are missing or $\text{max_surrogate}=0$, send the observation in the majority direction.

The 0 value corresponds to the behavior of the tree function, and 2 (the default) corresponds to the recommendations of Breiman et al.

surrogate_style

An integer controlling selection of a best surrogate. The default, 0, instructs the program to use the total number of correct classifications for a potential surrogate, while 1 instructs the program to use the percentage of correct classification over the non-missing values of the surrogate. Thus, 0 penalizes potential surrogates with a large number of missing values.

n_tree

A positive integer specifying the number of trees to grow.

m_try

A positive integer specifying the number of variables to sample as split candidates at each tree node. The default values are $\sqrt{\text{num of vars}}$ for classification and $(\text{num of vars})/3$ for regression.

replace

A bool value specifying if the sampling of observations should be done with or without replacement.

cutoff

(Classification only) A vector of length equal to the number of classes specifying the dividing factors for the class votes. The default is $1/(\text{num of classes})$.

strata

A character string specifying the (factor) variable to use for stratified sampling.

sample_rate

A scalar or a vector of positive values specifying the percentage(s) of observations to sample for each tree: for unstratified sampling: A scalar of positive value specifying the percentage of observations to sample for each tree. The default is 1.0 for sampling with replacement (i.e., `replace=True`) and 0.632 for sampling without replacement (i.e., `replace=False`).

For stratified sampling: A vector of positive values of length equal to the number of strata specifying the percentages of observations to sample from the strata for each tree.

importance

A bool value specifying if the importance of predictors should be assessed.

seed

An integer that will be used to initialize the random number generator. The default is `random`. For reproducibility, you can specify the random seed either using `set.seed` or by setting this seed argument as part of your call.

compute_oob_error

An integer specifying whether and how to compute the prediction error for out-of-bag samples: <0: never. This option may reduce the computation time. =0: only once for the entire forest.

0: once for each addition of a tree. This is the default.

max_num_bins

The maximum number of bins to use to cut numeric data. The default is `min(1001, max(101, sqrt(num of obs)))`. For non-XDF data sources, as (`num of obs`) is unknown in advance, it is wisest to specify this argument directly. If set to 0, unit binning will be used instead of cutting. See the 'Details' section for more information.

max_unordered_levels

The maximum number of levels allowed for an unordered factor predictor for multiclass (>2) classification.

remove_missings

Bool value. If `True`, rows with missing values are removed and will not be included in the output data.

use_sparse_cube

Bool value. If `True`, sparse cube is used.

find_splits_in_parallel

Bool value. If `True`, optimal splits for each node are determined using parallelization methods; this will typically speed up computation as the number of nodes on the same level is increased.

row_selection

None. Not currently supported, reserved for future use.

transforms

None. Not currently supported, reserved for future use.

transform_objects

A dictionary of variables besides the data that are used in the transform function. See `rx_data_step` for examples.

transform_function

Name of the function that will be used to modify the data before the model is built. The variables used in the

transformation function must be specified in transform_objects. See rx_data_step for examples.

transform_variables

List of strings of the column names needed for the transform function.

transform_packages

None. Not currently supported, reserved for future use.

blocks_per_read

number of blocks to read for each chunk of data read from the data source.

report_progress

Integer value with options: 0: No progress is reported. 1: The number of processed rows is printed and updated. 2: Rows processed and timings are reported. 3: Rows processed and all timings are reported.

verbose

Integer value. If 0, no additional output is printed. If 1, additional summary information is printed.

compute_context

A RxComputeContext object for prediction.

kwargs

Additional parameters

Returns

A RxDForestResults object of dtree model.

See also

[rx_predict](#), [rx_predict_rx_dforest](#).

Example

```
import os
from revoscalepy import rx_dforest, rx_import, RxOptions, RxXdfData
sample_data_path = RxOptions.get_option("sampleDataDir")
ds = RxXdfData(os.path.join(sample_data_path, "kyphosis.xdf"))
kyphosis = rx_import(input_data = ds)

# classification
formula = "Kyphosis ~ Number + Start"
method = "class"
parms = {'prior': [0.8, 0.2], 'loss': [0, 2, 3, 0], 'split': "gini"}

dforest = rx_dforest(formula, data = kyphosis, pweights = "Age", method = method,
                     parms = parms, cost = [2, 3], max_num_bins = 100, importance = True,
                     n_tree = 3, m_try = 2, sample_rate = 0.5,
                     replace = False, seed = 0, compute_oob_error = 1)

# regression
formula = "Age ~ Number + Start"
method = "anova"
parms = {'prior': [0.8, 0.2], 'loss': [0, 2, 3, 0], 'split': "gini"}

dforest = rx_dforest(formula, data = kyphosis, pweights = "Kyphosis", method = method,
                     parms = parms, cost = [2, 3], max_num_bins = 100, importance = True,
                     n_tree = 3, m_try = 2, sample_rate = 0.5,
                     replace = False, seed = 0, compute_oob_error = 1)
```

rx_dtrees

7/12/2022 • 7 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_dtrees(formula, data, output_file=None,  
                      output_column_name='rxNode', write_model_vars=False,  
                      extra_vars_to_write=None, overwrite=False, pweights=None, fweights=None,  
                      method=None, parms=None, cost=None, min_split=None, min_bucket=None,  
                      max_depth=10, cp=0, max_compete=0, max_surrogate=0, use_surrogate=2,  
                      surrogate_style=0, x_val=2, max_num_bins=None, max_unordered_levels=32,  
                      remove_missings=False, compute_obs_node_id=None, use_sparse_cube=False,  
                      find_splits_in_parallel=True, prune_cp=0, row_selection=None,  
                      transforms=None, transform_objects=None, transform_function=None,  
                      transform_variables=None, transform_packages=None, blocks_per_read=1,  
                      report_progress=2, verbose=0, compute_context=None, xdf_compression_level=1,  
                      **kwargs)
```

Description

Fit classification and regression trees on an .xdf file or data frame for small or large data using parallel external memory algorithm.

Arguments

formula

Statistical model using symbolic formulas.

data

Either a data source object, a character string specifying a '.xdf' file, or a data frame object. If a Spark compute context is being used, this argument may also be an RxHiveData, RxOrcData, RxParquetData or RxSparkDataFrame object or a Spark data frame object from pyspark.sql.DataFrame.

output_file

Either an RxXdfData data source object or a character string specifying the '.xdf' file for storing the resulting node indices. If None, then no node indices are stored to disk. If the input data is a data frame, the node indices are returned automatically.

output_column_name

Character string to be used as a column name for the resulting node indices if output_file is not None. Note that make.names is used on outColName to ensure that the column name is valid. If the output_file is an RxOdbcData source, dots are first converted to underscores. Thus, the default outColName becomes "X_rxNode".

write_model_vars

Bool value. If True, and the output file is different from the input file, variables in the model will be written to the output file in addition to the node numbers. If variables from the input data set are transformed in the model, the transformed variables will also be written out.

extra_vars_to_write

None or list of strings of additional variables names from the input data or transforms to include in the

`output_file`. If `writeModelVars` is True, model variables will be included as well.

overwrite

Bool value. If True, an existing `output_file` with an existing column named `outColName` will be overwritten.

pweights

Character string specifying the variable of numeric values to use as probability weights for the observations.

fweights

Character string specifying the variable of integer values to use as frequency weights for the observations.

method

Character string specifying the splitting method. Currently, only "class" or "anova" are supported. The default is "class" if the response is a factor, otherwise "anova".

parms

Optional list with components specifying additional parameters for the "class" splitting method, as follows:

`prior`: A vector of prior probabilities. The priors must be positive and sum to 1. The default priors are proportional to the data counts.

`loss`: A loss matrix, which must have zeros on the diagonal and positive off-diagonal elements. By default, the off-diagonal elements are set to 1.

`split`: The splitting index, either gini (the default) or information.

If `parms` is specified, any of the components can be specified or omitted. The defaults will be used for missing components.

cost

A vector of non-negative costs, containing one element for each variable in the model. Defaults to one for all variables. When deciding which split to choose, the improvement on splitting on a variable is divided by its cost.

min_split

The minimum number of observations that must exist in a node before a split is attempted. By default, this is $\sqrt{\text{num of obs}}$. For non-XDF data sources, as (num of obs) is unknown in advance, it is wisest to specify this argument directly.

min_bucket

The minimum number of observations in a terminal node (or leaf). By default, this is `min_split/3`.

cp

Numeric scalar specifying the complexity parameter. Any split that does not decrease overall lack-of-fit by at least `cp` is not attempted.

max_compete

The maximum number of competitor splits retained in the output. These are useful model diagnostics, as they allow you to compare splits in the output with the alternatives.

max_surrogate

The maximum number of surrogate splits retained in the output. Setting this to 0 can greatly improve the performance of the algorithm; in some cases almost half the computation time is spent in computing surrogate splits.

use_surrogate

An integer specifying how surrogates are to be used in the splitting process: 0: Display-only; observations with a missing value for the primary split variable are not sent further down the tree.

1: Use surrogates, in order, to split observations missing the primary split variable. If all surrogates are missing, the observation is not split.

2: Use surrogates, in order, to split observations missing the primary split variable. If all surrogates are missing or max_surrogate=0, send the observation in the majority direction.

The 0 value corresponds to the behavior of the tree function, and 2 (the default) corresponds to the recommendations of Breiman et al.

x_val

The number of cross-validations to be performed along with the model building. Currently, 1:x_val is repeated and used to identify the folds. If not zero, the cptable component of the resulting model will contain both the mean (xerror) and standard deviation (xstd) of the cross-validation errors, which can be used to select the optimal cost-complexity pruning of the fitted tree. Set it to zero if external cross-validation will be used to evaluate the fitted model because a value of k increases the compute time to $(k+1)$ -fold over a value of zero.

surrogate_style

An integer controlling selection of a best surrogate. The default, 0, instructs the program to use the total number of correct classifications for a potential surrogate, while 1 instructs the program to use the percentage of correct classification over the non-missing values of the surrogate. Thus, 0 penalizes potential surrogates with a large number of missing values.

max_depth

The maximum depth of any tree node. The computations take much longer at greater depth, so lowering max_depth can greatly speed up computation time.

max_num_bins

The maximum number of bins to use to cut numeric data. The default is $\min(1001, \max(101, \sqrt{\text{num of obs}}))$. For non-XDF data sources, as (num of obs) is unknown in advance, it is wisest to specify this argument directly. If set to 0, unit binning will be used instead of cutting.

max_unordered_levels

The maximum number of levels allowed for an unordered factor predictor for multiclass (>2) classification.

remove_missing

Bool value. If True, rows with missing values are removed and will not be included in the output data.

compute_obs_node_id

Bool value or None. If True, the tree node IDs for all the observations are computed and returned. If None, the IDs are computed for data.frame with less than 1000 observations and are returned as the where component in the fitted rxDTTree object.

use_sparse_cube

Bool value. If True, sparse cube is used.

find_splits_in_parallel

Bool value. If True, optimal splits for each node are determined using parallelization methods; this will typically speed up computation as the number of nodes on the same level is increased.

prune_cp

Optional complexity parameter for pruning. If prune_cp > 0, prune.rxDTTree is called on the completed tree with the specified prune_cp and the pruned tree is returned. This contrasts with the cp parameter that determines which splits are considered in growing the tree. The option prune_cp="auto" causes rxDTTree to call the function rxDTTreeBestCp on the completed tree, then use its return value as the cp value for prune.rxDTTree.

row_selection

None. Not currently supported, reserved for future use.

transform_objects

A dictionary of variables besides the data that are used in the transform function. See rx_data_step for examples.

transform_function

Name of the function that will be used to modify the data before the model is built. The variables used in the transformation function must be specified in transform_objects. See rx_data_step for examples.

transform_variables

List of strings of the column names needed for the transform function.

transform_packages

None. Not currently supported, reserved for future use.

blocks_per_read

number of blocks to read for each chunk of data read from the data source.

report_progress

Integer value with options: 0: No progress is reported. 1: The number of processed rows is printed and updated. 2: Rows processed and timings are reported. 3: Rows processed and all timings are reported.

verbose

Integer value. If 0, no additional output is printed. If 1, additional summary information is printed.

compute_context

A RxComputeContext object for prediction.

kwargs

Additional parameters

Returns

A RxDTreeResults object of dtree model.

See also

[rx_predict](#) , [rx_predict_rx_dtreetree](#) .

Example

```

import os
from revoscalepy import rx_dtree, rx_import, RxOptions, RxXdfData
sample_data_path = RxOptions.get_option("sampleDataDir")
ds = RxXdfData(os.path.join(sample_data_path, "kyphosis.xdf"))
kyphosis = rx_import(input_data = ds)

# classification
formula = "Kyphosis ~ Number + Start"
method = "class"
parms = {'prior': [0.8, 0.2], 'loss': [0, 2, 3, 0], 'split': "gini"}
cost = [2,3]
dtree = rx_dtree(formula, data = kyphosis, pweights = "Age", method = method, parms = parms, cost = cost,
max_num_bins = 100)

# regression
formula = "Age ~ Number + Start"
method = "anova"
parms = {'prior': [0.8, 0.2], 'loss': [0, 2, 3, 0], 'split': "gini"}
cost = [2,3]
dtree = rx_dtree(formula, data = kyphosis, pweights = "Kyphosis", method = method, parms = parms, cost = cost,
max_num_bins = 100)

# transform function
def my_transform(dataset, context):
    dataset['arrdelay2'] = dataset['ArrDelay'] * 10
    dataset['crsdeptime2'] = dataset['CRSDepTime']
    # Use the follow code to set high/low values for new columns
    # rx_attributes metadata needs to be set last
    dataset['arrdelay2'].rx_attributes = {'rxLowHigh': [-860.0, 14900.0]}
    dataset['crsdeptime2'].rx_attributes = {'rxLowHigh': [0.016666999086737633, 23.983333587646484]}
    return dataset

data_path = RxOptions.get_option("sampleDataDir")
data = RxXdfData(os.path.join(data_path, "AirlineDemoSmall.xdf")).head(20)
form = "ArrDelay ~ arrdelay2 + crsdeptime2"
dtree = rx_dtree(form, data=data, transform_function=my_transform, transform_variables=["ArrDelay",
"CRSDepTime", "DayOfWeek"])

```

rx_exec

7/12/2022 • 4 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_exec(function: typing.Callable, args: typing.Union[dict,
    typing.List[dict]] = None, times_to_run: int = -1,
    task_chunk_size: int = None, compute_context=None,
    local_state: dict = {}, callback: typing.Callable = None,
    continue_on_failure: bool = True, **kwargs)
```

Description

Allows the distributed execution of an arbitrary R function in parallel, across nodes (computers) or cores of a “compute context”, such as a cluster. For example, you could use the function to execute multiple instances of a model concurrently. When used with base R functions, rxexec does not lift the memory constraints of the underlying functions, or make a single-threaded process a multi-threaded one. The fundamentals of the function still apply; what changes is the execution framework of those functions.

Arguments

function

The function to be executed; the nodes or cores on which it is run are determined by the currently-active compute context and by the other arguments of rx_exec.

args

Arguments passed to the function ‘function’ each time it is executed. for multiple executions with different parameters, args should be a list where each element is a dict. each dict element contains the set of named parameters to pass the function on each execution.

times_to_run

Specifies the number of times ‘function’ should be executed. in case of different parameters for each execution, the length of args must equal times_to_run. in case where length of args equals 1 or args is not specified, function will be called times_to_run number of times with the same arguments.

task_chunk_size

Specifies the number of tasks to be executed per compute element. by submitting tasks in chunks, you can avoid some of the overhead of starting new python processes over and over. for example, if you are running thousands of identical simulations on a cluster, it makes sense to specify the task_chunk_size so that each worker can do its allotment of tasks in a single python process.

compute_context

A RxComputeContext object.

local_state

Dictionary with variables to be passed to a RxInSqlServer compute context not supported in RxSpark compute context.

callback

An optional function to be called with the results from each execution of ‘function’

continue_on_failure

None or logical value. If True, the default, then if an individual instance of a job fails due to a hardware or network failure, an attempt will be made to rerun that job. (Python errors, however, will cause immediate failure as usual.) Furthermore, should a process instance of a job fail due to a user code failure, the rest of the processes will continue, and the failed process will produce a warning when the output is collected. Additionally, the position in the returned list where the failure occurred will contain the error as opposed to a result.

****kwargs**

Contains named arguments to be passed to function. Alternative to using args param. For a function foo that takes two arguments named 'x' and 'y', kwargs can be used like: rx_exec(function=foo, x=5, y=4) (for a single execution with x=5, y=4) rx_exec(function=foo, x=[1,2,3,4,5], y=6) (for multiple executions with different values of x and y=6) rx_exec(function=foo, x=[1,2,3], y=[4,5,6]) (for multiple executions with different values of x and y) Note: if you want to pass in a list as an argument, you must wrap it in another list. For a function foobar that takes an argument named 'the_list', kwargs can be used like: rx_exec(function=foobar, the_list=[[0,1,2,3]]) (single execution with the_list = [0,1,2,3]) rx_exec(function=foobar, the_list=[[0,1,2],[3,4,5],[6,7,8]]) (multiple executions)

Returns

If local compute context or a waiting compute context is active, a list containing the return value of each execution of 'function' with specified parameters. If a non-waiting compute context is active, a jobInfo object, or a list of a jobInfo objects for multiple executions.

See also

[RxComputeContext](#) , [RxLocalSeq](#) , [RxLocalParallel](#) , [RxInSqlServer](#) , [rx_get_compute_context](#) , [rx_set_compute_context](#) .

Example

```
###  
# Local parallel rx_exec with different parameters for each execution  
###  
import os  
from revoscalepy import RxLocalParallel, rx_set_compute_context, rx_exec, rx_btrees, RxOptions, RxXdfData  
  
sample_data_path = RxOptions.get_option("sampleDataDir")  
in_mort_ds = RxXdfData(os.path.join(sample_data_path, "mortDefaultSmall.xdf"))  
rx_set_compute_context(RxLocalParallel())  
formula = "creditScore ~ yearsEmploy"  
args = [  
    {'data' : in_mort_ds, 'formula' : formula, 'n_tree': 3},  
    {'data' : in_mort_ds, 'formula' : formula, 'n_tree': 5},  
    {'data' : in_mort_ds, 'formula' : formula, 'n_tree': 7}  
]  
  
models = rx_exec(rx_btrees, args = args)  
# Alternatively  
models = rx_exec(rx_btrees, data=in_mort_ds, formula=formula, n_tree=[3,5,7])  
  
###  
## SQL rx_exec  
###  
  
from revoscalepy import RxSqlServerData, RxInSqlServer, rx_exec  
formula = "ArrDelay ~ CRSDepTime + DayOfWeek"  
connection_string="Driver=SQL Server;Server=.;Database=RevoTestDB;Trusted_Connection=TRUE"  
query="select top 100 [ArrDelay],[CRSDepTime],[DayOfWeek] FROM airlinedemosmall"
```

```

ds = RxSqlServerData(sql_query = query, connection_string = connection_string)
cc = RxInSqlServer(
    connection_string = connection_string,
    num_tasks = 1,
    auto_cleanup = False,
    console_output = True,
    execution_timeout_seconds = 0,
    wait = True
)

def remote_call(dataset):
    from revoscalepy import rx_data_step
    df = rx_data_step(dataset)
    return len(df)

results = rx_exec(function = remote_call, args={'dataset': ds}, compute_context = cc)
print(results)

###  

## Run rx_exec in RxSpark compute context  

###  

from revoscalepy import *

# start Spark app
spark_cc = rx_spark_connect()

# define function to conditional check
def my_check_fun(id_param):
    if id_param == 13:
        raise Exception("Ensure to fail")
    return id_param

# prepare args for each run with total 20 runs
elem_arg = []
for i in range(0,20):
    elem_arg.append({"id_param": i})

# get result and print
results = rx_exec(my_check_fun, elem_arg, continue_on_failure=True)
print(results)

# stop Spark app
rx_spark_disconnect(spark_cc)

## End(Not run)

```

rx_exec_by

7/12/2022 • 3 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_exec_by(input_data: typing.Union[revoscalepy.datasource.RxDataSource.RxDataSource,
    pandas.core.frame.DataFrame, str], keys: typing.List[str] = None,
    function: typing.Callable = None,
    function_parameters: dict = None,
    filter_function: typing.Callable = None,
    compute_context: revoscalepy.computecontext.RxComputeContext.RxComputeContext = None,
    **kwargs) -> revoscalepy.functions.RxExecBy.RxExecByResults
```

Description

Partition input data source by keys and apply a user-defined function on individual partitions. If the input data source is already partitioned, apply a user-defined function directly on the partitions. Currently supported in local, localpar, RxInSqlServer and RxSpark compute contexts.

Arguments

input_data

A data source object supported in currently active compute context, e.g. 'RxSqlServerData' for 'RxInSqlServer'. In 'RxLocalSeq' and 'RxLocalParallel', a character string specifying a '.xdf' file, or a data frame object can be also used. If a Spark compute context is being used, this argument may also be an RxHiveData, RxOrcData, RxParquetData or RxSparkDataFrame object or a Spark data frame object from `pyspark.sql.DataFrame`.

keys

List of strings of variable names to be used for partitioning the input data set.

function

The user function to be executed. The user function takes 'keys' and 'data' as two required input arguments where 'keys' determines the partitioning values and 'data' is a data source object of the corresponding partition. 'data' can be a RxXdfData object or a RxODBCData object, which can be transformed to a pandas data frame by using `rx_data_step`. 'keys' is a dict where keys of the dict are variable names and values are variable values of the partition. The nodes or cores on which it is running are determined by the currently active compute context.

function_parameters

A dict which defines a list of additional arguments for the user function func.

filter_function

An user function that takes a Panda data frame of keys/values as an input argument, applies filter to the keys/values and returns a data frame containing rows whose keys/values satisfy the filter conditions. The input data frame has similar format to the results returned by `rx_partition` which comprises of partitioning variables and an additional variable of partition data source. This `filter_function` allows user to control what data partitions to be applied by the user function 'function'. 'filter_function' currently is not supported in RxHadoopMR and RxSpark compute contexts.

compute_context

A RxComputeContext object.

kwargs

Additional arguments.

Returns

A RxExecByResults object inherited from dataframe with each row to be the result of each partition. The indexes of dataframe are keys, columns are 'result' and 'status'. 'result': the object returned from the user function from each partition. 'status': 'OK', if the user function on each partition runs success, otherwise, the exception object.

See also

[rx_partition](#) . [RxXdfData](#) .

Example

```
###  
# Run rx_exec_by in local compute context  
###  
import os  
from revoscalepy import RxLocalParallel, rx_set_compute_context, rx_exec_by, RxOptions, RxXdfData  
data_path = RxOptions.get_option("sampleDataDir")  
  
input_file = os.path.join(data_path, "claims.xdf")  
input_ds = RxXdfData(input_file)  
  
# count number of rows  
def count(data, keys):  
    from revoscalepy import rx_data_step  
    df = rx_data_step(data)  
    return len(df)  
  
rx_set_compute_context(RxLocalParallel())  
local_cc_results = rx_exec_by(input_data = input_ds, keys = ["car.age", "type"], function = count)  
print(local_cc_results)  
  
###  
# Run rx_exec_by in SQL Server compute context with data table specified  
###  
from revoscalepy import RxInSqlServer, rx_set_compute_context, rx_exec_by, RxOptions, RxSqlServerData  
connection_string = 'Driver=SQL Server;Server=.;Database=RevoTestDb;Trusted_Connection=True;'  
ds = RxSqlServerData(table = "AirlineDemoSmall", connection_string=connection_string)  
  
def count(keys, data):  
    from revoscalepy import rx_import  
    df = rx_import(data)  
    return len(df)  
  
# filter function  
def filter_weekend(partitioned_df):  
    return (partitioned_df.loc[(partitioned_df.DayOfWeek == "Saturday") | (partitioned_df.DayOfWeek == "Sunday")])  
  
sql_cc = RxInSqlServer(connection_string = connection_string, num_tasks = 4)  
rx_set_compute_context(sql_cc)  
sql_cc_results = rx_exec_by(ds, keys = ["DayOfWeek"], function = count, filter_function = filter_weekend)  
print(sql_cc_results)  
  
###  
# Run rx_exec_by in RxSpark compute context  
###  
from revoscalepy import *  
  
# start Spark app  
spark_cc = rx_spark_connect()
```

```
spark_cc = RxSparkContext()
```

```
# define function to compute average delay
def average_delay(keys, data):
    df = rx_data_step(data)
    return df['ArrDelay'].mean(skipna=True)

# define colInfo
col_info = {
    'ArrDelay': {'type': "numeric"},
    'CRSDepTime': {'type': "numeric"},
    'DayOfWeek': {'type': "string"}
}

# create text data source with airline data
text_data = RxTextData(
    file = "/share/sample_data/AirlineDemoSmall.csv",
    first_row_is_column_names = True,
    column_info = col_info,
    file_system = RxHdfsFileSystem())

# group text_data by day of week and get average delay on each day
result = rx_exec_by(text_data, keys = ["DayOfWeek"], function = average_delay)

# get result in pandas dataframe format and sort multiindex
result_dataframe = result.to_dataframe()
result_dataframe.sort_index()
print(result_dataframe)

# stop Spark app
rx_spark_disconnect(spark_cc)
```

rx_get_compute_context

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_get_compute_context() -> revoscalepy.computecontext.RxComputeContext.RxComputeContext
```

Description

Gets the active compute context for revoscalepy computations

Arguments

compute_context

Character string specifying class name or description of the specific class to instantiate, or an existing RxComputeContext object. Choices include: "RxLocalSeq" or "local", "RxInSqlServer".

Returns

rx_set_compute_context returns the previously active compute context invisibly. rx_get_compute_context returns the active compute context.

See also

[RxComputeContext](#) , [RxLocalSeq](#) , [RxInSqlServer](#) , [rx_set_compute_context](#) .

Example

```
from revoscalepy import RxLocalSeq, RxInSqlServer, rx_get_compute_context, rx_set_compute_context

local_cc = RxLocalSeq()
sql_server_cc = RxInSqlServer('Driver=SQL Server;Server=.;Database=RevoTestDb;Trusted_Connection=True;')
previous_cc = rx_set_compute_context(sql_server_cc)
rx_get_compute_context()
```

rx_get_info

7/12/2022 • 3 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_get_info(data, get_var_info: bool = False,  
                        get_block_sizes: bool = False, get_value_labels: bool = None,  
                        vars_to_keep: list = None, vars_to_drop: list = None,  
                        start_row: int = 1, num_rows: int = 0,  
                        compute_info: bool = False, all_nodes: bool = False,  
                        verbose: int = 0)
```

Description

Get basic information about a revoscalepy data source or data frame.

Arguments

data

A data frame, a character string specifying an ".xdf", or an RxDataSource object. If a local compute context is being used, this argument may also be a list of data sources, in which case the output will be returned in a named list. See the details section for more information. If a Spark compute context is being used, this argument may also be an RxHiveData, RxOrcData, RxParquetData or RxSparkDataFrame object or a Spark data frame object from pyspark.sql.DataFrame.

get_var_info

Bool value. If True, variable information is returned.

get_block_sizes

Bool value. If True, block sizes are returned in the output for an ".xdf" file, and when printed the first 10 block sizes are shown.

get_value_labels

Bool value. If True and get_var_info is True or None, factor value labels are included in the output.

vars_to_keep

List of strings of variable names for which information is returned. If None or get_var_info is False, argument is ignored. Cannot be used with vars_to_drop.

vars_to_drop

List of strings of variable names for which information is not returned. If None or get_var_info is False, argument is ignored. Cannot be used with vars_to_keep.

start_row

Starting row for retrieval of data of a data frame or ".xdf" file.

num_rows

Number of rows of data to retrieve of a data frame or ".xdf" file.

compute_info

Bool value. If True, and get_var_info is True, variable information (e.g., high/low values) for non-xdf data sources will be computed by reading through the data set. If True, and get_var_info is False, the number of variables will be gotten from non-xdf data sources (but not the number of rows).

all_nodes

Bool value. Ignored if the active RxComputeContext compute context is local or RxForEachDoPar. Otherwise, if True, a list containing the information for the data set on each node in the active compute context will be returned. If False, only information on the data set on the master node will be returned. Note that the determination of the master node is not controlled by the end user.

verbose

Integer value. If 0, no additional output is printed. If 1, additional summary information is printed for an ".xdf" file.

Returns

List containing the following possible elements:

```
fileName: Character string containing the file name and path (if an  
".xdf" file).  
  
objName: Character string containing object name (if not an ".xdf" file).  
class: Class of the object.  
length: Length of the object if it is not an ".xdf" file or data frame.  
numCompositeFiles: Number of composite data files(if a composite ".xdf" file).  
  
 numRows: Number of rows in the data set.  
 numVars: Number of variables in the data set.  
 numBlocks: Number of blocks in the data set.  
 varInfo: List of variable information where each element is a list  
  
     describing a variable.  
  
 rowsPerBlock: Integer vector containing number of rows in each block  
 (if get_block_sizes is set to True). Set to None if data is a data frame.  
  
 data: Data frame containing the data (if num_rows > 0)
```

See also

[rx_data_step](#), [rx_get_var_info](#).

Example

```
import os  
from revoscalepy import RxOptions, RxXdfData, rx_get_info  
  
sample_data_path = RxOptions.get_option("sampleDataDir")  
claims_ds = RxXdfData(os.path.join(sample_data_path, "claims.xdf"))  
info = rx_get_info(claims_ds, get_var_info=True)  
print(info)
```

Output:

```
File name:C:\swarm\workspace\bigAnalytics-9.4.0\python\revoscalepy\revoscalepy\data\sample_data\claims.xdf
Number of observations:128.0
Number of variables:6.0
Number of blocks:1.0
Compression type:zlib
Variable information:
Var 1: RowNum, Type: integer, Storage: int32, Low/High: (1.0000, 128.0000)
Var 2: age
  8 factor levels: ['17-20', '21-24', '25-29', '30-34', '35-39', '40-49', '50-59', '60+']
Var 3: car.age
  4 factor levels: ['0-3', '4-7', '8-9', '10+']
Var 4: type
  4 factor levels: ['A', 'B', 'C', 'D']
Var 5: cost, Type: numeric, Storage: float32, Low/High: (11.0000, 850.0000)
Var 6: number, Type: numeric, Storage: float32, Low/High: (0.0000, 434.0000)
```

rx_get_job_info

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_get_job_info(job) -> revoscalepy.computecontext.RxRemoteJob.RxRemoteJob
```

Description

The object returned from a non-waiting, distributed computation contains job information together with other information. The job information is used internally by such functions as *rx_get_job_status*, *rx_get_job_output*, and *rx_get_job_results*. It is sometimes useful to extract it for its own sake, as it contains complete information on the job's compute context as well as other information needed by the distributed computing resources.

For most users, the principal use of this function is to determine whether a given object actually contains job information. If the return value is not *None*, then the object contains job information. Note, however, that the structure of the job information is subject to change, so code that attempts to manipulate it directly is not guaranteed to be forward-compatible.

Arguments

job

An object containing jobInfo information, such as that returned from a non-waiting, distributed computation.

Returns

The job information, if present, or None.

See also

[rx_get_job_status](#)

Example

```
from revoscalepy import RxInSqlServer
from revoscalepy import rx_exec
from revoscalepy import rx_get_job_info

connection_string = 'Driver=SQL Server;Server=.;Database=RevoTestDb;Trusted_Connection=True;'

# Setting wait to False allows the job to be run asynchronously
# Setting console_output to True allows us to get the console output of the distributed computing job
compute_context = RxInSqlServer(connection_string=connection_string,
                                 num_tasks=1,
                                 console_output=True,
                                 wait=False)

def hello_from_sql():
    import time
    print('Hello from SQL server')
    time.sleep(3)
    return 'We just ran Python code asynchronously on a SQL server!'

job = rx_exec(function=hello_from_sql, compute_context=compute_context)

#Get the job information
info = rx_get_job_info(job)
print(info)
```

rx_get_job_output

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_get_job_output(job_info: revoscalepy.computecontext.RxRemoteJob.RxRemoteJob) -> str
```

Description

During a job run, the state of the output is non-deterministic (that is, it may or may not be on disk, and what is on disk at any given point in time may not reflect the actual completion state of a job).

If *auto_cleanup* has been set to *True* on the distributed computing job's compute context, the console output will not persist after the job completes.

Unlike *rx_get_job_results*, this function does not remove any job information upon retrieval.

Arguments

job_info

The distributed computing job for which to retrieve the job output

Returns

str that contains the console output for the nodes participating in the distributed computing job

See also

[rx_get_job_status](#)

Example

```
from revoscalepy import RxInSqlServer
from revoscalepy import rx_exec
from revoscalepy import rx_get_job_output
from revoscalepy import rx_wait_for_job

connection_string = 'Driver=SQL Server;Server=.;Database=RevoTestDb;Trusted_Connection=True;'

# Setting wait to False allows the job to be run asynchronously
# Setting console_output to True allows us to get the console output of the distributed computing job
compute_context = RxInSqlServer(connection_string=connection_string,
                                 num_tasks=1,
                                 console_output=True,
                                 wait=False)

def hello_from_sql():
    import time
    print('Hello from SQL server')
    time.sleep(3)
    return 'We just ran Python code asynchronously on a SQL server!'

job = rx_exec(function=hello_from_sql, compute_context=compute_context)

# Wait for the job to finish or fail whatever the case may be
rx_wait_for_job(job)

# Print out what our code printed on sql
output = rx_get_job_output(job)
print(output)
```

rx_get_job_results

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_get_job_results(job_info: revoscalepy.computecontext.RxRemoteJob.RxRemoteJob,  
    console_output: bool = None,  
    auto_cleanup: bool = None) -> list
```

Description

Obtain distributed computing results and processing status.

Arguments

job_info

A job object as returned by rx_exec or a revoscalepy analysis function, if available.

console_output

None or bool value. If True, the console output from all of the processes is printed to the user console. If False, no console output is displayed. Output can be retrieved with the function rxGetJobOutput for a non-waiting job. If not None, this flag overrides the value set in the compute context when the job was submitted. If None, the setting in the compute context will be used.

auto_cleanup

None or bool value. If True, the default behavior is to clean up any artifacts created by the distributed computing job. If False, then the artifacts are not deleted, and the results may be acquired using rxGetJobResults, and the console output via rxGetJobOutput until the rxCleanupJobs is used to delete the artifacts. If not None, this flag overwrites the value set in the compute context when the job was submitted. If you routinely set auto_cleanup=False, you will eventually fill your hard disk with compute artifacts. If you set auto_cleanup=True and experience performance degradation on a Windows XP client, consider setting auto_cleanup=False.

Returns

Either the results of the run (prepended with console output if the console_output argument is set to True) or a message saying that the results are not available because the job has not finished, has failed, or was deleted.

See also

[rx_get_job_status](#)

Example

```
from revoscalepy import RxInSqlServer
from revoscalepy import rx_exec
from revoscalepy import rx_wait_for_job
from revoscalepy import rx_get_job_results

connection_string = 'Driver=SQL Server;Server=.;Database=RevoTestDb;Trusted_Connection=True;'

# Setting wait to False allows the job to be run asynchronously
# Setting console_output to True allows us to get the console output of the distributed computing job
compute_context = RxInSqlServer(connection_string=connection_string,
                                 num_tasks=1,
                                 console_output=True,
                                 wait=False)

def hello_from_sql():
    import time
    print('Hello from SQL server')
    time.sleep(3)
    return 'We just ran Python code asynchronously on a SQL server!'

job = rx_exec(function=hello_from_sql, compute_context=compute_context)

# Wait for the job to finish
rx_wait_for_job(job)

# Print out what we returned from SQL
results = rx_get_job_results(job)
print(results)
```

rx_get_jobs

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_get_jobs(compute_context:  
    revoscalepy.computecontext.RxRemoteComputeContext,  
    exact_match: bool = False,  
    start_time: <module 'datetime' from 'C:\\swarm\\workspace\\bigAnalytics-  
9.4.0\\runtime\\Python\\lib\\datetime.py'> = None,  
    end_time: <module 'datetime' from 'C:\\swarm\\workspace\\bigAnalytics-  
9.4.0\\runtime\\Python\\lib\\datetime.py'> = None,  
    states: list = None, verbose: bool = True) -> list
```

Description

Returns a list of job objects associated with the given compute context and matching the specified parameters.

Arguments

compute_context

A compute context object.

exact_match

Determines if jobs are matched using the full compute context, or a simpler subset. If True, only jobs which use the same context object are returned. If False, all jobs which have the same headNode (if available) and ShareDir are returned.

start_time

A time, specified as a POSIXct object. If specified, only jobs created at or after start_time are returned.

end_time

A time, specified as a POSIXct object. If specified, only jobs created at or before end_time are returned.

states

If specified (as a list of strings of states that can include "none", "finished", "failed", "canceled", "undetermined" "queued" or "running"), only jobs in those states are returned. Otherwise, no filtering is performed on job state.

verbose

If True (the default), a brief summary of each job is printed as it is found. This includes the current job status as returned by rx_get_job_status, the modification time of the job, and the current job ID (this is used as the component name in the returned list of job information objects). If no job status is returned, the job status shows none.

Returns

Returns a list of job information objects based on the compute context.

See also

Example

```
from revoscalepy import RxInSqlServer
from revoscalepy import rx_exec
from revoscalepy import rx_get_jobs

connection_string = 'Driver=SQL Server;Server=.;Database=RevoTestDb;Trusted_Connection=True;'

# Setting wait to False allows the job to be run asynchronously
# Setting console_output to True allows us to get the console output of the distributed computing job
compute_context = RxInSqlServer(connection_string=connection_string,
                                 num_tasks=1,
                                 console_output=True,
                                 wait=False)

def hello_from_sql():
    import time
    print('Hello from SQL server')
    time.sleep(3)
    return 'We just ran Python code asynchronously on a SQL server!'

job = rx_exec(function=hello_from_sql, compute_context=compute_context)

job_list = rx_get_jobs(compute_context)
print(job_list)
```

rx_get_job_status

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_get_job_status(job_info: revoscalepy.computecontext.RxRemoteJob.RxRemoteJob) ->
revoscalepy.computecontext.RxJob.RxRemoteJobStatus
```

Description

Obtain distributed computing processing status for the specified job.

Arguments

job_info

A job object as returned by rx_exec or a revoscalepy analysis function, if available.

Returns

A RxRemoteJobStatus enumeration value that designates the status of the remote job

See also

[rx_get_job_results](#) [RxRemoteJobStatus](#)

Example

```
from revoscalepy import RxInSqlServer
from revoscalepy import rx_exec
from revoscalepy import rx_get_job_status
from revoscalepy import rx_wait_for_job

connection_string = 'Driver=SQL Server;Server=.;Database=RevoTestDb;Trusted_Connection=True;'

# Setting wait to False allows the job to be run asynchronously
# Setting console_output to True allows us to get the console output of the distributed computing job
compute_context = RxInSqlServer(connection_string=connection_string,
                                 num_tasks=1,
                                 console_output=True,
                                 wait=False)

def hello_from_sql():
    import time
    print('Hello from SQL server')
    time.sleep(3)
    return 'We just ran Python code asynchronously on a SQL server!'

job = rx_exec(function=hello_from_sql, compute_context=compute_context)

# Poll initial status
status = rx_get_job_status(job)

print(status)

# Wait for the job to finish or fail whatever the case may be
rx_wait_for_job(job)

# Poll final status
status = rx_get_job_status(job)

print(status)
```

rx_get_partitions

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_get_partitions(input_data: revoscalepy.datasource.RxXdfData.RxXdfData = None,  
**kwargs)
```

Description

Get partitions enumeration of a partitioned .xdf file data source.

Arguments

input_data

An existing partitioned data source object which was created by RxXdfData with create_partition_set = True and constructed by rx_partition.

Returns

A Pandas data frame with (n+1) columns, the first n columns are partitioning columns specified by vars_to_partition in rx_partition and the (n+1)th column is a data source column where each row contains an Xdf data source object of one partition.

See also

[rx_exec_by](#) . [RxXdfData](#) . [rx_partition](#) .

Example

```
import os, tempfile  
from revoscalepy import RxOptions, RxXdfData, rx_partition, rx_get_partitions  
data_path = RxOptions.get_option("sampleDataDir")  
  
# input xdf data source  
xdf_file = os.path.join(data_path, "claims.xdf")  
xdf_ds = RxXdfData(xdf_file)  
  
# create a partitioned xdf data source object  
out_xdf_file = os.path.join(tempfile._get_default_tempdir(), "outPartitions")  
out_xdf = RxXdfData(out_xdf_file, create_partition_set = True)  
  
# do partitioning for input data set  
partitions = rx_partition(input_data = xdf_ds, output_data = out_xdf, vars_to_partition =  
["car.age","type"], append = "none", overwrite = True)  
  
# use rx_get_partitions to load an existing partitioned xdf  
out_xdf_1 = RxXdfData(out_xdf_file)  
partitions_1 = rx_get_partitions(out_xdf_1)  
print(partitions_1)
```

rx_get_pyspark_connection

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_get_pyspark_connection(compute_context: revoscalepy.computecontext.RxSpark.RxSpark)
```

Description

Gets a PySpark connection from the current Spark compute context.

Arguments

compute_context

Compute context get created by rx_spark_connect.

Returns

Object of python.context.SparkContext.

Example

```
from revoscalepy import rx_spark_connect, rx_get_pyspark_connection
from pyspark.sql import SparkSession
cc = rx_spark_connect(interop = "pyspark")
sc = rx_get_pyspark_connection(cc)
spark = SparkSession(sc)
df = spark.createDataFrame([('Monday',3012),('Friday',1354),('Sunday',5452)], ["DayOfWeek", "Sale"])
```

rx_get_var_info

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_get_var_info(data, get_value_labels: bool = True,  
    vars_to_keep: list = None, vars_to_drop: list = None,  
    compute_info: bool = False, all_nodes: bool = False)
```

Description

Get variable information for a revoscalepy data source or data frame, including variable names, descriptions, and value labels

Arguments

data

a data frame, a character string specifying an ".xdf", or an RxDataSource object. If a local compute context is being used, this argument may also be a list of data sources, in which case the output will be returned in a named list. See the details section for more information. If a Spark compute context is being used, this argument may also be an RxHiveData, RxOrcData, RxParquetData or RxSparkDataFrame object or a Spark data frame object from pyspark.sql.DataFrame.

get_value_labels

bool value. If True and get_var_info is True or None, factor value labels are included in the output.

vars_to_keep

list of strings of variable names for which information is returned. If None or get_var_info is False, argument is ignored. Cannot be used with vars_to_drop.

vars_to_drop

list of strings of variable names for which information is not returned. If None or get_var_info is False, argument is ignored. Cannot be used with vars_to_keep.

compute_info

bool value. If True, variable information (e.g., high/low values) for non-xdf data sources will be computed by reading through the data set.

all_nodes

bool value. Ignored if the active RxComputeContext compute context is local or RxForeachDoPar. Otherwise, if True, a list containing the information for the data set on each node in the active compute context will be returned. If False, only information on the data set on the master node will be returned. Note that the determination of the master node is not controlled by the end user.

Returns

list with named elements corresponding to the variables in the data set.

Each list element is also a list with following possible elements:

```
description: character string specifying the variable description
varType: character string specifying the variable type
storage: character string specifying the storage type
low: numeric giving the low values, possibly generated through a
     temporary factor transformation F()

high: numeric giving the high values, possibly generated through a
     temporary factor transformation F()

levels: (factor only) a list of strings containing the factor levels
valueInfoCodes: list of strings of value codes, for informational
purposes only

valueInfoLabels: list of strings of value labels that is the same
length as valueInfoCodes, used for informational purposes only
```

See also

[rx_data_step](#), [rx_get_info](#).

Example

```
import os
from revoscalepy import RxOptions, RxXdfData, rx_get_var_info

sample_data_path = RxOptions.get_option("sampleDataDir")
claims_ds = RxXdfData(os.path.join(sample_data_path, "claims.xdf"))
info = rx_get_var_info(claims_ds)
print(info)
```

Output:

```
Var 1: RowNum, Type: integer, Storage: int32, Low/High: (1.0000, 128.0000)
Var 2: age
  8 factor levels: ['17-20', '21-24', '25-29', '30-34', '35-39', '40-49', '50-59', '60+']
Var 3: car.age
  4 factor levels: ['0-3', '4-7', '8-9', '10+']
Var 4: type
  4 factor levels: ['A', 'B', 'C', 'D']
Var 5: cost, Type: numeric, Storage: float32, Low/High: (11.0000, 850.0000)
Var 6: number, Type: numeric, Storage: float32, Low/High: (0.0000, 434.0000)
```

rx_get_var_names

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_get_var_names(data)
```

Description

Read the variable names for data source or data frame

Arguments

data

an RxDataSource object, a character string specifying the ".xdf" file, or a data frame. If a Spark compute context is being used, this argument may also be an RxHiveData, RxOrcData, RxParquetData or RxSparkDataFrame object or a Spark data frame object from pyspark.sql.DataFrame.

Returns

list of strings containing the names of the variables in the data source or data frame.

See also

[rx_data_step](#), [rx_get_info](#), [rx_get_var_info](#).

Example

```
import os
from revoscalepy import RxOptions, RxXdfData, rx_get_var_names

sample_data_path = RxOptions.get_option("sampleDataDir")
claims_ds = RxXdfData(os.path.join(sample_data_path, "claims.xdf"))
info = rx_get_var_names(claims_ds)
print(info)
```

Output:

```
RowNum age car.age type cost number
```

rx_import

7/12/2022 • 8 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_import(input_data: typing.Union[revoscalepy.datasource.RxDataSource.RxDataSource,
    pandas.core.frame.DataFrame, str], output_file=None,
    vars_to_keep: list = None, vars_to_drop: list = None,
    row_selection: str = None, transforms: dict = None,
    transform_objects: dict = None, transform_function: <built-
    in function callable> = None,
    transform_variables: dict = None,
    transform_packages: dict = None, append: str = None,
    overwrite: bool = False, number_rows: int = None,
    strings_as_factors: bool = None, column_classes: dict = None,
    column_info: dict = None, rows_per_read: int = None,
    type: str = None, max_rows_by_columns: int = None,
    report_progress: int = None, verbose: int = None,
    xdf_compression_level: int = None,
    create_composite_set: bool = None,
    blocks_per_composite_file: int = None)
```

Description

Import data and store as an .xdf file on disk or in-memory as a data.frame object.

Arguments

input_data

A character string with the path for the data to import (delimited, fixed format, ODBC, or XDF). Alternatively, a data source object representing the input data source can be specified. If a Spark compute context is being used, this argument may also be an RxHiveData, RxOrcData, RxParquetData or RxSparkDataFrame object or a Spark data frame object from pyspark.sql.DataFrame.

output_file

A character string representing the output '.xdf' file or an RxXdfData object. If None, a data frame will be returned in memory. If a Spark compute context is being used, this argument may also be an RxHiveData, RxOrcData, RxParquetData or RxSparkDataFrame object.

vars_to_keep

List of strings of variable names to include when reading from the input data file. If None, argument is ignored. Cannot be used with vars_to_drop. Not supported for ODBC or fixed format text files.

vars_to_drop

List of strings of variable names to exclude when reading from the input data file. If None, argument is ignored. Cannot be used with vars_to_keep. Not supported for ODBC or fixed format text files.

row_selection

None. Not currently supported, reserved for future use.

transforms

None. Not currently supported, reserved for future use.

transform_objects

A dictionary of variables besides the data that are used in the transform function. See `rx_data_step` for examples.

transform_function

Name of the function that will be used to modify the data. The variables used in the transformation function must be specified in `transform_objects`. See `rx_data_step` for examples.

transform_variables

List of strings of the column names needed for the transform function.

transform_packages

None. Not currently supported, reserved for future use.

append

Either "none" to create a new '.xdf' file or "rows" to append rows to an existing '.xdf' file. If `output_file` exists and `append` is "none", the `overwrite` argument must be set to True. Ignored if a data frame is returned.

overwrite

Bool value. If True, the existing `output_file` will be overwritten. Ignored if a data frame is returned.

number_rows

Integer value specifying the maximum number of rows to import. If set to -1, all rows will be imported.

strings_as_factors

Bool value indicating whether or not to automatically convert strings to factors on import. This can be overridden by specifying "character" in `column_classes` and `column_info`. If True, the factor levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use `column_info` with specified "levels".

column_classes

Dictionary of column name to strings specifying the column types to use when converting the data. The element names for the vector are used to identify which column should be converted to which type.

Allowable column types are: "bool" (stored as uchar), "integer" (stored as int32), "float32" (the default for floating point data for '.xdf' files), "numeric" (stored as float64 as in R), "character" (stored as string), "factor" (stored as uint32), "ordered" (ordered factor stored as uint32. Ordered factors are treated the same as factors in RevoScaleR analysis functions.), "int16" (alternative to integer for smaller storage space), "uint16" (alternative to unsigned integer for smaller storage space), "Date" (stored as Date, i.e. float64. Not supported for import types "textFast", "fixedFast", or "odbcFast"). "POSIXct" (stored as POSIXct, i.e. float64. Not supported for import types "textFast", "fixedFast", or "odbcFast"). Note for "factor" and "ordered" types, the levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use `column_info` with specified "levels".

Note that equivalent types share the same bullet in the list above; for some types we allow both 'R-friendly' type names, as well as our own, more specific type names for '.xdf' data.

Note also that specifying the column as a "factor" type is currently equivalent to "string" - for the moment, if you wish to import a column as factor data you must use the `column_info` argument, documented below.

column_info

List of named variable information lists. Each variable information list contains one or more of the named elements given below. When importing fixed format data, either `column_info` or an '.sts' schema file should be supplied. For fixed format text files, only the variables specified will be imported. For all text types, the information supplied for `column_info` overrides that supplied for `column_classes`. Currently available properties for a column information list are:

```
type: Character string specifying the data type for the column. See  
column_classes argument description for the available types. If the  
type is not specified for fixed format data, it will be read as  
character data.  
  
newName: Character string specifying a new name for the variable.  
description: character string specifying a description for the  
variable.  
  
levels: List of strings containing the levels when type =  
"factor". If the levels property is not provided, factor levels  
will be determined by the values in the source column. If levels  
are provided, any value that does not match a provided level will  
be converted to a missing value.  
  
newLevels: New or replacement levels specified for a column of type  
"factor". It must be used in conjunction with the levels argument.  
After reading in the original data, the labels for each level will  
be replaced with the newLevels.  
  
low: The minimum data value in the variable (used in computations  
using the F() function.)  
  
high: The maximum data value in the variable (used in computations  
using the F() function.)  
  
start: The left-most position, in bytes, for the column of a fixed  
format file respectively. When all elements of column_info have start,  
the text file is designated as a fixed format file. When none of  
the elements have it, the text file is designated as a delimited  
file. Specification of start must always be accompanied by  
specification of width.  
  
width: The number of characters in a fixed-width character column  
or the column of a fixed format file. If width is specified for a  
character column, it will be imported as a fixed-width character  
variable. Any characters beyond the fixed width will be ignored.  
Specification of width is required for all columns of a fixed  
format file.  
  
decimalPlaces: The number of decimal places.
```

rows_per_read

Number of rows to read at a time.

type

Character string set specifying file type of input_data. This is ignored if input_data is a data source. Possible values are: "auto": File type is automatically detected by looking at file extensions and argument values.

"textFast": Delimited text import using faster, more limited import mode. By default variables containing the values True and False or T and F will be created as bool variables.

"text": Delimited text import using enhanced, slower import mode. This allows for importing Date and POSIXct data types, handling the delimiter character inside a quoted string, and specifying decimal character and thousands separator. (See RxTextData.)

"fixedFast": Fixed format text import using faster, more limited import mode. You must specify a '.sts' format file or column_info specifications with start and width for each variable.

"fixed": Fixed format text import using enhanced, slower import mode. This allows for importing Date and POSIXct data types and specifying decimal character and thousands separator. You must specify a '.sts' format file or column_info specifications with start and width for each variable.

"odbcFast": ODBC import using faster, more limited import mode. "odbc": ODBC import using slower, enhanced import on Windows. (See RxOdbcData.)

max_rows_by_columns

The maximum size of a data frame that will be read in if output_file is set to None, measured by the number of rows times the number of columns. If the number of rows times the number of columns being imported exceeds this, a warning will be reported and a smaller number of rows will be read in than requested. If max_rows_by_columns is set to be too large, you may experience problems from loading a huge data frame into memory.

report_progress

Integer value with options: 0: no progress is reported. 1: the number of processed rows is printed and updated. 2: rows processed and timings are reported. 3: rows processed and all timings are reported.

verbose

Integer value. If 0, no additional output is printed. If 1, information on the import type is printed if type is set to auto.

xdf_compression_level

Integer in the range of -1 to 9. The higher the value, the greater the amount of compression - resulting in smaller files but a longer time to create them. If xdfCompressionLevel is set to 0, there will be no compression and files will be compatible with the 6.0 release of Revolution R Enterprise. If set to -1, a default level of compression will be used.

create_composite_set

Bool value or None. If True, a composite set of files will be created instead of a single '.xdf' file. A directory will be created whose name is the same as the '.xdf' file that would otherwise be created, but with no extension. Subdirectories 'data' and 'metadata' will be created. In the 'data' subdirectory, the data will be split across a set of '.xdff' files (see blocks_per_composite_file below for determining how many blocks of data will be in each file). In the 'metadata' subdirectory there is a single '.xdfm' file, which contains the meta data for all of the '.xdff' files in the 'data' subdirectory.

blocks_per_composite_file

Integer value. If create_composite_set=True, this will be the number of blocks put into each '.xdff' file in the composite set. If the output_file is an RxXdfData object, set the value for blocks_per_composite_file there instead.

kwargs

Additional arguments to be passed directly to the underlying data source objects to be imported.

Returns

If an output_file is not specified, an output data frame is returned. If an output_file is specified, an RxXdfData data source is returned that can be used in subsequent revoscalepy analysis.

Example

```
import os
from revoscalepy import rx_import, RxOptions, RxXdfData
sample_data_path = RxOptions.get_option("sampleDataDir")
ds = RxXdfData(os.path.join(sample_data_path, "kyphosis.xdf"))
kyphosis = rx_import(input_data = ds)
kyphosis.head()
```

RxInSqlServer

7/12/2022 • 2 minutes to read • [Edit Online](#)

```
revoscalepy.RxInSqlServer(connection_string: str, num_tasks: int = 1,  
    auto_cleanup: bool = True, console_output: bool = None,  
    execution_timeout_seconds: int = None, wait: bool = True,  
    packages_to_load: list = None)
```

Description

Creates a compute context for running revoscalepy analyses inside Microsoft SQL Server. Currently only supported in Windows.

Arguments

connection_string

An ODBC connection string used to connect to the Microsoft SQL Server database.

num_tasks

Number of tasks (processes) to run for each computation. This is the maximum number of tasks that will be used; SQL Server may start fewer processes if there is not enough data, if too many resources are already being used by other jobs, or if num_tasks exceeds the MAXDOP (maximum degree of parallelism) configuration option in SQL Server. Each of the tasks is given data in parallel, and does computations in parallel, and so computation time may decrease as num_tasks increases. However, that may not always be the case, and computation time may even increase if too many tasks are competing for machine resources. Note that RxOptions.set_option("NumCoresToUse", n) controls how many cores (actually, threads) are used in parallel within each process, and there is a trade-off between NumCoresToUse and NumTasks that depends upon the specific algorithm, the type of data, the hardware, and the other jobs that are running.

wait

Bool value, if True, the job will be blocking and will not return until it has completed or has failed. If False, the job will be non-blocking and return immediately, allowing you to continue running other Python code. The client connection with SQL Server must be maintained while the job is running, even in non-blocking mode.

console_output

Bool value, if True, causes the standard output of the Python process started by SQL Server to be printed to the user console. This value may be overwritten by passing a non-None bool value to the consoleOutput argument provided in rx_exec and rx_get_job_results.

auto_cleanup

Bool value, if True, the default behavior is to clean up the temporary computational artifacts and delete the result objects upon retrieval. If False, then the computational results are not deleted, and the results may be acquired using rx_get_job_results, and the output via rx_get_job_output until the rx_cleanup_jobs is used to delete the results and other artifacts. Leaving this flag set to False can result in accumulation of compute artifacts which you may eventually need to delete before they fill up your hard drive.

execution_timeout_seconds

Integer value, defaults to 0 which means infinite wait.

packages_to_load

Optional list of strings specifying additional packages to be loaded on the nodes when jobs are run in this compute context.

See also

[RxComputeContext](#), [RxLocalSeq](#), [rx_get_compute_context](#), [rx_set_compute_context](#).

Example

```
## Not run:
from revoscalepy import RxSqlServerData, RxInSqlServer, rx_lin_mod

connection_string="Driver=SQL Server;Server=.;Database=RevoTestDB;Trusted_Connection=True"

cc = RxInSqlServer(
    connection_string = connection_string,
    num_tasks = 1,
    auto_cleanup = False,
    console_output = True,
    execution_timeout_seconds = 0,
    wait = True
)

query="select top 100 [ArrDelay],[CRSDepTime],[DayOfWeek] FROM airlinedemosmall"
data_source = RxSqlServerData(
    sql_query = "select top 100 * from airlinedemosmall",
    connection_string = connection_string,
    column_info = {
        "ArrDelay" : { "type" : "integer" },
        "DayOfWeek" : {
            "type" : "factor",
            "levels" : [ "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday" ]
        }
    }
)

formula = "ArrDelay ~ CRSDepTime + DayOfWeek"
lin_mod = rx_lin_mod(formula, data = data_source, compute_context = cc)
print(lin_mod)
## End(Not run)
```

rx_hadoop_command

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_hadoop_command(cmd: str) -> revoscalepy.functions.RxHadoopUtils.RxHadoopCommandResults
```

Description

Executes arbitrary Hadoop commands and performs standard file operations in Hadoop.

Arguments

cmd

A character string containing a valid Hadoop command, that is, the cmd portion of Hadoop Command. Embedded quotes are not permitted.

Returns

Class RxHadoopCommandResults

See also

[rx_hadoop_make_dir](#) [rx_hadoop_copy_from_local](#) [rx_hadoop_remove_dir](#) [rx_hadoop_file_exists](#)
[rx_hadoop_list_files](#)

Example

```
from revoscalepy import rx_hadoop_command
rx_hadoop_command("version")
```

rx_hadoop_copy_from_local

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_hadoop_copy_from_local(source: str, dest: str)
```

Description

Wraps the Hadoop *fs -copyFromLocal* command.

Arguments

source

A character string specifying file(s) in Local File System

dest

A character string specifying the destination of a copy in HDFS If *source* includes more than one file, *dest* must be a directory.

Example

```
from revoscalepy import rx_hadoop_copy_from_local
rx_hadoop_copy_from_local("/tmp/foo.txt", "/user/RevoShare/newUser")
```

rx_hadoop_copy_to_local

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_hadoop_copy_to_local(source: str, dest: str)
```

Description

Wraps the Hadoop *fs -copyToLocal* command.

Arguments

source

A character string specifying file(s) to be copied in HDFS

dest

A character string specifying the destination of a copy in Local File System If *source* includes more than one file, *dest* must be a directory.

Example

```
from revoscalepy import rx_hadoop_copy_to_local
rx_hadoop_copy_to_local("/user/RevoShare/foo.txt", "/tmp/foo.txt")
```

rx_hadoop_copy

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_hadoop_copy(source: str, dest: str)
```

Description

Wraps the Hadoop *fs -cp* command.

Arguments

source

A character string specifying file(s) to be copied in HDFS

dest

A character string specifying the destination of a copy in HDFS If *source* includes more than one file, *dest* must be a directory.

Example

```
from revoscalepy import rx_hadoop_copy
rx_hadoop_copy("/user/RevoShare/newUser/foo.txt", "/user/RevoShare/newUser/bar.txt")
```

rx_hadoop_file_exists

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_hadoop_file_exists(path: typing.Union[list, str])
```

Description

Wraps the Hadoop *fs -test -e* command.

Arguments

path

Character string or list. A list of paths or A character string specifying location of one or more files or directories.

Returns

A bool value specifying whether file exists.

Example

```
from revoscalepy import rx_hadoop_file_exists
rx_hadoop_file_exists("/user/RevoShare/newUser/foo.txt")
```

rx_hadoop_list_files

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_hadoop_list_files(path: typing.Union[list, str],  
                                 recursive=False)
```

Description

Wraps the Hadoop *fs -ls* or *-lsr* command.

Arguments

path

Character string or list. A list of paths or A character string specifying location of one or more files or directories.

recursive

Bool value. If True, directory listings are recursive.

Example

```
from revoscalepy import rx_hadoop_list_files  
rx_hadoop_list_files("/user/RevoShare/newUser")
```

rx_hadoop_make_dir

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_hadoop_make_dir(path: typing.Union[list, str])
```

Description

Wraps the Hadoop *fs -mkdir -p* command.

Arguments

path

Character string or list. A list of paths or A character string specifying location of one or more directories.

Example

```
from revoscalepy import rx_hadoop_make_dir
rx_hadoop_make_dir("/user/RevoShare/newUser")
```

rx_hadoop_move

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_hadoop_move(source: str, dest: str)
```

Description

Wraps the Hadoop *fs -mv* command.

Arguments

source

A character string specifying file(s) to be moved in HDFS

dest

A character string specifying the destination of move in HDFS If *source* includes more than one file, *dest* must be a directory.

Example

```
from revoscalepy import rx_hadoop_move
rx_hadoop_move("/user/RevoShare/newUser/foo.txt", "/user/RevoShare/newUser/bar.txt")
```

rx_hadoop_remove_dir

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_hadoop_remove_dir(path: typing.Union[list, str],  
    skip_trash=False)
```

Description

Wraps the Hadoop *fs -rm -r* or *fs -rm -r -skipTrash* command.

Arguments

path

Character string or list. A list of paths or A character string specifying location of one or more directories.

:param skip_trash: If True, removal bypasses the trash folder, if one has been set up.

Example

```
from revoscalepy import rx_hadoop_remove_dir  
rx_hadoop_remove_dir("/user/RevoShare/newUser", skip_trash = True)
```

rx_hadoop_remove

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_hadoop_remove(path: typing.Union[list, str], skip_trash=False)
```

Description

Wraps the Hadoop *fs -rm* or *fs -rm -skipTrash* command.

Arguments

path

Character string or list. A list of paths or A character string specifying location of one or more files.

:param skip_trash: If True, removal bypasses the trash folder, if one has been set up.

Example

```
from revoscalepy import rx_hadoop_remove
rx_hadoop_remove("/user/RevoShare/newUser/foo.txt", skip_trash = True)
```

RxHdfsFileSystem

7/12/2022 • 2 minutes to read • [Edit Online](#)

```
revoscalepy.RxHdfsFileSystem(host_name=None, port=None)
```

Description

Main generator class for RxHdfsFileSystem.

Returns

An RxHdfsFileSystem file system object. This object may be used in [RxOptions](#), [RxTextData](#), [RxXdfData](#), [RxParquetData](#), or [RxOrcData](#) to set the file system.

See also

[RxOptions](#) [RxTextData](#) [RxXdfData](#) [RxParquet](#) [RxOrcData](#) [RxFileSystem](#)

Example

```
from revoscalepy import RxHdfsFileSystem
fs = RxHdfsFileSystem()
print(fs.file_system_type())
from revoscalepy import RxXdfData
xdf_data = RxXdfData("/tmp/hdfs_file", file_system = fs)
```

RxHiveData

7/12/2022 • 2 minutes to read • [Edit Online](#)

```
revoscalepy.RxHiveData(query=None, table=None, save_as_temp_table=False,  
write_factors_as_indexes: bool = False, column_info=None)
```

Description

Main generator for class RxHiveData, which extends RxSparkData.

Arguments

query

Character string specifying a Hive query, e.g. "select * from >>sample_<< table". Cannot be used with 'table'.

table

Character string specifying the name of a Hive table, e.g. "sample_table". Cannot be used with 'query'.

column_info

List of named variable information lists. Each variable information list contains one or more of the named elements given below. Currently available properties for a column information list are: type: Character string specifying the data type for the column.

Supported types are:

```
"bool" (stored as uchar),  
"integer" (stored as int32),  
"int16" (alternative to integer for smaller storage space),  
"float32" (stored as FloatType),  
"numeric" (stored as float64),  
"character" (stored as string),  
"factor" (stored as uint32),  
"Date" (stored as Date, i.e. float64.)  
"POSIXct" (stored as POSIXct, i.e. float64.)
```

levels: List of strings containing the levels when type = "factor". If the levels property is not provided, factor levels will be determined by the values in the source column. If levels are provided, any value that does not match a provided level will be converted to a missing value.

save_as_temp_table

Bool value, only applicable when using as output with "table" parameter. If "TRUE" register a temporary Hive table in Spark memory system otherwise generate a persistent Hive table. The temporary Hive table is always cached in Spark memory system.

write_factors_as_indexes

Bool value, If True, when writing to an RxHiveData data source, underlying factor indexes will be written instead of the string representations.

Returns

object of class `RxHiveData`.

Example

```
import os
from revoscalepy import rx_data_step, RxOptions, RxHiveData
sample_data_path = RxOptions.get_option("sampleDataDir")
colInfo = {"DayOfWeek": {"type": "factor"}}
ds1 = RxHiveData(table = "AirlineDemo")
result = rx_data_step(ds1)
ds2 = RxHiveData(query = "select * from hivesampletable")
result = rx_data_step(ds2)
```

rx_lin_mod

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_lin_mod(formula, data, pweights=None, fweights=None,
    cube=False, cube_predictions=False,
    row_selection=None, transforms=None,
    transform_objects=None,
    transform_function=typing.Union[str, <built-in function callable>] = None,
    transform_variables=None,
    transform_packages=None, drop_first=False,
    drop_main=True, cov_coef=False,
    cov_data=False, blocks_per_read=1,
    report_progress=None, verbose=0,
    compute_context=None, **kwargs)
```

Description

Fit linear models on small or large data sets.

Arguments

formula

Statistical model using symbolic formulas.

data

Either a data source object, a character string specifying a .xdf file, or a data frame object. If a Spark compute context is being used, this argument may also be an RxHiveData, RxOrcData, RxParquetData or RxSparkDataFrame object or a Spark data frame object from pyspark.sql.DataFrame.

pweights

Character string specifying the variable to use as probability weights for the observations.

fweights

Character string specifying the variable to use as frequency weights for the observations.

cube

Bool flag. If True and the first term of the predictor variables is categorical (a factor or an interaction of factors), the regression is performed by applying the Frisch-Waugh-Lovell Theorem, which uses a partitioned inverse to save on computation time and memory.

cube_predictions

Bool flag. If True and cube is True the predicted values are computed and included in the countDF component of the returned value. This may be memory intensive.

row_selection

None. Not currently supported, reserved for future use.

transforms

None. Not currently supported, reserved for future use.

transform_objects

A dictionary of variables besides the data that are used in the transform function. See rx_data_step for examples.

transform_function

Name of the function that will be used to modify the data before the model is built. The variables used in the transform function must be specified in transform_objects. See rx_data_step for examples.

transform_variables

List of strings of the column names needed for the transform function.

transform_packages

None. Not currently supported, reserved for future use.

drop_first

Bool flag. If False, the last level is dropped in all sets of factor levels in a model. If that level has no observations (in any of the sets), or if the model as formed is otherwise determined to be singular, then an attempt is made to estimate the model by dropping the first level in all sets of factor levels. If True, the starting position is to drop the first level. Note that for cube regressions, the first set of factors is excluded from these rules and the intercept is dropped.

drop_main

Bool value. If True, main-effect terms are dropped before their interactions.

cov_coef

Bool flag. If True and if cube is False, the variance-covariance matrix of the regression coefficients is returned.

cov_data

Bool flag. If True and if cube is False and if constant term is included in the formula, then the variance-covariance matrix of the data is returned.

blocks_per_read

Number of blocks to read for each chunk of data read from the data source.

report_progress

Integer value with options: 0: No progress is reported. 1: The number of processed rows is printed and updated. 2: Rows processed and timings are reported. 3: Rows processed and all timings are reported.

verbose

Integer value. If 0, no additional output is printed. If 1, additional summary information is printed.

compute_context

A RxComputeContext object for prediction.

kwargs

Additional parameters

Returns

A RxLinModResults object of linear model.

See also

[rx_logit](#).

Example

```
import os
import tempfile
from revoscalepy import RxOptions, RxXdfData, rx_lin_mod

sample_data_path = RxOptions.get_option("sampleDataDir")
in_mort_ds = RxXdfData(os.path.join(sample_data_path, "mortDefaultSmall.xdf"))

lin_mod = rx_lin_mod("creditScore ~ yearsEmploy", in_mort_ds)
```

rx_list_keys

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_list_keys(src: revoscalepy.datasource.RxOdbcData.RxOdbcData,  
    key: str = None, version: str = None,  
    key_name: str = 'id', version_name: str = 'version')
```

Description

Retrieves object keys from ODBC data sources.

Details

Enumerates all keys or versions for a given key, depending on the parameters. When key is None, the function enumerates all unique keys in the table. Otherwise, it enumerates all versions for the given key. Returns a single column data frame.

The key and the version should be of some SQL character type (CHAR, VARCHAR, NVARCHAR, etc.) supported by the data source. The value column should be a binary type (VARBINARY for instance). Some conversions to other types might work, however, they are dependent on the ODBC driver and on the underlying package functions.

Arguments

value

The object being stored into the data source.

key

A character string identifying the object. The intended use is for the key+version to be unique.

version

None or a character string which carries the version of the object. Combined with key identifies the object.

key_name

Character string specifying the column name for the key in the underlying table.

value_name

Character string specifying the column name for the objects in the underlying table.

version_name

Character string specifying the column name for the version in the underlying table.

Returns

rx_read_object returns an object. rx_write_object and rx_delete_object return bool, True on success. rx_list_keys returns a single column data frame containing strings.

Example

```
from pandas import DataFrame
from numpy import random
from revoscalepy import RxOdbcData, rx_write_object, rx_read_object, rx_list_keys, rx_delete_object

connection_string = 'Driver=SQL Server;Server=.;Database=RevoTestDb;Trusted_Connection=True;'
dest = RxOdbcData(connection_string, table = "dataframe")

df = DataFrame(random.randn(10, 5))

status = rx_write_object(dest, key = "myDf", value = df)

read_obj = rx_read_object(dest, key = "myDf")

keys = rx_list_keys(dest)

rx_delete_object(dest, key = "myDf")
```

RxLocalSeq

7/12/2022 • 2 minutes to read • [Edit Online](#)

```
revoscalepy.RxLocalSeq
```

Description

Creates a local compute context object. Computations using rx_exec will be processed sequentially. This is the default compute context.

Returns

Object of class `RxLocalSeq`.

See also

`RxComputeContext`, `RxInSqlServer`, `rx_get_compute_context`, `rx_set_compute_context`.

Example

```
from revoscalepy import RxLocalSeq
localcc = RxLocalSeq()
```

rx_logit

7/12/2022 • 3 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_logit()
```

Description

Use rx_logit to fit logistic regression models for small or large data sets.

Arguments

formula

Statistical model using symbolic formulas. Dependent variable must be binary. It can be a bool variable, a factor with only two categories, or a numeric variable with values in the range (0,1). In the latter case it will be converted to a bool.

data

Either a data source object, a character string specifying a '.xdf' file, or a data frame object. If a Spark compute context is being used, this argument may also be an RxHiveData, RxOrcData, RxParquetData or RxSparkDataFrame object or a Spark data frame object from pyspark.sql.DataFrame.

pweights

Character string specifying the variable to use as probability weights for the observations.

fweights

Character string specifying the variable to use as frequency weights for the observations.

cube

Bool flag. If True and the first term of the predictor variables is categorical (a factor or an interaction of factors), the regression is performed by applying the Frisch-Waugh-Lovell Theorem, which uses a partitioned inverse to save on computation time and memory.

cube_predictions

Bool flag. If True and cube is True the estimated model is evaluated (predicted) for each cell in the cube, fixing the non-cube variables in the model at their mean values, and these predictions are included in the countDF component of the returned value. This may be time and memory intensive for large models.

variable_selection

A list specifying various parameters that control aspects of stepwise regression. If it is an empty list (default), no stepwise model selection will be performed. If not, stepwise regression will be performed and cube must be False.

row_selection

None. Not currently supported, reserved for future use.

transforms

None. Not currently supported, reserved for future use.

transform_objects

A dictionary of variables besides the data that are used in the transform function. See rx_data_step for examples.

transform_function

Name of the function that will be used to modify the data before the model is built. The variables used in the transform function must be specified in transform_objects. See rx_data_step for examples.

transform_variables

List of strings of the column names needed for the transform function.

transform_packages

None. Not currently supported, reserved for future use.

drop_first

Bool flag. If False, the last level is dropped in all sets of factor levels in a model. If that level has no observations (in any of the sets), or if the model as formed is otherwise determined to be singular, then an attempt is made to estimate the model by dropping the first level in all sets of factor levels. If True, the starting position is to drop the first level. Note that for cube regressions, the first set of factors is excluded from these rules and the intercept is dropped.

drop_main

Bool value. If True, main-effect terms are dropped before their interactions.

cov_coef

Bool flag. If True and if cube is False, the variance-covariance matrix of the regression coefficients is returned.

cov_data

Bool flag. If True and if cube is False and if constant term is included in the formula, then the variance-covariance matrix of the data is returned.

initial_values

Starting values for the Iteratively Reweighted Least Squares algorithm used to estimate the model coefficients.

coef_label_style

Character string specifying the coefficient label style. The default is "Revo".

blocks_per_read

Number of blocks to read for each chunk of data read from the data source.

max_iterations

Maximum number of iterations.

coefficient_tolerance

Convergence tolerance for coefficients. If the maximum absolute change in the coefficients (step), divided by the maximum absolute coefficient value, is less than or equal to this tolerance at the end of an iteration, the estimation is considered to have converged. To disable this test, set this value to 0.

gradient_tolerance

This argument is deprecated.

objective_function_tolerance

Convergence tolerance for the objective function.

report_progress

Integer value with options: 0: No progress is reported. 1: The number of processed rows is printed and updated.

2: Rows processed and timings are reported. 3: Rows processed and all timings are reported.

verbose

Integer value.

compute_context

A RxComputeContext object for prediction.

kwargs

Additional parameters

Returns

A RxLogitResults object of linear model.

See also

[rx_lin_mod](#).

Example

```
import os
from revoscalepy import rx_logit, RxOptions, RxXdfData
sample_data_path = RxOptions.get_option("sampleDataDir")
kyphosis = RxXdfData(os.path.join(sample_data_path, "kyphosis.xdf"))

logit = rx_logit('Kyphosis ~ Age + Number + Start', data = kyphosis)
```

RxMissingValues

7/12/2022 • 2 minutes to read • [Edit Online](#)

`revoscalepy.RxMissingValues`

Description

revoscalepy package uses Pandas dataframe as the abstraction to hold the data and manipulate it. Pandas dataframe in turn uses NumPy ndarray to hold homogeneous data efficiently and provides fast access and manipulation functions on the ndarray. Python and NumPy ndarray in particular only has numpy.NaN value for float types to indicate missing values. For other primitive datatypes, like int, there isn't a missing value notation. If one uses Python's None object to represent missing value in a series of data, say int, the whole ndarray's dtype changes to object. This makes dealing with data with missing values quite inefficient during processing.

This class provides missing values for various NumPy data types which one can use to mark missing values in a sequence of data in ndarray.

It provides missing value for following types:

- `int8()` for `numpy.int8` with value of `numpy.iinfo(np.int8).min`
- `uint8()` for `numpy.uint8` with value of `numpy.iinfo(np.uint8).max`
- `int16()` for `numpy.int16` with value of `numpy.iinfo(np.int16).min`
- `uint16()` for `numpy.uint16` with value of `numpy.iinfo(np.uint16).max`
- `int32()` for `numpy.int32` with value of `numpy.iinfo(np.int32).min`
- `uint32()` for `numpy.uint32` with value of `numpy.iinfo(np.uint32).max`
- `int64()` for `numpy.int64` with value of `numpy.iinfo(np.int64).min`
- `uint64()` for `numpy.uint64` with value of `numpy.iinfo(np.uint64).max`
- `float16()` for `numpy.float16` with value of `numpy.NaN`
- `float32()` for `numpy.float32` with value of `numpy.NaN`
- `float64()` for `numpy.float64` with value of `numpy.NaN`

Example

```

## Not run:
from revoscalepy import RxXdfData, RxSqlServerData, RxInSqlServer
from revoscalepy import RxOptions, rx_logit, RxMissingValues
from revoscalepy.functions.RxLogit import RxLogitResults
import os

def transform_late (data, context):
    import numpy

    #
    # create a new 'Late' column based on 'ArrDelay' int32 column
    #
    data['Late'] = data['ArrDelay'] > 15

    #
    # replace all 'Late' with NaN for 'NA' or missing values in ArrDelay
    #
    data.loc[data['ArrDelay'] == RxMissingValues.int32(), ('Late')] = numpy.NaN

    return data

transformVars = ["ArrDelay"]
sample_data_path = RxOptions.get_option("sampleDataDir")
xdf_file = os.path.join(sample_data_path, "AirlineDemoSmall.xdf")

data = RxXdfData(xdf_file)
model = rx_logit("Late~CRSDepTime + DayOfWeek", data=data, transform_function=transform_late,
transform_variables=transformVars)

assert isinstance(model, RxLogitResults)
## End(Not run)

```

RxNativeFileSystem

7/12/2022 • 2 minutes to read • [Edit Online](#)

```
revoscalepy.RxNativeFileSystem
```

Description

Main generator class for objects representing the native file system.

Returns

An RxNativeFileSystem file system object. This object may be used in [RxOptions](#), [RxTextData](#), or [RxXdfData](#) to set the file system.

See also

[RxOptions](#) [RxTextData](#) [RxXdfData](#) [RxFileSystem](#)

Example

```
from revoscalepy import RxNativeFileSystem
fs = RxNativeFileSystem()
print(fs.file_system_type())
```

RxOdbcData

7/12/2022 • 4 minutes to read • [Edit Online](#)

```
revoscalepy.RxOdbcData(connection_string: str = None,  
                         table: str = None, sql_query: str = None,  
                         dbms_name: str = None, database_name: str = None,  
                         use_fast_read: bool = True, trim_space: bool = True,  
                         row_buffering: bool = True, return_data_frame: bool = True,  
                         string_as_factors: bool = False, column_classes: dict = None,  
                         column_info: dict = None, rows_per_read: int = 500000,  
                         verbose: int = 0, write_factors_as_indexes: bool = False,  
                         **kwargs)
```

Description

Main generator for class RxOdbcData, which extends RxDataSource.

Arguments

connection_string

None or character string specifying the connection string.

table

None or character string specifying the table name. Cannot be used with sqlQuery.

sql_query

None or character string specifying a valid SQL select query. Cannot be used with table.

dbms_name

None or character string specifying the Database Management System (DBMS) name.

database_name

None or character string specifying the name of the database.

use_fast_read

Bool specifying whether or not to use a direct ODBC connection.

trim_space

Bool specifying whether or not to trim the white character of string data for reading.

row_buffering

Bool specifying whether or not to buffer rows on read from the database. If you are having problems with your ODBC driver, try setting this to False.

return_data_frame

Bool indicating whether or not to convert the result from a list to a data frame (for use in rxReadNext only). If False, a list is returned.

string_as_factors

Bool indicating whether or not to automatically convert strings to factors on import. This can be overridden by specifying "character" in column_classes and column_info. If True, the factor levels will be coded in the order

encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use column_info with specified "levels".

column_classes

Dictionary of column name to strings specifying the column types to use when converting the data. The element names for the vector are used to identify which column should be converted to which type.

Allowable column types are:

```
"bool" (stored as uchar),  
"integer" (stored as int32),  
"float32" (the default for floating point data for '.xdf' files),  
"numeric" (stored as float64 as in R),  
"character" (stored as string),  
"factor" (stored as uint32),  
"int16" (alternative to integer for smaller storage space),  
"uint16" (alternative to unsigned integer for smaller storage space),  
"Date" (stored as Date, i.e. float64)
```

Note for "factor" type, the levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use column_info with specified "levels". Note that equivalent types share the same bullet in the list above; for some types we allow both 'R-friendly' type names, as well as our own, more specific type names for '.xdf' data. Note also that specifying the column as a "factor" type is currently equivalent to "string" - for the moment, if you wish to import a column as factor data you must use the column_info argument, documented below.

column_info

List of named variable information lists. Each variable information list contains one or more of the named elements given below. The information supplied for column_info overrides that supplied for column_classes.

Currently available properties for a column information list are:

```
type: character string specifying the data type for the column. See  
      column_classes argument description for the available types. Specify  
      "factorIndex" as the type for 0-based factor indexes. levels must also  
      be specified.  
  
newName: character string specifying a new name for the variable.  
description: character string specifying a description for the variable.  
levels: list of strings containing the levels when type = "factor". If  
  
      the levels property is not provided, factor levels will be determined  
      by the values in the source column. If levels are provided, any value  
      that does not match a provided level will be converted to a missing  
      value.  
  
newLevels: new or replacement levels specified for a column of type  
      "factor". It must be used in conjunction with the levels argument.  
      After reading in the original data, the labels for each level will be  
      replaced with the newLevels.  
  
low: the minimum data value in the variable (used in computations using  
      the F() function.  
  
high: the maximum data value in the variable (used in computations  
      using the F() function.
```

rows_per_read

Number of rows to read at a time.

verbose

integer value. If 0, no additional output is printed. If 1, information on the odbc data source type (odbc or odbcFast) is printed.

write_factors_as_indexes

Bool value, If True, when writing to an RxOdbcData data source, underlying factor indexes will be written instead of the string representations.

kwargs

Additional arguments to be passed directly to the underlying functions.

Returns

Object of class RxOdbcData.

Example

```
from revoscalepy import RxOdbcData, RxOptions, rx_write_object, rx_read_object, RxXdfData
from numpy import array_equal
import os

# Establish a connection to an ODBC data source
connection_string = 'Driver=SQL Server;Server=.;Database=RevoTestDb;Trusted_Connection=True;'
dest = RxOdbcData(connection_string, table = "data")

# Write an array to the database
my_array = [1,2,3]
rx_write_object(dest = dest, key = "my_array", value = my_array)

# Retrieve the array from the database
array_ds = rx_read_object(src = dest, key = "my_array")
array_equal(my_array, array_ds) # True

# Write a XDF object to the database
sample_data_path = RxOptions.get_option("sampleDataDir")
kyphosis = RxXdfData(os.path.join(sample_data_path, "kyphosis.xdf"))
rx_write_object(dest = dest, key = "kyphosis", value = kyphosis)

# Retrieve the data from the database
kyphosis_ds = rx_read_object(src = dest, key="kyphosis")
```

RxOptions

7/12/2022 • 3 minutes to read • [Edit Online](#)

`revoscalepy.RxOptions`

Description

Functions to specify and retrieve options needed for revoscalepy computations. These need to be set only once to carry out multiple computations.

Arguments

unitTestDataDir

Character string specifying path to revoscalepy's test data directory.

sampleDataDir

Character string specifying path to revoscalepy's sample data directory.

blocksPerRead

Default value to use for blocksPerRead argument for many revoscalepy functions. Represents the number of blocks to read within each read chunk.

reportProgress

Default value to use for reportProgress argument for many revoscalepy functions. Options are:

```
0: no progress is reported.  
1: the number of processed rows is printed and updated.  
2: rows processed and timings are reported.  
3: rows processed and all timings are reported.
```

RowDisplayMax

Integer value specifying the maximum number of rows to display when using the verbose argument in revoscalepy functions. The default of -1 displays all available rows.

MemStatsReset

Boolean integer. If 1, reset memory status

MemStatsDiff

Boolean integer. If 1, the change of memory status is shown.

NumCoresToUse

Integer value specifying the number of cores to use. If set to a value higher than the number of available cores, the number of available cores will be used. If set to -1, the number of available cores will be used. Increasing the number of cores to use will also increase the amount of memory required for revoscalepy analysis functions.

NumDigits

Controls the number of digits to use when converting numeric data to or from strings, such as when printing numeric values or importing numeric data as strings. The default is the current value of options()\$digits, which defaults to 7. Beyond fifteen digits, however, results are likely to be unreliable.

ShowTransformFn

Bool value. If True, the transform function is shown.

DataPath

List of strings containing paths to search for local data sources. The default is to search just the current working directory. This will be ignored if dataPath is specified in the active compute context. See the Details section for more information regarding the path format.

OutDataPath

List of strings containing paths for writing new output data files. New data files will be written to the first path that exists. The default is to write to the current working directory. This will be ignored if outDataPath is specified in the active compute context.

XdfCompressionLevel

Integer in the range of -1 to 9. The higher the value, the greater the amount of compression - resulting in smaller files but a longer time to create them.

FileSystem

Character string or RxFileSystem object indicating type of file system; "native" or RxNativeFileSystem object can be used for the local operating system, or an RxHdfsFileSystem object for the Hadoop file system.

UseSparseCube

Bool value. If True, sparse cube is used.

RngBfferSize

A positive integer scalar specifying the buffer size for the Parallel Random Number Generators (RNGs) in MKL.

DropMain

Bool value. If True, main-effect terms are dropped before their interactions.

CoefLabelStyle

Character string specifying the coefficient label style. The default is "Revo".

NumTasks

Integer value. The default numTasks use in RxInSqlServer.

unixPythonPath

The path to Python executable on a Unix/Linux node. By default it points to a path corresponding to this client's version.

traceLevel

Specifies the traceLevel that ML server will run with. This parameter controls ML Server Logging features as well as Runtime Tracing of ScalePy functions. Levels are inclusive, (i.e. level 3:INFO includes levels 2:WARN and 1:ERROR log messages). The options are:

```
0: DISABLED - Tracing/Logging disabled.  
1: ERROR - ERROR coded trace points are logged to MRS log files  
2: WARN - WARN and ERROR coded trace points are logged to MRS log files.  
3: INFO - INFO, WARN, and ERROR coded trace points are logged to MRS  
        log files.  
  
4: DEBUG - All trace points are logged to MRS log files.  
5: RESERVED - If set, will log at DEBUG granularity  
6: RESERVED - If set, will log at DEBUG granularity  
7: TRACE - ScaleR functions Runtime Tracing is activated and MRS log  
        level is set to DEBUG granularity.
```

Returns

For RxOptions, a list containing the original RxOptions is returned.

If there is no argument specified, the list is returned explicitly, otherwise the list is returned as an invisible object.

For RxGetOption, the current value of the requested option is returned.

Example

```
from revoscalepy import RxOptions
sample_data_path = RxOptions.get_option("sampleDataDir")
```

RxOrcData

7/12/2022 • 2 minutes to read • [Edit Online](#)

```
revoscalepy.RxOrcData(file: str = None, column_info=None,  
                      file_system: revoscalepy.datasource.RxFileSystem.RxFileSystem = None,  
                      write_factors_as_indexes: bool = False)
```

Description

Main generator for class RxOrcData, which extends RxSparkData.

Arguments

file

Character string specifying the location of the data. e.g. "/tmp/AirlineDemoSmall.orc".

column_info

List of named variable information lists. Each variable information list contains one or more of the named elements given below. Currently available properties for a column information list are: type: Character string specifying the data type for the column.

```
Supported types are:  
"bool" (stored as uchar),  
"integer" (stored as int32),  
"int16" (alternative to integer for smaller storage space),  
"float32" (stored as FloatType),  
"numeric" (stored as float64),  
"character" (stored as string),  
"factor" (stored as uint32),  
"Date" (stored as Date, i.e. float64.)
```

levels: List of strings containing the levels when type = "factor". If the levels property is not provided, factor levels will be determined by the values in the source column. If levels are provided, any value that does not match a provided level will be converted to a missing value.

file_system

Character string or RxFileSystem object indicating type of file system; It supports native HDFS and other HDFS compatible systems, e.g., Azure Blob and Azure Data Lake. Local file system is not supported.

write_factors_as_indexes

Bool value, if True, when writing to an RxOrcData data source, underlying factor indexes will be written instead of the string representations.

Returns

object of class `RxOrcData`.

Example

```
import os
from revoscalepy import rx_data_step, RxOptions, RxOrcData
sample_data_path = RxOptions.get_option("sampleDataDir")
colInfo = {"DayOfWeek": {"type": "factor"}}
ds = RxOrcData(os.path.join(sample_data_path, "AirlineDemoSmall.orc"))
result = rx_data_step(ds)
```

RxParquetData

7/12/2022 • 2 minutes to read • [Edit Online](#)

```
revoscalepy.RxParquetData(file: str = None, column_info=None,  
    file_system: revoscalepy.datasource.RxFileSystem.RxFileSystem = None,  
    write_factors_as_indexes: bool = False)
```

Description

Main generator for class RxParquetData, which extends RxSparkData.

Arguments

file

Character string specifying the location of the data. e.g. "/tmp/AirlineDemoSmall.parquet".

column_info

List of named variable information lists. Each variable information list contains one or more of the named elements given below. Currently available properties for a column information list are: type: Character string specifying the data type for the column.

```
Supported types are:  
"bool" (stored as uchar),  
"integer" (stored as int32),  
"int16" (alternative to integer for smaller storage space),  
"float32" (stored as FloatType),  
"numeric" (stored as float64),  
"character" (stored as string),  
"factor" (stored as uint32),  
"Date" (stored as Date, i.e. float64.)  
"POSIXct" (stored as POSIXct, i.e. float64.)
```

levels: List of strings containing the levels when type = "factor". If the levels property is not provided, factor levels will be determined by the values in the source column. If levels are provided, any value that does not match a provided level will be converted to a missing value.

file_system

Character string or RxFileSystem object indicating type of file system; It supports native HDFS and other HDFS compatible systems, e.g., Azure Blob and Azure Data Lake. Local file system is not supported.

write_factors_as_indexes

Bool, if True, when writing to an RxParquetData data source, underlying factor indexes will be written instead of the string representations.

Returns

Object of class `RxParquetData`.

Example

```
import os
from revoscalepy import rx_data_step, RxOptions, RxParquetData
sample_data_path = RxOptions.get_option("sampleDataDir")
colInfo = {"DayOfWeek": {"type": "factor"}}
ds = RxParquetData(os.path.join(sample_data_path, "AirlineDemoSmall.parquet"))
result = rx_data_step(ds)
```

rx_partition

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_partition(input_data: typing.Union[revoscalepy.datasource.RxDataSource.RxDataSource,
    pandas.core.frame.DataFrame, str],
    output_data: revoscalepy.datasource.RxXdfData.RxXdfData,
    vars_to_partition: typing.Union[typing.List[str], str],
    append: str = 'rows', overwrite: bool = False,
    **kwargs) -> pandas.core.frame.DataFrame
```

Description

Partition input data sources by key values and save the results to a partitioned .xdf file on disk.

Arguments

input_data

Either a data source object, a character string specifying a '.xdf' file, or a Pandas data frame object.

output_data

A partitioned data source object created by RxXdfData with create_partition_set = True.

vars_to_partition

List of strings of variable names to be used for partitioning the input data set.

append

Either "none" to create a new file or "rows" to append rows to an existing file. If output_data exists and append is "none", the overwrite argument must be set to True.

overwrite

Bool value. If True, an existing output_data will be overwritten. overwrite is ignored if appending rows.

kwargs

Additional arguments.

Returns

A Pandas data frame of partitioning values and data sources, each row in the Pandas data frame represents one partition and the data source in the last variable holds the data of a specific partition.

See also

[rx_exec_by](#) . [RxXdfData](#) .

Example

```
import os, tempfile
from revoscalepy import RxOptions, RxXdfData, rx_partition
data_path = RxOptions.get_option("sampleDataDir")

# input xdf data source
xdf_file = os.path.join(data_path, "claims.xdf")
xdf_ds = RxXdfData(xdf_file)

# create a partitioned xdf data source object
out_xdf_file = os.path.join(tempfile._get_default_tempdir(), "outPartitions")
out_xdf = RxXdfData(out_xdf_file, create_partition_set = True)

# do partitioning for input data set
partitions = rx_partition(input_data = xdf_ds, output_data = out_xdf, vars_to_partition =
["car.age","type"])
print(partitions)
```

rx_predict

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_predict(model_object=None, data=None, output_data=None, **kwargs)
```

Description

Generic function to compute predicted values and residuals using rx_lin_mod, rx_logit, rx_dtree, rx_dforest and rx_btrees objects.

Arguments

model_object

object returned from a call to rx_lin_mod, rx_logit, rx_dtree, rx_dforest and rx_btrees. Objects with multiple dependent variables are not supported.

data

a data frame or an RxXdfData data source object to be used for predictions. If a Spark compute context is being used, this argument may also be an RxHiveData, RxOrcData, RxParquetData or RxSparkDataFrame object or a Spark data frame object from pyspark.sql.DataFrame.

output_data

an RxXdfData data source object or existing data frame to store predictions.

kwargs

additional parameters

Returns

a data frame or a data source object of prediction results.

See also

[rx_predict_default](#), [rx_predict_rx_dtree](#), [rx_predict_rx_dforest](#).

Example

```
import os
from revoscalepy import RxOptions, RxXdfData, rx_lin_mod, rx_predict, rx_data_step

sample_data_path = RxOptions.get_option("sampleDataDir")
mort_ds = RxXdfData(os.path.join(sample_data_path, "mortDefaultSmall.xdf"))
mort_df = rx_data_step(mort_ds)

lin_mod = rx_lin_mod("creditScore ~ yearsEmploy", mort_df)
pred = rx_predict(lin_mod, data = mort_df)
print(pred.head())
```

Output:

```
Rows Read: 100000, Total Rows Processed: 100000, Total Chunk Time: 0.058 seconds
Rows Read: 100000, Total Rows Processed: 100000, Total Chunk Time: 0.006 seconds
Computation time: 0.039 seconds.
Rows Read: 100000, Total Rows Processed: 100000, Total Chunk Time: Less than .001 seconds
  creditScore_Pred
0      700.089114
1      699.834355
2      699.783403
3      699.681499
4      699.783403
```

Note: Function `rx_predict` does not run predictions but chooses the appropriate function `rx_predict_default`, `rx_predict_rx_dtrees`, or `rx_predict_rx_dforest` based on the model which was given to it. Each of them has a different set of parameters described by their documentation.

rx_predict_default

7/12/2022 • 3 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_predict_default(model_object=None,
    data: revoscalepy.datasource.RxDataSource.RxDataSource = None,
    output_data: typing.Union[revoscalepy.datasource.RxDataSource.RxDataSource,
    str] = None, compute_standard_errors: bool = False,
    interval: typing.Union[list, str] = 'none',
    confidence_level: float = 0.95, compute_residuals: bool = False,
    type: typing.Union[list, str] = None,
    write_model_vars: bool = False,
    extra_vars_to_write: typing.Union[list, str] = None,
    remove_missings: bool = False, append: typing.Union[list,
    str] = None, overwrite: bool = False,
    check_factor_levels: bool = True,
    predict_var_names: typing.Union[list, str] = None,
    residual_var_names: typing.Union[list, str] = None,
    interval_var_names: typing.Union[list, str] = None,
    std_errors_var_names: typing.Union[list, str] = None,
    blocks_per_read: int = 0, report_progress: int = None,
    verbose: int = 0, xdf_compression_level: int = 0,
    compute_context: revoscalepy.computecontext.RxComputeContext.RxComputeContext = None,
    **kwargs)
```

Description

Compute predicted values and residuals using rx_lin_mod and rx_logit objects.

Arguments

model_object

Object returned from a call to rx_lin_mod and rx_logit. Objects with multiple dependent variables are not supported.

data

a data frame or an RxXdfData data source object to be used for predictions. If a Spark compute context is being used, this argument may also be an RxHiveData, RxOrcData, RxParquetData or RxSparkDataFrame object or a Spark data frame object from pyspark.sql.DataFrame.

output_data

A character string specifying the output '.xdf' file, a RxXdfData object, RxTextData object, a RxOdbcData data source, or a RxSqlServerData data source to store predictions.

compute_standard_errors

Bool value. If True, the standard errors for each dependent variable are calculated.

interval

Character string defining the type of interval calculation to perform. Supported values are "none", "confidence", and "prediction".

confidence_level

Numeric value representing the confidence level on the interval [0, 1].

compute_residuals

bool Value. If True, residuals are computed.

type

The type of prediction desired. Supported choices are: "response" and "link". If type = "response", the predictions are on the scale of the response variable. For instance, for the binomial model, the predictions are in the range (0,1). If type = "link", the predictions are on the scale of the linear predictors. Thus for the binomial model, the predictions are of log-odds.

write_model_vars

Bool value. If True, and the output data set is different from the input data set, variables in the model will be written to the output data set in addition to the predictions (and residuals, standard errors, and confidence bounds, if requested). If variables from the input data set are transformed in the model, the transformed variables will also be included.

extra_vars_to_write

None or list of strings of additional variables names from the input data or transforms to include in the output_data. If write_model_vars is True, model variables will be included as well.

remove_missings

Bool value. If True, rows with missing values are removed.

append

either "none" to create a new file or "rows" to append rows to an existing file. If output_data exists and append is "none", the overwrite argument must be set to True. Ignored for data frames.

overwrite

Bool value. If True, an existing output_data will be overwritten. overwrite is ignored if appending rows. Ignored for data frames.

check_factor_levels

Bool value.

predict_var_names

List of strings specifying name(s) to give to the prediction results.

residual_var_names

List of strings specifying name(s) to give to the residual results.

interval_var_names

None or a list of strings defining low and high confidence interval variable names, respectively. If None, the strings "_Lower" and "_Upper" are appended to the dependent variable names to form the confidence interval variable names.

std_errors_var_names

None or a list of strings defining variable names corresponding to the standard errors, if calculated. If None, the string "_StdErr" is appended to the dependent variable names to form the standard errors variable names.

blocks_per_read

Number of blocks to read for each chunk of data read from the data source. If the data and output_data are the same file, blocks_per_read must be 1.

report_progress

Integer value with options: 0: No progress is reported. 1: The number of processed rows is printed and updated. 2: Rows processed and timings are reported. 3: Rows processed and all timings are reported.

verbose

Integer value. If 0, no additional output is printed. If 1, additional summary information is printed.

xdf_compression_level

Integer in the range of -1 to 9 indicating the compression level for the output data if written to an .xdf file.

compute_context

A RxComputeContext object for prediction.

kwargs

Additional parameters

Returns

A data frame or a data source object of prediction results.

See also

[rx_predict](#).

Example

```
import os
from revoscalepy import RxOptions, RxXdfData, rx_lin_mod, rx_predict_default, rx_data_step

sample_data_path = RxOptions.get_option("sampleDataDir")
mort_ds = RxXdfData(os.path.join(sample_data_path, "mortDefaultSmall.xdf"))
mort_df = rx_data_step(mort_ds)

lin_mod = rx_lin_mod("creditScore ~ yearsEmploy", mort_df)
pred = rx_predict_default(lin_mod, data = mort_df, compute_residuals = True, write_model_vars = True)
pred.head()
```

rx_predict_rx_dtreet

7/12/2022 • 3 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_predict_rx_dtreet(model_object=None,
    data: revoscalepy.datasource.RxDataSource.RxDataSource = None,
    output_data: typing.Union[revoscalepy.datasource.RxDataSource.RxDataSource,
    str] = None, predict_var_names: list = None,
    write_model_vars: bool = False,
    extra_vars_to_write: list = None, append: typing.Union[list,
    str] = 'none', overwrite: bool = False, type: typing.Union[list,
    str] = None, remove_missings: bool = False,
    compute_residuals: bool = False, residual_type: typing.Union[list,
    str] = 'usual', residual_var_names: list = None,
    blocks_per_read: int = None, report_progress: int = None,
    verbose: int = 0, xdf_compression_level: int = None,
    compute_context=None, **kwargs)
```

Description

Calculate predicted or fitted values for a data set from an rx_dtreet object.

Arguments

model_object

Object returned from a call to rx_dtreet.

data

A data frame or an RxXdfData data source object to be used for predictions. If a Spark compute context is being used, this argument may also be an RxHiveData, RxOrcData, RxParquetData or RxSparkDataFrame object or a Spark data frame object from pyspark.sql.DataFrame.

output_data

An RxXdfData data source object or existing data frame to store predictions.

predict_var_names

List of strings specifying name(s) to give to the prediction results

write_model_vars

Bool value. If True, and the output data set is different from the input data set, variables in the model will be written to the output data set in addition to the predictions (and residuals, standard errors, and confidence bounds, if requested). If variables from the input data set are transformed in the model, the transformed variables will also be included.

extra_vars_to_write

None or list of strings of additional variables names from the input data or transforms to include in the output_data. If write_model_vars is True, model variables will be included as well.

append

Either "none" to create a new file or "rows" to append rows to an existing file. If output_data exists and append is "none", the overwrite argument must be set to True. Ignored for data frames.

overwrite

Bool value. If True, an existing output_data will be overwritten. overwrite is ignored if appending rows. Ignored for data frames.

type

the type of prediction desired. Supported choices are: "vector", "prob", "class", and "matrix".

remove_missings

Bool value. If True, rows with missing values are removed.

compute_residuals

Bool value. If True, residuals are computed.

residual_type

Indicates the type of residual desired.

residual_var_names

List of strings specifying name(s) to give to the residual results.

blocks_per_read

Number of blocks to read for each chunk of data read from the data source. If the data and output_data are the same file, blocks_per_read must be 1.

report_progress

Integer value with options: 0: No progress is reported. 1: The number of processed rows is printed and updated. 2: Rows processed and timings are reported. 3: Rows processed and all timings are reported.

verbose

Integer value. If 0, no additional output is printed. If 1, additional summary information is printed.

xdf_compression_level

Integer in the range of -1 to 9 indicating the compression level for the output data if written to an .xdf file.

compute_context

A RxComputeContext object for prediction.

kwargs

Additional parameters

Returns

A data frame or a data source object of prediction results.

See also

[rx_predict](#).

Example

```
import os
from revoscalepy import rx_dtree, rx_predict_rx_dtree, rx_import, RxOptions, RxXdfData
sample_data_path = RxOptions.get_option("sampleDataDir")
ds = RxXdfData(os.path.join(sample_data_path, "kyphosis.xdf"))
kyphosis = rx_import(input_data = ds)

# classification
formula = "Kyphosis ~ Number + Start"
method = "class"
parms = {'prior': [0.8, 0.2], 'loss': [0, 2, 3, 0], 'split': "gini"}
cost = [2,3]
dtree = rx_dtree(formula, data = kyphosis, pweights = "Age", method = method, parms = parms, cost = cost,
max_num_bins = 100)
rx_pred = rx_predict_rx_dtree(dtree, data = kyphosis)
rx_pred.head()

# regression
formula = "Age ~ Number + Start"
method = "anova"
parms = {'prior': [0.8, 0.2], 'loss': [0, 2, 3, 0], 'split': "gini"}
cost = [2,3]
dtree = rx_dtree(formula, data = kyphosis, pweights = "Kyphosis", method = method, parms = parms, cost =
cost, max_num_bins = 100)
rx_pred = rx_predict_rx_dtree(dtree, data = kyphosis)
rx_pred.head()
```

rx_predict_rx_dforest

7/12/2022 • 3 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_predict_rx_dforest(model_object=None,
    data: revoscalepy.datasource.RxDataSource.RxDataSource = None,
    output_data: typing.Union[revoscalepy.datasource.RxDataSource.RxDataSource,
    str] = None, predict_var_names: list = None,
    write_model_vars: bool = False,
    extra_vars_to_write: list = None, append: typing.Union[list,
    str] = 'none', overwrite: bool = False, type: typing.Union[list,
    str] = None, cutoff: list = None,
    remove_missings: bool = False, compute_residuals: bool = False,
    residual_type: typing.Union[list, str] = 'usual',
    residual_var_names: list = None, blocks_per_read: int = None,
    report_progress: int = None, verbose: int = 0,
    xdf_compression_level: int = None, compute_context=None, **kwargs)
```

Description

Calculate predicted or fitted values for a data set from an rx_dforest or rx_btrees object.

Arguments

model_object

object returned from a call to rx_dtrees.

data

a data frame or an RxXdfData data source object to be used for predictions. If a Spark compute context is being used, this argument may also be an RxHiveData, RxOrcData, RxParquetData or RxSparkDataFrame object or a Spark data frame object from pyspark.sql.DataFrame.

output_data

an RxXdfData data source object or existing data frame to store predictions.

predict_var_names

list of strings specifying name(s) to give to the prediction results

write_model_vars

bool value. If True, and the output data set is different from the input data set, variables in the model will be written to the output data set in addition to the predictions (and residuals, standard errors, and confidence bounds, if requested). If variables from the input data set are transformed in the model, the transformed variables will also be included.

extra_vars_to_write

None or list of strings of additional variables names from the input data or transforms to include in the output_data. If write_model_vars is True, model variables will be included as well.

append

either "none" to create a new file or "rows" to append rows to an existing file. If output_data exists and append is "none", the overwrite argument must be set to True. Ignored for data frames.

overwrite

bool value. If True, an existing output_data will be overwritten. overwrite is ignored if appending rows. Ignored for data frames.

type

the type of prediction desired. Supported choices are: "response", "prob", and "vote".

cutoff

(Classification only) a vector of length equal to the number of classes specifying the dividing factors for the class votes. The default is the one used when the decision forest is built.

remove_missings

bool value. If True, rows with missing values are removed.

compute_residuals

bool value. If True, residuals are computed.

residual_type

Indicates the type of residual desired.

residual_var_names

list of strings specifying name(s) to give to the residual results.

blocks_per_read

number of blocks to read for each chunk of data read from the data source. If the data and output_data are the same file, blocks_per_read must be 1.

report_progress

integer value with options: 0: no progress is reported. 1: the number of processed rows is printed and updated. 2: rows processed and timings are reported. 3: rows processed and all timings are reported.

verbose

integer value. If 0, no additional output is printed. If 1, additional summary information is printed.

xdf_compression_level

integer in the range of -1 to 9 indicating the compression level for the output data if written to an .xdf file.

compute_context

a RxComputeContext object for prediction.

kwargs

additional parameters

Returns

a data frame or a data source object of prediction results.

See also

[rx_predict](#).

Example

```
import os
from revoscalepy import rx_dforest, rx_predict_rx_dforest, rx_import, RxOptions, RxXdfData
sample_data_path = RxOptions.get_option("sampleDataDir")
ds = RxXdfData(os.path.join(sample_data_path, "kyphosis.xdf"))
kyphosis = rx_import(input_data = ds)

# classification
formula = "Kyphosis ~ Number + Start"
method = "class"
parms = {'prior': [0.8, 0.2], 'loss': [0, 2, 3, 0], 'split': "gini"}

dforest = rx_dforest(formula, data = kyphosis, pweights = "Age", method = method,
                     parms = parms, cost = [2, 3], max_num_bins = 100, importance = True,
                     n_tree = 3, m_try = 2, sample_rate = 0.5,
                     replace = False, seed = 0, compute_oob_error = 1)
rx_pred = rx_predict_rx_dforest(dforest, data = kyphosis)
rx_pred.head()

# regression
formula = "Age ~ Number + Start"
method = "anova"
parms = {'prior': [0.8, 0.2], 'loss': [0, 2, 3, 0], 'split': "gini"}

dforest = rx_dforest(formula, data = kyphosis, pweights = "Kyphosis", method = method,
                     parms = parms, cost = [2, 3], max_num_bins = 100, importance = True,
                     n_tree = 3, m_try = 2, sample_rate = 0.5,
                     replace = False, seed = 0, compute_oob_error = 1)
rx_pred = rx_predict_rx_dforest(dforest, data = kyphosis)
rx_pred.head()
```

rx_privacy_control

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_privacy_control(enable: bool = None)
```

Description

Used for opting out of telemetry data collection, which is enabled by default.

Arguments

opt_in

A bool value that specifies whether to opt in (True) or out (False) of anonymous data usage collection. If not specified, the value is returned.

Example

```
import revoscalepy
revoscalepy.rx_privacy_control(False)
```

rx_read_object

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_read_object(src: revoscalepy.datasource.RxOdbcData.RxOdbcData,  
    key: str = None, version: str = None,  
    key_name: str = 'id', value_name: str = 'value',  
    version_name: str = 'version', deserialize: bool = True,  
    decompress: str = 'zip')
```

Description

Retrieves an ODBC data source object by its key value.

Details

Loads an object from the ODBC data source, decompressing and unserializing it (by default) in the process.
Returns the object. If the data source parameter defines a query, the key and the version parameters are ignored.

The key and the version should be of some SQL character type (CHAR, VARCHAR, NVARCHAR, etc.) supported by the data source. The value column should be a binary type (VARBINARY for instance). Some conversions to other types might work, however, they are dependent on the ODBC driver and on the underlying package functions.

Arguments

src

The object being stored into the data source.

key

A character string identifying the object. The intended use is for the key+version to be unique.

version

None or a character string which carries the version of the object. Combined with key identifies the object.

key_name

Character string specifying the column name for the key in the underlying table.

value_name

Character string specifying the column name for the objects in the underlying table.

version_name

Character string specifying the column name for the version in the underlying table.

serialize

Bool value. Dictates whether the object is to be serialized. Only raw values are supported if serialization is off.

deserialize

logical value. Defines whether the object is to be de-serialized.

decompress

Character string defining the compression algorithm to use for memDecompress.

Returns

rx_read_object returns an object. rx_write_object and rx_delete_object return bool, True on success. rx_list_keys returns a single column data frame containing strings.

Example

```
from pandas import DataFrame
from numpy import random
from revoscalepy import RxOdbcData, rx_write_object, rx_read_object, rx_list_keys, rx_delete_object

connection_string = 'Driver=SQL Server;Server=.;Database=RevoTestDb;Trusted_Connection=True;'
dest = RxOdbcData(connection_string, table = "dataframe")

df = DataFrame(random.randn(10, 5))

status = rx_write_object(dest, key = "myDf", value = df)

read_obj = rx_read_object(dest, key = "myDf")

keys = rx_list_keys(dest)

rx_delete_object(dest, key = "myDf")
```

rx_read_xdf

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_read_xdf(file: str, vars_to_keep: list = None,  
                        vars_to_drop: list = None, row_var_name: str = None,  
                        start_row: int = 1, num_rows: int = None,  
                        return_data_frame: bool = True,  
                        strings_as_factors: bool = False,  
                        max_rows_by_columns: int = None, report_progress: int = None,  
                        read_by_block: bool = False, cpp_interp: list = None)
```

Description

Read data from an ".xdf" file into a data frame.

Arguments

file

Either an RxXdfData object or a character string specifying the ".xdf" file.

vars_to_keep

List of strings of variable names to include when reading from the input data file. If None, argument is ignored. Cannot be used with vars_to_drop.

vars_to_drop

List of strings of variable names to exclude when reading from the input data file. If None, argument is ignored. Cannot be used with vars_to_keep.

row_var_name

Optional character string specifying the variable in the data file to use as row names for the output data frame.

start_row

Starting row for retrieval.

num_rows

Number of rows of data to retrieve. If -1, all are read.

return_data_frame

Bool indicating whether or not to create a data frame. If False, a list is returned.

strings_as_factors

Bool indicating whether or not to convert strings into factors.

max_rows_by_columns

The maximum size of a data frame that will be read in, measured by the number of rows times the number of columns. If the number of rows times the number of columns being extracted from the ".xdf" file exceeds this, a warning will be reported and a smaller number of rows will be read in than requested. If max_rows_by_columns is set to be too large, you may experience problems from loading a huge data frame into memory. To extract a subset of rows and/or columns from an ".xdf" file, use rx_data_step.

report_progress

Integer value with options: 0: No progress is reported. 1: The number of processed rows is printed and updated. 2: Rows processed and timings are reported. 3: Rows processed and all timings are reported.

read_by_block

Read data by blocks. This argument is deprecated.

cpp_interp

List of information sent to C++ interpreter.

Returns

a data frame.

See also

[rx_import](#), [rx_data_step](#).

Example

```
import os
from revoscalepy import RxOptions, rx_read_xdf

sample_data_path = RxOptions.get_option("sampleDataDir")
mort_file = os.path.join(sample_data_path, "mortDefaultSmall.xdf")
mort_df = rx_read_xdf(mort_file, num_rows = 10)
print(mort_df)
```

Output:

```
Rows Processed: 10
Time to read data file: 0.00 secs.
Time to convert to data frame: less than .001 secs.
   creditScore  houseAge  yearsEmploy  ccDebt  year  default
0            691        16             9    6725  2000         0
1            691         4             4    5077  2000         0
2            743        18             3    3080  2000         0
3            728        22             1    4345  2000         0
4            745        17             3    2969  2000         0
5            539        15             3    4588  2000         0
6            724         6             5    4057  2000         0
7            659        14             3    6456  2000         0
8            621        18             3    1861  2000         0
9            720        14             7    4568  2000         0
```

RxRemoteComputeContext

7/12/2022 • 2 minutes to read • [Edit Online](#)

```
revoscalepy.RxRemoteComputeContext(description: str, version: str,  
id: str = None, wait: bool = True)
```

RxRemoteJob

7/12/2022 • 2 minutes to read • [Edit Online](#)

```
revoscalepy.RxRemoteJob(compute_context: revoscalepy.computecontext.RxComputeContext.RxComputeContext,  
job_id: str = None)
```

Description

Closes the remote job, purging all associated job-related data. You can reference a job by its ID, or call close() to purge jobs in a given compute context.

```
close()
```

Closes the remote job, purging all the data associated with the job

```
deserialize_job(job_id: str) -> revoscalepy.computecontext.RxRemoteJob.RxRemoteJob
```

Deserializes a RxRemoteJob given the job ID

Returns deserialize_jobs

The job that was deserialized

```
deserialize_jobs() -> list
```

Deserializes the existing jobs that have not been cleaned up by calling close().

Returns resolve_context

The serialized jobs

```
() -> revoscalepy.computecontext.RxComputeContext.RxComputeContext
```

Resolves the `RxComputeContext` that is associated with the job

Returns RxComputeContext

The `RxComputeContext` that is associated with the job or `None` if the compute context isn't valid

RxRemoteJobStatus

7/12/2022 • 2 minutes to read • [Edit Online](#)

`revoscalepy.RxRemoteJobStatus`

Description

This enumeration represents the execution status of a remote Python job. The enumeration is returned from the `rx_get_job_status` method and will indicate the current status of the job. The enumeration has the following values indicating job status:

FINISHED - The job has run to a successful completion

FAILED - The job has failed

CANCELED - The job was canceled before it could run to successful completion by using `rx_cancel_job`

UNDETERMINED - The job's status could not be determined, typically meaning it has been executed on the server and the job has already been cleaned up by calling `rx_get_job_results` or `rx_cleanup_jobs`

RUNNING - The distributed computing job is currently executing on the server

QUEUED - The distributed computing job is currently awaiting execution on the server

See also

`rx_get_job_status`

Example

```
from revoscalepy import RxInSqlServer
from revoscalepy import rx_exec
from revoscalepy import RxRemoteJobStatus
from revoscalepy import rx_get_job_status
from revoscalepy import rx_wait_for_job

connection_string = 'Driver=SQL Server;Server=.;Database=RevoTestDb;Trusted_Connection=True;'

# Setting wait to False allows the job to be run asynchronously
# Setting console_output to True allows us to get the console output of the distributed computing job
compute_context = RxInSqlServer(connection_string=connection_string,
                                 num_tasks=1,
                                 console_output=True,
                                 wait=False)

def hello_from_sql():
    import time
    print('Hello from SQL server')
    time.sleep(3)
    return 'We just ran Python code asynchronously on a SQL server!'

job = rx_exec(function=hello_from_sql, compute_context=compute_context)

# Poll initial status
status = rx_get_job_status(job)

print(status)

if status == RxRemoteJobStatus.RUNNING or status == RxRemoteJobStatus.QUEUED :
    # Wait for the job to finish or fail whatever the case may be
    rx_wait_for_job(job)

# Poll final status
status = rx_get_job_status(job)

print(status)
```

rx_set_compute_context

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_set_compute_context(compute_context:  
revoscalepy.computecontext.RxComputeContext.RxComputeContext) ->  
revoscalepy.computecontext.RxComputeContext.RxComputeContext
```

Description

Sets the active compute context for revoscalepy computations

Arguments

compute_context

Character string specifying class name or description of the specific class to instantiate, or an existing RxComputeContext object. Choices include: "RxLocalSeq" or "local", "RxInSqlServer".

Returns

rx_set_compute_context returns the previously active compute context invisibly. rx_get_compute_context returns the active compute context.

See also

[RxComputeContext](#) , [RxLocalSeq](#) , [RxInSqlServer](#) , [rx_get_compute_context](#) .

Example

```
from revoscalepy import RxLocalSeq, RxInSqlServer, rx_get_compute_context, rx_set_compute_context  
  
local_cc = RxLocalSeq()  
sql_server_cc = RxInSqlServer('Driver=SQL Server;Server=.;Database=RevoTestDb;Trusted_Connection=True;')  
previous_cc = rx_set_compute_context(sql_server_cc)  
rx_get_compute_context()
```

rx_set_var_info

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_set_var_info(var_info, data)
```

Description

Set the variable information for an .xdf file, including variable names, descriptions, and value labels, or set attributes for variables in a data frame

Arguments

var_info

list containing lists of variable information for variables in the XDF data source or data frame.

data

an RxDataSource object, a character string specifying the .xdf file, or a data frame.

Returns

If the input data is a data frame, a data frame is returned containing variables with the new attributes. If the input data represents an .xdf file or composite file, an RxXdfData object representing the modified file is returned.

See also

[rx_get_info](#) , [rx_get_var_info](#) .

Example

```

import os
import tempfile
from shutil import copy2
from revoscalepy import RxOptions, RxXdfData, rx_import, rx_get_var_info, rx_set_var_info

sample_data_path = RxOptions.get_option("sampleDataDir")
src_claims_file = os.path.join(sample_data_path, "claims.xdf")
dest_claims_file = os.path.join(tempfile.gettempdir(), "claims_new.xdf")
copy2(src_claims_file, dest_claims_file)
var_info = rx_get_var_info(src_claims_file)
print(var_info)

claims_labels = ['Type1', 'Type2', 'Type3', 'Type4']
var_info['type']['levels'] = claims_labels
var_info['cost']['low'] = 10
var_info['cost']['high'] = 1000
rx_set_var_info(var_info, dest_claims_file)

new_var_info = rx_get_var_info(dest_claims_file)
print(new_var_info)

```

Output:

```

Var 1: RowNum, Type: integer, Storage: int32, Low/High: (1.0000, 128.0000)
Var 2: age
  8 factor levels: ['17-20', '21-24', '25-29', '30-34', '35-39', '40-49', '50-59', '60+']
Var 3: car.age
  4 factor levels: ['0-3', '4-7', '8-9', '10+']
Var 4: type
  4 factor levels: ['A', 'B', 'C', 'D']
Var 5: cost, Type: numeric, Storage: float32, Low/High: (11.0000, 850.0000)
Var 6: number, Type: numeric, Storage: float32, Low/High: (0.0000, 434.0000)

Var 1: RowNum, Type: integer, Storage: int32, Low/High: (1.0000, 128.0000)
Var 2: age
  8 factor levels: ['17-20', '21-24', '25-29', '30-34', '35-39', '40-49', '50-59', '60+']
Var 3: car.age
  4 factor levels: ['0-3', '4-7', '8-9', '10+']
Var 4: type
  4 factor levels: ['Type1', 'Type2', 'Type3', 'Type4']
Var 5: cost, Type: numeric, Storage: float32, Low/High: (10.0000, 1000.0000)
Var 6: number, Type: numeric, Storage: float32, Low/High: (0.0000, 434.0000)

```

rx_serialize_model

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_serialize_model(model, realtime_scoring_only=False) -> bytes
```

Description

Serialize the given python model.

Arguments

model

Supported models are "rx_logit", "rx_lin_mod", "rx_dtree", "rx_btrees", "rx_dforest" and MicrosoftML models.

realtime_scoring_only

Boolean flag indicating if model serialization is for real-time only. Default to False

Returns

Bytes of the serialized model.

Example

```
import os
from revoscalepy import RxOptions, RxXdfData, rx_serialize_model, rx_lin_mod
sample_data_path = RxOptions.get_option("sampleDataDir")
ds = RxXdfData(os.path.join(sample_data_path, "AirlineDemoSmall.xdf"))
linmod = rx_lin_mod("ArrDelay~DayOfWeek", ds)
s_linmod = rx_serialize_model(linmod, realtime_scoring_only = False)
```

RxSpark

7/12/2022 • 2 minutes to read • [Edit Online](#)

```
revoscalepy.RxSpark(hdfs_share_dir: str = '/user/RevoShare\766RR78ROCWFDMK$',  
                     share_dir: str = '/var/RevoShare\766RR78ROCWFDMK$',  
                     user: str = '766RR78ROCWFDMK$', name_node: str = None,  
                     master: str = 'yarn', port: int = None,  
                     auto_cleanup: bool = True, console_output: bool = False,  
                     packages_to_load: list = None, idle_timeout: int = 3600,  
                     num_executors: int = None, executor_cores: int = None,  
                     executor_mem: str = None, driver_mem: str = None,  
                     executor_overhead_mem: str = None, extra_spark_config: str = '',  
                     spark_reduce_method: str = 'auto', tmp_fs_work_dir: str = None,  
                     persistent_run: bool = False, wait: bool = True, **kwargs)
```

Description

Creates the compute context for running revoscalepy analysis on Spark. Note that the use of `rx_spark_connect()` is recommended over `RxSpark()` as `rx_spark_connect()` supports persistent mode with in-memory caching. Run `help(revoscalepy.rx_spark_connect)` for more information.

RxSparkData

7/12/2022 • 2 minutes to read • [Edit Online](#)

```
revoscalepy.RxSparkData(column_info: dict = None,  
                         df_source: str = None, df_type: str = None,  
                         write_factors_as_indexes: bool = False, **kwargs)
```

Description

Base class for all revoscalepy Spark data sources, including RxHiveData, RxOrcData, RxParquetData and RxSparkDataFrame. It is intended to be called from those subclasses instead of directly.

RxSparkDataFrame

7/12/2022 • 2 minutes to read • [Edit Online](#)

```
revoscalepy.RxSparkDataFrame(data=None, column_info=None,  
    write_factors_as_indexes: bool = False)
```

Description

Main generator for class RxSparkDataFrame, which extends RxSparkData.

Arguments

data

Spark data frame obj from pyspark.sql.DataFrame.

column_info

List of named variable information lists. Each variable information list contains one or more of the named elements given below. Currently available properties for a column information list are: type: Character string specifying the data type for the column.

```
Supported types are:  
"bool" (stored as uchar),  
"integer" (stored as int32),  
"int16" (alternative to integer for smaller storage space),  
"float32" (stored as FloatType),  
"numeric" (stored as float64),  
"character" (stored as string),  
"factor" (stored as uint32),  
"Date" (stored as Date, i.e. float64.)  
"POSIXct" (stored as POSIXct, i.e. float64.)
```

levels: List of strings containing the levels when type = "factor". If the levels property is not provided, factor levels will be determined by the values in the source column. If levels are provided, any value that does not match a provided level will be converted to a missing value.

write_factors_as_indexes

Bool value, if True, when writing to an RxSparkDataFrame data source, underlying factor indexes will be written instead of the string representations.

Returns

Object of class `RxSparkDataFrame`.

Example

```
import os
from revoscalepy import rx_data_step, RxSparkDataFrame
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
df = spark.createDataFrame([('Monday',3012),('Friday',1354),('Sunday',5452)], ["DayOfWeek", "Sale"])
col_info = {"DayOfWeek": {"type":"factor"}}
ds = RxSparkDataFrame(df, column_info=col_info)
out_ds = RxSparkDataFrame(write_factors_as_indexes=True)
rx_data_step(ds, out_ds)
out_df = out_ds.spark_data_frame
out_df.take(2)
# [Row(DayOfWeek=0, Sale=3012), Row(DayOfWeek=2, Sale=1354)]
```

rx_spark_cache_data

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_spark_cache_data(in_data: revoscalepy.datasource.RxDataSource.RxDataSource,  
                                cache: bool)
```

Description

Use this function to set the cache flag to control whether data objects should be cached in Spark memory system, applicable for RxXdfData, RxHiveData, RxParquetData, RxOrcData and RxSparkDataFrame.

Arguments

in_data

A data source that can be RxXdfData, RxHiveData, RxParquetData, RxOrcData or RxSparkDataFrame. RxTextData is currently not supported.

cache

Bool value controlling whether the data should be cached or not. If TRUE the data will be cached in the Spark memory system after the first use for performance enhancement.

Returns

Data object with new cache flag.

Example

```
from revoscalepy import RxHiveData, rx_spark_connect, rx_spark_cache_data  
hive_data = RxHiveData(table = "mytable")  
  
# The cache flag of hiveData is now set to TRUE.  
hive_data = rx_spark_cache_data(hive_data, True)  
  
# set cache value to False  
hive_data = rx_spark_cache_data(hive_data, False)
```

rx_spark_connect

7/12/2022 • 5 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_spark_connect(hdfs_share_dir: str = '/user/RevoShare\\766RR78ROCFDMK$',  
    share_dir: str = '/var/RevoShare\\766RR78ROCFDMK$',  
    user: str = '766RR78ROCFDMK$', name_node: str = None,  
    master: str = 'yarn', port: int = None,  
    auto_cleanup: bool = True, console_output: bool = False,  
    packages_to_load: list = None, idle_timeout: int = None,  
    num_executors: int = None, executor_cores: int = None,  
    executor_mem: str = None, driver_mem: str = None,  
    executor_overhead_mem: str = None, extra_spark_config: str = '',  
    spark_reduce_method: str = 'auto', tmp_fs_work_dir: str = None,  
    interop: typing.Union[list, str] = None,  
    reset: bool = False) -> revoscalepy.computecontext.RxSpark.RxSpark
```

Description

Creates the compute context object with RxSpark and then immediately starts the remote Spark application.

Arguments

hdfs_share_dir

Character string specifying the file sharing location within HDFS. You must have permissions to read and write to this location. When you are running in Spark local mode, this parameter will be ignored and it will be forced to be equal to the parameter share_dir.

share_dir

Character string specifying the directory on the master (perhaps edge) node that is shared among all the nodes of the cluster and any client host. You must have permissions to read and write in this directory.

user

Character string specifying the username for submitting job to the Spark cluster. Defaults to the username of the user running the python (that is, the value of `getpass.getuser()`).

name_node

Character string specifying the Spark name node hostname or IP address. Typically you can leave this at its default value. If set to a value other than "default" or the empty string (see below), this must be an address that can be resolved by the data nodes and used by them to contact the name node. Depending on your cluster, it may need to be set to a private network address such as "master.local". If set to the empty string, "", then the master process will set this to the name of the node on which it is running, as returned by `platform.node()`. If you are running in Spark local mode, this parameter defaults to `file:///`; otherwise it defaults to

```
RxOptions.get_option("hdfsHost") .
```

master

Character string specifying the master URL passed to Spark. The value of this parameter could be "yarn", "standalone" and "local". The formats of Spark's master URL (except for mesos) specified in this website <https://spark.apache.org/docs/latest/submitting-applications.html> are also supported.

port

Integer value specifying the port used by the name node for HDFS. Needs to be able to be cast to an integer. Defaults to RxOptions.get_option("hdfsPort").

auto_cleanup

Bool value, if True, the default behavior is to clean up the temporary computational artifacts and delete the result objects upon retrieval. If False, then the computational results are not deleted. Leaving this flag set to False can result in accumulation of compute artifacts which you may eventually need to delete before they fill up your hard drive.

console_output

Bool value, if True, causes the standard output of the python processes to be printed to the user console.

packages_to_load

Optional character vector specifying additional modules to be loaded on the nodes when jobs are run in this compute context.

idle_timeout

Integer value, specifying the number of seconds of being idle (i.e., not running any Spark job) before system kills the Spark process. Setting a value greater than 600 is recommended. Defaults to 3600. Setting a negative value will not timeout. pyspark interoperation will have no timeout.

num_executors

Integer value, specifying the number of executors in Spark, which is equivalent to parameter –num-executors in spark-submit app. If not specified, the default behavior is to launch as many executors as possible, which may use up all resources and prevent other users from sharing the cluster. Defaults to RxOptions.get_option("spark.numExecutors").

executor_cores

Integer value, specifying the number of cores per executor, which is equivalent to parameter –executor-cores in spark-submit app. Defaults to RxOptions.get_option("spark.executorCores").

executor_mem

Character string specifying memory per executor (e.g., 1000M, 2G), which is equivalent to parameter –executor-memory in spark-submit app. Defaults to RxOptions.get_option("spark.executorMem").

driver_mem

Character string specifying memory for driver (e.g., 1000M, 2G), which is equivalent to parameter –driver-memory in spark-submit app. Defaults to RxOptions.get_option("spark.driverMem").

executor_overhead_mem

Character string specifying memory overhead per executor (e.g. 1000M, 2G), which is equivalent to setting spark.yarn.executor.memoryOverhead in YARN. Increasing this value will allocate more memory for the python process and the ScaleR engine process in the YARN executors, so it may help resolve job failure or executor lost issues. Defaults to RxOptions.get_option("spark.executorOverheadMem").

extra_spark_config

Character string specifying extra arbitrary Spark properties (e.g., “–conf spark.speculation=true –conf spark.yarn.queue=aQueue”), which is equivalent to additional parameters passed into spark-submit app.

spark_reduce_method

Spark job collects all parallel tasks' results to reduce as final result. This parameter decides reduce strategy: oneStage/twoStage/auto. oneStage: reduce parallel tasks to one result with one reduce function; twoStage: reduce parallel tasks to square root size with the first reduce function, then reduce to final result with the second

reduce function; auto: spark will smartly select oneStage or twoStage.

tmp_fs_work_dir

Character string specifying the temporary working directory in worker nodes. It defaults to /dev/shm to utilize in-memory temporary file system for performance gain, when the size of /dev/shm is less than 1G, the default value would switch to /tmp for guarantee sufficiency. You can specify a different location if the size of /dev/shm is insufficient. Please make sure that YARN run-as user has permission to read and write to this location

interop

None or string or list of strings. Current supported interoperation values are, 'pyspark': active revoscalepy Spark compute context in existing pyspark application to support the usage of both pyspark and revoscalepy functionalities. For jobs running in AzureML, interop value 'pyspark' must be used.

reset

Bool value, if True, all cached Spark Data Frames will be freed and all existing Spark applications that belong to the current user will be shut down.

Example

```
#####
from revoscalepy import rx_spark_connect, rx_spark_disconnect
cc = rx_spark_connect()
rx_spark_disconnect(cc)

#####
##### pyspark interoperation example #####
from pyspark.sql import SparkSession
from pyspark import SparkContext
from revoscalepy import rx_spark_connect, rx_get_pyspark_connection, RxSparkDataFrame, rx_summary,
rx_data_step
spark = SparkSession.builder.master("yarn").getOrCreate()
cc = rx_spark_connect(interop = "pyspark")

# sc is equivalent to result of pyspark API call SparkContext.getOrCreate()
sc = rx_get_pyspark_connection(cc)

#####
##### algorithms with input from spark data frame #####
df = spark.createDataFrame([('Monday',3012),('Friday',1354),('Sunday',5452)], ["DayOfWeek", "Sale"])
input_df = RxSparkDataFrame(df)
rx_summary("~.", input_df)

#####
##### ETL with output to spark data frame #####
output_df = RxSparkDataFrame()
rx_data_step(input_df, output_df, vars_to_drop = ['DayOfWeek'])
odf = output_df.spark_data_frame
odf.take(2)
# [Row(Sale=3012), Row(Sale=1354)]
```

rx_spark_disconnect

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_spark_disconnect(compute_context=None)
```

Description

Shuts down the remote Spark application and switches to a local compute context. All rx* function calls after this will run in a local compute context. In pyspark-interop mode, if Spark application is started by pyspark APIs, rx_spark_disconnect will not shut down the remote Spark application but disassociate from it. Run 'help(revoscalepy.rx_spark_connect)' for more information about interop.

Arguments

compute_context

Spark compute context to be terminated by rx_spark_disconnect. If input is None, then current compute context will be used.

Example

```
from revoscalepy import rx_spark_connect, rx_spark_disconnect
cc = rx_spark_connect()
rx_spark_disconnect(cc)
```

rx_spark_list_data

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_spark_list_data(show_description: bool,  
                               compute_context: revoscalepy.computecontext.RxComputeContext = None)
```

Description

Use these functions to manage the objects cached in the Spark memory system. These functions are only applicable when using RxSpark compute context.

Arguments

show_description

Bool value, indicating whether or not to print out the detail to console.

compute_context

RxSpark compute context object.

Returns

List of all objects cached in Spark memory system for rxSparkListData.

Example

```
from revoscalepy import RxOrcData, rx_spark_connect, rx_spark_list_data, rx_lin_mod, rx_spark_cache_data  
rx_spark_connect()  
col_info = {"DayOfWeek": {"type": "factor"}}  
df = RxOrcData(file = "/share/sample_data/AirlineDemoSmallOrc", column_info = col_info)  
df = rx_spark_cache_data(df, True)  
  
# After the first run, a Spark data object is added into the list  
rx_lin_mod("ArrDelay ~ DayOfWeek", data = df)  
rx_spark_list_data(True)
```

rx_spark_remove_data

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_spark_remove_data(remove_list,  
    compute_context: revoscalepy.computecontext.RxComputeContext = None)
```

Description

Use these functions to manage the objects cached in the Spark memory system. These functions are only applicable when using RxSpark compute context.

Arguments

remove_list

Cached object or a list of cached objects need to be deleted.

compute_context

RxSpark compute context object.

Returns

No return objs for rxSparkRemoveData.

Example

```
from revoscalepy import RxOrcData, rx_spark_connect, rx_spark_disconnect, rx_spark_list_data,  
rx_spark_cache_data, rx_spark_remove_data, rx_lin_mod  
cc=rx_spark_connect()  
col_info = {"DayOfWeek": {"type": "factor"}}  
  
df = RxOrcData(file = "/share/sample_data/AirlineDemoSmallOrc", column_info = col_info)  
df = rx_spark_cache_data(df, True)  
  
# After the first run, a Spark data object is added into the list  
rx_lin_mod("ArrDelay ~ DayOfWeek", data = df)  
  
rx_spark_remove_data(df)  
rx_spark_list_data(True)  
rx_spark_disconnect(cc)
```

RxSqlServerData

7/12/2022 • 4 minutes to read • [Edit Online](#)

```
revoscalepy.RxSqlServerData(connection_string: str = None,  
    table: str = None, sql_query: str = None,  
    row_buffering: bool = True, return_data_frame: bool = True,  
    string_as_factors: bool = False, column_classes: dict = None,  
    column_info: dict = None, rows_per_read: int = 50000,  
    verbose: bool = False, use_fast_read: bool = True,  
    server: str = None, database_name: str = None,  
    user: str = None, password: str = None,  
    write_factors_as_indexes: bool = False, **kwargs)
```

Description

Main generator for class RxSqlServerData, which extends RxDataSource.

Arguments

connection_string

None or character string specifying the connection string.

table

None or character string specifying the table name. Cannot be used with sql_query.

sql_query

None or character string specifying a valid SQL select query. Cannot contain hidden characters such as tabs or newlines. Cannot be used with table. If you want to use TABLESAMPLE clause in sqlQuery, set row_buffering argument to False.

row_buffering

Bool value specifying whether or not to buffer rows on read from the database. If you are having problems with your ODBC driver, try setting this to False.

return_data_frame

Bool value indicating whether or not to convert the result from a list to a data frame (for use in rxReadNext only). If False, a list is returned.

string_as_factors

Bool value indicating whether or not to automatically convert strings to factors on import. This can be overridden by specifying "character" in column_classes and column_info. If True, the factor levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use column_info with specified "levels".

column_classes

Dictionary of column name to strings specifying the column types to use when converting the data. The element names for the vector are used to identify which column should be converted to which type. Allowable column types are: "bool" (stored as uchar), "integer" (stored as int32), "float32" (the default for floating point data for '.xdf' files), "numeric" (stored as float64 as in R), "character" (stored as string), "factor" (stored as uint32), "int16" (alternative to integer for smaller storage space), "uint16" (alternative to unsigned integer for smaller storage space), "Date" (stored as Date, i.e. float64)

Note for "factor" type, the levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use column_info with specified "levels". Note that equivalent types share the same bullet in the list above; for some types we allow both 'R-friendly' type names, as well as our own, more specific type names for '.xdf' data. Note also that specifying the column as a "factor" type is currently equivalent to "string" - for the moment, if you wish to import a column as factor data you must use the column_info argument, documented below.

column_info

List of named variable information lists. Each variable information list contains one or more of the named elements given below. The information supplied for column_info overrides that supplied for column_classes. Currently available properties for a column information list are: type: Character string specifying the data type for the column. See

column_classes argument description for the available types. Specify "factorIndex" as the type for 0-based factor indexes. levels must also be specified.

newName: Character string specifying a new name for the variable. description: character string specifying a description for the variable. levels: List of strings containing the levels when type = "factor". If the levels property is not provided, factor levels will be determined by the values in the source column. If levels are provided, any value that does not match a provided level will be converted to a missing value.

newLevels: New or replacement levels specified for a column of type "factor". It must be used in conjunction with the levels argument. After reading in the original data, the labels for each level will be replaced with the newLevels.

low: The minimum data value in the variable (used in computations using the F() function).

high: The maximum data value in the variable (used in computations using the F() function).

rows_per_read

Number of rows to read at a time.

verbose

Integer value. If 0, no additional output is printed. If 1, information on the odbc data source type (odbc or odbcFast) is printed.

use_fast_read

Bool value, specifying whether or not to use a direct ODBC connection. On Linux systems, this is the only ODBC connection available.

server

Target SQL Server instance. Can also be specified in the connection string with the Server keyword.

database_name

Database on the target SQL Server instance. Can also be specified in the connection string with the Database keyword.

user

SQL login to connect to the SQL Server instance. SQL login can also be specified in the connection string with the uid keyword.

password

Password associated with the SQL login. Can also be specified in the connection string with the pwd keyword.

write_factors_as_indexes

Bool value, if True, when writing to an RxOdbcData data source, underlying factor indexes will be written instead

of the string representations.

kwargs

Additional arguments to be passed directly to the underlying functions.

Returns

object of class `RxSqlServerData`.

Example

```
from revoscalepy import RxSqlServerData, rx_data_step

connection_string="Driver=SQL Server;Server=.;Database=RevoTestDB;Trusted_Connection=True"
query="select top 100 [ArrDelay],[CRSDepTime],[DayOfWeek] FROM airlinedemosmall"

ds = RxSqlServerData(sql_query = query, connection_string = connection_string)
df = rx_data_step(ds)
df.head()
```

rx_summary

7/12/2022 • 3 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_summary(formula: str, data, by_group_out_file=None,  
summary_stats: list = None, by_term: bool = True, pweights=None,  
fweights=None, row_selection: str = None, transforms=None,  
transform_objects=None, transform_function=None, transform_variables=None,  
transform_packages=None, transform_environment=None,  
overwrite: bool = False, use_sparse_cube: bool = None,  
remove_zero_counts: bool = None, blocks_per_read: int = None,  
rows_per_block: int = 100000, report_progress: int = None,  
verbose: int = 0, compute_context=None, **kwargs)
```

Description

Produce univariate summaries of objects in revoscalepy.

Arguments

formula

Statistical model using symbolic formulas. The formula typically does not contain a response variable, i.e. it should be of the form ~ terms.

data

either a data source object, a character string specifying a '.xdf' file, or a data frame object to summarize. If a Spark compute context is being used, this argument may also be an RxHiveData, RxOrcData, RxParquetData or RxSparkDataFrame object or a Spark data frame object from pyspark.sql.DataFrame.

by_group_out_file

None, a character string or vector of character strings specifying .xdf file names(s), or an RxXdfData object or list of RxXdfData objects. If not None, and the formula includes computations by factor, the by-group summary results will be written out to one or more '.xdf' files. If more than one .xdf file is created and a single character string is specified, an integer will be appended to the base by_group_out_file name for additional file names. The resulting RxXdfData objects will be listed in the categorical component of the output object.

summary_stats

A list of strings containing one or more of the following values: "Mean", "StdDev", "Min", "Max", "ValidObs", "MissingObs", "Sum".

by_term

bool variable. If True, missings will be removed by term (by variable or by interaction expression) before computing summary statistics. If False, observations with missings in any term will be removed before computations.

pweights

Character string specifying the variable to use as probability weights for the observations.

fweights

Character string specifying the variable to use as frequency weights for the observations.

row_selection

None. Not currently supported, reserved for future use.

transforms

None. Not currently supported, reserved for future use.

transform_objects

None. Not currently supported, reserved for future use.

transform_function

Variable transformation function.

transform_variables

List of strings of input data set variables needed for the transformation function.

transform_packages

None. Not currently supported, reserved for future use.

transform_environment

None. Not currently supported, reserved for future use.

overwrite

Bool value. If True, an existing byGroupOutFile will be overwritten. overwrite is ignored byGroupOutFile is None.

use_sparse_cube

Bool value. If True, sparse cube is used.

remove_zero_counts

Bool flag. If True, rows with no observations will be removed from the output for counts of categorical data. By default, it has the same value as useSparseCube. For large summary computation, this should be set to True, otherwise the Python interpreter may run out of memory even if the internal C++ computation succeeds.

blocks_per_read

Number of blocks to read for each chunk of data read from the data source.

rows_per_block

Maximum number of rows to write to each block in the by_group_out_file (if it is not None).

report_progress

Integer value with options: 0: No progress is reported. 1: The number of processed rows is printed and updated. 2: Rows processed and timings are reported. 3: Rows processed and all timings are reported.

verbose

Integer value. If 0, no additional output is printed. If 1, additional summary information is printed.

compute_context

A valid RxComputeContext object.

kwargs

Additional arguments to be passed directly to the Revolution Compute Engine.

Returns

An RxSummary object containing the following elements: nobs.valid: Number of valid observations. nobs.missing: Number of missing observations. sDataFrame: Data frame containing summaries for continuous variables. categorical: List of summaries for categorical variables. categorical.type: Types of categorical

summaries: can be "counts", or "cube" (for crosstab counts) or "none" (if there is no categorical summaries).

formula: Formula used to obtain the summary.

Example

```
import os
from revoscalepy import rx_summary, RxOptions, RxXdfData
sample_data_path = RxOptions.get_option("sampleDataDir")
ds = RxXdfData(os.path.join(sample_data_path, "AirlineDemoSmall.xdf"))
summary = rx_summary("ArrDelay+DayOfWeek", ds)
print(summary)
```

RxTextData

7/12/2022 • 11 minutes to read • [Edit Online](#)

```
revoscalepy.RxTextData(file: str, strings_as_factors: bool = False,
    column_classes: dict = None, column_info: dict = None,
    vars_to_keep: list = None, vars_to_drop: list = None,
    missing_value_string: str = 'NA', rows_per_read: int = 500000,
    delimiter: str = None, combine_delimiters: bool = False,
    quote_mark: str = "", decimal_point: str = '.',
    thousands_separator: str = None,
    read_date_format: str = '[%y[-][/]%m[-][/]%d]',
    read_posixct_format: str = '%y[-][/]%m[-][/]%d [%H:%M[:%S]][%p]',
    century_cutoff: int = 20, first_row_is_column_names=None,
    rows_to_sniff: int = 10000, rows_to_skip: int = 0,
    return_data_frame: bool = True,
    default_read_buffer_size: int = 10000,
    default_decimal_column_type: str = None,
    default_missing_column_type: str = None,
    write_precision: int = 7, strip_zeros: bool = False,
    quoted_delimiters: bool = False, is_fixed_format: bool = None,
    use_fast_read: bool = True, create_file_set: bool = None,
    rows_per_out_file: int = None, verbose: int = 0,
    check_vars_to_keep: bool = False, file_system: typing.Union[str,
    revoscalepy.datasource.RxFileSystem.RxFileSystem] = 'native',
    input_encoding: str = 'utf-8',
    write_factors_as_indexes: bool = False)
```

Description

Main generator for class RxTextData, which extends RxDataSource.

Arguments

file

Character string specifying a text file. If it has an '.sts' extension, it is interpreted as a fixed format schema file. If the column_info argument contains start and width information, it is interpreted as a fixed format data file. Otherwise, it is treated as a delimited text data file. See the Details section for more information on using '.sts' files.

return_data_frame

Bool value indicating whether or not to convert the result from a list to a data frame (for use in rxReadNext only). If False, a list is returned.

strings_as_factors

Bool value indicating whether or not to automatically convert strings to factors on import. This can be overridden by specifying "character" in column_classes and column_info. If True, the factor levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use column_info with specified "levels".

column_classes

Dictionary of column names to strings specifying the column types to use when converting the data. The element names for the vector are used to identify which column should be converted to which type.

Allowable column types are:

```
"bool" (stored as uchar),  
"integer" (stored as int32),  
"float32" (the default for floating point data for '.xdf' files),  
"numeric" (stored as float64 as in R),  
"character" (stored as string),  
"factor" (stored as uint32),  
"ordered" (ordered factor stored as uint32. Ordered factors are treated  
the same as factors in RevoScaleR analysis functions.),  
  
"int16" (alternative to integer for smaller storage space),  
"uint16" (alternative to unsigned integer for smaller storage space),  
"Date" (stored as Date, i.e. float64. Not supported if use_fast_read =  
True.)  
  
"POSIXct" (stored as POSIXct, i.e. float64. Not supported if  
use_fast_read = True.)
```

Note for "factor" and "ordered" types, the levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use column_info with specified "levels".

column_info

List of named variable information lists. Each variable information list contains one or more of the named elements given below. When importing fixed format data, either column_info or an '.sts' schema file should be supplied. For such fixed format data, only the variables specified will be imported. For all text types, the information supplied for column_info overrides that supplied for column_classes.

Currently available properties for a column information list are:

```
type: Character string specifying the data type for the column. See  
column_classes argument description for the available types. If the  
type is not specified for fixed format data, it will be read as  
character data.  
  
newName: Character string specifying a new name for the variable.  
description: Character string specifying a description for the variable.  
levels: List of strings containing the levels when type = "factor". If  
the levels property is not provided, factor levels will be determined  
by the values in the source column. If levels are provided, any value  
that does not match a provided level will be converted to a missing  
value.  
  
newLevels: New or replacement levels specified for a column of type  
"factor". It must be used in conjunction with the levels argument.  
After reading in the original data, the labels for each level will be  
replaced with the newLevels.  
  
low: The minimum data value in the variable (used in computations using  
the F() function.  
  
high: The maximum data value in the variable (used in computations  
using the F() function.  
  
start: The left-most position, in bytes, for the column of a fixed  
format file respectively. When all elements of column_info have  
"start", the text file is designated as a fixed format file. When none  
of the elements have it, the text file is designated as a delimited  
file. Specification of start must always be accompanied by  
specification of width.  
  
width: The number of characters in a fixed-width character column or  
the column of a fixed format file. If width is specified for a  
character column, it will be imported as a fixed-width character  
variable. Any characters beyond the fixed width will be ignored.  
Specification of width is required for all columns of a fixed format  
file (if not provided in an '.sts' file).  
  
decimalPlaces: The number of decimal places.  
index: Column index in the original delimited text data file. It is  
used as an alternative to naming the variable information list if the  
original delimited text file does not contain column names. Ignored if  
a name for the list is specified. Should not be used with fixed format  
files.
```

vars_to_keep

List of strings of variable names to include when reading from the input data file. If None, argument is ignored.
Cannot be used with vars_to_drop.

vars_to_drop

List of strings of variable names to exclude when reading from the input data file. If None, argument is ignored.
Cannot be used with vars_to_keep.

missing_value_string

Character string containing the missing value code. It can be an empty string: "".

rows_per_read

Number of rows to read at a time.

delimiter

Character string containing the character to use as the separator between variables. If None and the
column_info argument does not contain "start" and "width" information (which implies a fixed-formatted file),

the delimiter is auto-sensed from the list "", " ", ";" and " ".

combine_delimiters

Bool value indicating whether or not to treat consecutive non-space (" ") delimiters as a single delimiter. Space " " delimiters are always combined.

quote_mark

Character string containing the quotation mark. It can be an empty string: "".

decimal_point

Character string containing the decimal point. Not supported when `use_fast_read` is set to True.

thousands_separator

Character string containing the thousands separator. Not supported when `use_fast_read` is set to True.

read_date_format

Character string containing the time date format to use during read operations. Not supported when `use_fast_read` is set to True. Valid formats are:

```
%c Skip a single character (see also %w).  
%Nc Skip N characters.  
%$c Skip the rest of the input string.  
%d Day of the month as integer (01-31).  
%m Month as integer (01-12) or as character string.  
%w Skip a whitespace delimited word (see also %c).  
%y Year. If less than 100, century_cutoff is used to determine the actual year.  
%Y Year as found in the input string.  
%[, ] input the %, [, and ] characters from the input string.  
[...] square brackets within format specifications indicate optional components; if present, they are used, but they need not be there.
```

read_posixct_format

Character string containing the time date format to use during read operations. Not supported when `use_fast_read` is set to True. Valid formats are:

```
%c Skip a single character (see also %w).  
%Nc Skip N characters.  
%$c Skip the rest of the input string.  
%d Day of the month as integer (01-31).  
%H Hour as integer (00-23).  
%m Month as integer (01-12) or as character string.  
%M Minute as integer (00-59).  
%n Milliseconds as integer (00-999).  
%N Milliseconds or tenths or hundredths of second.  
%p Character string defining 'am'/'pm'.  
%S Second as integer (00-59).  
%w Skip a whitespace delimited word (see also %c).  
%y Year. If less than 100, century_cutoff is used to determine the actual year.  
%Y Year as found in the input string.  
%[, ] input the %, [, and ] characters from the input string.  
[...] square brackets within format specifications indicate optional components; if present, they are used, but they need not be there.
```

century_cutoff

Integer specifying the changeover year between the twentieth and twenty-first century if two-digit years are read. Values less than `century_cutoff` are prefixed by 20 and those greater than or equal to `century_cutoff` are prefixed by 19. If you specify 0, all two digit dates are interpreted as years in the twentieth century. Not

supported when `use_fast_read` is set to True.

first_row_is_column_names

Bool indicating if the first row represents column names for reading text. If `first_row_is_column_names` is None, then column names are auto-detected. The logic for auto-detection is: if the first row contains all values that are interpreted as character and the second row contains at least one value that is interpreted as numeric, then the first row is considered to be column names; otherwise the first row is considered to be the first data row. Not relevant for fixed format data. As for writing, it specifies to write column names as the first row. If `first_row_is_column_names` is None, the default is to write the column names as the first row.

rows_to_sniff

Number of rows to use in determining column type.

rows_to_skip

Integer indicating number of leading rows to ignore. Only supported for `use_fast_read` = True.

default_read_buffer_size

Number of rows to read into a temporary buffer. This value could affect the speed of import.

default_decimal_column_type

Used to specify a column's data type when only decimal values (possibly mixed with missing (NA) values) are encountered upon first read of the data and the column's type information is not specified via `column_info` or `column_classes`. Supported types are "float32" and "numeric", for 32-bit floating point and 64-bit floating point values, respectively.

default_missing_column_type

Used to specify a given column's data type when only missings (NAs) or blanks are encountered upon first read of the data and the column's type information is not specified via `column_info` or `column_classes`. Supported types are "float32", "numeric", and "character" for 32-bit floating point, 64-bit floating point and string values, respectively. Only supported for `use_fast_read` = True.

write_precision

Integer specifying the precision to use when writing numeric data to a file.

strip_zeros

Bool value if True, if there are only zeros after the decimal point for a numeric, it will be written as an integer to a file.

quoted_delimiters

Bool value, if True, delimiters within quoted strings will be ignored. This requires a slower parsing process. Only applicable to `use_fast_read` is set to True; delimiters within quotes are always supported when `use_fast_read` is set to False.

is_fixed_format

Bool value, if true, the input data file is treated as a fixed format file. Fixed format files must have a '.sts' file or a `column_info` argument specifying the start and width of each variable. If False, the input data file is treated as a delimited text file. If None, the text file type (fixed or delimited text) is auto-determined.

use_fast_read

None or bool value. If True, the data file is accessed directly by the Microsoft R Services Compute Engine. If False, `StatTransfer` is used to access the data file. If None, the type of text import is auto-determined. `use_fast_read` should be set to False if importing Date or POSIXct data types, if the data set contains the delimiter character inside a quoted string, if the decimalPoint is not ".", if the thousandsSeparator is not ",", if the quoteMark is not "", or if `combineDelimiters` is set to True. `use_fast_read` should be set to True if `rowsToSkip` or `defaultMissingColType` are set. If `use_fast_read` is True, by default variables containing the values True and False

or T and F will be created as bool variables. use_fast_read cannot be set to False if an HDFS file system is being used. If an incompatible setting of use_fast_read is detected, a warning will be issued and the value will be changed.

create_file_set

Bool value or None. Used only when writing. If True, a file folder of text files will be created instead of a single text file. A directory will be created whose name is the same as the text file that would otherwise be created, but with no extension. In the directory, the data will be split across a set of text files (see rows_per_out_file below for determining how many rows of data will be in each file). If False, a single text file will be created. If None, a folder of files will be created if the input data set is a composite XDF file or a folder of text files. This argument is ignored if the text file is fixed format.

rows_per_out_file

Integer value or None. If a directory of text files is being created, this will be the number of rows of data put into each file in the directory.

verbose

Integer value. If 0, no additional output is printed. If 1, information on the text type (text or textFast) is printed.

check_vars_to_keep

bool value. If True variable names specified in vars_to_keep will be checked against variables in the data set to make sure they exist. An error will be reported if not found. If there are more than 500 variables in the data set, this flag is ignored and no checking is performed.

file_system

Character string or RxFileSystem object indicating type of file system; "native" or RxNativeFileSystem object can be used for the local operating system.

input_encoding

Character string indicating the encoding used by input text. May be set to either "utf-8" (the default), or "gb18030", a standard Chinese encoding. Not supported when use_fast_read is set to True.

write_factors_as_indexes

bool. If True, when writing to an RxTextData data source, underlying factor indexes will be written instead of the string representations. Not supported when use_fast_read is set to False.

Returns

Object of class `RxTextData`.

Example

```
import os
from revoscalepy import RxOptions, RxTextData, rx_get_info
sample_data_path = RxOptions.get_option("sampleDataDir")
text_data_source = RxTextData(os.path.join(sample_data_path, "AirlineDemoSmall.csv"))
results = rx_get_info(data = text_data_source)
print(results)
```

rx_wait_for_job

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_wait_for_job(job_info: revoscalepy.computecontext.RxRemoteJob.RxRemoteJob)
```

Description

Causes Python to block on an existing distributed job until completion. This effectively turns a non-blocking job into a blocking job.

Arguments

job_info

A jobInfo object.

See also

Example

```

from revoscalepy import RxInSqlServer
from revoscalepy import RxSqlServerData
from revoscalepy import rx_logit
from revoscalepy import rx_wait_for_job
from revoscalepy import rx_get_job_results

connection_string = 'Driver=SQL Server;Server=.;Database=RevoTestDb;Trusted_Connection=True;'
airline_data_source = RxSqlServerData(sql_query='select top 1000 * from airlinedemosmall',
                                      connection_string=connection_string,
                                      column_info={
                                          'ArrDelay': {'type': 'integer'},
                                          'CRSDepTime': {'type': 'float'},
                                          'DayOfWeek': {
                                              'type': 'factor',
                                              'levels': ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
                                                         'Friday',
                                                         'Saturday', 'Sunday']
                                          }
                                      })
# Setting wait to False allows the job to be run asynchronously
compute_context = RxInSqlServer(connection_string=connection_string, num_tasks=1, wait=False)

def transform_late(dataset, context):
    dataset['Late'] = dataset['ArrDelay'] > 15
    return dataset

# Since wait is set to False on the compute_context rx_logit will return a job rather than the model
job = rx_logit(formula='Late ~ DayOfWeek',
                data=airline_data_source,
                compute_context=compute_context,
                transform_function=transform_late,
                transform_variables=['ArrDelay'])

# Wait for the job to finish
rx_wait_for_job(job)
model = rx_get_job_results(job)
print(model)

```

rx_write_object

7/12/2022 • 2 minutes to read • [Edit Online](#)

Usage

```
revoscalepy.rx_write_object(dest: revoscalepy.datasource.RxOdbcData.RxOdbcData,  
    key: str = None, value=None, version: str = None,  
    key_name: str = 'id', value_name: str = 'value',  
    version_name: str = 'version', serialize: bool = True,  
    overwrite: bool = False, compress: str = 'zip')
```

Description

Stores objects in an ODBC data source. The APIs are modeled after a simple key value store.

Details

Stores an object into the ODBC data source. The object is identified by a key, and optionally, by a version (key+version). By default, the object is serialized and compressed. Returns True if successful.

The key and the version should be of some SQL character type (CHAR, VARCHAR, NVARCHAR, etc.) supported by the data source. The value column should be a binary type (VARBINARY for instance). Some conversions to other types might work, however, they are dependent on the ODBC driver and on the underlying package functions.

Arguments

key

A character string identifying the object. The intended use is for the key+version to be unique.

value

A python object serializable by dill.

dest

The object being stored into the data source.

version

None or a character string which carries the version of the object. Combined with key identifies the object.

key_name

Character string specifying the column name for the key in the underlying table.

value_name

Character string specifying the column name for the objects in the underlying table.

version_name

Character string specifying the column name for the version in the underlying table.

serialize

Bool value. Dictates whether the object is to be serialized. Only raw values are supported if serialization is off.

compress

Character string defining the compression algorithm to use for memCompress.

deserialize

Bool value. Defines whether the object is to be de-serialized.

overwrite

Bool value. If True, rx_write_object first removes the key (or the key+version combination) before writing the new value. Even when overwrite is False, rx_write_object may still succeed if there is no database constraint (or index) enforcing uniqueness.

Returns

rx_read_object returns an object. rx_write_object and rx_delete_object return bool, True on success. rx_list_keys returns a single column data frame containing strings.

Example

```
from pandas import DataFrame
from numpy import random
from revoscalepy import RxOdbcData, rx_write_object, rx_read_object, rx_list_keys, rx_delete_object

connection_string = 'Driver=SQL Server;Server=.;Database=RevoTestDb;Trusted_Connection=True;'
dest = RxOdbcData(connection_string, table = "dataframe")

df = DataFrame(random.randn(10, 5))

status = rx_write_object(dest, key = "myDf", value = df)

read_obj = rx_read_object(dest, key = "myDf")

keys = rx_list_keys(dest)

rx_delete_object(dest, key = "myDf")
```

RxXdfData

7/12/2022 • 2 minutes to read • [Edit Online](#)

```
revoscalepy.RxXdfData(file: str, vars_to_keep=None, vars_to_drop=None,
    return_data_frame=True, strings_as_factors=False, blocks_per_read=1,
    file_system: typing.Union[str,
        revoscalepy.datasource.RxFileSystem] = 'native',
    create_composite_set=None, create_partition_set=None,
    blocks_per_composite_file=3)
```

Description

Main generator for class RxXdfData, which extends RxDataSource.

Arguments

file

Character string specifying the location of the data. For single Xdf, it is a '.xdf' file. For composite Xdf, it is a directory like '/tmp/airline'. When using distributed compute contexts like RxSpark, a directory should be used since those compute contexts always use composite Xdf.

vars_to_keep

List of strings of variable names to keep around during operations. If None, argument is ignored. Cannot be used with vars_to_drop.

vars_to_drop

list of strings of variable names to drop from operations. If None, argument is ignored. Cannot be used with vars_to_keep.

return_data_frame

Bool value indicating whether or not to convert the result to a data frame.

strings_as_factors

Bool value indicating whether or not to convert strings into factors (for reader mode only).

blocks_per_read

Number of blocks to read for each chunk of data read from the data source.

file_system

Character string or RxFileSystem object indicating type of file system; "native" or RxNativeFileSystem object can be used for the local operating system. If None, the file system will be set to that in the current compute context, if available, otherwise the fileSystem option.

create_composite_set

Bool value or None. Used only when writing. If True, a composite set of files will be created instead of a single '.xdf' file. Subdirectories 'data' and 'metadata' will be created. In the 'data' subdirectory, the data will be split across a set of '.xdfd' files (see blocks_per_composite_file below for determining how many blocks of data will be in each file). In the 'metadata' subdirectory there is a single '.xdfm' file, which contains the meta data for all of the '.xdfd' files in the 'data' subdirectory. When the compute context is RxSpark, a composite set of files is always created.

create_partition_set

Bool value or None. Used only when writing. If True, a set of files for partitioned Xdf will be created when assigning this RxXdfData object for outData of rxPartition. Subdirectories 'data' and 'metadata' will be created. In the 'data' subdirectory, the data will be split across a set of '.xdf' files (each file stores data of a single data partition, see rxPartition for details). In the 'metadata' subdirectory there is a single '.xdfp' file, which contains the meta data for all of the '.xdf' files in the 'data' subdirectory. The partitioned Xdf object is currently supported only in rxPartition and rxGetPartitions

blocks_per_composite_file

Integer value. If create_composite_set=True, this will be the number of blocks put into each '.xdfd' file in the composite set. RxSpark compute context will optimize the number of blocks based upon HDFS and Spark settings.

Returns

Object of class `RxXdfData`.

Example

```
import os
from revoscalepy import rx_data_step, RxOptions, RxXdfData
sample_data_path = RxOptions.get_option("sampleDataDir")
ds = RxXdfData(os.path.join(sample_data_path, "kyphosis.xdf"))
kyphosis = rx_data_step(ds)
```

R Function Library Reference

7/12/2022 • 3 minutes to read • [Edit Online](#)

This section contains the R reference documentation for proprietary packages from Microsoft used for data science and machine learning on premises and at scale.

You can use these libraries and functions in combination with other open-source or third-party packages, but to use the *revo* packages, your R code must run against a service or on a computer that provides the interpreters.

LIBRARY DETAILS	DESCRIPTION
Supported platforms	Machine Learning Server 9.x Microsoft R Client (Windows and Linux) Microsoft R Server 9.1 and earlier SQL Server 2016 and later (Windows only) Azure HDInsight Azure Data Science Virtual Machines
Built on:	R 3.5.2 (included when you install a product that provides this package).

R function libraries

PACKAGE	VERSION	DESCRIPTION
MicrosoftML	1.5.0	A collection of functions in Microsoft R used for machine learning operations, including training and transformations, scoring, text and image analysis, and feature extraction for deriving values from existing data.
mrsdeploy	1.1.2	Deployment functions for interactive remote execution at the command line, plus web service functions for bundling R code blocks as discrete web services that can be deployed and managed on an R Server instance. Formerly known as DeployR.
olapR	1.0.0	A collection of functions for constructing MDX queries against an OLAP cube. Runs only on the Windows platform.
RevoIOQ	8.0.7	Installation and Operational Qualification test functions, used in conjunction with the RUnit package to run a set of unit tests. It has only one user-facing function, also called RevoIOQ .

PACKAGE	VERSION	DESCRIPTION
RevoMods	11.0.0	Microsoft modifications and extensions to standard R functions. Reference documentation is online only .
RevoPemaR	10.0.0	Developer functions for coding custom parallel external memory algorithms.
RevoScaleR	9.4	Data acquisition, manipulation and transformations, visualization, and analysis. RevoScaleR provides functions for the full range of statistical and analytical tasks. It's the backbone of R Server functionality.
RevoTreeView	10.0.0	Decision tree functions, including the <code>rxDTree</code> function. Reference documentation is online only .
RevoUtils	10.0.4	Utility functions useful when programming and developing R packages.
RevoUtilsMath	10.0.0	Microsoft's distribution of the Intel Math Kernel Library (MKL). Reference documentation is online only .
sqlrutils	1.0.0	A collection of functions for executing stored procedures against SQL Server.

How to get packages

The packages documented in this section are found only on installations of the Microsoft products or Azure services that provide them. Setup programs or scripts install the proprietary R packages from Microsoft and any package dependencies. Unless noted otherwise, all of the packages listed in the preceding table are installed with the product or service.

By default, packages are installed in the `\Program Files\Microsoft\ML Server\R_SERVER\library` folder on Windows, and in the `/opt/microsoft/mlserver/9.2.1` folder on the Linux native file system.

Ships in:

- [Machine Learning Server](#)
- [SQL Server 2017 Machine Learning Services \(Windows only\)](#)
- [SQL Server Machine Learning Server \(Standalone\)](#)
- [Microsoft R Client \(Windows and Linux\)](#)
- [Microsoft R Server 9.1 and earlier](#)
- [Azure HDInsight](#)
- [Azure Data Science Virtual Machines](#)

How to list packages and versions

To get the version of an R package installed on your computer, open an R console application and execute the following command: `installed.packages()`

To determine the version of the package, open its help page to view the version number just under the title:

```
library(help=<package-name>)
```

How to list functions in a package

To list all of the functions in a package, execute the following command:

```
ls("package:RevoScaleR")
```

To search for a function by full or partial string matching, add a pattern. For example, to return all functions that include the term "Spark":

```
> ls("package:RevoScaleR", pattern = "Spark")
[1] "rxGetSparklyrConnection" "RxSpark"                  "rxSparkCacheData"
"rxSparkDisconnect"          "rxSparkConnect"
[6] "rxSparkListData"         "rxSparkRemoveData"
```

How to view built-in help pages

Most R packages come with built-in help pages that open in separate window.

1. Open an R console tool, such as Rgui.exe or another R IDE.
2. List the packages using `installed.packages()`
3. Open help for a specific package using: `library(help=<package-name>)`
4. Open help for a specific function using: `?<function_name>`

R is case-sensitive. Be sure to type the package name using the correct case: for example,

```
library(help = "RevoScaleR").
```

Deprecated & discontinued packages

The following packages exist for backward compatibility but are no longer under active development:

- RevoRpeConnector
- RevoRsrConnector
- revolpe

For a list of deprecated or discontinued functions within an existing package, see [Deprecated, discontinued, or changed features](#).

Next steps

First, read the introduction to each package to learn about common use case scenarios:

- [MicrosoftML](#)
- [mrsdeploy](#)
- [olapR](#)
- [RevoPemaR](#)
- [RevoScaleR](#)
- [RevoUtils](#)
- [sqlrutils](#)

Next, add these packages by installing [R Client](#) or [Machine Learning Server](#). R Client is free. Machine Learning Server developer edition is also free.

Lastly, follow these tutorials for hands-on experience:

- [Explore R and RevoScaleR in 25 functions](#)
- [Quickstart: Run R Code in Machine Learning Server](#)

See also

[How-to guides in Machine Learning Server](#)
[Machine Learning Server](#)
[Additional learning resources and sample datasets](#)

RevoScaleR package

7/12/2022 • 9 minutes to read • [Edit Online](#)

The **RevoScaleR** library is a collection of portable, scalable, and distributable R functions for importing, transforming, and analyzing data at scale. You can use it for descriptive statistics, generalized linear models, k-means clustering, logistic regression, classification and regression trees, and decision forests.

Functions run on the **RevoScaleR** interpreter, built on open-source R, engineered to leverage the multithreaded and multinode architecture of the host platform.

PACKAGE DETAILS	DESCRIPTION
Current version:	9.4
Built on:	R 3.5.2
Package distribution:	Machine Learning Server R Client (Windows and Linux) R Server 9.1 and earlier SQL Server 2016 and later (Windows only) Azure HDInsight Azure Data Science Virtual Machines

How to use RevoScaleR

The **RevoScaleR** library is found in Machine Learning Server and Microsoft R products. You can use any R IDE to write R script calling functions in **RevoScaleR**, but the script must run on a computer having the interpreter and libraries.

RevoScaleR is often preloaded into tools that integrate with Machine Learning Server and R Client, which means you can call functions without having to load the library. If the library is not loaded, you can load **RevoScaleR** from the command line by typing `library(RevoScaleR)`.

Run it locally

This is the default. **RevoScaleR** runs locally on all platforms, including R Client. On a standalone Linux or Windows system, data and operations are local to the machine. On Hadoop, a local compute context means that data and operations are local to current execution environment (typically, an edge node).

Run in a remote compute context

RevoScaleR runs remotely on computers that have a server installation. In a remote compute context, the script running on a local R Client or Machine Learning Server shifts execution to a remote Machine Learning Server. For example, script running on Windows might shift execution to a Spark cluster to process data there.

On distributed platforms, such as Hadoop processing frameworks (Spark and MapReduce), set the compute context to [RxSpark](#) or [RxHadoopMR](#) and give the cluster name. In this context, if you call a function that can run in parallel, the task is distributed across data nodes in the cluster, where the operation is co-located with the data.

On SQL Server, set the compute context to [RxInSqlServer](#). There are two primary use cases for remote compute context:

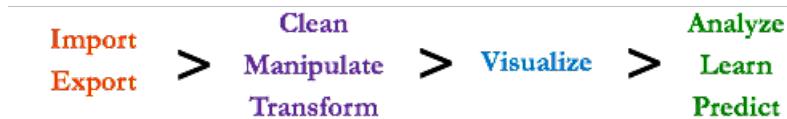
- Call R functions in T-SQL script or stored procedures running on SQL Server.

- Call **RevoScaleR** functions in R script executing in a SQL Server [compute context](#). In your script, you can set a compute context to shift execution of **RevoScaleR** operations to a remote SQL Server instance that has the **RevoScaleR** interpreter.

Some functions in **RevoScaleR** are specific to particular compute contexts. A filtered list of functions includes the following:

- [Computing on a Hadoop Cluster](#)
- [Computing on SQL Server](#)

Typical workflow



Whenever you want to perform an analysis using `RevoScaleR` functions, you should specify three distinct pieces of information:

- The **analytic function**, which specifies the analysis to be performed
- The **compute context**, which specifies where the computations should take place
- The **data source**, which is the data to be used

Functions by category

The library includes data transformation and manipulation, visualization, predictions, and statistical analysis functions. It also includes functions for controlling jobs, serializing data, and performing common utility tasks.

This section lists the functions by category to give you an idea of how each one is used. The table of contents lists functions in alphabetical order.

NOTE

Some function names begin with `rx` and others with `Rx`. The `Rx` function name prefix is used for class constructors for data sources and compute contexts.

1-Data source functions

FUNCTION NAME	DESCRIPTION
<code>RxDfData</code>	Creates an efficient XDF data source object.
<code>RxTextData</code>	Creates a comma-delimited text data source object.
<code>RxSasData</code>	Creates a SAS data source object.
<code>RxSpssData</code>	Creates an SPSS data source object.
<code>RxOdbcData</code>	Creates an ODBC data source object.
<code>RxTeradata</code>	Creates a Teradata data source object.

FUNCTION NAME	DESCRIPTION
RxSqlServerData	Creates a SQL Server data source object.

2-Import and save-as

FUNCTION NAME	DESCRIPTION
rxImport *	Creates an .xdf file or data frame from a data source (for example, text, SAS, SPSS data files, ODBC or Teradata connection, or data frame).
rxDataStep *	Transform and subset data. Creates an .xdf file, a comma-delimited text file, or data frame in memory (assuming you have sufficient memory to hold the output data) from an .xdf file or a data frame.
rxGetInfo *	Retrieves summary information from a data source or data frame.
rxSetInfo *	Sets a file description in an .xdf file or a description attribute in a data frame.
rxGetVarInfo	Retrieves variable information from a data source or data frame.
rxSetVarInfo	Modifies variable information in an .xdf file or data frame.
rxGetVarNames	Retrieves variable names from a data source or data frame.
rxCreateCollInfo	Generates a collInfo list from a data source.
rxCompressXdf	Compresses an existing .xdf file, or a directory of .xdf files.
rxIsOpen	Indicates whether a data source can be accessed.
rxOpen	Opens a data source for reading.
rxClose	Closes a data source.
rxReadNext	Read data from a source.
rxWriteNext	Writes the next chunk when moving data between RevoScaleR data sources.
rxSetFileSystem	Specify a file system type for data for import.
rxGetFileSystem	Retrieve the current file system type.
rxHdfsFileSystem	Creates an HDFS file system object.
rxNativeFileSystem	Creates a native file system object.

FUNCTION NAME	DESCRIPTION
rxSqlServerDropTable	Execute an SQL statement that drops a table.
rxSqlServerTableExists	Execute an SQL statement that checks for a table's existence.

* Signifies the most popular functions in this category.

3-Data transformation

FUNCTION NAME	DESCRIPTION
rxDataStep *	Transform and subset data. Creates an .xdf file, a comma-delimited text file, or data frame in memory (assuming you have sufficient memory to hold the output) from an .xdf file or a data frame.
rxFactors *	Recode a factor variable or convert non-factor variable into a factor in an .xdf file or data frame.
rxGetFuzzyDist	Get fuzzy distances for a character vector.
rxGetFuzzyKeys	Get fuzzy keys for a character vector.
rxSplit	Splits an .xdf file or data frame into multiple .xdf files or data frames.
rxSort	Multi-key sorting of the variables an .xdf file or data frame.
rxMerge	Merges two .xdf files or data frames using various merge types.
rxExecuteSQLDDL	SQL Server R Services only. Runs an arbitrary SQL DDL command.

* Signifies the most popular functions in this category.

4-Graphing functions

FUNCTION NAME	DESCRIPTION
rxHistogram	Creates a histogram from data.
rxLinePlot	Creates a line plot from data.
rxLorenz	Computes a Lorenz curve that can be plotted.
rxRocCurve	Computes and plots ROC curves from actual and predicted data.

5-Descriptive statistics

FUNCTION NAME	DESCRIPTION
rxQuantile *	Computes approximate quantiles for .xdf files and data frames without sorting.
rxSummary *	Basic summary statistics of data, including computations by group. Writing by group computations to .xdf file not supported.
rxCrossTabs *	Formula-based cross-tabulation of data.
rxCube *	Alternative formula-based cross-tabulation designed for efficient representation returning cube results. Writing output to .xdf file not supported.
rxMarginals	Marginal summaries of cross-tabulations.
as.xtabs	Converts cross tabulation results to an xtabs object.
rxChiSquaredTest	Performs Chi-squared Test on xtabs object. Used with small data sets and does not chunk data.
rxFisherTest	Performs Fisher's Exact Test on xtabs object. Used with small data sets and does not chunk data.
rxKendallCor	Computes Kendall's Tau Rank Correlation Coefficient using xtabs object.
rxPairwiseCrossTab	Apply a function to pairwise combinations of rows and columns of an xtabs object.
rxRiskRatio	Calculate the relative risk on a two-by-two xtabs object.
rxOddsRatio	Calculate the odds ratio on a two-by-two xtabs object.

* Signifies the most popular functions in this category.

6-Prediction functions

FUNCTION NAME	DESCRIPTION
rxLinMod *	Fits a linear model to data.
rxLogit *	Fits a logistic regression model to data.
rxGlm *	Fits a generalized linear model to data.
rxCovCor *	Calculate the covariance, correlation, or sum of squares / cross-product matrix for a set of variables.
rxDTree *	Fits a classification or regression tree to data.

FUNCTION NAME	DESCRIPTION
<code>rxBTrees</code> *	Fits a classification or regression decision forest to data using a stochastic gradient boosting algorithm.
<code>rxDForest</code> *	Fits a classification or regression decision forest to data.
<code>rxPredict</code> *	Calculates predictions for fitted models. Output must be an XDF data source.
<code>rxKmeans</code> *	Performs k-means clustering.
<code>rxNaiveBayes</code> *	Performs Naive Bayes classification.
<code>rxCov</code>	Calculate the covariance matrix for a set of variables.
<code>rxCor</code>	Calculate the correlation matrix for a set of variables.
<code>rxSSCP</code>	Calculate the sum of squares / cross-product matrix for a set of variables.
<code>rxRoc</code>	Receiver Operating Characteristic (ROC) computations using actual and predicted values from binary classifier system.

* Signifies the most popular functions in this category.

7-Compute context functions

FUNCTION NAME	DESCRIPTION
<code>RxComputeContext</code>	Creates a compute context.
<code>rxSetComputeContext</code>	Sets a compute context.
<code>rxGetComputeContext</code>	Gets the current compute context.
<code>RxSpark</code>	Creates an in-data, file-based Spark compute context. Computations are parallelized and distributed across the nodes of a Hadoop cluster via Apache Spark.
<code>RxHadoopMR</code>	Creates an in-data, file-based Hadoop compute context.
<code>RxInTeradata</code>	Creates an in-database compute context for Teradata.
<code>RxInSqlServer</code>	Creates an in-database compute context for SQL Server.
<code>RxLocalSeq</code>	Creates a local compute context for rxExec using sequential computations.
<code>RxLocalParallel</code>	Creates a local compute context for rxExec using the <code>**parallel*</code> package as backend.

FUNCTION NAME	DESCRIPTION
RxForeachDoPar	Creates a compute context for rxExec using the current foreach parallel backend.

8-Distributed computing

These functions and many more can be used for high performance computing and distributed computing. Learn more about the entire set of functions in [Distributed Computing](#).

FUNCTION NAME	DESCRIPTION
rxExec	Run an arbitrary R function on nodes or cores of a cluster.
rxRngNewStream	Support for Parallel Random Number Generation.
rxRngDelStream	Support for Parallel Random Number Generation.
rxRngGetStream	Support for Parallel Random Number Generation.
rxRngSetStream	Support for Parallel Random Number Generation.
rxGetAvailableNodes	Get all the available nodes on a distributed compute context.
rxgetNodeInfo	Get information on nodes specified for a distributed compute context.
rxPingNodes	Test round trip from user through computation node(s) in a cluster or cloud.
rxGetJobStatus	Get the status of a non-waiting distributed computing job.
rxGetJobResults	Get the return object(s) of a non-waiting distributed computing job.
rxGetJobOutput	Get the console output from a non-waiting distributed computing job.
rxGetJobs	Get the available distributed computing job information objects.
rxLocateFile	Get the first occurrence of a specified input file in a set of specified paths.

9-Utility functions

Some of the utility functions are operational in local compute context only. Check the documentation of individual functions to confirm.

FUNCTION NAME	DESCRIPTION
rxOptions	Gets or sets a specific option.

FUNCTION NAME	DESCRIPTION
rxGetOption	Retrieves a specific RevoScaleR option.
rxGetEnableThreadPool	Gets the current state of the thread pool, which on Linux can be either persistent or on-demand.
rxSetEnableThreadPool	Sets the thread pool state.
rxStepControl	Construct a variable.selection argument for rxLinMod.

10-Package management

FUNCTION NAME	DESCRIPTION
rxInstallPackages	Installs a package.
rxInstalledPackages	Returns the list of installed packages for a compute context.
rxFindPackage	Returns the path to one or more packages for a compute context.
rxRemovePackages	Removes installed packages from a compute context.
rxSqlLibPaths	Gets the search path for the library trees for packages while executing inside the SQL server.

Next steps

Add R packages to your computer by running setup:

- [Machine Learning Server](#)
- [R Client](#)

Next, follow these tutorials for hands-on experience:

- [Explore R and RevoScaleR in 25 functions](#)
- [Quickstart: Run R Code](#)

See also

[R Package Reference](#)

AirlineData87to08: Airline On-Time Performance Data

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Airline on-time performance data from 1987 to 2008.

Format

An .xdf file with 123534969 observations on the following 29 variables:

`Year`

year of the flight (stored as factor).

`Month`

month of the flight (stored as factor).

`DayOfMonth`

day of the month (1 to 31) (stored as integer).

`DayOfWeek`

day of the week (stored as factor).

`DepTime`

actual departure time (stored as float).

`CRSDepTime`

scheduled departure time (stored as float).

`ArrTime`

actual arrival time (stored as float).

`CRSArrTime`

scheduled arrival time (stored as float).

`UniqueCarrier`

carrier ID (stored as factor).

`FlightNum`

flight number (stored as factor).

`TailNum`

plane's tail number (stored as factor).

`ActualElapsedTime`

actual elapsed time of the flight, in minutes (stored as integer).

`CRSElapsedTime`

scheduled elapsed time of the flight, in minutes (stored as integer).

AirTime

airborne time for the flight, in minutes (stored as integer).

ArrDelay

arrival delay, in minutes (stored as integer).

DepDelay

departure delay, in minutes (stored as integer).

Origin

originating airport (stored as factor).

Dest

destination airport (stored as factor).

Distance

flight distance (stored as integer).

TaxiIn

taxi time from wheels down to arrival at the gate, in minutes (stored as integer).

TaxiOut

taxi time from departure from the gate to wheels up, in minutes (stored as integer).

Cancelled

cancellation status (stored as logical).

CancellationCode

cancellation code, if applicable (stored as factor).

Diverted

diversion status (stored as logical).

CarrierDelay

delay, in minutes, attributable to the carrier (stored integer).

WeatherDelay

delay, in minutes, attributable to weather factors (stored as integer).

NASDelay

delay, in minutes, attributable to the National Aviation System (stored as integer).

SecurityDelay

delay, in minutes, attributable to security factors (stored as integer).

LateAircraftDelay

delay, in minutes, attributable to late-arriving aircraft (stored as integer).

Details

This data set contains on-time performance data from 1987 to 2008. It is an .xdf file, which means that the data

are stored in *blocks*. The AirlineData87to08.xdf data file contains 832 blocks.

Source

American Statistical Association Statistical Computing Group, Data Expo '09.

<http://stat-computing.org/dataexpo/2009/the-data.html>

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

U.S. Department of Transportation, Bureau of Transportation Statistics, Research and Innovative Technology Administration. Airline On-Time Statistics. <http://www.bts.gov/xml/ontimesummarystatistics/src/index.xml>

See Also

[AirlineDemoSmall](#)

Examples

```
## Not run:  
  
airlineDemoBig <- rxDataStep(file = "AirlineData87to08.xdf",  
                           varsToKeep = c("ArrDelay", "DepDelay", "DayOfWeek"),  
                           startRow = 100000, numRows = 1000)  
summary(airlineDemoBig)  
## End(Not run)
```

AirlineDemoSmall: Small Airline Demonstration File

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

A small sample of airline on-time performance data.

Format

An .xdf file with 600000 observations on the following 3 variables:

`ArrDelay`

arrival delay, in minutes (stored as integer).

`CRSDepTime`

schedule departure time (stored as float32).

`DayOfWeek`

day of the week (stored as a factor).

Details

This data set is a small subsample of airline on-time performance data containing only 600,000 observations of 3 variables, so it will fit in memory on most systems. It is an .xdf file, which means that the data are stored in *blocks*. The AirlineDemoSmall.xdf data file contains 3 blocks. It is compressed using zlib compression. The AirlineDemoSmallUC.xdf contains the same data, but without compression.

The data are also available in text format in the file AirlineDemoSmall.csv. A smaller subset, with only 1010 rows and no missing data, is available in text format in the file AirlineDemo1kNoMissing.csv.

Source

American Statistical Association Statistical Computing Group, Data Expo '09.

<http://stat-computing.org/dataexpo/2009/the-data.html>

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

U.S. Department of Transportation, Bureau of Transportation Statistics, Research and Innovative Technology Administration. Airline On-Time Statistics. <http://www.bts.gov/xml/ontimesummarystatistics/src/index.xml>

See Also

[AirOnTime87to12](#)

Examples

```
airlineSmall <- file.path(rxGetOption("sampleDataDir"), "AirlineDemoSmall.xdf")
rxSummary(~ ArrDelay + CRSDepTime, data = airlineSmall)
```

AirOnTime87to12: Airline On-Time Performance Data

7/12/2022 • 4 minutes to read • [Edit Online](#)

Description

Airline on-time performance data from 1987 to 2012.

Format

AirOnTime87to12 is an .xdf file with 148617414 observations on the following 29 variables: AirOnTime7Pct is a 7 percent subsample with a subset of 9 variables: ArrDelay, ArrDel15, CRSDepTime, DayOfWeek, DepDelay, Dest, Origin, UniqueCarrier, Year.

`Year`

year of flight (stored as integer).

`Month`

month of the flight(stored as integer).

`DayofMonth`

day of the month (1 to 31) (stored as integer).

`DayOfWeek`

day of the week (stored as factor).

`FlightDate`

flight date (stored as Date).

`UniqueCarrier`

unique carrier code (stored as factor). When the same code has been used by multiple carriers, a numeric suffix is used for earlier users, for example, PA, PA(1), PA(2).

`TailNum`

plane's tail number (stored as factor).

`FlightNum`

flight number (stored as factor).

`OriginAirportID`

originating airport, airport ID (stored as factor). An identification number assigned by US DOT to identify a unique airport. Use this field for airport analysis across a range of years because an airport can change its airport code and airport codes can be reused.

`Origin`

originating airport (stored as factor).

`OriginState`

originating airport, state code (stored as factor).

DestAirportID

destination airport, airport ID (stored as factor). An identification number assigned by US DOT to identify a unique airport. Use this field for airport analysis across a range of years because an airport can change its airport code and airport codes can be reused.

Dest

destination airport (stored as factor).

DestState

destination airport, state code (stored as factor).

CRSDepTime

scheduled local departure time (stored as decimal float, for example, 12:45 is stored as 12.75).

DepTime

actual local departure time (stored as decimal float, for example, 12:45 is stored as 12.75).

DepDelay

difference in minutes between scheduled and actual departure time (stored as integer). Early departures show negative numbers.

DepDelayMinutes

difference in minutes between scheduled and actual departure time (stored as integer). Early departures set to 0.

DepDelay15

departure delay indicator, 15 minutes or more (stored as logical).

DepDelayGroups

departure delay intervals, every 15 minutes from < -15 to > 180 (stored as a factor).

TaxiOut

taxi time from departure from the gate to wheels off, in minutes (stored as integer).

WheelsOff

wheels off time in local time (stored as decimal float, for example, 12:45 is stored as 12.75).

WheelsOn

wheels on time in local time (stored as decimal float, for example, 12:45 is stored as 12.75).

TaxiIn

taxi in time in minutes (stored as integer).

CRSArrTime

scheduled arrival in local time (stored as decimal float, for example, 12:45 is stored as 12.75).

ArrTime

actual arrival time in local time (stored as decimal float, for example, 12:45 is stored as 12.75).

ArrDelay

difference in minutes between scheduled and actual arrival time (stored as integer). Early arrivals show negative numbers.

ArrDelayMinutes

difference in minutes between scheduled and actual arrival time (stored as integer). Early arrivals set to 0.

ArrDelay15

arrival delay indicator, 15 minutes or more (stored as logical).

ArrDelayGroups

arrival delay intervals, every 15 minutes from < -15 to > 180 (stored as a factor).

Cancelled

canceled flight indicator (stored as logical).

CancellationCode

cancellation code, if applicable (stored as factor).

Diverted

diverted flight indicator (stored as logical).

CRSElapsedTime

scheduled elapsed time of flight, in minutes (stored as integer).

ActualElapsedTime

actual elapsed time of flight, in minutes (stored as integer).

AirTime

flight time, in minutes (stored as integer).

Flights

number of flights (stored as integer).

Distance

distance between airports in miles (stored as integer).

DistanceGroup

distance intervals, every 250 miles, for flight segment (stored as a factor).

CarrierDelay

delay, in minutes, attributable to the carrier (stored as integer).

WeatherDelay

delay, in minutes, attributable to weather factors (stored as integer).

NASDelay

delay, in minutes, attributable to the National Aviation System (stored as integer).

SecurityDelay

delay, in minutes, attributable to security factors (stored as integer).

LateAircraftDelay

delay, in minutes, attributable to late-arriving aircraft (stored as integer).

MonthsSince198710

number of months since October 1987 (store as integer).

DaysSince19871001

number of days since 1 October 1987 (store as integer).

Details

These data set contain on-time performance data from 1987 to 2012. The data full data set is stored in 1024 *blocks* in an `zlib` compressed .xdf file.

Source

Research and Innovative Technology Administration (RITA), Bureau of Transportation Statistics.

http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

American Statistical Association Statistical Computing Group, Data Expo '09.

<http://stat-computing.org/dataexpo/2009/the-data.html>

U.S. Department of Transportation, Bureau of Transportation Statistics, Research and Innovative Technology Administration. Airline On-Time Statistics. <http://www.bts.gov/xml/ontimesummarystatistics/src/index.xml>

See Also

[AirlineDemoSmall](#)

Examples

```
## Not run:

#####
# Compute mean arrival delay by year using the full data set
#####
sumOut <- rxSummary(ArrDelayMinutes~Year = "AirOnTime87to12.xdf")
sumOut

#####
# To create the 7
# a subset of variables
#####
airVarsToKeep = c("Year", "DayOfWeek", "UniqueCarrier", "Origin", "Dest",
  "CRSDepTime", "DepDelay", "TaxiOut", "TaxiIn", "ArrDelay", "ArrDel15",
  "CRSElapsedTime", "Distance")

# Specify the locations for the data
bigAirData <- "C:/Microsoft/Data/AirOnTime87to12/AirOnTime87to12.xdf"
airData7Pct <- "C:/Microsoft/Data/AirOnTime7Pct.xdf"

set.seed(12)
rxDataStep(inData = bigAirData, outFile = airData7Pct,
  rowSelection = as.logical(rbinom(.rxNumRows, 1, .07)),
  varsToKeep = airVarsToKeep, blocksPerRead = 40,
  overwrite = TRUE)
rxGetInfo(airData7Pct)
rxGetVarInfo(airData7Pct)
## End(Not run)
```

CensusUS5Pct2000: IPUMS 2000 U.S. Census Data

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

The IPUMS 5% sample of the 2000 U.S. Census data in .xdf format.

Format

An .xdf file with 14,058,983 observations on 264 variables. For a complete description of the variables, see

<http://usa.ipums.org/usa-action/variables/group>.

Details

The data in CensusUS5Pct2000 comes from the *Integrated Public Use Microdata Series* (IPUMS) 5% 2000 U.S. Census sample, produced and maintained by the Minnesota Population Center at the University of Minnesota. The full data set at the time of initial conversion contained 14,583,731 rows with 265 variables. This data was converted to the .xdf format in 2006, using the SPSS dictionary provided on the IPUMS web site to obtain variable labels, value labels, and value codes. Variable names in the .xdf file are all lower case but are upper case in the IPUMS dictionary. In this version of the sample, observations with probability weights (`perwt`) equal to 0 have been removed. The "q" variables such as `qage` that take the values of 0 and 4 in the original data have been converted to logicals for more efficient storage. It was compressed and updated to a current .xdf file format in 2013.

This data set is available for download, but is not included in the standard **RevoScaleR** distribution. It is, however, used in several documentation examples and demo scripts. You are free to use this file (and subsamples of it) subject to the restrictions imposed by IPUMS, but do so at your own risk.

Source

Minnesota Population Center, University of Minnesota. *Integration Public Use Microdata Series*.

<http://www.ipums.org>.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[CensusWorkers](#)

Examples

```
## Not run:

dataPath = "C:/Microsoft/Data"
bigCensusData <- file.path(dataPath, "CensusUS5Pct2000.xdf")
censusEarly20s <- file.path(dataPath, "CensusEarly20s.xdf")
rxFDataStep(inData = bigCensusData, outFile = censusEarly20s,
            rowSelection = age >= 20 & age <= 25,
            varsToKeep = c("age", "incwage", "perwt", "sex", "wkswork1"))
rxGetInfo(censusEarly20s, getVarInfo = TRUE)
## End(Not run)
```

CensusWorkers: Subset of IPUMS 2000 U.S. Census Data

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

A small subset of the IPUMS 5% sample of the 2000 U.S. Census data containing information on the working population from Connecticut, Indiana, and Washington.

Format

An .xdf file with 351121 observations on the following 6 variables:

`age`

integer variable containing ages.

`incwage`

integer variable containing wage and salary income. The value 999999 from the original data has been converted to missing.

`perwt`

integer variable containing person weights, that is, a frequency weight for each person record. This is treated as integer data in R.

`sex`

factor variable sex with levels `Male` and `Female`.

`wkswork1`

integer variable containing the weeks worked the previous year.

`state`

factor variable containing the state with levels `Connecticut`, `Indiana`, and `Washington`.

Details

The 5% 2000 U.S. Census sample is a stratified 5% sample of households, produced and maintained by the Minnesota Population Center at the University of Minnesota as part of the *Integrated Public Use Microdata Series*, or IPUMS. The full data set contains 14,583,731 rows with 265 variables. This data was converted to the .xdf format, using the SPSS dictionary provided on the IPUMS web site to obtain variable labels, value labels, and value codes. The subsample was taken using the following rowSelection:

```
(age >= 20) & (age <= 65) & (wkswork1 > 20) & ``(stateicp == "Connecticut" | stateicp == "Washington" | stateicp == "Indiana")
```

Source

Minnesota Population Center, University of Minnesota. *Integration Public Use Microdata Series*.

<http://www.ipums.org>.

Author(s)

See Also

[CensusUS5Pct2000](#)

Examples

```
censusWorkers <- file.path(rxGetOption("sampleDataDir"),
                           "CensusWorkers.xdf")
rxSummary(~ age + incwage, data = censusWorkers)
```

claims: Auto Insurance Claims Data

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Observations on automobile insurance claims.

Format

An .xdf file with 127 observations on the following 6 variables:

`RowNum`

integer containing the row numbers.

`age`

driver's age (stored as a factor).

`car.age`

car's age (stored as a factor).

`type`

type of claim (stored as a factor).

`cost`

cost of each claim (stored as a float32).

`number`

number of claims per driver (stored as a float32).

Details

The claims.xdf file is an .xdf version of the `claims` data frame used in the White Book (*Statistical Models in S*).

The claims.txt file is a comma-separated text version of the same data. The file claimsExtra.txt is a simulated data set with four additional observations of the original six variables.

Source

Baxter, L.A., Coutts, S.M., and Ross, G.A.F. (1980) Applications of Linear Models in Motor Insurance. *Proceedings of the 21st International Congress of Actuaries*, Zurich, 11-29.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

John M. Chambers and Trevor Hastie, Ed., (1992) *Statistical Models in S*. Belmont, CA. Wadsworth.

Examples

```
claims <- rxDataStep(inData = file.path(rxGetOption("sampleDataDir"),
                           "claims.xdf"))
rxSummary(~ age + car.age, data = claims)
```

Kyphosis: Data on a Post-Operative Spinal Defect

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

The kyphosis data in .xdf format.

Format

An .xdf file with 81 observations on 4 variables. For a complete description of the variables, see [kyphosis](#).

Source

John M. Chambers and Trevor Hastie, Ed., (1992) *Statistical Models in S*. Belmont, CA. Wadsworth.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[kyphosis](#)

Examples

```
Kyphosis <- rxDataStep(inData = file.path(rxGetOption("sampleDataDir"),
                      "kyphosis.xdf"))
rxSummary(~ Kyphosis + Age + Number + Start, data = Kyphosis)
```

mortDefaultSmall: Smaller Mortgage Default Demonstration Data

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Simulated data on mortgage defaults.

Format

An .xdf file with 50000 observations on the following 6 variables:

`creditScore`

FICO credit scores (stored as an integer).

`houseAge`

house age, by range (stored as integer).

`yearsEmploy`

years employed at current employer (stored as integer).

`ccDebt`

the amount of debt (stored as integer).

`year`

year of the observation (stored as integer).

`default`

1 if the homeowner was in default on the loan, 0 otherwise (stored as an integer).

Details

This data set simulates mortgage default data. The data set is created by looping over the input data files mortDefaultSmall200x.csv, each of which contains a year's worth of simulated data.

Source

Microsoft Corporation.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

Examples

```
mortDefaultSmall <- rxDataStep(data = file.path(rxGetOption("sampleDataDir"),
                                         "mortDefaultSmall.xdf"))
rxSummary(~ creditScore + houseAge, data = mortDefaultSmall)
```

as.gbm: Conversion of an rxBTrees, rxDTree, or rpart object to a gbm Object

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Converts objects containing decision tree results to a gbm object.

Usage

```
## S3 method for class 'rxBTrees':  
as.gbm (x, ...)  
## S3 method for class 'rxDTree':  
as.gbm (x, ...)  
## S3 method for class 'rpart':  
as.gbm (x, use.weight = TRUE, cat.start = 0L, shrinkage = 1.0, ...)
```

Arguments

`x`

object of class rxBTrees, rxDTree, or rpart.

`use.weight`

a logical value (default being `TRUE`) specifying if the majority splitting direction at a node should be decided based on the sum of case weights or the number of observations when the split variable at the node is a factor or ordered factor but a certain level is not present (or not defined for the factor).

`cat.start`

an integer specifying the starting position to add the categorical splits from the current tree in the list of all the categorical splits in the collection of trees.

`shrinkage`

numeric scalar specifying the learning rate of the boosting procedure for the current tree.

`...`

additional arguments to be passed directly to `as.gbm.rpart`.

Details

These functions convert an existing object of class rxBTrees, rxDTree, or rpart to an object of class `gbm`, respectively. The underlying structure of the output object will be a subset of that produced by an equivalent call to `gbm`. Often, this method can be used to coerce an object for use with the `pmmI` package. **RevoScaleR** model objects that contain `transforms` or a `transformFunc` are not supported.

Value

an object of class gbm.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

`rxBTrees`, `rxDTree`, `rpart`, `gbm`, `as.rpart`.

Examples

```
## Not run:

# If the pmml and the gbm packages are installed
library(pmml)
library(gbm)

mydata <- infert
form <- case ~ age + parity + education + spontaneous + induced
ntree <- 2

fit.btrees <- rxBTrees(form, data = mydata, nTree = ntree,
  lossFunction = "gaussian", learningRate = 0.1)
fit.btrees
fit.btrees.gbm <- as.gbm(fit.btrees)
predict(fit.btrees.gbm, newdata = mydata, n.trees = ntree)
pmml(fit.btrees.gbm)

fit.gbm <- gbm(form, data = mydata, n.trees = ntree,
  distribution = "gaussian", shrinkage = 0.1)
fit.gbm
predict(fit.gbm, newdata = mydata, n.trees = ntree)
pmml(fit.gbm)
## End(Not run)
```

as.glm: Conversion of a RevoScaleR rxLogit or rxGlm object to a glm Object

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Converts objects containing generalized linear model results to a glm object.

Usage

```
## S3 method for class 'rxLogit':  
as.glm (x, ...)  
## S3 method for class 'rxGlm':  
as.glm (x, ...)
```

Arguments

x

object of class rxLogit or rxGlm.

...

additional arguments (currently not used).

Details

This function converts an existing object of class `rxLogit` or `rxGlm` to an object of class `glm`. The underlying structure of the output object will be a subset of that produced by an equivalent call to `glm`. Often, this method can be used to coerce an object for use with the `pmmI` package. **RevoScaleR** model objects that contain `transforms` or a `transformFunc` are not supported.

Value

an object of class `lm`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

`rxLogit`, `rxGlm`, `glm`, `as.lm`, `as.kmeans`, `as.rpart`, `as.xtabs`.

Examples

```
## Not run:

# If the pmml package is installed
library(pmml)
form <- case ~ age + parity + spontaneous + induced
rxObj <- rxLogit(form, data=infert, reportProgress = 0)
pmml(as.glm(rxObj))
## End(Not run)
```

as.kmeans: Conversion of a RevoScaleR rxKmeans object to a kmeans Object

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Converts objects containing RevoScaleR-computed k-means clusters to an R kmeans object.

Usage

```
## S3 method for class 'rxKmeans':  
as.kmeans (x, ...)
```

Arguments

x

object of class "rxKmeans".

...

additional arguments (currently not used).

Details

This function converts an existing object of class "rxKmeans" an object of class "kmeans". The underlying structure of the output object will be a subset of that produced by an equivalent call to `kmeans`. Often, this method can be used to coerce an object for use with the `pmml` package. **RevoScaleR** model objects that contain `transforms` or a `transformFunc` are not supported.

Value

an object of class "kmeans".

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[as.lm](#), [as.glm](#), [as.rpart](#), [as.xtabs](#), [rxKmeans](#).

Examples

```
## Not run:

# If the pmml package is installed
library(pmml)
form <- ~ height + weight
set.seed(17)
irow <- unique(sample.int(nrow(women), 4L, replace = TRUE))[seq(2)]
centers <- women[irow,, drop = FALSE]
rxKmeansObj <- rxKmeans(form, data = women, centers = centers)
pmml(as.kmeans(rxKmeansObj))
## End(Not run)
```

as.lm: Conversion of a RevoScaleR rxLinMod object to an lm Object

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Converts objects containing linear model results to an lm object.

Usage

```
## S3 method for class 'rxLinMod':  
as.lm (x, ...)
```

Arguments

x

object of class rxLinMod.

...

additional arguments (currently not used).

Details

This function converts an existing object of class rxLinMod an object of class lm. The underlying structure of the output object will be a subset of that produced by an equivalent call to lm. Often, this method can be used to coerce an object for use with the **pmml** package. RevoScaleR model objects that contain `transforms` or a `transformFunc` are not supported.

Value

an object of class lm.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxLinMod](#), lm, [as.glm](#), [as.kmeans](#), [as.rpart](#), [as.xtabs](#).

Examples

```
## Not run:  
  
# If the pmml package is installed  
library(pmml)  
rxObj <- rxLinMod(Sepal.Length ~ Sepal.Width + Petal.Length, data=iris)  
pmml(as.lm(rxObj))  
## End(Not run)
```

as.naiveBayes: Conversion of a RevoScaleR rxNaiveBayes object to a e1071 naiveBayes object

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Converts RevoScaleR rxNaiveBayes objects to a (limited) e1071 naiveBayes object.

Usage

```
## S3 method for class 'rxNaiveBayes':  
as.naiveBayes (x, ...)
```

Arguments

x

object of class rxNaiveBayes.

...

additional arguments (currently not used).

Details

This function converts an existing object of class `rxNaiveBayes` to an object of class `naiveBayes` in the `e1071` package. The underlying structure of the output object will be a subset of that produced by an equivalent call to `naiveBayes`. `RevoScaleR` model objects that contain `transforms` or a `transformFunc` are not supported.

Value

an object of class `naiveBayes` from `e1071`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

`rxNaiveBayes`, `as.lm`, `as.kmeans`, `as.glm`, `as.gbm`, `as.xtabs`.

Examples

```
## Not run:  
  
# If the e1071 package is installed  
library(e1071)  
rxBayes1 <- rxNaiveBayes(Kyphosis ~ Start + Age + Number + Start, data = kyphosis)  
as.naiveBayes(rxBayes1)  
## End(Not run)
```

as.randomForest: Conversion of an rxDForest, rxDTree, or rpart object to an randomForest Object

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Converts objects containing decision tree results to an randomForest object.

Usage

```
## S3 method for class 'rxDForest':  
as.randomForest (x, ...)  
## S3 method for class 'rxDTree':  
as.randomForest (x, ...)  
## S3 method for class 'rpart':  
as.randomForest (x, use.weight = TRUE, ties.method = c("random", "first", "last"), ...)
```

Arguments

x

object of class rxDForest, rxDTree, or rpart.

use.weight

a logical value (default being `TRUE`) specifying if the majority splitting direction at a node should be decided based on the sum of case weights or the number of observations when the split variable at the node is a factor or ordered factor but a certain level is not present (or not defined for the factor).

ties.method

a character string specifying how ties are handled when deciding the majority direction, with the default being `"random"`. Refer to `max.col` for details.

...

additional arguments to be passed directly to `as.randomForest.rpart`.

Details

These functions convert an existing object of class rxDForest, rxDTree, or rpart to an object of class `randomForest`, respectively. The underlying structure of the output object will be a subset of that produced by an equivalent call to `randomForest`. Often, this method can be used to coerce an object for use with the `pmml` package. **RevoScaleR** model objects that contain `transforms` or a `transformFunc` are not supported.

Value

an object of class `randomForest`.

Author(s)

See Also

[rxDForest](#), [rxDTree](#), [rpart](#), [randomForest](#), [as.rpart](#).

Examples

```
## Not run:

# If the pmml and the randomForest packages are installed
library(pmml)
library(randomForest)

mydata <- infert
form <- case ~ age + parity + education + spontaneous + induced
ntree <- 2

fit.dforest <- rxDForest(form, data = mydata, nTree = ntree)      #, cp = 0.01, seed = 1
fit.dforest
fit.dforest.rf <- as.randomForest(fit.dforest)
predict(fit.dforest.rf, newdata = mydata)
pmml(fit.dforest.rf)

fit.rf <- randomForest(form, data = mydata, ntree = ntree)
fit.rf
predict(fit.rf, newdata = mydata)
pmml(fit.rf)
## End(Not run)
```

as.rpart: Conversion of a RevoScaleR rxDTree object to a rpart Object

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Converts objects containing decision tree results to a rpart object.

Usage

```
## S3 method for class 'rxDTreer':  
as.rpart (x, ...)
```

Arguments

x

object of class rxDTree.

...

additional arguments (currently not used).

Details

This function converts an existing object of class rxDTree an object of class rpart . The underlying structure of the output object will be a subset of that produced by an equivalent call to rpart . Often, this method can be used to coerce an object for use with the pmml package. RevoScaleR model objects that contain transforms or a transformFunc are not supported.

Value

an object of class rpart.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxDTreer](#), [as.lm](#), [as.kmeans](#), [as.glm](#), [as.xtabs](#).

Examples

```
## Not run:

# If the pmml package is installed
library(pmml)
infert.nrow <- nrow(infert)
infert.sub <- sample(infert.nrow, infert.nrow / 2)
infert.dtree <- rxDTree(case ~ age + parity + education + spontaneous + induced,
  data = infert[infert.sub, ], cp = 0.01)
infert.dtree
pmml(as.rpart(infert.dtree))
## End(Not run)
```

as.xtabs: Conversion of a RevoScaleR Cross Tabulation Object to an xtabs Object

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Converts objects containing cross tabulation results to an xtabs object.

Usage

```
## S3 method for class 'rxCrossTabs':  
as.xtabs (x, ...)  
## S3 method for class 'rxCube':  
as.xtabs (x, ...)
```

Arguments

x

object of class rxCrossTabs or rxCube.

...

additional arguments (currently not used).

Details

This function converts an existing object of class rxCrossTabs or rxCube into an xtabs object. The underlying structure of the output object will be the same as that produced by an equivalent call to xtabs. However, you should expect the `"call"` attribute to be different, since it is a copy of the original call stored in the rxCrossTabs or rxCube input object. Often, this method can be used to coerce an object for use with the pmml package. RevoScaleR model objects that contain `transforms` or a `transformFunc` are not supported.

Value

an object of class xtabs.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxCrossTabs](#), [rxCube](#), [xtabs](#), [as.lm](#), [as.kmeans](#), [as.rpart](#), [as.glm](#).

Examples

```

# Define function to compare xtabs and as.xtabs output
"as.xtabs.check" <- function(convertedXtabsObject, xtabsObject)
{
  attr(convertedXtabsObject, "call") <- attr(xtabsObject, "call") <- NULL
  all.equal(convertedXtabsObject, xtabsObject)
}

# Create a data set
set.seed(100)
divs <- letters[1:5]
glads <- c("spartacus", "crixus")
romeDF <- data.frame( division = rep(divs, 5L),
                      score = runif(25, min = 0, max = 10),
                      rank = runif(25, min = 1, max = 100),
                      gladiator = c(rep(glads[1L], 12L), rep(glads[2L], 13L)),
                      arena = sample(c("colosseum", "ludus", "market"), 25L, replace = TRUE))

# Compare xtabs and as.xtabs(rxCrossTabs(..., returnXtabs = FALSE))
# results for a 3-way interaction with no dependent variables
z1 <- rxCrossTabs(~ division : gladiator : arena, data = romeDF, returnXtabs = FALSE)
z2 <- xtabs(~ division + gladiator + arena, romeDF)
as.xtabs.check(as.xtabs(z1), z2) # TRUE

# Compare xtabs and as.xtabs(rxCube(...)) results for a 3-way interaction
# with no dependent variable
z1 <- rxCube(~ division : gladiator : arena, data = romeDF, means = FALSE)
z2 <- xtabs(~ division + gladiator + arena, romeDF)
as.xtabs.check(as.xtabs(z1), z2) # TRUE

```

prune.rxDTree: Pruning an rxDTree Decision Tree

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Prune a decision tree created by `rxDTee` and return the smaller tree.

Usage

```
prune.rxDTree(tree, cp, ... )
```

Arguments

`tree`

object returned from a call to `rxDTee`.

`cp`

a complexity parameter specifying the complexity at which to prune the tree. Generally, you should examine the `cptable` component of the `tree` object to determine a suitable value for `cp`.

`...`

additional arguments to be passed to other methods. (There are, in fact, no other methods called by `prune.rxDTree`.)

Details

The `prune.rxDTree` function can be used as a `prune` method for objects of class `rxDTee`, provided the `rpart` package is attached prior to attaching `RevoScaleR`.

Value

an object of class `"rxDTee"` representing the pruned tree. It is a list with components similar to those of class `"rpart"` with the following distinctions:

- `where` - A vector of integers indicating the node to which each point is allocated. This information is always returned if the data source is a data frame. If the data source is not a data frame and `outFile` is specified that is, not `NULL`, the node indices are written/appended to the specified file with a column name as defined by `outColName`.

For other components, see `rpart.object` for details.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

- Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. (1984) *Classification and Regression Trees*. Wadsworth.
- Therneau, T. M. and Atkinson, E. J. (2011) *An Introduction to Recursive Partitioning Using the RPART Routines*.
- Yael Ben-Haim and Elad Tom-Tov (2010) A streaming parallel decision tree algorithm. *Journal of Machine Learning Research* 11, 849–872.

See Also

rpart, rpart.control, rpart.object.

Examples

```
claimsData <- file.path(system.file("SampleData", package="RevoScaleR"), "claims.xdf")
claimsTree <- rxDTree(type ~ cost + number, data=claimsData, minSplit=20)
claimsTreePruned <- prune.rxDTree(claimsTree, cp=0.04)
```

rxAddInheritance: Add Inheritance to Compatible Objects

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Add a specific S3 class to the `"class"` of a compatible object.

Usage

```
rxAddInheritance(x, inheritanceClass, ...)

## S3 method for class `rxDTree':
rxAddInheritance (x, inheritanceClass = "rpart",     ... )
```

Arguments

`x`

the object to which inheritance is to be added. Specifically, for `rxAddInheritance.rxDTree`, an object of class `rxDTree`.

`inheritanceClass`

the class to be inherited from. Specifically, for `rxAddInheritance.rxDTree`, the class `"rpart"`.

`...`

additional arguments to be passed directly to other methods.

Details

The `rxAddInheritance` function is designed to be a general way to add inheritance to S3 objects that are compatible with the class for which inheritance is to be added. Note, however, that this approach is exactly as dangerous and error-prone as simply calling `class(x) <- c(class(x), inheritanceClass)`.

It is expected that classes that are designed to mimic existing R classes, but not rely on them, will have their own specific `rxAddInheritance` methods to add the R class inheritance on request.

In the case of `rxDTree` objects, these were designed to be compatible with the `"rpart"` class, so adding `rpart` inheritance is guaranteed to work.

Value

The original object, modified to include `inheritanceClass` as part of its `"class"` attribute.

In the case of `rxDTree` objects, the object may also be modified to include the `functions` component if it was previously `NULL`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxDTree](#).

Examples

```
mtcarTree <- rxDTree(mpg ~ disp, data=mtcars)
mtcarRpart <- rxAddInheritance(mtcarTree)
require(rpart)
printcp(mtcarRpart)
```

rxBTrees: Parallel External Memory Algorithm for Stochastic Gradient Boosted Decision Trees

7/12/2022 • 13 minutes to read • [Edit Online](#)

Description

Fit stochastic gradient boosted decision trees on an .xdf file or data frame for small or large data using parallel external memory algorithm.

Usage

```
rxBTrees(formula, data,
         outFile = NULL, writeModelVars = FALSE, overwrite = FALSE,
         pweights = NULL, fweights = NULL, cost = NULL,
         minSplit = NULL, minBucket = NULL, maxDepth = 1, cp = 0,
         maxCompete = 0, maxSurrogate = 0, useSurrogate = 2, surrogateStyle = 0,
         nTree = 10, mTry = NULL, replace = FALSE,
         strata = NULL, sampRate = NULL, importance = FALSE, seed = sample.int(.Machine$integer.max, 1),
         lossFunction = "bernoulli", learningRate = 0.1,
         maxNumBins = NULL, maxUnorderedLevels = 32, removeMissings = FALSE,
         useSparseCube = rxGetOption("useSparseCube"), findSplitsInParallel = TRUE,
         scheduleOnce = FALSE,
         rowSelection = NULL, transforms = NULL, transformObjects = NULL, transformFunc = NULL,
         transformVars = NULL, transformPackages = NULL, transformEnvir = NULL,
         blocksPerRead = rxGetOption("blocksPerRead"), reportProgress = rxGetOption("reportProgress"),
         verbose = 0, computeContext = rxGetOption("computeContext"),
         xdfCompressionLevel = rxGetOption("xdfCompressionLevel"),
         ... )

## S3 method for class `rxBTrees':
print (x, by.class = FALSE, ...)

## S3 method for class `rxBTrees':
plot (x, type = "l", lty = 1:5, lwd = 1, pch = NULL, col = 1:6,
       main = deparse(substitute(x)), by.class = FALSE, ... )
```

Arguments

`formula`

formula as described in [rxFormula](#). Currently, formula functions are not supported.

`data`

either a data source object, a character string specifying a .xdf file, or a data frame object.

`outFile`

either an RxXdfData data source object or a character string specifying the .xdf file for storing the resulting OOB predictions. If `NULL` or the input data is a data frame, then no OOB predictions are stored to disk. If `rowSelection` is specified and not `NULL`, then `outFile` cannot be the same as the `data` since the resulting set of OOB predictions will generally not have the same number of rows as the original data source.

`writeModelVars`

logical value. If `TRUE`, and the output file is different from the input file, variables in the model will be written to the output file in addition to the OOB predictions. If variables from the input data set are transformed in the model, the transformed variables will also be written out.

`overwrite`

logical value. If `TRUE`, an existing `outFile` with an existing column named `outColName` will be overwritten.

`pweights`

character string specifying the variable of numeric values to use as probability weights for the observations.

`fweights`

character string specifying the variable of integer values to use as frequency weights for the observations.

`cost`

a vector of non-negative costs, containing one element for each variable in the model. Defaults to one for all variables. When deciding which split to choose, the improvement on splitting on a variable is divided by its cost.

`minSplit`

the minimum number of observations that must exist in a node before a split is attempted. By default, this is `sqrt(num of obs)`. For non-XDF data sources, as `(num of obs)` is unknown in advance, it is wisest to specify this argument directly.

`minBucket`

the minimum number of observations in a terminal node (or leaf). By default, this is `minsplit/3`.

`maxDepth`

the maximum depth of any tree node. The computations take much longer at greater depth, so lowering `maxDepth` can greatly speed up computation time.

`cp`

numeric scalar specifying the complexity parameter. Any split that does not decrease overall lack-of-fit by at least `cp` is not attempted.

`maxCompete`

the maximum number of competitor splits retained in the output. These are useful model diagnostics, as they allow you to compare splits in the output with the alternatives.

`maxSurrogate`

the maximum number of surrogate splits retained in the output. See the Details for a description of how surrogate splits are used in the model fitting. Setting this to 0 can greatly improve the performance of the algorithm; in some cases almost half the computation time is spent in computing surrogate splits.

`useSurrogate`

an integer specifying how surrogates are to be used in the splitting process:

- `0` - display-only; observations with a missing value for the primary split variable are not sent further down the tree.
- `1` - use surrogates, in order, to split observations missing the primary split variable. If all surrogates are missing, the observation is not split.
- `2` - use surrogates, in order, to split observations missing the primary split variable. If all surrogates are missing or `maxSurrogate=0`, send the observation in the majority direction.

The `0` value corresponds to the behavior of the `tree` function, and `2` (the default) corresponds to the recommendations of Breiman et al.

surrogateStyle

an integer controlling selection of a best surrogate. The default, `0`, instructs the program to use the total number of correct classifications for a potential surrogate, while `1` instructs the program to use the percentage of correct classification over the non-missing values of the surrogate. Thus, `0` penalizes potential surrogates with a large number of missing values.

nTree

a positive integer specifying the number of boosting iterations, which is generally the number of trees to grow except for `multinomial` loss function, where the number of trees to grow for each boosting iteration is equal to the number of levels of the categorical response.

mTry

a positive integer specifying the number of variables to sample as split candidates at each tree node. The default values are `sqrt(num of vars)` for classification and `(num of vars)/3` for regression.

replace

a logical value specifying if the sampling of observations should be done with or without replacement.

strata

a character string specifying the (factor) variable to use for stratified sampling.

sampRate

a scalar or a vector of positive values specifying the percentage(s) of observations to sample for each tree:

- for unstratified sampling: a scalar of positive value specifying the percentage of observations to sample for each tree. The default is 1.0 for sampling with replacement (that is, `replace=TRUE`) and 0.632 for sampling without replacement (that is, `replace=FALSE`).
- for stratified sampling: a vector of positive values of length equal to the number of strata specifying the percentages of observations to sample from the strata for each tree.

importance

a logical value specifying if the importance of predictors should be assessed.

seed

an integer that will be used to initialize the random number generator. The default is `random`. For reproducibility, you can specify the random seed either using `set.seed` or by setting this `seed` argument as part of your call.

lossFunction

character string specifying the name of the loss function to use. The following options are currently supported:

- `"gaussian"` - regression: for numeric responses.
- `"bernoulli"` - regression: for 0-1 responses.
- `"multinomial"` - classification: for categorical responses with two or more levels.

learningRate

numeric scalar specifying the learning rate of the boosting procedure.

maxNumBins

the maximum number of bins to use to cut numeric data. The default is `min(1001, max(101, sqrt(num of obs)))`. For non-XDF data sources, as `(num of obs)` is unknown in advance, it is wisest to specify this argument directly. If set to `0`, unit binning will be used instead of cutting. See the 'Details' section for more information.

`maxUnorderedLevels`

the maximum number of levels allowed for an unordered factor predictor for multiclass (>2) classification.

`removeMissings`

logical value. If `TRUE`, rows with missing values are removed and will not be included in the output data.

`useSparseCube`

logical value. If `TRUE`, sparse cube is used.

`findSplitsInParallel`

logical value. If `TRUE`, optimal splits for each node are determined using parallelization methods; this will typically speed up computation as the number of nodes on the same level is increased. Note that when it is `TRUE`, the number of nodes being processed in parallel is also printed to the console, interleaved with the number of rows read from the input data set.

`scheduleOnce`

EXPERIMENTAL. logical value. If `TRUE`, rxBTrees will be run with `rxExec`, which submits only one job to the scheduler and thus can speed up computation on small data sets particularly in the RxHadoopMR compute context.

`rowSelection`

name of a logical variable in the data set (in quotes) or a logical expression using variables in the data set to specify row selection. For example, `rowSelection = "old"` will use only observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` will use only observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

`transforms`

an expression of the form `list(name = expression, ...)` representing the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

`transformObjects`

a named list containing objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

`transformFunc`

variable transformation function. The ".rxSetLowHigh" attribute must be set for transformed variables if they are to be used in `formula`. See `rxTransform` for details.

`transformVars`

character vector of input data set variables needed for the transformation function. See `rxTransform` for details.

`transformPackages`

character vector defining additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, e.g., those explicitly defined in RevoScaleR functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.

`transformEnvir`

user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

`blocksPerRead`

number of blocks to read for each chunk of data read from the data source.

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`verbose`

integer value. If `0`, no verbose output is printed during calculations. Integer values from `1` to `2` provide increasing amounts of information are provided.

`computeContext`

a valid `RxComputeContext`. The `RxHadoopMR` compute context distributes the computation among the nodes specified by the compute context; for other compute contexts, the computation is distributed if possible on the local computer.

`xdfCompressionLevel`

integer in the range of -1 to 9 indicating the compression level for the output data if written to an `.xdf` file. The higher the value, the greater the amount of compression - resulting in smaller files but a longer time to create them. If `xdfCompressionLevel` is set to 0, there will be no compression and files will be compatible with the 6.0 release of Revolution R Enterprise. If set to -1, a default level of compression will be used.

`...`

additional arguments to be passed directly to the Microsoft R Services Compute Engine and to `rxExec` when `scheduleOnce` is set to `TRUE`.

`x`

an object of class `rxBTrees`.

`type, lty, lwd, pch, col, main`

see `plot.default` and `matplot` for details.

`by.class`

(classification with multinomial loss function only) logical value. If `TRUE`, the out-of-bag error estimate will be broken down by classes.

Details

`rxBTrees` is a parallel external memory algorithm for stochastic gradient boosted decision trees targeted for very large data sets. It is based on the gradient boosting machine of Jerome Friedman and Trevor Hastie and Robert Tibshirani and modeled after the `gbm` package of Greg Ridgeway with contributions from others, using the tree-fitting algorithm introduced in `rxDTree`.

In a decision forest, a number of decision trees are fit to bootstrap samples of the original data. Observations omitted from a given bootstrap sample are termed "out-of-bag" observations. For a given observation, the

decision forest prediction is determined by the result of sending the observation through all the trees for which it is out-of-bag. For classification, the prediction is the class to which a majority assigned the observation, and for regression, the prediction is the mean of the predictions.

For each tree, the out-of-bag observations are fed through the tree to estimate out-of-bag error estimates. The reported out-of-bag error estimates are cumulative (that is, the λ th element represents the out-of-bag error estimate for all trees through the λ th).

Value

an object of class `"rxBTrees"` inherited from class `"rxDForest"`. It is a list with the following components, similar to those of class `"rxDForest"`:

`ntree`

The number of trees.

`mtry`

The number of variables tried at each split.

`type`

One of `"class"` (for classification) or `"anova"` (for regression).

`forest`

a list containing the entire forest.

`oob.err`

a data frame containing the out-of-bag error estimate. For classification forests, this includes the OOB error estimate, which represents the proportion of times the predicted class is not equal to the true class, and the cumulative number of out-of-bag observations for the forest. For regression forests, this includes the OOB error estimate, which here represents the sum of squared residuals of the out-of-bag observations divided by the number of out-of-bag observations, the number of out-of-bag observations, the out-of-bag variance, and the "pseudo-R-Squared", which is 1 minus the quotient of the `oob.err` and `oob.var`.

`init.pred`

The initial prediction value(s).

`params`

The input parameters passed to the underlying code.

`formula`

The input formula.

`call`

The original call to `rxBTrees`.

Note

Like `rxDTree`, `rxBTrees` requires multiple passes over the data set and the maximum number of passes can be computed as follows for loss functions other than `multinomial`:

- quantile computation: 1 pass for computing the quantiles for all continuous variables,
- recursive partition: $\maxDepth + 1$ passes per tree for building the tree on the entire dataset,

- leaf prediction estimation: 1 pass per tree for estimating the optimal terminal node predictions,
- out-of-bag prediction: 1 pass per tree for computing the out-of-bag error estimates.

For `multinomial` loss function, the number of passes except for the quantile computation needs to be multiplied by the number of levels of the categorical response.

`rxBTrees` uses random streams and RNGs in parallel computation for sampling. Different threads on different nodes will be using different random streams so that different but equivalent results might be obtained for different number of threads.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

Y. Freund and R.E. Schapire (1997) A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1), 119--139.

G. Ridgeway (1999). The state of boosting. *Computing Science and Statistics* 31, 172--181.

J.H. Friedman, T. Hastie, R. Tibshirani (2000). Additive Logistic Regression: a Statistical View of Boosting. *Annals of Statistics* 28(2), 337--374.

J.H. Friedman (2001). Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics* 29(5), 1189--1232.

J.H. Friedman (2002). Stochastic Gradient Boosting. *Computational Statistics and Data Analysis* 38(4), 367--378.

Greg Ridgeway with contributions from others, `gbm`: Generalized Boosted Regression Models (R package),

<https://cran.r-project.org/web/packages/gbm/index.html>

See Also

`rxDForest`, `rxDForestUtils`, `rxPredict.rxDForest`.

Examples

```

library(RevoScaleR)
set.seed(1234)

# multi-class classification
iris.sub <- c(sample(1:50, 25), sample(51:100, 25), sample(101:150, 25))
iris.form <- Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width
iris.btrees <- rxBTrees(iris.form, data = iris[iris.sub, ], nTree = 50,
    importance = TRUE, lossFunction = "multinomial", learningRate = 0.1)

iris.btrees
plot(iris.btrees, by.class = TRUE)
rxVarImpPlot(iris.btrees)

iris.pred <- rxPredict(iris.btrees, iris[-iris.sub, ], type = "class")
table(iris.pred[["Species_Pred"]], iris[-iris.sub, "Species"])

# binary response
require(rpart)
kyphosis.nrow <- nrow(kyphosis)
kyphosis.sub <- sample(kyphosis.nrow, kyphosis.nrow / 2)
kyphosis.form <- Kyphosis ~ Age + Number + Start
kyphosis.btrees <- rxBTrees(kyphosis.form, data = kyphosis[kyphosis.sub, ],
    maxDepth = 6, minSplit = 2, nTree = 50,
    lossFunction = "bernoulli", learningRate = 0.1)

kyphosis.btrees
plot(kyphosis.btrees)

kyphosis.prob <- rxPredict(kyphosis.btrees, kyphosis[-kyphosis.sub, ], type = "response")
table(kyphosis.prob[["Kyphosis_prob"]] > 0.5, kyphosis[-kyphosis.sub, "Kyphosis"])

# regression with .xdf file
claims.xdf <- file.path(rxGetOption("sampleDataDir"), "claims.xdf")
claims.form <- cost ~ age + car.age + type
claims.btrees <- rxBTrees(claims.form, data = claims.xdf,
    maxDepth = 6, minSplit = 2, nTree = 50,
    lossFunction = "gaussian", learningRate = 0.1)

claims.btrees
plot(claims.btrees)

```

rxCancelJob: Cancel Distributed Computing Job

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Causes R to cancel an existing distributed computing job.

Usage

```
rxCancelJob( jobInfo, consoleOutput = NULL)
```

Arguments

`jobInfo`

a `jobInfo` object, such as that returned by `rxExec` or one of the RevoScaleR analysis functions in a non-waiting compute context, or the current contents of the `rsgLastPendingJob` object.

`consoleOutput`

If `NULL`, the `consoleOutput` value assigned to the job by the compute context at launch is used. If `TRUE`, any console output present at the time the job is canceled is displayed. If `FALSE`, no console output is displayed.

Details

This function does not attempt to retrieve any output objects; if the output is desired, the `consoleOutput` flag can be used to display it. This function does, however, remove all job-related artifacts from the distributed computing resources, including any job results.

On Windows, if you press ESC to interrupt a job and then call `rxCancelJob`, you may get an error message if the job was not completely submitted before you pressed ESC.

Value

`TRUE` if the job is successfully canceled; `FALSE` otherwise.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxGetJobs](#), RxHadoopMR.

Examples

```
## Not run:  
  
rxCancelJob( rxgLastPendingJob, consoleOutput = TRUE )  
rxCancelJob( myNonWaitingJob, consoleOutput = FALSE )  
## End(Not run)
```

rxChiSquaredTest: Chi-squared Test, Fisher's Exact Test, and Kendall's Tau Rank Correlation Coefficient

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Chi-squared Test, Fisher's Exact Test, and Kendall's Tau Rank Correlation Coefficient

Usage

```
rxChiSquaredTest(x, ...)
rxFisherTest(x, ...)
rxKendallCor(x, type = "b", seed = NULL, ...)
```

Arguments

`x`

an object of class `xtabs`, `rxCrossTabs`, or `rxCube`.

`type`

character string specifying the version of Kendall's correlation. Supported types are `"a"`, `"b"` or `"c"`.

`seed`

seed for random number generator. If `NULL`, the seed is not set.

`...`

additional arguments to the underlying function.

Value

an object of class `rxMultiTest`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxMultiTest](#), [rxPairwiseCrossTab](#), [rxRiskRatio](#), [rxOddsRatio](#).

Examples

```
# Create a data frame with admissions data
Admit <- factor(c(rep('Admitted', 46), rep('Rejected', 668)))
Gender <- factor(c(rep('M', 22), rep('F', 24), rep('M', 351), rep('F', 317)))
Admissions <- data.frame(Admit, Gender)

# For most efficient computations, return an xtabs object from rxCrossTabs
adminXtabs <- rxCrossTabs(~Admit:Gender, data = Admissions, returnXtabs = TRUE)
rxChiSquaredTest(adminXtabs)
rxFisherTest(adminXtabs)
rxKendallCor(adminXtabs, type = "b")
```

rxCleanupJobs: Cleanup of a Distributed Computing Job or Jobs.

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Removes artifacts created while executing a distributed computing job.

Usage

```
rxCleanupJobs(jobInfoList, force = FALSE, verbose = TRUE)
```

Arguments

`jobInfoList`

`rxJobInfo` object or a list of job objects that can be obtained from [rxGetJobs](#).

`force`

logical scalar. If `TRUE`, forces removal of job directories even if there are retrievable results or if the current job state is undetermined.

`verbose`

logical scalar. If `TRUE`, will print the directories/records being deleted.

Details

If `jobInfoList` is a `jobInfo` object, `rxCleanupJobs` attempts to remove the artifacts. However, if the job has successfully completed and `force=FALSE`, `rxCleanupJobs` issues a warning saying to either set `force=TRUE` or use [rxGetJobResults](#) to get the results and delete the artifacts.

If `jobInfoList` is a list of jobs, `rxCleanupJobs` attempts to apply the cleanup rules for a single job to each element in the list.

Value

This function is called for its side effects (removing job artifacts); it does not have a useful return value.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxGetJobs](#), [rxGetJobOutput](#), [RxSpark](#), [RxHadoopMR](#), [rxGetJobResults](#)

Examples

```
## Not run:  
  
rxCleanupJobs(jobInfoList = myJobs, force = TRUE)  
  
rxCleanupJobs(rxGetJobs(rxGetOption("computeContext")))  
## End(Not run)
```

rxCompareContexts: Compare Two Compute Context Objects

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Determines if two compute contexts are equivalent.

Usage

```
rxCompareContexts(context1, context2, exactMatch = FALSE)
```

Arguments

`context1`

The first compute context to be compared.

`context2`

The second compute context to be compared.

`exactMatch`

Determines if compute contexts are matched simply by the location specified for the compute context, or by all fields in the full compute context. The location is specified by the `headNode` (if available) and `shareDir` parameters. See Details for more information.

Details

If `exactMatch=FALSE`, only the shared directory `shareDir` and the cluster head name `headNode` (if available) are compared. Otherwise, all slots are compared. However, if the `nodes` slot in either compute context is `NULL`, that slot is also omitted from the comparison. Note also that case is ignored for node name comparisons, and for LSF node lists, near matching of node names and partial domain information will be used when comparing node names.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

Examples

```
## Not run:  
  
rxCompareContexts( myContext1, myContext2 )  
## End(Not run)
```

rxCompressXdf: Compress .xdf files

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Compress one or more .xdf files

Usage

```
rxCompressXdf(inFile, outFile = NULL, xdfCompressionLevel = 1, overwrite = FALSE, reportProgress = rxGetOption("reportProgress"))
```

Arguments

inFile

An .xdf file name, an RxXdfData data source, a directory containing .xdf files, or a vector of .xdf file names or RxXdfData data sources to compress

outFile

An .xdf file name, an RxXdfData data source, a directory, or a vector of .xdf file names or RxXdfData data sources to contain the compressed files.

xdfCompressionLevel

integer in the range of -1 to 9. The higher the value, the greater the amount of compression - resulting in smaller files but a longer time to create them. If `xdfCompressionLevel` is set to 0, there will be no compression and files will be compatible with the 6.0 release of Revolution R Enterprise. If set to -1, a default level of compression will be used.

overwrite

If `outFile` is specified and is different from `inFile`, `overwrite` must be set to `TRUE` to have `outFile` overwritten.

reportProgress

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

Details

`rxCompressXdf` uses ZLIB to compress `.xdf` files in blocks. The auto compression level of -1 is equivalent to approximately 6. Typically setting the `xdfCompressionLevel` to 1 will provide an adequate amount of compression at the fastest speed.

Value

A vector of [RxXdfData](#) data sources

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxImport](#), [rxDataStep](#), [RxXdfData](#),

Examples

```
# Get location of sample uncompressed .xdf file
sampleXdf <- file.path(rxGetOption("sampleDataDir"), "AirlineDemoSmallUC.xdf")

# Create name for a temporary file
compressXdf <- tempfile(pattern = ".rxCompress", fileext = ".xdf")

# Create a new compressed .xdf file from the sample file
newDS <- rxCompressXdf(inFile = sampleXdf, outFile = compressXdf, xdfCompressionLevel = 1)

# Get information about files and compare sizes
sampleFileInfo <- file.info(sampleXdf)
compressFileInfo <- file.info(compressXdf)
sampleFileInfo$size
compressFileInfo$size

# Clean-up
file.remove(compressXdf)
```

RxComputeContext-class: Class RxComputeContext

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Base class for all RevoScaleR Compute Contexts.

Objects from the Class

A virtual class: No objects may be created from it.

Generator

The generator for classes that extend RxComputeContext is [RxComputeContext](#).

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

##See Also

[RxHadoopMR](#), [RxSpark](#), [RxInSqlServer](#), [RxLocalSeq](#), [RxLocalParallel](#), [RxForeachDoPar](#).

Examples

```
myComputeContext <- RxComputeContext("local")
is(myComputeContext, "RxComputeContext")
# [1] TRUE
is(myComputeContext, "RxLocalSeq")
# [1] TRUE
```

RxComputeContext: RevoScaleR Compute Contexts: Class Generator

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

This is the main generator for RxComputeContext S4 classes.

Usage

```
RxComputeContext( computeContext, ...)
```

Arguments

`computeContext`

character string specifying class name or description of the specific class to instantiate, or an existing `RxComputeContext` object. Choices include: "RxLocalSeq" or "local", "RxLocalParallel" or "localpar", "RxSpark" or "spark", "RxHadoopMR" or "hadoopmr", "RxInSqlServer" or "sqlserver", and "RxForeachDoPar" or "dopar".

`...`

any other arguments are passed to the class generator determined from `context`.

Details

This is a wrapper to specific class generator functions for the RevoScaleR compute context classes. For example, the `RxInSqlServer` class uses function `RxInSqlServer` as a generator. Therefore either `RxInSqlServer(...)` or `RxComputeContext("RxInSqlServer", ...)` will create an `RxInSqlServer` instance.

Value

A type of `RxComputeContext` compute context object. This object may be used to in `rxSetComputeContext` or `rxOptions` to set the compute context.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxComputeContext-class](#), [RxHadoopMR](#), [RxSpark](#), [RxInSqlServer](#), [RxLocalSeq](#), [RxLocalParallel](#), [RxForeachDoPar](#), [rxSetComputeContext](#), [rxOptions](#), [rxExec](#).

Examples

```
# Setup to run analyses on a SQL Server
## Not run:

# Note: for improved security, read connection string from a file, such as
# connectionString <- readLines("connectionString.txt")

connectionString <- "Server=MyServer;Database=MyDatabase;UID=MyUser;PWD=MyPassword"
sqlQuery <- "WITH nb AS (SELECT 0 AS n UNION ALL SELECT n+1 FROM nb where n < 9) SELECT
n1.n+10*n2.n+100*n3.n+1 AS n, ABS(CHECKSUM(NewId()))"

myServer <- RxComputeContext("RxInSqlServer",
  sqlQuery = sqlQuery, connectionString = connectionString)
rxSetComputeContext(computeContext = myServer )
## End(Not run)
```

rxCovCor: Covariance/Correlation Matrix

7/12/2022 • 6 minutes to read • [Edit Online](#)

Description

Calculate the covariance, correlation, or sum of squares / cross-product matrix for a set of variables.

Usage

```
rxCovCor(formula, data, pweights = NULL, fweights = NULL, rowSelection = NULL,
          transforms = NULL, transformObjects = NULL,
          transformFunc = NULL, transformVars = NULL,
          transformPackages = NULL, transformEnvir = NULL,
          keepAll = TRUE, varTol = 1e-12, type = "Cov",
          blocksPerRead = rxGetOption("blocksPerRead"),
          reportProgress = rxGetOption("reportProgress"), verbose = 0,
          computeContext = rxGetOption("computeContext"), ...)

rxCov(formula, data, pweights = NULL, fweights = NULL, rowSelection = NULL,
       transforms = NULL, transformObjects = NULL,
       transformFunc = NULL, transformVars = NULL,
       transformPackages = NULL, transformEnvir = NULL,
       keepAll = TRUE, varTol = 1e-12,
       blocksPerRead = rxGetOption("blocksPerRead"),
       reportProgress = rxGetOption("reportProgress"), verbose = 0,
       computeContext = rxGetOption("computeContext"), ...)

rxCor(formula, data, pweights = NULL, fweights = NULL, rowSelection = NULL,
      transforms = NULL, transformObjects = NULL,
      transformFunc = NULL, transformVars = NULL,
      transformPackages = NULL, transformEnvir = NULL,
      keepAll = TRUE, varTol = 1e-12,
      blocksPerRead = rxGetOption("blocksPerRead"),
      reportProgress = rxGetOption("reportProgress"), verbose = 0,
      computeContext = rxGetOption("computeContext"), ...)

rxSSCP(formula, data, pweights = NULL, fweights = NULL, rowSelection = NULL,
        transforms = NULL, transformObjects = NULL,
        transformFunc = NULL, transformVars = NULL,
        transformPackages = NULL, transformEnvir = NULL,
        keepAll = TRUE, varTol = 1e-12,
        blocksPerRead = rxGetOption("blocksPerRead"),
        reportProgress = rxGetOption("reportProgress"), verbose = 0,
        computeContext = rxGetOption("computeContext"), ...)

## S3 method for class `rxCovCor':
print (x, header = TRUE, ...)
```

Arguments

`formula`

`formula`, as described in [rxFormula](#), with all the terms on the right-hand side of the `~` separated by `+` operators. Each term may be a single variable, a transformed variable, or the interaction of (transformed) variables separated by the `:` operator. e.g. `~ x1 + log(x2) + x3 : x4`

`data`

either a data source object, a character string specifying a .xdf file, or a data frame object.

pweights

character string specifying the variable to use as probability weights for the observations. Only one of `pweights` and `fweights` may be specified at a time.

fweights

character string specifying the variable to use as frequency weights for the observations. Only one of `pweights` and `fweights` may be specified at a time.

rowSelection

name of a logical variable in the data set (in quotes) or a logical expression using variables in the data set to specify row selection. For example, `rowSelection = "old"` will use only observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` will use only observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

transforms

an expression of the form `list(name = expression, ...)` representing the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

transformObjects

a named list containing objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

transformFunc

variable transformation function. See [rxTransform](#) for details.

transformVars

character vector of input data set variables needed for the transformation function. See [rxTransform](#) for details.

transformPackages

character vector defining additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, e.g., those explicitly defined in RevoScaleR functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.

transformEnvir

user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

keepAll

logical value. If `TRUE`, all of the columns are kept in the returned matrix. If `FALSE`, columns (and corresponding rows in the returned matrix) that are symbolic linear combinations of other columns, see alias, are dropped.

varTol

numeric tolerance used to identify columns in the data matrix that have near zero variance. If the variance of a column is less than or equal to `varTol` and `keepAll=TRUE`, that column is dropped from the data matrix.

type

character string specifying the type of matrix to return. The supported types are:

- "SSCP" : Sums of Squares / Cross Products matrix.
- "Cov" : covariance matrix.
- "Cor" : correlation matrix.

The `type` argument is case insensitive, e.g. "SSCP" and "sscp" are equivalent.

blocksPerRead

number of blocks to read for each chunk of data read from the data source.

reportProgress

integer value with options:

- 0 : no progress is reported.
- 1 : the number of processed rows is printed and updated.
- 2 : rows processed and timings are reported.
- 3 : rows processed and all timings are reported.

verbose

integer value. If 0, no additional output is printed. If 1, additional summary information is printed.

computeContext

a valid RxComputeContext. The `RxSpark` and `RxHadoopMR` compute contexts distribute the computation among the nodes specified by the compute context; for other compute contexts, the computation is distributed if possible on the local computer.

...

additional arguments to be passed directly to the Revolution Compute Engine.

x

an object of class "rxCovCor".

logical value. If TRUE, header information is printed.

Details

The `rxCovCor`, and the appropriate convenience functions `rxCov`, `rxCor` and `rxSSCP`, calculates either the covariance, Pearson's correlation, or a sums of squares/cross-product matrix, which may or may not use probability or frequency weights.

The sums of squares/cross-product matrix differs from the other two output types in that an initial column of 1 s or square root of the weights, if specified, is added to the data matrix prior to multiplication so the first row and first column must be dropped from the output to obtain the cross-product of just the specified data matrix.

Value

For `rxCovCor`, an object of class "rxCovCor" with the following list elements:

CovCor

numeric matrix representing either the (weighted) covariance, correlation, or sum of squares/cross-product.

StdDevs

For type = "Cor" and "Cov", numeric vector of (weighted) standard deviations of the columns. For type = "SSCP", the standard deviations are not calculated and the return value is `numeric(0)`.

Means

numeric vector containing the (weighted) column means.

valid.obs

number of valid observations.

missing.obs

number of missing observations.

SumOfWeights

either the sum of the weights or `NA` if no weights are specified.

DroppedVars

character vector containing the names of the data columns that were dropped during the calculations.

DroppedVarIndexes

integer vector containing the indices of the data columns that were dropped during the calculations.

params

parameters sent to Microsoft R Services Compute Engine.

call

the matched call.

formula

formula as described in [rxFormula](#).

For `rxCov`, a covariance matrix.

For `rxCor`, a correlation matrix.

For `rxSSCP`, a sum of squares/cross-product matrix.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

`cov`, `cor`, [rxCovData](#), [rxCorData](#).

Examples

```
# Obtain all components from rxCorCov
form <- ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width
allCov <- rxCorCov(form, data = iris, type = "Cov")
allCov

# Direct access to covariance or correlation matrix
rxCorCov(form, data = iris, reportProgress = 0)
cov(iris[,1:4])
rxCorCov(form, data = iris, reportProgress = 0)
cor(iris[,1:4])

# Cross-product of data matrix (need to drop first row and column of output)
rxSSCP(form, data = iris, reportProgress = 0)[-1, -1]
crossprod(as.matrix(iris[,1:4]))
```

rxCovCoef: Covariance and Correlation Matrices for Linear Model Coefficients and Explanatory Variables

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Obtain covariance and correlation matrices for the coefficient estimates within `rxLinMod`, `rxLogit`, and `rxGlm` objects and explanatory variables within `rxLinMod` and `rxLogit` objects.

Usage

```
rxCovCoef(x)
rxCorCoef(x)
rxCovData(x)
rxCorData(x)
```

Arguments

`x`

object of class `rxLinMod`, `rxLogit`, or `rxGlm` that satisfies conditions in the Details section.

Details

For `rxCovCoef` and `rxCorCoef`, the `rxLinMod`, `rxLogit`, or `rxGlm` object must have been fit with `covCoef = TRUE` and `cube = FALSE`. The degrees of freedom must be greater than 0.

For `rxCovData` and `rxCorData`, the `rxLinMod` or `rxLogit` object must have been fit with an intercept term as well as with `covData = TRUE` and `cube = FALSE`.

Value

If `p` is the number of columns in the model matrix, then

For `rxCovCoef` a $p \times p$ numeric matrix containing the covariances of the model coefficients.

For `rxCorCoef` a $p \times p$ numeric matrix containing the correlations amongst the model coefficients.

For `rxCovData` a $(p - 1) \times (p - 1)$ numeric matrix containing the covariances of the non-intercept terms in the model matrix.

For `rxCorData` a $(p - 1) \times (p - 1)$ numeric matrix containing the correlations amongst the non-intercept terms in the model matrix.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

`rxLinMod`, `rxLogit`, `rxCorCor`.

Examples

```
## Example 1
# Get the covariance matrix of the estimated model coefficients
kyphXdfFileName <- file.path(rxGetOption("sampleDataDir"), "kyphosis.xdf")
kyphLogitWithCovCoef <-
  rxLogit(Kyphosis ~ Age + Number + Start, data = kyphXdfFileName,
           covCoef = TRUE, reportProgress = 0)
rxCovCoef(kyphLogitWithCovCoef)

# Compare results with results from stats::glm function
data(kyphosis, package = "rpart")
kyphGlmSummary <-
  summary(glm(Kyphosis ~ Age + Number + Start, data = kyphosis,
              family = binomial()))
kyphGlmSummary[["cov.scaled"]]

## Example 2
# Get the covariance matrix of the data
kyphXdfFileName <- file.path(rxGetOption("sampleDataDir"), "kyphosis.xdf")
kyphLogitWithCovData <-
  rxLogit(Kyphosis ~ Age + Number + Start, data = kyphXdfFileName,
           covData = TRUE, reportProgress = 0)
rxCovData(kyphLogitWithCovData)

# Compare results with stats::cov function
cov(kyphosis[2:4])

## Example 3
# Find the correlation matrices for both the coefficient estimates and the
# explanatory variables
rxCorCoef(kyphLogitWithCovCoef)
rxCorData(kyphLogitWithCovData)
```

rxCreateCollInfo: Function to generate a 'collinfo' list from a data source

7/12/2022 • 3 minutes to read • [Edit Online](#)

Description

Generates a `colInfo` list from a data source that can be used in `rxImport` or an `RxDatasource` constructor.

Usage

```
rxCreateCollInfo(data, includeLowHigh = FALSE, factorsOnly = FALSE,  
                 varsToKeep = NULL, sortLevels = FALSE, computeInfo = TRUE,  
                 useFactorIndex = FALSE)
```

Arguments

`data`

An `RxDatasource` object, a character string containing an .xdf file name, or a data frame. An object returned from `rxGetVarInfo` is also supported.

`includeLowHigh`

If `TRUE`, the low/high values will be included in the `colInfo` object. Note that this will override any actual low/high values in the data set if the `colInfo` object is applied to a different data source.

`factorsOnly`

If `TRUE`, only column information for factor variables will be included in the output.

`varsToKeep`

`NULL` to include all variables, or character vector of variables to include.

`sortLevels`

If `TRUE`, factor levels will be sorted. If factor levels represent integers, they will be put in numeric order.

`computeInfo`

If `TRUE`, a pass through the data will be taken for non-xdf data sources in order to compute factor levels and low/high values.

`useFactorIndex`

If `TRUE`, the `factorIndex` variable type will be used instead of `factor`.

Details

This function can be used to ensure consistent factor levels when importing a series of text files to xdf. It is also useful for repeated analysis on non-xdf data sources.

Value

A `colInfo` list that can be used as input for `rxImport` and in data sources such as `RxTextData` and `RxSqlServerData`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

`RxDatasource-class`, `RxTextData`, `RxSqlServerData`, `RxSpssData`, `RxSasData`, `RxOdbcData`, `RxTeradata`, `RxXdfData`, `rxImport`.

Examples

```

# Get the low/high values and factor levels before using a data source
# for import or analysis

# Create a text data source, specifying the 'yearsEmploy' should be a factor
mort1 <- file.path(rxGetOption("sampleDataDir"), "mortDefaultSmall2000.csv")
mort1DS <- RxTextData(file = mort1, colClasses = c(yearsEmploy = "factor", default = "logical"))

# By default, rxCreateColInfo will make a pass through the data to compute factor levels
# and low/high values. We'll also request that the levels be sorted
mortColInfo <- rxCreateColInfo(data = mort1DS, includeLowHigh = TRUE, sortLevels = TRUE)

# Re-create the data source, now using the computed colInfo
mort1DS <- RxTextData(file = mort1, colInfo = mortColInfo)
# Import the data
mort1DF <- rxImport(mort1DS)
levels(mort1DF$yearsEmploy)

# Or use the text data source directly in an analysis
# (not needing a pass through the data to compute the factor levels)
logitObj <- rxLogit(default~yearsEmploy, data = mort1DS)

#####
# Train a model on one imported data set, then score using another
# Train a model on the first year of the data, importing it from text to a data frame

mort1 <- file.path(rxGetOption("sampleDataDir"), "mortDefaultSmall2000.csv")
mort1DS <- RxTextData(file = mort1, colClasses = c(yearsEmploy = "factor", default = "logical"))
# Since we haven't specified factor levels, they will be created 'first come, first serve'
mort1DF <- rxImport(mort1DS)
levels(mort1DF$yearsEmploy)
# Estimate a logit model
logitObj <- rxLogit(default~yearsEmploy, data = mort1DF)

# Now import the second year of data
mort2 <- file.path(rxGetOption("sampleDataDir"), "mortDefaultSmall2001.csv")
mort2DS <- RxTextData(file = mort2, colClasses = c(yearsEmploy = "factor", default = "logical"))
mort2DF <- rxImport(mort2DS)
# The levels are in a different order
levels(mort2DF$yearsEmploy)

# If we try to use the model estimated from the first data set to predict on the second,
# predOut <- rxPredict(logitObj, data = mort2DF)
# We will get an error
#ERROR: order of factor levels in the data are inconsistent with
#the order of the model coefficients

# Instead, we can extract the colInfo from the first data set
mortColInfo <- rxCreateColInfo(data = mort1DF)
# And use it when importing the second
mort2DS <- RxTextData(file = mort2, colInfo = mortColInfo)
mort2DF <- rxImport(mort2DS)
predOut <- rxPredict(logitObj, data = mort2DF)
head(predOut)

```

rxCrossTabs: Cross Tabulation

7/12/2022 • 12 minutes to read • [Edit Online](#)

Description

Use `rxCrossTabs` to create contingency tables from cross-classifying factors using a formula interface. It performs equivalent computations to the `rxCube` function, but returns its results in a different way.

Usage

```
rxCrossTabs(formula, data, pweights = NULL, fweights = NULL, means = FALSE,
           marginals = FALSE, cube = FALSE, rowSelection = NULL,
           transforms = NULL, transformObjects = NULL,
           transformFunc = NULL, transformVars = NULL,
           transformPackages = NULL, transformEnvir = NULL,
           useSparseCube = rxGetOption("useSparseCube"),
           removeZeroCounts = useSparseCube, returnXtabs = FALSE, na.rm = FALSE,
           blocksPerRead = rxGetOption("blocksPerRead"),
           reportProgress = rxGetOption("reportProgress"), verbose = 0,
           computeContext = rxGetOption("computeContext"), ...)

## S3 method for class `rxCrossTabs':
print (x, output, header = TRUE, marginals = FALSE,
       na.rm = FALSE, ...)

## S3 method for class `rxCrossTabs':
summary (object, output, type = "%", na.rm = FALSE, ...)

## S3 method for class `rxCrossTabs':
as.list (x, output, marginals = FALSE, na.rm = FALSE, ...)

## S3 method for class `rxCrossTabs':
mean (x, marginals = TRUE, na.rm = FALSE, ...)
```

Arguments

formula

formula as described in `rxFormula` with the categorical cross-classifying variables (separated by `:`) on the right hand side.

data

either a data source object, a character string specifying a .xdf file, or a data frame object containing the cross-classifying variables.

pweights

character string specifying the variable to use as probability weights for the observations.

fweights

character string specifying the variable to use as frequency weights for the observations.

means

logical value. If `TRUE`, the mean values of the contingency table are also stored in the output object along with

the sums and counts. By default, if the mean values are stored, the `print` and `summary` methods display them. However, the `output` argument in those methods can be used to override this behavior by setting `output` equal to `"sums"` or `"counts"`.

`marginals`

logical value. If `TRUE`, a list of marginal table values is stored as an attribute named `"marginals"` for each of the contingency tables. Each `marginals` list contains entries for the row, column and grand totals or means, depending on the type of data table. To access them directly, use the [rxMarginals](#) function.

`cube`

logical value. If `TRUE`, the C++ cube functionality is called.

`rowSelection`

name of a logical variable in the data set (in quotes) or a logical expression using variables in the data set to specify row selection. For example, `rowSelection = "old"` will use only observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` will use only observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the `expression` function.

`transforms`

an expression of the form `list(name = expression, ...)` representing the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the `expression` function.

`transformObjects`

a named list containing objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

`transformFunc`

variable transformation function. The variables used in the transformation function must be specified in `transformVars` if they are not variables used in the model. See [rxTransform](#) for details.

`transformVars`

character vector of input data set variables needed for the transformation function. See [rxTransform](#) for details.

`transformPackages`

character vector defining additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, e.g., those explicitly defined in RevoScaleR functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.

`transformEnvir`

user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

`useSparseCube`

logical value. If `TRUE`, sparse cube is used. For large crosstab computation, R may run out of memory due to the resulting expanded contingency tables even if the internal C++ computation succeeds. In which cases, try to use `rxCube` instead.

`removeZeroCounts`

logical flag. If `TRUE`, rows with no observations will be removed from the contingency tables. By default, it has the same value as `useSparseCube`. Please note this affects only those zeroed counts in the final contingency table for which there are no observations in the input data. However, if the input data contains a row with *frequency* zero it will be reported in the final contingency table. This should be set to `TRUE` if the total number of combinations of factor values on the right-hand side of the `formula` is significant and as a result R might run out of memory when handling the resulting large contingency table.

`returnXtabs`

logical flag. If `TRUE`, an object of class `xtabs` is returned. Note that the only difference between the structures of an equivalent `xtabs` call output and the output of `rxCrossTabs(..., returnXtabs = TRUE)` is that they will contain different `"call"` attributes. Note also that `xtabs` expects the cross-classifying variables in the `formula` to be separated by plus (+) symbols whereas `rxCrossTabs` expects them to be separated by a colon (:) symbols.

`na.rm`

logical value. If `TRUE`, `NA` values are removed when calculating the marginal means of the contingency tables.

`blocksPerRead`

number of blocks to read for each chunk of data read from the data source.

`reportProgress`

integer value: Options are:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`verbose`

integer value. If `0`, no additional output is printed. If `1`, additional summary information is printed.

`computeContext`

a valid `RxComputeContext`. The `RxSpark` and `RxHadoopMR` compute contexts distribute the computation among the nodes specified by the compute context; for other compute contexts, the computation is distributed if possible on the local computer.

`...`

for `rxCrossTabs`, additional arguments to be passed directly to the base computational function.

`x, object`

objects of class `rxCrossTabs`.

`output`

character string used to specify the type of output to display. Choices are `"sums"`, `"counts"` and `"means"`.

logical value. If `TRUE`, header information is printed.

`type`

character string used to specify the summary to create. Choices are `"%"` or `"percentages"` and `"chisquare"` to summarize the cross-tabulation results with percentages or performs a chi-squared test for independence of factors, respectively.

Details

The output is returned in a list and the `print` and `summary` methods can be used to display and summarize the contingency table(s) contained in each element of the output list. The `print` method produces an output similar to that of the `xtabs` function. The `summary` method produces a summary table for each output contingency table and displays the column, row, and total table percentages as well as the counts.

Value

an object of class `rxCrossTabs` that contains a list of elements described as follows:

sums

list of contingency tables whose values are cross-tabulation sums. *This object is NULL if there are no dependent variables specified in the formula.* The names of the list objects are built using the dependent variables specified in the formula (if they exist) along with the independent variable factor levels corresponding to each contingency table. For example, `z <- rxCrossTabs(ncontrols ~ agegp + alcgp + tobgp, esoph); names(z$sums)` will return the character vector with elements `"ncontrols, tobgp = 0-9g/day"`, `"ncontrols, tobgp = 10-19"`, `"ncontrols, tobgp = 20-29"`, `"ncontrols, tobgp = 30+"`. Typically, the user should rely on the `print` or `summary` methods to display the cross tabulation results but you can also directly access an individual contingency table using its name in R's standard list data access methods. For example, to access the "ncontrols, tobgp = 10-19" table containing cross tabulation summations you would use `z$sums[["ncontrols, tobgp = 10-19"]]` or equivalently `z$sums[[2]]`. To print the entire list of cross-tabulation summations one would issue `print(z, output="sums")`.

counts

list of contingency tables whose values are cross-tabulation counts. The names of the list objects are equivalent to those of the 'sums' output list.

means

list of contingency tables containing cross tabulation mean values. *This object is NULL if there are no dependent variables specified in the formula.* The 'means' list is returned only if the user has specified `means=TRUE` in the call to `rxCrossTabs`. If `means=FALSE` in the call, mean values still may be calculated and returned using the `print` and `summary` methods with an `output="means"` argument. In this case, the mean values are calculated dynamically. If you wish to have quick access to the means, use `means=TRUE` in the call to `rxCrossTabs`. The names of the list objects are equivalent to those of the 'sums' output list.

call

original call to the underlying `rxCrossTabs.formula` method.

chisquare

list of chi-square tests, one for each cross-tabulation table. Each entry contains the results of a chi-squared test for independence of factors as used in the `summary` method for the `xtabs` function. The names of the list objects are equivalent to those of the 'sums' output list.

formula

formula used in the `rxCrossTabs` call.

depvars

character vector of dependent variable names as extracted from the formula.

Author(s)

See Also

[xtabs](#), [rxMarginals](#), [rxCube](#), [as.xtabs](#), [rxChiSquaredTest](#), [rxFisherTest](#), [rxKendallCor](#), [rxPairwiseCrossTab](#), [rxRiskRatio](#), [rxOddsRatio](#), [rxTransform](#).

Examples

```
# Basic data.frame source example
admissions <- as.data.frame(UCBAdmissions)
admissCTabs <- rxCrossTabs(Freq ~ Gender : Admit, data = admissions)

# print different outputs and summarize different types
print(admissCTabs) # same as print(admissCTabs, output = "sums")
print(admissCTabs, output = "counts")
print(admissCTabs, output = "means")
summary(admissCTabs) # same as summary(admissCTabs, type = "%")
summary(admissCTabs, output="means", type = "%")
summary(admissCTabs, type = "chisquare")

# Example using multiple dependent variables in formula
rxCrossTabs(ncontrols ~ agegp : alcgp : tobgp, data = esoph)
rxCrossTabs(ncases ~ agegp : alcgp : tobgp, data = esoph)
esophCTabs <-
    rxCrossTabs(cbind(ncases, ncontrols) ~ agegp : alcgp : tobgp, esoph)
esophCTabs

# Obtaining the mean values
esophMeans <- mean(esophCTabs, marginals = FALSE)
esophMeans
esophMeans <- mean(esophCTabs, marginals = TRUE)
esophMeans

# XDF example: small subset of census data
censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers.xdf")
censusCTabs <- rxCrossTabs(wkswork1 ~ sex : F(age), data = censusWorkers,
    pweights = "perwt", blocksPerRead = 3)
censusCTabs
barplot(censusCTabs$sums$wkswork1/1e6, xlab = "Age (years)",
    ylab = "Population (millions)", beside = TRUE,
    legend.text = c("Male", "Female"))

# perform a census crosstab, limiting the analysis to ages
# on the interval [20, 65]. Verify the age range from the output.
censusXtabAge.20.65 <- rxCrossTabs(wkswork1 ~ sex : F(age), data = censusWorkers,
    rowSelection = age >= 20 & age <= 65)
ageRange <- range(as.numeric(colnames(censusXtabAge.20.65$sums$wkswork1)))
(ageRange[1] >= 20 & ageRange[2] <=65)

# Create a data frame
myDF <- data.frame(sex = c("Male", "Male", "Female", "Male"),
    age = c(20, 20, 12, 15), score = 1.1:4.1, sport=c(1:3,2))

# Use the 'transforms' argument to dynamically transform the
# variables of the data source. Here, we form a named list of
# transformation expressions. To avoid evaluation when assigning
# to a local variable, we wrap the transformation list with expression().
transforms <- expression(list(
    ageHalved = age/2,
    sport = factor(sport, labels=c("tennis", "golf", "football"))))
rxCrossTabs(score ~ sport:sex, data = myDF, transforms = transforms)
rxCrossTabs(~ sport : F(ageHalved, low = 7, high = 10), data = myDF,
    transforms = transforms)
```

```

# Arithmetic formula expression only (no transformFunc specification).
rxCrossTabs(log(score) ~ F(age) : sex, data = myDF)

# No transformFunc or formula arithmetic expressions.
rxCrossTabs(score ~ F(age) : sex, data = myDF)

# Transform a categorical variable to a continuous one and use it
# as a response variable in the formula for cross-tabulation.
# The transformation is equivalent to doing the following, which
# is reflected in the cross-tabulation results.
#
#   > as.numeric(as.factor(c(20,20,12,15))) - 1
#   [1] 2 2 0 1
#
# Note that the effect of N() is to return the factor codes
myDF <- data.frame(sex = c("Male", "Male", "Female", "Male"),
                    age = factor(c(20, 20, 12, 15)), score = factor(1.1:4.1))
rxCrossTabs(N(age) ~ sex : score, data = myDF)

# To transform a categorical variable (like age) that has numeric levels
# (as opposed to codes), use the following construction:
myDF <- data.frame(sex = c("Male", "Male", "Female", "Male"),
                    age = factor(c(20, 20, 12, 15)), score = factor(1.1:4.1))
rxCrossTabs(as.numeric(levels(age))[age] ~ sex : score, data = myDF)

# this should break because 'age' is a categorical variable
## Not run:

try(rxCrossTabs(age ~ sex + score, data = myDF))
## End(Not run)

# frequency weighting
fwts <- 1:4
sex <- c("Male", "Male", "Female", "Male")
age <- c(20, 20, 12, 15)
score <- 1.1:4.1

myDF1 <- data.frame(sex = sex, age = age, score = factor(score), fwts = fwts)
myDF2 <- data.frame(sex = rep(sex, fwts), age = rep(age, fwts),
                     score = factor(rep(score, fwts)))

mySums1 <- rxCrossTabs(age ~ sex : score, data = myDF1,
                       fweights = "fwts")$sums$age[c("Male", "Female"),]
mySums2 <- rxCrossTabs(age ~ sex : score,
                       data = myDF2)$sums$age[c("Male", "Female"),]
all.equal(mySums1, mySums2)

# Compare xtabs and rxCrossTabs(..., returnXtabs = TRUE)
# results for 3-way interaction, one dependent variable
set.seed(100)
divs <- letters[1:5]
glads <- c("spartacus", "crixus")
romeDF <- data.frame( division = rep(divs, 5L),
                      score = runif(25, min = 0, max = 10),
                      rank = runif(25, min = 1, max = 100),
                      gladiator = c(rep(glads[1L], 12L), rep(glads[2L], 13L)),
                      arena = sample(c("colosseum", "ludus", "market"), 25L, replace = TRUE))

z1 <- rxCrossTabs(score ~ division : gladiator : arena, data = romeDF, returnXtabs = TRUE)
z2 <- xtabs(score ~ division + gladiator + arena, romeDF)
all.equal(z1, z2, check.attributes = FALSE) # all the same except "call" attribute

# Compare xtabs and rxCrossTabs(..., returnXtabs = TRUE)
# results for 3-way interaction, multiple dependent variable
z1 <- rxCrossTabs(cbind(score, rank) ~ division : gladiator : arena, data = romeDF, returnXtabs = TRUE,
                  means = TRUE)
z2 <- xtabs(cbind(score, rank) ~ division + gladiator + arena, romeDF)

```

```
all.equal(z1, z2, check.attributes = FALSE) # all the same except "call" attribute

# Compare xtabs and rxCrossTabs(..., returnXtabs = TRUE)
# results for 3-way interaction, no dependent variable
z1 <- rxCrossTabs( ~ division : gladiator : arena, data = romeDF, returnXtabs = TRUE, means = TRUE)
z2 <- xtabs(~ division + gladiator + arena, romeDF)
all.equal(z1, z2, check.attributes = FALSE) # all the same except "call" attribute

# removeZeroCounts
admissions <- as.data.frame(UCBAdmissions)
admissions[admissions$Dept == "F", "Freq"] <- 0

# removeZeroCounts does not make a difference for the zero values observed from input data
crossTab1 <- rxCrossTabs(Freq ~ Dept : Gender, data = admissions, removeZeroCounts = TRUE)
crossTab2 <- rxCrossTabs(Freq ~ Dept : Gender, data = admissions)
all.equal(as.data.frame(crossTab1$sums$Freq), as.data.frame(crossTab2$sums$Freq))

# removeZeroCounts removes the missing values that are not observed from input data
admissions_NoZero <- admissions[admissions$Dept != "F",]
crossTab1 <- rxCrossTabs(Freq ~ Dept : Gender, data = admissions, removeZeroCounts = TRUE, rowSelection =
(Freq != 0))
crossTab2 <- rxCrossTabs(Freq ~ Dept : Gender, data = admissions_NoZero, removeZeroCounts = TRUE)
all.equal(as.data.frame(crossTab1$sums$Freq), as.data.frame(crossTab2$sums$Freq))
```

rxCube: Cross Tabulation

7/12/2022 • 7 minutes to read • [Edit Online](#)

Description

Use `rxCube` to create efficiently represented contingency tables from cross-classifying factors using a formula interface. It performs equivalent calculations to the `rxCrossTabs` function, but returns its results in a different way.

Usage

```
rxCube(formula, data, outFile = NULL, pweights = NULL, fweights = NULL,
       means = TRUE, cube = TRUE, rowSelection = NULL,
       transforms = NULL, transformObjects = NULL,
       transformFunc = NULL, transformVars = NULL,
       transformPackages = NULL, transformEnvir = NULL,
       overwrite = FALSE,
       useSparseCube = rxGetOption("useSparseCube"),
       removeZeroCounts = useSparseCube, returnDataFrame = FALSE,
       blocksPerRead = rxGetOption("blocksPerRead"),
       rowsPerBlock = 100000,
       reportProgress = rxGetOption("reportProgress"), verbose = 0,
       computeContext = rxGetOption("computeContext"), ...)

## S3 method for class `rxCube':
print (x, header = TRUE, ...)

## S3 method for class `rxCube':
summary (object, header = TRUE, ...)

## S3 method for class `rxCube':
as.data.frame (x, row.names = NULL, optional = FALSE, ...)

## S3 method for class `rxCube':
subset (x, ...)

## S3 method for class `rxCube':
[ (x, ...)
```

Arguments

formula

formula as described in `rxFormula` with the cross-classifying variables (separated by `:`) on the right hand side. Independent variables must be factors. If present, the dependent variable must be numeric.

data

either a data source object, a character string specifying a .xdf file, or a data frame object containing the cross-classifying variables.

outFile

`NULL`, a character string specifying a .xdf file, or an `RxDfData` object. If not `NULL`, the cube results will be written out to an .xdf file and an `RxDfData` object will be returned. `outFile` is not supported when using distributed compute contexts.

pweights

character string specifying the variable to use as probability weights for the observations.

fweights

character string specifying the variable to use as frequency weights for the observations.

means

logical flag. If `TRUE` (default), the mean values of the dependent variable are returned. Otherwise, the variable summations are returned.

cube

logical flag. If `TRUE`, the C++ cube functionality is called.

rowSelection

name of a logical variable in the data set (in quotes) or a logical expression using variables in the data set to specify row selection. For example, `rowSelection = "old"` will use only observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` will use only observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

transforms

an expression of the form `list(name = expression, ...)` representing the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

transformObjects

a named list containing objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

transformFunc

variable transformation function. The variables used in the transformation function must be specified in `transformVars` if they are not variables used in the model. See [rxTransform](#) for details.

transformVars

character vector of input data set variables needed for the transformation function. See [rxTransform](#) for details.

transformPackages

character vector defining additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, e.g., those explicitly defined in RevoScaleR functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.

transformEnvir

user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

overwrite

logical value. If `TRUE`, an existing `outFile` will be overwritten. `overwrite` is ignored `outFile` is `NULL`.

`useSparseCube`

logical value. If `TRUE`, sparse cube is used.

`removeZeroCounts`

logical flag. If `TRUE`, rows with no observations will be removed from the output. By default, it has the same value as `useSparseCube`. For large cube computation, this should be set to `TRUE`, otherwise R may run out of memory even if the internal C++ computation succeeds.

`returnDataFrame`

logical flag. If `TRUE`, a data frame is returned, otherwise a list is returned. Ignored if `outFile` is specified and is not `NULL`. See the Details section for more information.

`blocksPerRead`

number of blocks to read for each chunk of data read from the data source.

`rowsPerBlock`

maximum number of rows to write to each block in the `outFile` (if it is not `NULL`).

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`verbose`

integer value. If `0`, no additional output is printed. If `1`, additional summary information is printed.

`computeContext`

a valid RxComputeContext. The `RxSpark` and `RxHadoopMR` compute contexts distribute the computation among the nodes specified by the compute context; for other compute contexts, the computation is distributed if possible on the local computer.

`...`

additional arguments to be passed directly to the Revolution Compute Engine.

`x, object`

output objects from rxCube function.

logical value. If `TRUE`, header information is printed.

`row.names`

the `row.names` argument passed unaltered to the underlying `as.data.frame.list` function.

`optional`

the `optional` argument passed unaltered to the underlying `as.data.frame.list` function.

Details

The output of the `rxCube` function is essentially the same as that produced by `rxCrossTabs` except that it is presented in a different format. While the `rxCrossTabs` function produces lists of contingency tables (where

each table is a matrix), the `rxCube` function outputs a single list (or data frame, or .xdf file) containing one column for each variable specified in the formula, plus a `"Counts"` column. The columns corresponding to *independent* variables contain the factor levels of that variable, replicated as necessary. If a dependent variable is specified in the formula, an output column of the same name is produced and contains the mean values of the categories defined by the interaction of the independent/categorical variables. The `"Counts"` column contains the counts of the interactions used to form the corresponding means.

Value

- `outfile` is not NULL: an `RxxdfData` object representing the output .xdf file. In this case, the value for `returnDataFrame` is ignored.
- `returnDataFrame = FALSE`: an object of class `rxCube` that is also of class `"list"`. This is the default.
- `returnDataFrame = TRUE`: an object of class `"data.frame"`.

In all cases, the names of the output columns are those of the variables defined in the formula plus a `"Counts"` column. See the Details section for more information regarding the content of these columns.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

`xtabs`, `rxCrossTabs`, `as.xtabs`, `rxTransform`.

Examples

```
# Basic data.frame source example
admissions <- as.data.frame(UCBAdmissions)
admissCube <- rxCube(Freq ~ Gender : Admit, data = admissions)
admissCube

# XDF example: small subset of census data
censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers.xdf")
censusCube <- rxCube(wkswork1 ~ sex : F(age), data = censusWorkers,
  pweights = "perwt", blocksPerRead = 3, returnDataFrame = TRUE)
censusCube
censusCube$age <- as.integer(as.character(censusCube$F_age))
rxLinePlot(wkswork1 ~ age, groups=sex, data = censusCube)

# perform a census cube, limiting the analysis to ages
# on the interval [20, 65]. Verify the age range from the output.
censusCubeAge.20.65 <- rxCube(wkswork1 ~ sex : F(age), data = censusWorkers,
  rowSelection = age >= 20 & age <= 65)
ageRange <- range(as.numeric(as.character(censusCubeAge.20.65$F_age)))
(ageRange[1] >= 20 & ageRange[2] <=65)

# Create a local data.frame and define a transformation
# function to be applied to the data prior to processing.
myDF <- data.frame(sex = c("Male", "Male", "Female", "Male"),
  age = factor(c(20,20,12,15)), score = 1.1:4.1, sport=c(1:3,2))

# Use the 'transforms' argument to dynamically transform the
# variables of the data source. Here, we form a named list of
# transformation expressions. To avoid evaluation when assigning
# to a local variable, we wrap the transformation list with expression().
transforms <- expression(list(
  scoreDoubled = score * 2,
```

```

sport = factor(sport, labels=c("tennis", "golf", "football")))

rxCube(scoreDoubled ~ sport : sex, data = myDF, transforms = transforms,
       removeZeroCounts=TRUE)

# Arithmetic formula expression only (no transformFunc specification).
rxCube(log(score) ~ age : sex, data = myDF)

# No transformFunc or arithmetic expressions in formula.
rxCube(score ~ age : sex, data = myDF)

# Transform a categorical variable to a continuous one and use it
# as a response variable in the formula for cross-tabulation.
# The transformation is equivalent to doing the following, which
# is reflected in the cross-tabulation results.
#
#   > as.numeric(as.factor(c(20,20,12,15))) - 1
#   [1] 2 2 0 1
myDF <- data.frame(sex = c("Male", "Male", "Female", "Male"),
                    age = factor(c(20, 20, 12, 15)), score = 1.1:4.1)
rxCube(N(age) ~ sex : F(score), data = myDF)

# this should break because 'age' is a categorical variable
## Not run:

try(rxCube(age ~ sex : score, data = myDF))
## End(Not run)

# frequency weighting
fwts <- 1:4
sex <- c("Male", "Male", "Female", "Male")
age <- c(20, 20, 12, 15)
score <- 1.1:4.1

myDF1 <- data.frame(sex = sex, age = age, score = factor(score), fwts = fwts)
myDF2 <- data.frame(sex = rep(sex, fwts), age = rep(age, fwts),
                     score = factor(rep(score, fwts)))

myCube1 <- rxCube(age ~ sex : score, data = myDF1, fweights = "fwts")
myCube2 <- rxCube(age ~ sex : score, data = myDF2)
all.equal(myCube1, myCube2, check.attributes = FALSE)

```

RxDataSource-class: Class RxDataSource

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Base class for all Microsoft R Services Compute Engine data sources.

Objects from the Class

A virtual class: No objects may be created from it.

Generator

The generator for classes that extend RxDataSource is [rxNewDataSource](#).

Methods

The following methods are defined for classes that extend RxDataSource:

```
names  
signature(x = "RxDataSource") : ...  
[rxOpen](rxOpen-methods.md)  
signature(src = "RxDataSource") : ...  
[rxClose](rxOpen-methods.md)  
signature(src = "RxDataSource") : ...  
[rxReadNext](rxOpen-methods.md)  
signature(src = "RxDataSource") : ...  
[rxWriteNext](rxOpen-methods.md)  
signature(from = "data.frame", to = "RxDataSource", verbose = 0) : ...
```

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxXdfData-class](#), [RxXdfData](#), [RxTextData](#), [RxSasData](#), [RxSpssData](#), [RxOdbcData](#), [RxTeradata](#), [rxNewDataSource](#), [rxOpen](#), [rxReadNext](#), [rxWriteNext](#).

Examples

```
fileName <- file.path(rxGetOption("sampleDataDir"), "claims.xdf")
ds <- rxNewDataSource("RxXdfData", fileName)
is(ds, "RxXdfData")
# [1] TRUE
is(ds, "RxDataSource")
# [1] TRUE
```

RxDataSource: RevoScaleR Data Source: Class Generator

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

A generator for RxDataSource S4 classes.

Usage

```
RxDataSource( class = NULL, dataSource = NULL, ...)
```

Arguments

class

an optional character string specifying class name of the data source to be created, such as [RxTextData](#), [RxSpssData](#), [RxSasData](#), [RxOdbcData](#), or [RxTeradata](#). If the class of the input dataSource differs from `class`, contents of overlapping slots will be copied from the input data source to the newly constructed data source.

dataSource

an optional data source object to be cloned.

...

any other arguments to be applied to the newly constructed data source.

Details

If `class` is specified, the returned object will be of that class. If not, the returned object will have the class of `dataSource`. If both `class` and `dataSource` are specified, the contents of matching slots will be copied from the input `dataSource` to the output object. Additional arguments will then be applied.

Value

A type of RxDataSource data source object.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxDataSource-class](#), [RxTextData](#), [RxSqlServerData](#), [RxSpssData](#), [RxSasData](#), [RxOdbcData](#), [RxTeradata](#), [RxXdfData](#).

Examples

```
claimsSpssName <- file.path(rxGetOption("sampleDataDir"), "claims.sav")
claimsTextName <- file.path(rxGetOption("sampleDataDir"), "claims.txt")
claimsColInfo <- list(
  age = list(type = "factor",
             levels = c("17-20", "21-24", "25-29", "30-34", "35-39", "40-49", "50-59", "60+")),
  car.age = list(type = "factor", levels = c("0-3", "4-7", "8-9", "10+")),
  type = list(type = "factor", levels = c("A", "B", "C", "D")))
)
textDS <- RxTextData(file = claimsTextName, colInfo = claimsColInfo)

# Create an SPSS data source from the text data source
# The 'colInfo' will be carried over to the SPSS data source
# and the file name will be replaced
spssDS <- RxDataSource(class = "RxSpssData", dataSource = textDS, file = claimsSpssName)
```

rxDataStep: Data Step for RevoScaleR data sources

7/12/2022 • 13 minutes to read • [Edit Online](#)

Description

Transform data from an input data set to an output data set. The rxDataStep function is multi-threaded.

Usage

```
rxDataStep(inData = NULL, outFile = NULL, varsToKeep = NULL, varsToDrop = NULL,
           rowSelection = NULL, transforms = NULL, transformObjects = NULL,
           transformFunc = NULL, transformVars = NULL,
           transformPackages = NULL, transformEnvir = NULL,
           append = "none", overwrite = FALSE, rowVarName = NULL,
           removeMissingsOnRead = FALSE, removeMissings = FALSE,
           computeLowHigh = TRUE, maxRowsByCols = 3000000,
           rowsPerRead = -1, startRow = 1, numRows = -1,
           returnTransformObjects = FALSE,
           blocksPerRead = rxGetOption("blocksPerRead"),
           reportProgress = rxGetOption("reportProgress"),
           xdfCompressionLevel = rxGetOption("xdfCompressionLevel"), ...)
```

Arguments

inData

A data source object of these types: [RxTextData](#), [RxXdfData](#), [RxSasData](#), [RxSpssData](#), [RxOdbcData](#), [RxSqlServerData](#), [RxTeradata](#). If not using a distributed compute context such as RxHadoopMR, a data frame, a character string specifying the input .xdf file, or `NULL` can also be used. If `NULL`, a data set will be created automatically with a single variable, `.rxRowNums`, containing row numbers. It will have a total of `numRows` rows with `rowsPerRead` rows in each block.

outFile

A character string specifying the output .xdf file or any of these data sources: [RxTextData](#), [RxXdfData](#), [RxSasData](#), [RxSpssData](#), [RxOdbcData](#), [RxSqlServerData](#), [RxTeradata](#), [RxHiveData](#), [RxParquetData](#), [RxOrcData](#). If `NULL`, a data frame will be returned from `rxDataStep` unless `returnTransformObjects` is set to `TRUE`. Setting `outFile` to `NULL` and `returnTransformObjects=TRUE` allows chunkwise computations on the data without modifying the existing data or creating a new data set. `outFile` can also be a delimited [RxTextData](#) data source if using a native file system and not appending.

varsToKeep

A character vector of variable names to include when reading from the input data file. If `NULL`, argument is ignored. Cannot be used with `varsToDrop` or when `outFile` is the same as the input data file. Variables used in transformations or row selection will be retained even if not specified in `varsToKeep`. If `newName` is used in `colInfo` in a non-xdf data source, the `newName` should be used in `varsToKeep`. Not supported for [RxTeradata](#), [RxOdbcData](#), or [RxSqlServerData](#) data sources.

varsToDrop

A character vector of variable names to exclude when reading from the input data file. If `NULL`, argument is ignored. Cannot be used with `varsToKeep` or when `outFile` is the same as the input data file. Variables used in transformations or row selection will be retained even if specified in `varsToDelete`. If `newName` is used in `colInfo` in a non-xdf data source, the `newName` should be used in `varsToDelete`. Not supported for [RxTeradata](#), [RxOdbcData](#), or [RxSqlServerData](#) data sources.

`rowSelection`

The name of a logical variable in the data set or a logical expression using variables in the data set to specify row selection. For example, `rowSelection = old` will use only observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` will use only observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

`transforms`

An expression of the form `list(name = expression, ...)` representing the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function. When using function call in the expression, `transformObject` should be used to pass the function name to remote context. In addition, calling and enclosing environment of the function are very important if the function uses undefined variable.

`transformObjects`

A named list containing objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

`transformFunc`

A variable transformation function. The recommended way to do variable transformation. See [rxTransform](#) for details.

`transformVars`

A character vector of input data set variables needed for the transformation function. See [rxTransform](#) for details.

`transformPackages`

A character vector defining additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, e.g., those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.

`transformEnvir`

A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

`append`

Either `"none"` to create a new files, `"rows"` to append rows to an existing file, or `"cols"` to append columns to an existing file. If `outFile` exists and `append` is `"none"`, the `overwrite` argument must be set to `TRUE`. Ignored for data frames. You cannot append to [RxTextData](#) or append columns to [RxTeradata](#) data sources, and appending is not supported for composite .xdf files or when using the RxHadoopMR compute context.

`overwrite`

A logical value. If `TRUE`, an existing `outFile` will be overwritten, or if appending columns existing columns with the same name will be overwritten. `overwrite` is ignored if appending rows. Ignored for data frames.

`rowVarName`

A character string or `NULL`. If `inData` is a `data.frame`: If `NULL`, the data frame's row names will be dropped. If a character string, an additional variable of that name will be added to the data set containing the data frame's row names. If a `data.frame` is being returned, a variable with the name `rowVarName` will be removed as a column from the data frame and will be used as the row names.

`removeMissingsOnRead`

A logical value. If `TRUE`, rows with missing values will be removed on read.

`removeMissings`

A logical value. If `TRUE`, rows with missing values will not be included in the output data.

`computeLowHigh`

A logical value. If `FALSE`, low and high values will not automatically be computed. This should only be set to `FALSE` in special circumstances, such as when `append` is being used repeatedly. Ignored for data frames.

`maxRowsByCols`

The maximum size of a data frame that will be returned if `outFile` is set to `NULL` and `inData` is an .xdf file, measured by the number of rows times the number of columns. If the number of rows times the number of columns being created from the .xdf file exceeds this, a warning will be reported and the number of rows in the returned data frame will be truncated. If `maxRowsByCols` is set to be too large, you may experience problems from loading a huge data frame into memory.

`rowsPerRead`

The number of rows to read for each chunk of data read from the input data source. Use this argument for finer control of the number of rows per block in the output data source. If greater than 0, `blocksPerRead` is ignored. Cannot be used if `inFile` is the same as `outFile`. The default value of `-1` specifies that data should be read by blocks according to the `blocksPerRead` argument.

`startRow`

The starting row to read from the input data source. Cannot be used if `inFile` is the same as `outFile`.

`numRows`

The number of rows to read from the input data source. If `rowSelection` or `removeMissings` are used, the output data set may have fewer rows than specified by `numRows`. Cannot be used if `inFile` is the same as `outFile`.

`returnTransformObjects`

A logical value. If `TRUE`, the list of `transformObjects` will be returned instead of a data frame or data source object. If the input `transformObjects` have been modified, by using `.rxSet` or `.rxModify` in the `transformFunc`, the updated values will be returned. Any data returned from the `transformFunc` is ignored. If no `transformObjects` are used, `NULL` is returned. This argument allows for user-defined computations within a `transformFunc` without creating new data. `returnTransformObjects` is not supported in distributed compute contexts such as RxHadoopMR.

`blocksPerRead`

The number of blocks to read for each chunk of data read from the data source. Ignored for data frames or if `rowsPerRead` is positive.

reportProgress

An integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

xdfCompressionLevel

An integer in the range of -1 to 9. The higher the value, the greater the amount of compression - resulting in smaller files but a longer time to create them. If `xdfCompressionLevel` is set to 0, there will be no compression and files will be compatible with the 6.0 release of Revolution R Enterprise. If set to -1, a default level of compression will be used.

...

Additional arguments to be passed to the input data source. If `stringsAsFactors` is specified and the input data source is `RxXdfData`, strings will be converted to factors when returning a data frame to R.

Value

For `rxDataStep`, if `returnTransformObjects` is `FALSE`)and an `outFile` is specified, an `RxXdfData` data source is returned invisibly. If no `outFile` is specified, a data frame is returned invisibly. Either can be used in subsequent RevoScaleR analysis.

If `returnTransformObjects` is `TRUE`, the `transformObjects` list as modified by the transformation function is returned invisibly. When working with an `RxInSqlServer` compute context, both the input and output data sources must be `RxSqlServerData`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxImport](#), [RxXdfData](#), [RxTextData](#), [RxSqlServerData](#), [RxTeradata](#), [rxGetInfo](#), [rxGetVarInfo](#), [rxTransform](#)

Examples

```
# Create a data frame
set.seed(100)
myData <- data.frame(x = 1:100, y = rep(c("a", "b", "c", "d"), 25),
                      z = rnorm(100), w = runif(100))

# Get a subset rows and columns from the data frame
myDataSubset <- rxDataStep(inData = myData,
                           varsToKeep = c("x", "w", "z"), rowSelection = z > 0)
rxGetInfo(myDataSubset, getVarInfo = TRUE)

# Create a simple .xdf file from the data frame (for use below)
inputFile <- file.path(tempdir(), "testInput.xdf")
rxDataStep(inData = myData, outFile = inputFile, overwrite = TRUE)

# Redo, creating a multi-block .xdf file from the data frame
rxDataStep(inData = myData, outFile = inputFile, rowsPerRead = 50,
           overwrite = TRUE)
rxGetInfo(inputFile)
```

```

# Subsetting input file
# Subset rows and columns, creating a new .xdf file
outputFile <- file.path(tempdir(), "testOutput.xdf")
rxDataStep(inData = inputFile, outFile = outputFile,
           varsToKeep = c("x", "w", "z"), rowSelection = z > 0,
           overwrite = TRUE)
rxGetInfo(outputFile)

# Use transforms list with data frame input and output data
# Add new columns
myNewData <- rxDataStep(inData = myData,
                        transforms = list(a = w > 0, b = 100 * z))
names(myNewData)

# Use transformFunc to add new columns
myXformFunc <- function(dataList) {
  dataList$b <- 100 * dataList$z
  return (dataList)
}
myNewData <- rxDataStep(inData = myData,
                        transformFunc = myXformFunc)
names(myNewData)

# Use transforms to remove columns
rxDataStep(inData = inputFile, outFile = outputFile,
           transforms = list(w = NULL), overwrite = TRUE)
rxGetInfo(outputFile)

# use transformFunc to remove columns
xform <- function(dataList) {
  dataList$w <- NULL
  return (dataList)
}
rxDataStep(inData = inputFile, outFile = outputFile,
           transformFunc = xform, overwrite = TRUE)
rxGetInfo(outputFile)

# use transform to change data type
rxDataStep(inData = inputFile, outFile = outputFile, transformVars = c("x", "y"),
           transforms = list(x = as.numeric(x), y = as.numeric(y)), overwrite = TRUE)
rxGetVarInfo(outputFile)

# use transformFunc to change data type
myXform <- function(dataList) {
  dataList <- lapply(dataList, as.numeric)
  return (dataList)
}
rxDataStep(inData = inputFile, outFile = outputFile,
           transformFunc = myXform, overwrite = TRUE)
rxGetVarInfo(outputFile)

# use transform to create new data
rxDataStep(inData = inputFile, outFile = outputFile,
           transforms = list(maxZ = max(z), minW = min(w), z = NULL, w = NULL),
           varsToDelete = c("x", "y"), overwrite = TRUE)
rxGetVarInfo(outputFile)

# use transformFunc to create new data
xform <- function(dataList){
  outList <- list()
  outList$maxZ <- max(dataList$z)
  outList$minW <- min(dataList$w)
  return (outList)
}
rxDataStep(inData = inputFile, outFile = outputFile,
           transformFunc = xform, overwrite = TRUE)
rxGetVarInfo(outputFile)

```

```

# add new column by calling function in expression for "transform"
# "transform" validate the expression at server, so function name should be passed to
# remote context. R looking up undefined variable in calling environment, dynamic scoping
# should be used to find "const" used in function "myTransform"
const <- 10
myTransform <- function(x){
  const <- get("const", parent.frame())
  x * const
}
rxDataStep(inData = inputFile, outFile = outputFile,
           transforms = list(x10 = myTransform(x)), transformObjects = list(myTransform = myTransform,
const = const),
           overwrite = TRUE)
rxGetVarInfo(outputFile)

# using "transform" and "transformEnvir" to add new columns
env <- new.env()
env$constant <- 10
env$myTransform <- function(x){
  x * constant
}
environment(env$myTransform) <- env
data <- rxDataStep(inData=inputFile, outFile = outputFile,
                   transforms = list(b = myTransform(x)), transformEnvir = env, overwrite = TRUE)
rxGetVarInfo(outputFile)

# Specify which variables to keep in a new data file
varsToKeep <- c("x", "w", "z")
rxDataStep(inData = inputFile, outFile = outputFile, varsToKeep = varsToKeep,
           transforms = list(a = w > 0, b = 100 * z), overwrite = TRUE)
rxGetInfo(outputFile)

# Alternatively specify which variables to drop
varsToDrop <- "y"
rxDataStep(inData = inputFile, outFile = outputFile, varsToDrop = varsToDrop,
           transforms = list(a = w > 0, b = 100 * z), overwrite = TRUE)
rxGetInfo(outputFile)

# Read a specific number of rows of data into a data frame
myData <- rxDataStep(inData = inputFile, startRow = 5, numRows = 10)
rxGetInfo(myData)

# Create a selection variable to take a roughly 25% random sample
# of each block of data read (in this case 1).
rxDataStep(inData = inputFile, outFile = outputFile, rowSelection = selVar,
           transforms = list(
             selVar = as.logical(rbinom(.rxNumRows, 1, .25))
           ), overwrite = TRUE)
rxGetInfo(outputFile)

# Create a new variable containing row numbers for a multi-block data file
# Note that .rxStartRow and .rxNumRows are not supported in
# a distributed compute context such as RxHadoopMR
myDataSource <- rxDataStep(inData = inputFile, outFile = outputFile,
                           transforms = list(
                             rowNum = .rxStartRow : (.rxStartRow + .rxNumRows - 1)),
                           overwrite = TRUE )

# Use the returned data source object in another call to rxDataStep
myMiddleData <- rxDataStep(inData = myDataSource, startRow = 55,
                           numRows = 5)
myMiddleData

# Create a new factor variable from a continuous variable using the cut function
rxDataStep(inData = inputFile, outFile = outputFile,
           transforms = list(
             wFactor = cut(w, breaks = c(0, .33, .66, 1), labels = c("Low", "Med", "High"))
           ), overwrite=TRUE)
rxGetInfo( outputFile, numRows = 10 )

```

```

# Create a new data set with simulated data. A row number variable will
# be automatically created. It will have 20 rows in each block,
# with a total of 100 rows
#(This functionality not supported in distributed compute contexts.)
outFile <- tempfile(pattern = ".rxTest1", fileext = ".xdf")
newDS <- rxDataStep( outFile = outFile, numRows = 100, rowsPerRead = 20,
  transforms = list(
    x = (.rxRowNums - 50)/25,
    pnormx = pnorm(x),
    dnormx = dnorm(x)))
# Compute summary statistics on the new data file
rxSummary(~., newDS)
file.remove(outFile)

# Use the data step to chunk through the data and compute
# the sum of a squared deviation
inFile <- file.path(rxGetOption("unitTestDataDir"), "test100r5b.xdf")
myFun <- function( dataList )
{
  chunkSumDev <- sum((dataList$y - toMyCenter)^2, na.rm = TRUE)
  toTotalSumDev <- toTotalSumDev + chunkSumDev
  return(NULL)
}
myCenter <- 40
newTransformVals <- rxDataStep(inData = inFile,
  transformFunc = myFun,
  transformObjects = list(
    toMyCenter = myCenter,
    toTotalSumDev = 0),
  transformVars = "y", returnTransformObjects = TRUE)

newTransformVals[["toTotalSumDev"]]

## Not run:

# These examples need to be modified to substitute appropriate paths.
# Convert an xdf file to csv on HDFS
myData <- data.frame(textVar = c("a", "b", "c", "d"), intVar = as.integer(c(1:2, NA, 4)))
# write data frame to an xdf file in local file system
# test1.xdf will be written out to local file system in the current working directory
xdffile <- "test1.xdf"
xdfds <- RxXdfData(xdffile)
rxDataStep(inData = myData, outFile = xdfds, overwrite=TRUE)
# use RxTextData to write a csv file to HDFS
# /user/RevoShare/myuser is HDFS path that must exist for testCsv.csv to be written successfully.
csvfile <- "/user/RevoShare/myuser/testCsv.csv"
# We are writing a csv with no header (column names), no quotes and empty string for missing values (NA)
hdfsFS <- RxHdfsFileSystem()
myTextDS <- RxTextData(csvfile, missingValueString = "NA", firstRowIsColNames = FALSE, quoteMark = "", fileSystem=hdfsFS)
# This will write out a csv file to HDFS
rxDataStep(inData = xdfds, outFile = myTextDS, overwrite = TRUE)

# Convert a composite xdf file to csv writing to HDFS using RxTextData.
# Create a test xdf file.
myData <- data.frame(textVar = c("a", "b", "c", "d"), intVar = as.integer(c(1:2, NA, 4)))
hdfsFS <- RxHdfsFileSystem()
# composite xdf file path in HDFS. Path "/user/RevoShare/myuser/test1" must exist.
xdffile <- "/user/RevoShare/myuser/test1"
xdfds <- RxXdfData(xdffile, fileSystem = hdfsFS)
rxDataStep(inData = myData, outFile = xdfds, overwrite=TRUE)
# use RxTextData to write a csv file to HDFS
# /user/RevoShare/myuser is HDFS path that must exist for testCsv.csv to be written successfully.
csvfile <- "/user/RevoShare/myuser/testCsv.csv"
# We are writing a csv with no header, no quotes and empty string for missing values (NA)
myTextDS <- RxTextData(csvfile, missingValueString = "", firstRowIsColNames = FALSE, quoteMark = "", fileSystem = hdfsFS)
# This will write out a csv file to HDFS

```

```
rxDataStep(inData = xdfDS, outFile = myTextDS, overwrite = TRUE)
## End(Not run)
```

rxDForest: Parallel External Memory Algorithm for Classification and Regression Decision Forests

7/12/2022 • 12 minutes to read • [Edit Online](#)

Description

Fit classification and regression decision forests on an .xdf file or data frame for small or large data using parallel external memory algorithm.

Usage

```
rxDForest(formula, data,
          outFile = NULL, writeModelVars = FALSE, overwrite = FALSE,
          pweights = NULL, fweights = NULL, method = NULL, parms = NULL, cost = NULL,
          minSplit = NULL, minBucket = NULL, maxDepth = 10, cp = 0,
          maxCompete = 0, maxSurrogate = 0, useSurrogate = 2, surrogateStyle = 0,
          nTree = 10, mTry = NULL, replace = TRUE, cutoff = NULL,
          strata = NULL, sampRate = NULL, importance = FALSE, seed = sample.int(.Machine$integer.max, 1),
          computeOobError = 1,
          maxNumbins = NULL, maxUnorderedLevels = 32, removeMissings = FALSE,
          useSparseCube = rxGetOption("useSparseCube"), findSplitsInParallel = TRUE,
          scheduleOnce = FALSE,
          rowSelection = NULL, transforms = NULL, transformObjects = NULL, transformFunc = NULL,
          transformVars = NULL, transformPackages = NULL, transformEnvir = NULL,
          blocksPerRead = rxGetOption("blocksPerRead"), reportProgress = rxGetOption("reportProgress"),
          verbose = 0, computeContext = rxGetOption("computeContext"),
          xdfCompressionLevel = rxGetOption("xdfCompressionLevel"),
          ... )

## S3 method for class `rxDForest':
plot(x, type = "l", main = deparse(substitute(x)),
      ... )
```

Arguments

formula

formula as described in [rxFormula](#). Currently, formula functions are not supported.

data

either a data source object, a character string specifying a .xdf file, or a data frame object.

outFile

either an RxXdfData data source object or a character string specifying the .xdf file for storing the resulting OOB predictions. If `NULL` or the input data is a data frame, then no OOB predictions are stored to disk. If `rowSelection` is specified and not `NULL`, then `outFile` cannot be the same as the `data` since the resulting set of OOB predictions will generally not have the same number of rows as the original data source.

writeModelVars

logical value. If `TRUE`, and the output file is different from the input file, variables in the model will be written to the output file in addition to the OOB predictions. If variables from the input data set are transformed in the model, the transformed variables will also be written out.

overwrite

logical value. If `TRUE`, an existing `outFile` with an existing column named `outColName` will be overwritten.

pweights

character string specifying the variable to use as probability weights for the observations.

fweights

character string specifying the variable to use as frequency weights for the observations.

method

character string specifying the splitting method. Currently, only `"class"` or `"anova"` are supported. The default is `"class"` if the response is a factor, otherwise `"anova"`.

parms

optional list with components specifying additional parameters for the `"class"` splitting method, as follows:

- `prior` - a vector of prior probabilities. The priors must be positive and sum to 1. The default priors are proportional to the data counts.
- `loss` - a loss matrix, which must have zeros on the diagonal and positive off-diagonal elements. By default, the off-diagonal elements are set to 1.
- `split` - the splitting index, either `gini` (the default) or `information`.
If `parms` is specified, any of the components can be specified or omitted. The defaults will be used for missing components.

cost

a vector of non-negative costs, containing one element for each variable in the model. Defaults to one for all variables. When deciding which split to choose, the improvement on splitting on a variable is divided by its cost.

minSplit

the minimum number of observations that must exist in a node before a split is attempted. By default, this is `sqrt(num of obs)`. For non-XDF data sources, as `(num of obs)` is unknown in advance, it is wisest to specify this argument directly.

minBucket

the minimum number of observations in a terminal node (or leaf). By default, this is `minsplit /3`.

maxDepth

the maximum depth of any tree node. The computations take much longer at greater depth, so lowering `maxDepth` can greatly speed up computation time.

cp

numeric scalar specifying the complexity parameter. Any split that does not decrease overall lack-of-fit by at least `cp` is not attempted.

maxCompete

the maximum number of competitor splits retained in the output. These are useful model diagnostics, as they allow you to compare splits in the output with the alternatives.

maxSurrogate

the maximum number of surrogate splits retained in the output. See the Details for a description of how surrogate splits are used in the model fitting. Setting this to 0 can greatly improve the performance of the algorithm; in some cases almost half the computation time is spent in computing surrogate splits.

useSurrogate

an integer specifying how surrogates are to be used in the splitting process:

- `0` - display-only; observations with a missing value for the primary split variable are not sent further down the tree.
- `1` - use surrogates,in order, to split observations missing the primary split variable. If all surrogates are missing, the observation is not split.
- `2` - use surrogates, in order, to split observations missing the primary split variable. If all surrogates are missing or `maxSurrogate=0`, send the observation in the majority direction.
The `0` value corresponds to the behavior of the `tree` function, and `2` (the default) corresponds to the recommendations of Breiman et al.

surrogateStyle

an integer controlling selection of a best surrogate. The default, `0`, instructs the program to use the total number of correct classifications for a potential surrogate, while `1` instructs the program to use the percentage of correct classification over the non-missing values of the surrogate. Thus, `0` penalizes potential surrogates with a large number of missing values.

nTree

a positive integer specifying the number of trees to grow.

mTry

a positive integer specifying the number of variables to sample as split candidates at each tree node. The default values is `sqrt(num of vars)` for classification and `(num of vars)/3` for regression.

replace

a logical value specifying if the sampling of observations should be done with or without replacement.

cutoff

(Classification only) a vector of length equal to the number of classes specifying the dividing factors for the class votes. The default is `1/(num of classes)`.

strata

a character string specifying the (factor) variable to use for stratified sampling.

sampRate

a scalar or a vector of positive values specifying the percentage(s) of observations to sample for each tree:

- for unstratified sampling: a scalar of positive value specifying the percentage of observations to sample for each tree. The default is 1.0 for sampling with replacement (i.e., `replace=TRUE`) and 0.632 for sampling without replacement (i.e., `replace=FALSE`).
- for stratified sampling: a vector of positive values of length equal to the number of strata specifying the percentages of observations to sample from the strata for each tree.

importance

a logical value specifying if the importance of predictors should be assessed.

seed

an integer that will be used to initialize the random number generator. The default is random. For reproducibility, you can specify the random seed either using `set.seed` or by setting this `seed` argument as part of your call.

computeOobError

an integer specifying whether and how to compute the prediction error for out-of-bag samples:

- `<0` - never. This option may reduce the computation time.
- `=0` - only once for the entire forest.
- `>0` - once for each addition of a tree. This is the default.

`maxNumBins`

the maximum number of bins to use to cut numeric data. The default is `min(1001, max(101, sqrt(num of obs)))`. For non-XDF data sources, as `(num of obs)` is unknown in advance, it is wisest to specify this argument directly. If set to `0`, unit binning will be used instead of cutting. See the 'Details' section for more information.

`maxUnorderedLevels`

the maximum number of levels allowed for an unordered factor predictor for multiclass (>2) classification.

`removeMissings`

logical value. If `TRUE`, rows with missing values are removed and will not be included in the output data.

`useSparseCube`

logical value. If `TRUE`, sparse cube is used.

`findSplitsInParallel`

logical value. If `TRUE`, optimal splits for each node are determined using parallelization methods; this will typically speed up computation as the number of nodes on the same level is increased. Note that when it is `TRUE`, the number of nodes being processed in parallel is also printed to the console, interleaved with the number of rows read from the input data set.

`scheduleOnce`

EXPERIMENTAL. logical value. If `TRUE`, rxDForest will be run with `rxExec`, which submits only one job to the scheduler and thus can speed up computation on small data sets particularly in the RxHadoopMR compute context.

`rowSelection`

name of a logical variable in the data set (in quotes) or a logical expression using variables in the data set to specify row selection. For example, `rowSelection = "old"` will use only observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` will use only observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

`transforms`

an expression of the form `list(name = expression, ...)` representing the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

`transformObjects`

a named list containing objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

`transformFunc`

variable transformation function. The ".rxSetLowHigh" attribute must be set for transformed variables if they are to be used in `formula`. See `rxTransform` for details.

`transformVars`

character vector of input data set variables needed for the transformation function. See [rxTransform](#) for details.

`transformPackages`

character vector defining additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, e.g., those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.

`transformEnvir`

user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

`blocksPerRead`

number of blocks to read for each chunk of data read from the data source.

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`verbose`

integer value. If `0`, no verbose output is printed during calculations. Integer values from `1` to `2` provide increasing amounts of information are provided.

`computeContext`

a valid **RxComputeContext**. The `RxSpark` and `RxHadoopMR` compute contexts distribute the computation among the nodes specified by the compute context; for other compute contexts, the computation is distributed if possible on the local computer.

`xdfCompressionLevel`

integer in the range of -1 to 9 indicating the compression level for the output data if written to an `.xdf` file. The higher the value, the greater the amount of compression - resulting in smaller files but a longer time to create them. If `xdfCompressionLevel` is set to 0, there will be no compression and files will be compatible with the 6.0 release of Revolution R Enterprise. If set to -1, a default level of compression will be used.

`...`

additional arguments to be passed directly to the Microsoft R Services Compute Engine and to [rxExec](#) when `scheduleOnce` is set to `TRUE`.

`x`

an object of class `rxDForest`.

`type, main`

see `plot.default` for details.

Details

`rxDForest` is a parallel external memory decision forest algorithm targeted for very large data sets. It is modeled on the random forest ideas of Leo Breiman and Adele Cutler and the `randomForest` package of Andy Liaw and Matthew Weiner, using the tree-fitting algorithm introduced in `rxDTree`.

In a decision forest, a number of decision trees are fit to bootstrap samples of the original data. Observations omitted from a given bootstrap sample are termed "out-of-bag" observations. For a given observation, the decision forest prediction is determined by the result of sending the observation through all the trees for which it is out-of-bag. For classification, the prediction is the class to which a majority assigned the observation, and for regression, the prediction is the mean of the predictions.

For each tree, the out-of-bag observations are fed through the tree to estimate out-of-bag error estimates. The reported out-of-bag error estimates are cumulative (that is, the i th element represents the out-of-bag error estimate for all trees through the i th).

Value

an object of class `"rxDForest"`. It is a list with the following components, similar to those of class `"randomForest"`:

`ntree`

The number of trees.

`mtry`

The number of variables tried at each split.

`type`

One of `"class"` (for classification) or `"anova"` (for regression).

`forest`

a list containing the entire forest.

`oob.err`

a data frame containing the out-of-bag error estimate. For classification forests, this includes the OOB error estimate, which represents the proportion of times the predicted class is not equal to the true class, and the cumulative number of out-of-bag observations for the forest. For regression forests, this includes the OOB error estimate, which here represents the sum of squared residuals of the out-of-bag observations divided by the number of out-of-bag observations, the number of out-of-bag observations, the out-of-bag variance, and the "pseudo-R-Squared", which is 1 minus the quotient of the `oob.err` and `oob.var`.

`confusion`

(classification only) the confusion matrix of the prediction (based on out-of-bag data).

`cutoff`

(classification only) The cutoff vector.

`params`

The input parameters passed to the underlying code.

`formula`

The input formula.

`call`

The original call to `rxDForest`.

Note

Like `rxDTree`, `rxDForest` requires multiple passes over the data set and the maximum number of passes can be computed as follows:

- quantile computation: `1` pass for computing the quantiles for all continuous variables,
- recursive partition: `maxDepth + 1` passes per tree for building the tree on the entire dataset,
- out-of-bag prediction: `1` pass per tree for computing the out-of-bag error estimates.

`rxDForest` uses random streams and RNGs in parallel computation for sampling. Different threads on different nodes will be using different random streams so that different but equivalent results might be obtained for different number of threads.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

Breiman, L. (2001) Random Forests. *Machine Learning* **45**(1), 5--32.

Liaw, A., and Weiner, M. (2002). Classification and Regression by randomForest. *R News* **2**(3), 18--22.

Intel Math Kernel Library, Vector Statistical Library Notes. See section **Random Streams and RNGs in Parallel Computation.**

http://software.intel.com/sites/products/documentation/doclib/mkl_sa/11/vslnotes/vslnotes.pdf

See Also

[rxDTree](#), [rxPredict.rxDTree](#), [rxDForestUtils](#), [rxRngNewStream](#).

Examples

```
set.seed(1234)

# classification
iris.sub <- c(sample(1:50, 25), sample(51:100, 25), sample(101:150, 25))
iris.dforest <- rxDForest(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
  data = iris[iris.sub, ])
iris.dforest

table(rxPredict(iris.dforest, iris[-iris.sub, ], type = "class")[[1]],
  iris[-iris.sub, "Species"])

# regression
infert.nrow <- nrow(infert)
infert.sub <- sample(infert.nrow, infert.nrow / 2)
infert.dforest <- rxDForest(case ~ age + parity + education + spontaneous + induced,
  data = infert[infert.sub, ], cp = 0.01)
infert.dforest

hist(rxPredict(infert.dforest, infert[-infert.sub, ])[[1]] -
  infert[-infert.sub, "case"])

# .xdf file
claimsXdf <- file.path(rxGetOption("sampleDataDir"), "claims.xdf")
claims.dforest <- rxDForest(cost ~ age + car.age + type,
  data = claimsXdf)
```

rxDForestUtils: Utility Functions for rxDForest

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Utility Functions for rxDForest.

Usage

```
rxVarImpPlot(x, sort = TRUE, n.var = 30, main = deparse(substitute(x)),    ... )  
rxLeafSize(x, use.weight = TRUE)  
rxTreeDepth(x)  
rxTreeSize(x, terminal = TRUE)  
rxVarUsed(x, by.tree = FALSE, count = TRUE)  
rxGetTree(x, k = 1)
```

Arguments

`x`

an object of class `rxDForest` or `rxDTree`.

`sort`

logical value. If `TRUE`, the variables will be sorted in decreasing importance.

`n.var`

an integer specifying the number of variables to show when `sort=FALSE`.

`main`

a character string specifying the main title for the plot.

`...`

other arguments to be passed on to dotchart.

`use.weight`

logical value. If `TRUE`, the leaf size is measured by the total weight of its observations instead of the total number of its observations.

`terminal`

logical value. If `TRUE`, only the terminal nodes will be counted.

`by.tree`

logical value. If `TRUE`, the list of variables used will be broken down by trees.

`count`

logical value. If `TRUE`, the frequencies that variables appear in trees will be returned.

`k`

an integer specifying the index of the tree to be extracted.

Value

- `rxVarImpPlot` - plots a dotchart of the variable importance as measured by the decision forest.
- `rxLeafSize` - returns the size of the terminal nodes in the decision forest.
- `rxTreeDepth` - returns the depth of trees in the decision forest.
- `rxTreeSize` - returns the size of trees in terms of the number of nodes in the decision forest.
- `rxVarUsed` - finds out the variables actually used in the decision forest.
- `rxGetTree` - extracts a single tree from the decision forest.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

`randomForest`.

See Also

[rxDForest](#), [rxDTree](#), [rxBTrees](#).

Examples

```
set.seed(1234)

# classification
iris.sub <- c(sample(1:50, 25), sample(51:100, 25), sample(101:150, 25))
iris.dforest <- rxDForest(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
                           data = iris[iris.sub, ], importance = TRUE)

rxVarImpPlot(iris.dforest)
rxTreeSize(iris.dforest)
rxVarUsed(iris.dforest)
rxGetTree(iris.dforest)
```

RxDistributedHpa-class: Class RxDistributedHpa

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Compute context class for clusters supporting RevoScaleR High Performance Analytics and High Performance Computing.

Extends

Class RxComputeContext, directly.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxComputeContext-class](#), [RxHadoopMR](#), [RxSpark](#), [RxInSqlServer](#).

rxDistributeJob: Distribute job across nodes of a cluster

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Allows distributed execution of a function in parallel across nodes (computers) of a 'compute context' such as a cluster. A helper functions checks to see if the 'compute context' is appropriate.

Usage

```
rxDistributeJob(matchCallList, matchCall)
rxIsDistributedContext( computeContext = NULL, data = NULL)
```

Arguments

`matchCallList`

a list containing the function name and arguments, adjusting environments as needed

`matchCall`

a call in which all of the specified arguments are specified by their full names; typically the result of `match.call`

`computeContext`

`NULL` or an [RxComputeContext](#) object.

`data`

`NULL` or an [RxDataSource](#) object. If specified, compatibility of the data source with the 'dataDistType' in the compute context will be checked.

Details

An example of usage can be found in the [RevoPemaR](#) package.

Value

The result of the distributed computation.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxExec](#), [RxComputeContext](#), [rxSetComputeContext](#), [rxGetComputeContext](#)

Examples

```
## Not run:  
  
# Example is provided in the RevoPemaR package  
  
## End(Not run)
```

rxDTree: Parallel External Memory Algorithm for Classification and Regression Trees

7/12/2022 • 12 minutes to read • [Edit Online](#)

Description

Fit classification and regression trees on an .xdf file or data frame for small or large data using parallel external memory algorithm.

Usage

```
rxDTree(formula, data,
        outFile = NULL, outColName = ".rxNode", writeModelVars = FALSE, extraVarsToWrite = NULL, overwrite =
        FALSE,
        pweights = NULL, fweights = NULL, method = NULL, parms = NULL, cost = NULL,
        minSplit = max(20, sqrt(numObs)), minBucket = round(minSplit/3), maxDepth = 10, cp = 0,
        maxCompete = 0, maxSurrogate = 0, useSurrogate = 2, surrogateStyle = 0, xVal = 2,
        maxNumBins = NULL, maxUnorderedLevels = 32, removeMissings = FALSE,
        computeObsNodeId = NULL, useSparseCube = rxGetOption("useSparseCube"), findSplitsInParallel = TRUE,
        pruneCp = 0,
        rowSelection = NULL, transforms = NULL, transformObjects = NULL, transformFunc = NULL,
        transformVars = NULL, transformPackages = NULL, transformEnvir = NULL,
        blocksPerRead = rxGetOption("blocksPerRead"), reportProgress = rxGetOption("reportProgress"),
        verbose = 0, computeContext = rxGetOption("computeContext"),
        xdfCompressionLevel = rxGetOption("xdfCompressionLevel"),
        ... )
```

Arguments

formula

formula as described in [rxFormula](#). Currently, formula functions are not supported.

data

either a data source object, a character string specifying a .xdf file, or a data frame object.

outFile

either an RxXdfData data source object or a character string specifying the .xdf file for storing the resulting node indices. If `NULL`, then no node indices are stored to disk. If the input data is a data frame, the node indices are returned automatically. If `rowSelection` is specified and not `NULL`, then `outFile` cannot be the same as the `data` since the resulting set of node indices will generally not have the same number of rows as the original data source.

outColName

character string to be used as a column name for the resulting node indices if `outFile` is not `NULL`. Note that `make.names` is used on `outColName` to ensure that the column name is valid. If the `outFile` is an `RxOdbcData` source, dots are first converted to underscores. Thus, the default `outColName` becomes `"X_rxNode"`.

writeModelVars

logical value. If `TRUE`, and the output file is different from the input file, variables in the model will be written to the output file in addition to the node numbers. If variables from the input data set are transformed in the model, the transformed variables will also be written out.

`extraVarsToWrite`

`NULL` or character vector of additional variables names from the input data or transforms to include in the `outFile`. If `writeModelVars` is `TRUE`, model variables will be included as well.

`overwrite`

logical value. If `TRUE`, an existing `outFile` with an existing column named `outColName` will be overwritten.

`pweights`

character string specifying the variable of numeric values to use as probability weights for the observations.

`fweights`

character string specifying the variable of integer values to use as frequency weights for the observations.

`method`

character string specifying the splitting method. Currently, only `"class"` or `"anova"` are supported. The default is `"class"` if the response is a factor, otherwise `"anova"`.

`parms`

optional list with components specifying additional parameters for the `"class"` splitting method, as follows:

- `prior` - a vector of prior probabilities. The priors must be positive and sum to 1. The default priors are proportional to the data counts.
- `loss` - a loss matrix, which must have zeros on the diagonal and positive off-diagonal elements. By default, the off-diagonal elements are set to 1.
- `split` - the splitting index, either `gini` (the default) or `information`.
If `parms` is specified, any of the components can be specified or omitted. The defaults will be used for missing components.

`cost`

a vector of non-negative costs, containing one element for each variable in the model. Defaults to one for all variables. When deciding which split to choose, the improvement on splitting on a variable is divided by its cost.

`minSplit`

the minimum number of observations that must exist in a node before a split is attempted. By default, this is `sqrt(num of obs)`. For non-XDF data sources, as `(num of obs)` is unknown in advance, it is wisest to specify this argument directly.

`minBucket`

the minimum number of observations in a terminal node (or leaf). By default, this is `minsplit /3`.

`cp`

numeric scalar specifying the complexity parameter. Any split that does not decrease overall lack-of-fit by at least `cp` is not attempted.

`maxCompete`

the maximum number of competitor splits retained in the output. These are useful model diagnostics, as they allow you to compare splits in the output with the alternatives.

`maxSurrogate`

the maximum number of surrogate splits retained in the output. See the Details for a description of how surrogate splits are used in the model fitting. Setting this to 0 can greatly improve the performance of the algorithm; in some cases almost half the computation time is spent in computing surrogate splits.

useSurrogate

an integer specifying how surrogates are to be used in the splitting process:

- 0 - display-only; observations with a missing value for the primary split variable are not sent further down the tree.
- 1 - use surrogates, in order, to split observations missing the primary split variable. If all surrogates are missing, the observation is not split.
- 2 - use surrogates, in order, to split observations missing the primary split variable. If all surrogates are missing or `maxSurrogate=0`, send the observation in the majority direction.

The 0 value corresponds to the behavior of the `tree` function, and 2 (the default) corresponds to the recommendations of Breiman et al.

xVal

the number of cross-validations to be performed along with the model building. Currently, `1:xVal` is repeated and used to identify the folds. If not zero, the `cptable` component of the resulting model will contain both the mean (`xerror`) and standard deviation (`xstd`) of the cross-validation errors, which can be used to select the optimal cost-complexity pruning of the fitted tree. Set it to zero if external cross-validation will be used to evaluate the fitted model because a value of k increases the compute time to $(k+1)$ -fold over a value of zero.

surrogateStyle

an integer controlling selection of a best surrogate. The default, 0, instructs the program to use the total number of correct classifications for a potential surrogate, while 1 instructs the program to use the percentage of correct classification over the non-missing values of the surrogate. Thus, 0 penalizes potential surrogates with a large number of missing values.

maxDepth

the maximum depth of any tree node. The computations take much longer at greater depth, so lowering `maxDepth` can greatly speed up computation time.

maxNumBins

the maximum number of bins to use to cut numeric data. The default is `min(1001, max(101, sqrt(num of obs)))`. For non-XDF data sources, as `(num of obs)` is unknown in advance, it is wisest to specify this argument directly. If set to 0, unit binning will be used instead of cutting. See the 'Details' section for more information.

maxUnorderedLevels

the maximum number of levels allowed for an unordered factor predictor for multiclass (>2) classification.

removeMissings

logical value. If TRUE, rows with missing values are removed and will not be included in the output data.

computeObsNodeId

logical value or NULL. If TRUE, the tree node IDs for all the observations are computed and returned. If NULL, the IDs are computed for data.frame with less than 1000 observations and are returned as the `where` component in the fitted `rxDTree` object.

useSparseCube

logical value. If TRUE, sparse cube is used.

`findSplitsInParallel`

logical value. If `TRUE`, optimal splits for each node are determined using parallelization methods; this will typically speed up computation as the number of nodes on the same level is increased. Note that when it is `TRUE`, the number of nodes being processed in parallel is also printed to the console, interleaved with the number of rows read from the input data set.

`pruneCp`

Optional complexity parameter for pruning. If `pruneCp > 0`, `prune.rxDTree` is called on the completed tree with the specified `pruneCp` and the pruned tree is returned. This contrasts with the `cp` parameter that determines which splits are considered in *growing* the tree. The option `pruneCp="auto"` causes `rxDTTree` to call the function `rxDTTreeBestCp` on the completed tree, then use its return value as the `cp` value for `prune.rxDTree`.

`rowSelection`

name of a logical variable in the data set (in quotes) or a logical expression using variables in the data set to specify row selection. For example, `rowSelection = "old"` will use only observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` will use only observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the `expression` function.

`transforms`

an expression of the form `list(name = expression, ...)` representing the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the `expression` function.

`transformObjects`

a named list containing objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

`transformFunc`

variable transformation function. The ".rxSetLowHigh" attribute must be set for transformed variables if they are to be used in `formula`. See [rxTransform](#) for details.

`transformVars`

character vector of input data set variables needed for the transformation function. See [rxTransform](#) for details.

`transformPackages`

character vector defining additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, e.g., those explicitly defined in RevoScaleR functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.

`transformEnvir`

user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

`blocksPerRead`

number of blocks to read for each chunk of data read from the data source.

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`verbose`

integer value. If `0`, no verbose output is printed during calculations. Integer values from `1` to `2` provide increasing amounts of information are provided.

`computeContext`

a valid `RxComputeContext`. The `RxSpark` and `RxHadoopMR` compute contexts distribute the computation among the nodes specified by the compute context; for other compute contexts, the computation is distributed if possible on the local computer.

`xdfCompressionLevel`

integer in the range of -1 to 9 indicating the compression level for the output data if written to an `.xdf` file. The higher the value, the greater the amount of compression - resulting in smaller files but a longer time to create them. If `xdfCompressionLevel` is set to 0, there will be no compression and files will be compatible with the 6.0 release of Revolution R Enterprise. If set to -1, a default level of compression will be used.

`...`

additional arguments to be passed directly to the Microsoft R Services Compute Engine.

Details

`rxDTTree` is a parallel external memory decision tree algorithm targeted for very large data sets.

It is modeled after `rpart` (Version 4.1-0) and inspired by the algorithm proposed by Yael Ben-Haim and Elad Tom-Tov (2010).

It uses a histogram as the approximate compressed representation of the data and builds the tree in a breadth-first fashion using horizontal parallelism.

`maxNumBins` specifies the maximum number of bins for the histogram of each continuous independent variable and thus controls the accuracy of the algorithm. Also, `rxDTTree` builds histograms with roughly equal number of observations in each bin and checks only the boundaries of the bins as candidate splits to find the best split. So it is possible that a suboptimal split is chosen if `maxNumBins` is too small. This may cause the tree to be different from one constructed by a standard algorithm. Increasing `maxNumBins` allows more accurate results but with increased time and memory usage..

Surrogate splits may be used to assign observations for which the primary split variable is missing. Surrogate splits compare the groups produced by the remaining predictor variables to the groups produced by the primary split variable, and the predictors are ranked by how well their groups match the primary predictor. The best match is used as the surrogate split.

Value

an object of class `"rxDTTree"` representing the fitted tree. It is a list with components similar to those of class `"rpart"` with the following distinctions:

- `where` - A vector of integers indicating the node to which each point is allocated. This information is always returned if the data source is a data frame. If the data source is not a data frame and `outFile` is specified. i.e.,

not `NULL`, the node indices are written/appended to the specified file with a column name as defined by `outColName`.

For other components, see `rpart.object` for details.

Note

`rxDTree` requires multiple passes over the data set and the maximum number of passes can be computed as follows:

- quantile computation: `1` pass for computing the quantiles for all continuous variables,
- recursive partition: `maxDepth + 1` passes for building the tree on the entire dataset,
- tree node assignment: `1` pass for computing the id of the leaf node that each observation falls into.

If cross validation is specified (i.e., `xVal > 0`), additional passes will be needed for each fold:

- recursive partition: `maxDepth + 1` passes for building the tree on the other folds,
- tree node assignment: `1` pass for predicting on the current fold.

resulting in `xVal * ((maxDepth + 1) + 1)` additional passes for cross validation.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. (1984) *Classification and Regression Trees*. Wadsworth.

Therneau, T. M. and Atkinson, E. J. (2011) *An Introduction to Recursive Partitioning Using the RPART Routines*.

Yael Ben-Haim and Elad Tom-Tov (2010) A streaming parallel decision tree algorithm. *Journal of Machine Learning Research* 11, 849--872.

See Also

`rpart`, `rpart.control`, `rpart.object`, [rxPredict.rxDTree](#), [rxAddInheritance](#), [rxForestUtils](#).

Examples

```

set.seed(1234)

# classification
iris.sub <- c(sample(1:50, 25), sample(51:100, 25), sample(101:150, 25))
iris.dtree <- rxDTree(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
  data = iris[iris.sub, ])
iris.dtree

table(rxPredict(iris.dtree, iris[-iris.sub, ], type = "class")[[1]],
  iris[-iris.sub, "Species"])

# regression
infert.nrow <- nrow(infert)
infert.sub <- sample(infert.nrow, infert.nrow / 2)
infert.dtree <- rxDTree(case ~ age + parity + education + spontaneous + induced,
  data = infert[infert.sub, ], cp = 0.01)
infert.dtree

hist(rxPredict(infert.dtree, infert[-infert.sub, ])[[1]] -
  infert[-infert.sub, "case"])

# use transformations with rxDTree
range(log(iris$Sepal.Width)) ## find out the overall low/high values of the transformed variable
#[1] 0.6931472 1.4816045

myTransform <- function(dataList)
{
  dataList$log.Sepal.Width <- log(dataList$Sepal.Width)
  attr(dataList$log.Sepal.Width, ".rxSetLowHigh") <- c(0.6931472, 1.4816045) # set the overall low/high
values
  return(dataList)
}

frm <- Sepal.Length ~ log.Sepal.Width + Petal.Length + Petal.Width + Species
model <- rxDTree(frm, iris, transformFunc = myTransform, transformVars = c("Sepal.Width"))
stopifnot("log.Sepal.Width" %in% names(model$variable.importance))

# .xdf file
claimsXdf <- file.path(rxGetOption("sampleDataDir"), "claims.xdf")
claims.dtree <- rxDTree(cost ~ age + car.age + type,
  data = claimsXdf)
claimsCp <- rxDTreeBestCp(claims.dtree)
claims.dtree1 <- prune.rxDTree(claims.dtree, cp=claimsCp)
claims.dtree2 <- rxDTree(cost ~ age + car.age + type,
  data = claimsXdf, pruneCp="auto")
## claims.dtree1 and claims.dtree2 should differ only in their
## "call" component
claims.dtree1[[3]] <- claims.dtree2[[3]] <- NULL
all.equal(claims.dtree1, claims.dtree2)

```

rxDTreeBestCp: Find the Best Value of cp for Pruning rxDTree Object

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Attempts to find the cp for optimal model pruning, where optimal is defined by default in terms of the 1 Standard Error criterion of Breiman, et al.

Usage

```
rxDTreeBestCp(x, nstd = 1L)
```

Arguments

x

an object of class `rxDTree`.

nstd

number of standard errors of cross-validation error to define optimality. The default, 1, causes `rxDTreeBestCp` to find the simplest model within one standard error of the minimum cross-validation error.

Details

The `rxDTreeBestCp` function is intended to help automate the `rxDTree` model fitting function, by applying the 1 Standard Error Criterion of Breiman, et al., to a fitted `rxDTree` object. Using the `pruneCp="auto"` option in `rxDTree` causes `rxDTreeBestCp` to be called on the fitted model and the result of that computation supplied as the argument to `prune.rxDTree`, thus allowing the model to be fitted and pruned in a single call.

Value

a numeric scalar.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. (1984) *Classification and Regression Trees*. Wadsworth.

See Also

[rxDTree](#), [prune.rxDTree](#).

Examples

```
claimsXdf <- file.path(rxGetOption("sampleDataDir"),"claims.xdf")
claims.dtree <- rxDTree(cost ~ age + car.age + type,
  data = claimsXdf)
claimsCp <- rxDTreeBestCp(claims.dtree)
claims.dtree1 <- prune.rxDTree(claims.dtree, cp=claimsCp)
claims.dtree2 <- rxDTree(cost ~ age + car.age + type,
  data = claimsXdf, pruneCp="auto")
## claims.dtree1 and claims.dtree2 should differ only in their
## "call" component
claims.dtree1[[3]] <- claims.dtree2[[3]] <- NULL
all.equal(claims.dtree1, claims.dtree2)
```

rxElemArg: Helper function for rxExec arguments

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Allows different argument values to be passed to different (named and unnamed) nodes or cores through the ellipsis argument for rxExec. A vector or list of the argument values is used.

Usage

```
rxElemArg(x)
```

Arguments

x

The list or vector to be applied across the set of computations.

Details

This function is designed for use only within a call to `rxExec`. `rxExec` allows for the processing of a user function on multiple nodes or cores. Arguments for the user function can be passed directly through the call to `rxExec`. By default, the same argument value will be passed to each of the nodes or cores. If instead, a vector or list of argument values is wrapped in a call to `rxElemArg`, a distinct argument value will be passed to each node or core.

If `timesToRun` is specified for `rxExec`, the length of the vector or list within `rxElemsArg` must have a length equal to `timesToRun`. If `timesToRun` is not specified and the `elemType` is set to `"cores"`, the length of the vector or list will determine the `timesToRun`.

If `elemArgs` is used in addition to the `rxElemArg` function, the length of both lists/vectors must be the same.

Value

x is returned with attributes modified for use by rxExec.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxExec](#)

Examples

```
## Not run:

# Setup a Spark compute context
myCluster <- RxSpark(nameNode = "my-name-service-server", port = 8020, wait = TRUE)
rxOptions( computeContext = myCluster )

# Create a function to be run on multiple cores
myFunc <- function( sampleSize = 100, seed = 5)
{
  set.seed(seed)
  mean( rnorm( sampleSize ) )
}

# Use the same sampleSize for every computation, but a separate seed.
# The function will run 20 times, because that is the length of the
# vector in rxElemArg
mySeeds <- runif(20, min=1, max=1000)
rxEexec( myFunc, 100, seed = rxElemArg( mySeeds ), elemType = "cores")

# Send different sample sizes and different seeds
# Both mySeeds and mySampleSizes must have the same length
mySampleSizes <- 2^c(1:20)
res <- rxEexec(myFunc, rxElemArg( mySampleSizes ), seed = rxElemArg( mySeeds ),
  elemType = "cores")
## End(Not run)
```

rxExec: Run A Function on Multiple Nodes or Cores

7/12/2022 • 9 minutes to read • [Edit Online](#)

Description

Allows distributed execution of a function in parallel across nodes (computers) or cores of a "compute context" such as a cluster.

Usage

```
rxExec(FUN, ... , elemArgs, elemType = "nodes", oncePerElem = FALSE, timesToRun = -1L,
       packagesToLoad = NULL, execObjects = NULL, taskChunkSize = NULL, quote = FALSE,
       consoleOutput = NULL, autoCleanup = NULL, continueOnFailure = TRUE,
       RNGseed = NULL, RNGkind = NULL, foreachOpts = NULL)
```

Arguments

`FUN`

the function to be executed; the nodes or cores on which it is run are determined by the currently-active compute context and by the other arguments of `rxExec`.

`...`

arguments passed to the function `FUN` each time it is executed. Separate argument values can be sent for each computation by wrapping a vector or list of argument values in `rxElemArg`.

`elemArgs`

a vector or list specifying arguments to `FUN`. This allows a different set of arguments to be passed to `FUN` each time it is executed. The length of the vector or list must match the number of times the function will be executed. Each of these elements will be passed in turn to `FUN`. Using a list of lists allows multiple named or unnamed parameters to be passed. If `elemArgs` has length 1, that argument is passed to all compute elements (and thus is an alternative to `...`). The elements of `elemArgs` may be named; if they are node names those elements will be passed to those nodes. Alternatively, they can be "rxElem1", "rxElem2" and so on. In this case, the list of returned values will have those corresponding names. See the Details section for more information. This is an alternative to using `rxElemArg` one or more times.

`elemType`

[Deprecated]. The distributed computing mode to be used. This parameter is currently deprecated and is not honored by any of the supported compute contexts. It might come back to use in the future.

`oncePerElem`

logical flag. If `TRUE` and `elemType="nodes"`, `FUN` will be run exactly once on each specified node. In this case, each element of the return list will be named with the name of the node that computed that element. If `FALSE`, a node may be used more than once (but never simultaneously). `oncePerElem` must be set to `FALSE` if `elemType="cores"`. This parameter is ignored if the active compute context is local.

`timesToRun`

integer specifying the total number of instances of the function `FUN` to run. If `timesToRun=-1`, the default, then

times is set to the length of the `elemArgs` argument, if it exists, else to the number of nodes or cores specified in the compute context object, if that is exact. In the latter case, if the `elementType="nodes"` and a single set of arguments is being passed to each node, each element of the return list will be named with the name of the node that computed that element. If `timesToRun` is not -1, it must be consistent with this other information.

`packagesToLoad`

optional character vector specifying additional packages to be loaded on the nodes for this job. If provided, these packages are loaded after any `packagesToLoad` specified in the current distributed compute context.

`execObjects`

optional character vector specifying additional objects to be exported to the nodes for this job, or an environment containing these objects. The specified objects are added to `FUN`'s environment, unless that environment is locked, in which case they are added to the environment in which `FUN` is evaluated. For purposes of efficiency, this argument should not be used for exporting large data objects. Passing large data through reference to a shared storage location (e.g., HDFS) is recommended.

`taskChunkSize`

optional integer scalar specifying the number of tasks to be executed per compute element, or worker. By submitting tasks in chunks, you can avoid some of the overhead of starting new R processes over and over. For example, if you are running thousands of identical simulations on a cluster, it makes sense to specify the `taskChunkSize` so that each worker can do its allotment of tasks in a single R process. This argument is incompatible with the `oncePerElem` argument; if both are supplied, this one is ignored. It is also incompatible with lists supplied to `elemArgs` with compute element names.

`quote`

logical flag. If `TRUE`, underlying calls to `do.call` have the corresponding flag set to `TRUE`. This is primarily of use to the `doRSR` package, but may be of use to other users.

`consoleOutput`

`NULL` or logical value. If `TRUE`, the console output from all of the processes is printed to the user console. Note that the output from different nodes or cores may be interleaved in an unpredictable way. If `FALSE`, no console output is displayed. Output can be retrieved with the function `rxGetJobOutput` for a non-waiting job. If not `NULL`, this flag overrides the value set in the compute context when the job was submitted. If `NULL`, the setting in the compute context will be used. This parameter is ignored if the active compute context is local.

`autoCleanup`

`NULL` or logical value. If `TRUE`, artifacts created by the distributed computing job are deleted when the results are returned or retrieved using `rxGetJobResults`. If `FALSE`, the artifacts are not deleted, and the results may be obtained repeatedly using `rxGetJobResults`, and the console output via `rxGetJobOutput` until `rxCleanupJobs` is used to delete the artifacts. If not `NULL`, this flag overrides the value set in the compute context when the job was submitted. If you routinely set `autoCleanup=FALSE`, you may eventually fill your hard disk with compute artifacts. If you set `autoCleanup=TRUE` and experience performance degradation on a Windows XP client, consider setting `autoCleanup=FALSE`. This parameter is ignored if the active compute context is local.

`continueOnFailure`

`NULL` or logical value. If `TRUE`, the default, then if an individual instance of a job fails due to a hardware or network failure, an attempt will be made to rerun that job. (R syntax errors, however, will cause immediate failure as usual.) Furthermore, should a process instance of a job fail due to a user code failure, the rest of the processes will continue, and the failed process will produce a warning when the output is collected. Additionally, the position in the returned list where the failure occurred will contain the error as opposed to a result. This parameter is ignored if the active compute context is local or `RxForeachDoPar`.

RNGseed

`NULL`, the string `"auto"`, or an integer to be used as the seed for parallel random number generation. See the Details section for a description of how the `"auto"` string is used.

RNGkind

`NULL` or a character string specifying the type of random number generator to be used. Allowable strings are the strings accepted by `rxRngNewStream`, `"auto"`, and, if the active compute context is local parallel, `"L'Ecuyer-CMRG"` (for compatibility with the parallel package). See the Details section for a description of how the `"auto"` string is used.

foreachOpts

`NULL` or a list containing options to be passed to the foreach parallel computing backend. See `foreach` for details.

Details

`rxExec` has very limited functionality for `RxInSqlServer` for CTP3; computations are performed sequentially. There are two primary sets of use cases: In the first set, each computing element (node or core) gets the same argument values; in this case, do not use `elemArgs` or `rxElemArg`. In the second, each element gets a different set of arguments; use `rxElemArg` for each argument that has different values, or an `elemArgs` whose length is equal to the number of times `FUN` will be executed.

Set 1 (All computing elements get the same arguments):

```
1 rxExec(FUN, arg1, arg2)
```

Set 2: Every computing element gets a different set of arguments. If `rxElemArg` is used, the length of the vector or list for the enclosed argument must equal the number of compute elements. For example,

```
rxExec(FUN, arg1 = 1, arg2 = rxElemArg(c(1:5)))
```

If `elemArgs` is a nested list, the individual lists are passed to the compute resources according to the following:

1 The argument lists can be named according to which compute resource each component list should be assigned. For example, `rxExec(FUN, elemArgs=list(compute1=list(arg1,arg2), compute2=list(arg3, arg4)))`. In this case, the list of arguments must be the same length as the list of nodes requested for the current compute context, and have the same names. If `oncePerElem=TRUE` and `elemType="nodes"`, then the computation will be performed once on each requested node, and each node is assured of getting the argument with its name. If `oncePerElem=FALSE`, there is no guarantee that each node will be used in the processing, so arguments intended for a particular node may not be used; they must still be provided, however.

The component names must be valid R syntactic names. If you have nodes on your cluster with names that are not valid R syntactic names, use the function `rxMakeRNodeNames` on the node name to determine the appropriate name to give the list component. When the return value is a list with elements named by compute node, the node names are as returned by the `rxMakeRNodeNames` function.

1 The arguments lists, if not named, will be passed to the compute resources allowed by the compute context according to their position in the list. This is useful when you don't care which nodes or cores the function is executed on but want different arguments to be executed on each resource. For example,

```
rxExec(FUN, elemArgs=list(list(arg1,arg2), list(arg3, arg4))) or  
rxExec(FUN, elemArgs=list(c(arg1, arg2), list(arg3, arg4)))
```

The arguments `RNGseed` and `RNGkind` can be used to control random number generation in the workers. By default, both are `NULL` and no special random number control is used. If either or both `RNGseed` and `RNGkind` are set to `"auto"`, a parallel random number stream is initialized on each worker, using the `"MT2203"` generator

and separate substreams for each worker. If other non-null valid values are supplied for these arguments, they are used as is for the `"MT2203"` generator, which supports multiple substreams, but for other `rxRngNewStream` - supported generators, the seed will be used as the starting point of a sequence of seeds, one for each worker. In the special case of a local parallel compute context, the `"L'Ecuyer-CMRG"` generator case can be specified, in which case the parallel package's `clusterSetRNGStream` function is called on the internally generated parallel cluster.

Value

If a waiting compute context is active, a list with an element for each job, where each element contains the value(s) returned by that job's function call(s). If a non-waiting compute context is active, a `jobInfo` object. See [rxGetJobResults](#).

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxElemArg](#), [rxGetJobResults](#), [rxGetJobStatus](#), [RxComputeContext](#), [rxSetComputeContext](#), [RxSpark](#), [RxHadoopMR](#), [RxForeachDoPar](#), [RxLocalParallel](#), [RxLocalSeq](#), [rxGetNodeInfo](#), [rxMakeRNodeNames](#) [rxRngNewStream](#)

Examples

```

## Not run:

## Run function with no parameters
rxExec(getwd)

## Pass the same set of arguments to each compute element
rxExec(list.files, all.files=TRUE, full.names=TRUE)

## Run function with the same vector sent as the first
## argument to each compute element
## The values 1 to 10 will be printed 10 times
x <- 1:10
rxExec(print, x, elemType = "cores", timesToRun = 10)

## Pass a different argument value to each compute element
## The values 1 to 10 will be printed once each
rxExec(print, rxElemArg( x ), elemType = "cores")

## Extract different columns from a data frame on different nodes
set.seed(100)
myData <- data.frame(x = 1:100, y = rep(c("a", "b", "c", "d"), 25),
z = rnorm(100), w = runif(100))
myVarsToKeep = list(
  c("x", "y"),
  c("x", "z"),
  c("x", "w"),
  c("y", "z"),
  c("z", "w"))
# myVarDataFrames will be a list of data frames
myVarDataFrames <- rxExec(rxDataStep, inData = myData, varsToKeep = rxElemArg(myVarsToKeep))

## Extract different rows from the data frame on different nodes
myRowSelection = list(
  expression(y == 'a'),
  expression(y == 'b'),
  expression(y == 'c'),
  expression(y == 'd'),
  expression(z > 0))

myRowDataFrames <- rxExec(rxDataStep, inData = myData, rowSelection = rxElemArg(myRowSelection))

## Use the taskChunkSize argument
rxExec(sqrt, rxElemArg(1:100), taskChunkSize=50)
## End(Not run)

```

rxExecBy: Partition Data by Key Values and Execute User Function on Each Partition

7/12/2022 • 4 minutes to read • [Edit Online](#)

Description

Partition input data source by keys and apply user defined function on individual partitions. If input data source is already partitioned, apply user defined function on partitions directly. Currently supported in `local`, `localpar`, [RxInSqlServer](#) and [RxSpark](#) compute contexts.

Usage

```
rxExecBy(inData, keys, func, funcParams = NULL, filterFunc = NULL, computeContext =
  rxGetOption("computeContext"), ...)
```

Arguments

inData

a data source object supported in currently active compute context, e.g., [RxSqlServerData](#) for [RxInSqlServer](#) and [RxHiveData](#) for [RxSpark](#). In `local` and `localpar` compute contexts, a character string specifying a .xdf file or a data frame object can be also used.

keys

character vector of variable names to specify the values in those variables are used for partitioning.

func

the user function to be executed. The user function takes `keys` and `data` as two required input arguments where `keys` determines the partitioning values and `data` is a data source object of the corresponding partition. `data` can be a [RxXdfData](#) object or a RxODBCData object, which can be transformed to a standard R data frame by using [rxDataStep](#) method. The nodes or cores on which it is running are determined by the currently active compute context.

funcParams

list of additional arguments for the user function `func`.

filterFunc

the user function that takes a data frame of keys values as an input argument, applies filter to the keys values and returns a data frame containing rows whose keys values satisfy the filter conditions. The input data frame has similar format to the results returned by [rxPartition](#) which comprises of partitioning variables and an additional variable of partition data source. This `filterFunc` allows user to control what data partitions to be applied by the user function `func`. `filterFunc` currently is not supported in RxHadoopMR and [RxSpark](#) compute contexts.

computeContext

a RxComputeContext object.

...

additional arguments to be passed directly to the Compute Engine.

Value

A list which is the same length as the number of partitions in the `inData` argument. Each element in the top level list contains a three element list described below.

`keys`

a list which contains key values for the partition.

`result`

the object returned from the invocation of the user function with `keys` values. When an error occurs during the invocation of the user function the value will be `NULL`.

`status`

a string which takes the value of either `"OK"` or `"Error"`. In RxSpark compute context, it may include additional warning and error messages.

Note

The user function can call any function defined in R packages which are loaded by default and pass parameters which are defined in base R, the default loaded packages, or user defined S3 classes. The user function won't work with global variables or functions in non-default loaded packages unless they are redefined or reloaded within the scope of the user function.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxExecByPartition](#), [rxImport](#), [rxDataStep](#), [RxTextData](#), [RxXdfData](#), [RxHiveData](#), [RxSqlServerData](#)

Examples

```
## Not run:  
  
#####  
# run analytics with local compute context  
#####  
  
# create an input xdf data source.  
inFile <- "claims.xdf"  
inFile <- file.path(dataPath = rxGetOption(opt = "sampleDataDir"), inFile)  
inXdfDataSource <- RxXdfData(file = inFile)  
  
# define an user function that builds a linear model  
.linMod" <- function(keys, data)  
{  
  result <- rxLinMod(formula = cost ~ number, data = data)  
  return(result$coefficients[[1]])  
}  
  
# set local compute context with 4-way parallel  
rxSetComputeContext("localpar")
```

```

rxOptions(numCoresToUse = 4)

# define a filter function
".carFilter" <- function(partDF)
{
  subset(partDF, car.age != "10+" & type == "A")
}

# call rxExecBy with no filterFunction
results1 <- rxExecBy(inData = inXdfDataSource, keys = c("car.age", "type"), func = .linMod)

# call rxExecBy with filterFunction
results2 <- rxExecBy(inData = inXdfDataSource, keys = c("car.age", "type"), func = .linMod, filterFunc =
.carFilter)

#####
# run analytics with SQL Server compute context
#####

# Note: for improved security, read connection string from a file, such as
# sqlServerConnString <- readLines("sqlServerConnString.txt")

sqlServerConnString <- "SERVER=hostname;DATABASE=RevoTestDB;UID=DBUser;PWD=Password;"
inTable <- paste0("airlinedemosmall")
sqlServerDataDS <- RxSqlServerData(table = inTable, connectionString = sqlServerConnString)

# user function
".Count" <- function(keys, data, params)
{
  myDF <- rxImport(inData = data)
  return (nrow(myDF))
}

# Set SQL Server compute context with level of parallelism = 2
sqlServerCC <- RxInSqlServer(connectionString = sqlServerConnString, numTasks = 4)
rxSetComputeContext(sqlServerCC)

# Execute rxExecBy in SQL Server compute context
sqlServerCCResults <- rxExecBy(inData = sqlServerDataDS, keys = c("DayOfWeek"), func = .Count)

#####
# run analytics with RxSpark compute context
#####

# start Spark app
sparkCC <- rxSparkConnect()

# define function to compute average delay
".AverageDelay" <- function(keys, data) {
  df <- rxDataStep(data)
  mean(df$ArrDelay, na.rm = TRUE)
}

# define colInfo
colInfo <-
list(
  ArrDelay = list(type = "numeric"),
  CRSDeptTime = list(type = "numeric"),
  DayOfWeek = list(type = "string")
)

# create text data source with airline data
textData <-
RxTextData(
  file = "/share/SampleData/AirlineDemoSmall.csv",
  firstRowIsColNames = TRUE,
  colInfo = colInfo,
  fileSystem = RxHdfsFileSystem()
)

```

```
# group textData by day of week and get average delay on each day
objs <- rxExecBy(textData, keys = c("DayOfWeek"), func = .AverageDelay)

# transform objs to a data frame
do.call(rbind, lapply(objs, unlist))

# stop Spark app
rxSparkDisconnect(sparkCC)
## End(Not run)
```

rxExecByPartition: RevoScaleR By Group Parallelism

7/12/2022 • 6 minutes to read • [Edit Online](#)

Description

This feature allows users to run analytics computation in parallel on individual data partitions split from an input data source based on the specified variables. In **RevoScaleR** version 9.1.0, we provide the necessary rx functions to be executed for functionalities of By-group parallelism. This document will describe different scenarios of By-group parallelism, running in a number of supported compute contexts.

Details

By-group Parallelism provides functionalities that allow users perform the following typical operations:

- * Create new data partitions: given an input data set and a list of partitioning variables, split the data set into multiple small data sets based on the values of the specified partitioning variables.
- * Append new data to existing data partitions: given an input data set and a list of partitioning variables, split the data set and append data in partitioned data sets to the existing corresponding data partitions. ... * Perform analytics computation on data partitions in parallel multiple times as needed.
- * Do all the above three operations in one step initially and then repeat computation multiple times.

In **RevoScaleR** version 9.1.0, we introduce three new rx functions for partitioning data set and performing computation in parallel:

- * [rxExecBy\(\)](#) - to partition an input data source and execute user function on each partition in parallel. If the input data source is already partitioned, the function will skip the partitioning step and directly trigger computation for user function on partitions.
- * [rxPartition\(\)](#) - to partition an input data source and store data partitions on disk. For this functionality, a new xdf file format, called Partitioned Xdf (PXdf) is introduced for storing data partitions as well as partition metadata information on disk. Partitioned Xdf file can then be loaded into an in-memory Partitioned Xdf object using `RxXdfData` to be used for performing computation on data partitions repeatedly by [rxExecBy](#).
- * [rxGetPartitions\(\)](#) - to enumerate unique partitioning values in an existing partitioned Xdf and return it as a data frame

Running analytics computation in parallel on partitioned data set with [rxExecBy\(\)](#)

Input data set provided as a data source object can be data sources of different types, such as data frame, text data, Xdf files, ODBC data source, SQL Server data source, etc. In version 9.1.0, [rxExecBy\(\)](#) is supported in local compute context, SQL Server compute context and Spark compute context. Depending on input data sources and compute context, [rxExecBy\(\)](#) will execute in different modes which are summarized in the following table:

COL 1	COL 2	COL 3
Data Source \ Compute Context	Local	SQL Server
Data frame, text data, xdf, other data sources that are not ODBC	Generate PXdf for partitions and execute computation on PXdf	--

COL 1	COL 2	COL 3
SQL Server data source or ODBC data source with <i>query</i> specified	Generate PXdf for partitions and execute computation on PXdf	Generate temporary Composite Xdf and PXdf for partitions and execute computation on PXdf
SQL Server data source or ODBC data source with <i>table</i> specified	Do streaming with SQL rewrite partition queries and execute computation on streaming partitions	Do streaming with SQL rewrite partition queries and execute computation on streaming

As shown in the table, when running analytics on local compute context, PXdf is first temporarily generated and saved on disk; then computation are applied on the generated PXdf. The example for running this scenario can be found in the [rxExecBy\(\)](#) documentation. It's worth to note that the temporary PXdf generated will be removed once the computation is completed. If user plans to run the analytics multiple times with the same data set and different user functions, it would be more efficient to go with the following recommended flow:

1 Create a new partitioned Xdf object with [RxXdfData\(\)](#) by specifying `createPartitionSet = TRUE`.

1 Construct data partitions for the newly created PXdf object from an input data set and save it to disk with [rxPartition\(\)](#).

1 Run analysis with user defined function on the data partitions of the PXdf object with [rxExecBy\(\)](#). This step can be repeated multiple times with different user defined functions and different subsets of data partitions using a `filterFunc` specified as an argument of [rxExecBy\(\)](#).

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxXdfData](#), [rxExecBy](#), [rxPartition](#), [rxGetPartitions](#), [rxSplit](#), [rxExec](#), [rxImport](#)

Examples

```
## Not run:

#####
# Run analytics on data partitions in one operation
#####

# create an input text data source
inFile <- "claims.txt"
inFile <- file.path(dataPath = rxgetOption(opt = "sampleDataDir"), inFile)
inTxtDS <- RxTextData(file = inFile, delimiter = ",")

# define an user function that builds a linear model.
".linMod" <- function(keys, data)
{
  result <- rxLinMod(formula = cost ~ number, data = data)
  return(result$coefficients[[1]])
}

# set local compute context with 4-way parallel
rxSetComputeContext("localpar")
rxOptions(numCoresToUse = 4)

# run the .linMod function on partitions split from the input Xdf data source
```

```

# based on the variable 'car.age'.
localCCResults <- rxExecBy(inData = inTxtDS, keys = c("car.age"), func = .linMod)

#####
# Run analytics on data partitions multiple times with different user functions
#####

# create an input Xdf data source
inFile <- "claims.xdf"
inFile <- file.path(dataPath = rxGetOption(opt = "sampleDataDir"), inFile)
inXdfDS <- RxXdfData(file = inFile)

# create an output partitioned Xdf data source
outFile <- file.path(tempdir(), "partitionedClaims")
outPartXdfDataSource <- RxXdfData(file = outFile, createPartitionSet = TRUE)

# construct and save the partitioned Xdf to disk
partDF <- rxPartition(inData = inXdfDS, outData = outPartXdfDataSource, varsToPartition = c("car.age",
"type"))

# define an user function that counts number of rows in each partition
".Count" <- function(keys, data)
{
  myDF <- rxImport(inData = data)
  return(nrow(myDF))
}

# local compute context
rxSetComputeContext("localpar")
rxOptions(numCoresToUse = 4)

# call rxExecBy to execute the user function on partitions
results1 <- rxExecBy(inData = outPartXdfDataSource, keys = c("car.age", "type"), func = .Count)

# define another user function
".linMod" <- function(keys, data)
{
  result <- rxLinMod(formula = cost ~ number, data = data)
  return(result$coefficients[[1]])
}

# call rxExecBy to execute the new user function on the same set of partitions
results2 <- rxExecBy(inData = outPartXdfDataSource, keys = c("car.age", "type"), func = .linMod)

# clean-up: delete the partitioned Xdf
unlink(outFile, recursive = TRUE, force = TRUE)

#####
# Append new data to existing partitions and run analytics
#####

# create two sets of data frames from Xdf data source
inFile <- "claims.xdf"
inFile <- file.path(dataPath = rxGetOption(opt = "sampleDataDir"), inFile)
inXdfDS <- RxXdfData(file = inFile)
inDF <- rxImport(inData = inXdfDS)

df1 <- inDF[1:50,]
df2 <- inDF[51:nrow(inDF),]

# create an output partitioned Xdf data source
outFile <- file.path(tempdir(), "partitionedClaims")
outPartXdfDataSource <- RxXdfData(file = outFile, createPartitionSet = TRUE)

# construct the partitioned Xdf from the first data set df1 and save to disk
partDF1 <- rxPartition(inData = df1, outData = outPartXdfDataSource, varsToPartition = c("car.age",
"type"), append = "none", overwrite = TRUE)

# define an user function that counts number of rows in each partition
".Count" <- function(keys, data)

```

```

{
  myDF <- rxImport(inData = data)
  return(nrow(myDF))
}

# local compute context
rxSetComputeContext("localpar")
rxOptions(numCoresToUse = 4)

# call rxExecBy to execute the user function on partitions created from the first data set
results1 <- rxExecBy(inData = outPartXdfDataSource, keys = c("car.age", "type"), func = .Count)

# append data from the second data set to the existing partitioned Xdf
partDF2 <- rxPartition(inData = df2, outData = outPartXdfDataSource, varsToPartition = c("car.age",
"type"))

# define another user function
".linMod" <- function(keys, data)
{
  result <- rxLinMod(formula = cost ~ number, data = data)
  return(result$coefficients[[1]])
}

# call rxExecBy to execute the new user function on partitions created from the data combined
# from both data sets df1 and df2
results2 <- rxExecBy(inData = outPartXdfDataSource, keys = c("car.age", "type"), func = .linMod)

# clean-up: delete the partitioned Xdf
unlink(outFile, recursive = TRUE, force = TRUE)
## End(Not run)

```

rxExecuteSQLDDL: Execute SQL Command for Data Manipulation, Definition, or Control

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Execute a command to define, manipulate, or control SQL data (but not return data).

Usage

```
rxExecuteSQLDDL(src, ...)
```

Arguments

`src`

An RxOdbcData data source.

`...`

Additional arguments, typically `sSQLString=` supplied with a character string specifying the SQL command to be executed. The typical SQL commands are `CREATE TABLE` and `DROP TABLE`.

Value

Returns `NULL`; executed for its side-effect (the manipulation of the SQL data source).

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxOdbcData](#)

Examples

```
## Not run:

# Note: for improved security, read connection string from a file, such as
# conString <- readLines("conString.txt")

conString <- "Driver=Teradata;DBNAME=myDb;Database=test;Uid=tester;pwd=pwd;"
outOdbcDS <- RxOdbcData(table = "MyClaims",
                         connectionString = conString,
                         useFastRead=TRUE)
rxOpen(outOdbcDS, "w")
rxExecuteSQLDDL(outOdbcDS, sSQLString = paste("CREATE TABLE [MyClaims]([RowNum] [int] NULL, [age] [char](5)
NULL, ",
                                               "[car_age] [char](3) NULL, [type] [char](1) NULL, ", " [cost] [float] NULL, [number] [float] NULL);",
                                               sep=""))
inTextData <- RxTextData(file = file.path(rxGetOption("sampleDataDir"), "claims.txt"),
                         stringsAsFactors = TRUE, useFastRead = TRUE)
outOdbcDS <- RxOdbcData(table = "MyClaims",
                         connectionString = conString,
                         useFastRead=TRUE)
rxDataStep(inData = inTextData, outFile = outOdbcDS)
## End(Not run)
```

pow: Formula Expression Functions for RevoScaleR

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Formula expression functions for **RevoScaleR**.

Usage

```
pow(x, exponent)
```

Arguments

x

variable name

exponent

number to use as an exponent

Details

Many of the primary analysis functions in **RevoScaleR** require a formula argument. Arithmetic expressions in those arguments provide flexibility in the model. This man page lists the functions in **RevoScaleR** that are not native to R.

`pow` performs the power transformation `x^exponent`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxFormula](#), [rxTransform](#), [rxCrossTabs](#), [rxCube](#), [rxLinMod](#), [rxLogit](#), [rxSummary](#).

Examples

```
# define local data.frame
admissions <- as.data.frame(UCBAdmissions)

# cross-tabulation
rxCrossTabs(pow(Freq, 2) ~ Gender : Admit, data = admissions)
rxCrossTabs(Freq^2 ~ Gender : Admit, data = admissions)

# linear modeling
rxLinMod(pow(Freq, 2) ~ Gender + Admit, data = admissions)
rxLinMod(Freq^2 ~ Gender + Admit, data = admissions)

# model summarization
rxSummary(~ pow(Freq, 2) + Gender + Admit, data = admissions)
rxSummary(~ I(Freq^2) + Gender + Admit, data = admissions)
```

rxFactors: Factor Variable Recoding

7/12/2022 • 11 minutes to read • [Edit Online](#)

Description

Recodes a factor variable by mapping one set of factor levels and indices to a new set. Can also be used to convert non-factor variable into a factor.

Usage

```
rxFactors(inData, factorInfo, sortLevels = FALSE, otherLevel = NULL,
          outFile = NULL, varsToKeep = NULL, varsToDelete = NULL,
          overwrite = FALSE, maxRowsByCols = NULL,
          blocksPerRead = rxGetOption("blocksPerRead"),
          reportProgress = rxGetOption("reportProgress"), verbose = 0,
          xdfCompressionLevel = rxGetOption("xdfCompressionLevel"), ...)
```

Arguments

inData

either an RxXdfData object, a character string specifying the .xdf file, or a data frame.

factorInfo

character vector of variable names, a list of named variable information lists, or empty or `NULL`. If `sortLevels` is set to `TRUE`, the levels of the variables named in the character vector will all be sorted; if `sortLevels` is `TRUE` and `factorInfo` is empty or `NULL`, all factors will be sorted. If a `factorInfo` list is provided, each variable information list contains one or more of the named elements given below.

- Currently available properties for a column information list are:
- `levels` - optional vector, containing values to match in converting non-factor data to factor levels. If `levels = NULL`, all of the unique values in the data are converted to `levels` in the order encountered. However, the user can override this behavior and sort the resulting `levels` alphabetically by setting `sortLevels = TRUE`. The user may also specify a subset of the data to convert to `levels`. In this case, if `otherLevel = NULL`, all data values *not* found in the `levels` subset will be converted to missing (`NA`) values. For example, if a variable `x` is comprised of integer data `1, 2, 3, 4, 5`, then
- `` - `factorInfo = list(x = list(levels = 2:4, otherLevel = NULL))` will convert `x` into a factor with data `NA, "2", "3", "4", NA` with levels `"2", "3", "4"`. Alternatively, the user may wish to place all of those unspecified values into a single category, say `"other"`. In that case, use `otherLevel = "other"` along with the subset `levels` specification. Note that the `levels` vector may be any type, e.g., 'integer', 'numeric', 'character'. However, behind the scenes, it is always converted to type 'character', as are the data values being converted. The resulting strings are matched with those of the data to populate the categories.
- `otherLevel` - character string defining the level to assign to all factor values that are not listed in the `newLevels` field, if `newLevels` is specified. If `otherLevel = NULL`, the default, the factor levels that are not listed in `newLevels` will be left unchanged and in their original order. If specified, the value set here overrides the default argument of the same in the primary argument list.

- `sortLevels` - logical scalar. If `TRUE`, the resulting levels will be sorted alphabetically. If the input variable is not a factor and levels are not specified, this will be ignored and levels will be in the order in which they are encountered.
- `varName` - character string defining the name of an existing data variable to recode. If this field is left unspecified, then the name of the corresponding list element in `factorInfo` will be used. For example, all of the following are acceptable and equivalent `factorInfo` specifications for alphabetically sorting the levels of an existing factor variable named `"myFacVar"`:
 - `` - `factorInfo = list(myFacVar = list(sortLevels = TRUE))`
 - `` - `factorInfo = list(list(sortLevels = TRUE, varName = "myFacVar"))`
 - `` - `factorInfo = list(myFacVar = list(sortLevels = TRUE, varName = "myFacVar"))`
 However, if you wish to rename a variable after conversion (keeping the old variable in tact), there is only one acceptable format: the variable to be recoded must appear in the `varName` field while the new variable name for the converted data must appear as the name of the corresponding list element. For example, to sort the levels of an existing factor variable `"myFacVar"` and store the result in a new variable `"myNewVar"`, you would issue:
- `` - `factorInfo = list(myNewFacVar = list(sortLevels = TRUE, varName = "myFacVar"))`
- `newLevels` - a character vector or list, possibly with named elements, used to rename the levels of a factor. While `levels` provides a means of filtering the data the user wishes to import and convert to factor levels, `newLevels` is used to alter converted or existing levels by renaming, collapsing, or sorting them. See the Examples section below for typical use cases of `newLevels`.
- `description` - character string defining a description for the recoded variable.

`sortLevels`

the default value to use for the `sortLevels` field in the `factorInfo` list.

`otherLevel`

the default value to use for the `otherLevel` field in the `factorInfo` list.

`outFile`

either an RxXdfData object, a character string specifying the .xdf file, or `NULL`. If `outFile = NULL`, a data frame is returned. When writing to HDFS, `outFile` must be an `RxXdfData` object representing a new composite XDF.

`varsToKeep`

character vector of variable names to include in the data file. If `NULL`, argument is ignored. Cannot be used with `varsToDelete`.

`varsToDelete`

character vector of variable names to not include in the data file. If `NULL`, argument is ignored. Cannot be used with `varsToKeep`.

`overwrite`

logical value. If `TRUE`, an existing `outFile` will be overwritten. Ignored if a dataframe is returned.

`maxRowsByCols`

this argument is used only when `inData` is referring to an .xdf file (character string defining a path to an existing .xdf file or an RxXdfData object) and we wish to return the output as a data frame (`outFile = NULL`). In this case, and behind the scenes, the output is written to a temporary .xdf file and `rxDataStep` is subsequently

called to convert the output into a data frame. The `maxRowsByCols` argument is passed directly in the `rxDataStep` call, giving the user some control over the conversion. See [rxDataStep](#) for more details on the `maxRowsByCols` argument.

blocksPerRead

number of blocks to read for each chunk of data read from the data source. If the `data` and `outFile` are the same file, `blocksPerRead` must be 1.

reportProgress

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

verbose

integer value. If `0`, no additional output is printed. If `1`, additional summary information is printed.

xdfCompressionLevel

integer in the range of -1 to 9. The higher the value, the greater the amount of compression - resulting in smaller files but a longer time to create them. If `xdfCompressionLevel` is set to 0, there will be no compression and files will be compatible with the 6.0 release of Revolution R Enterprise. If set to -1, a default level of compression will be used.

...

additional arguments to be passed directly to the Microsoft R Services Compute Engine.

Details

Factors are variables that represent categories. An example is a variable named `"state"` whose values are the levels `"Alabama"`, `"Alaska"`, ..., `"Wyoming"`. There are two parts to a factor variable:

1 a vector of N (number of observations) integer indexes with values in the range of `1:k`, where K is the number of categories.

1 a vector of K strings (characters) that are used when the vector is displayed and in some other situations.

For instance, when state levels are alphabetical, all observations for which `state == "Alabama"` will have the index `1`, `state == "Washington"` values correspond to index `47`, and so on.

Recoding a factor means changing from one set of indices to another. For instance, if the levels for `"state"` are currently arranged in the order in which they were encountered when importing a .csv file, and it is desired to put them in alphabetical order, then it is necessary to change the index for every observation.

If numeric data is converted to a factor, a maximum precision of 6 is used. So, for example, the values 7.123456 and 7.12346 would be placed in the same category.

To recode a categorical or factor variable into a continuous variable within a formula use `N()`. To recode continuous variable to a categorical or factor variable within a formula use `F()`. See [rxFormula](#).

To rename the levels of a factor variable in an .xdf file (without change the levels themselves), use [rxSetVarInfoXdf](#).

Value

if `outFile` is `NULL`, then a data frame is returned. Otherwise, the results are written to the specified `outFile` file and an RxXdfData object is returned *invisibly* corresponding to the output file.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxFormula](#), [rxSetVarInfoXdf](#), [rxImport](#), [rxDataStep](#).

Examples

```
###  
# Example 1: Recoding levels in alphabetical order  
###  
  
# Use the 'warpbreaks' data frame found in the 'datasets' package  
# Note that the 'tension' factor variable has levels that are not  
# alphabetically ordered.  
rxGetVarInfo( warpbreaks )  
  
# Reorder all factor levels that are not in alphabetical order  
recodedDF1 <- rxFactors(inData = warpbreaks, sortLevels = TRUE)  
rxGetVarInfo( recodedDF1 )  
  
# Specify that only 'tension' levels should be reordered alphabetically  
recodedDF2 <- rxFactors(inData = warpbreaks, sortLevels = TRUE,  
    factorInfo = c("tension"))  
rxGetVarInfo( recodedDF2 )  
  
# Specify that only 'tension' levels should be reordered alphabetically using a list  
recodedDF3 <- rxFactors(inData = warpbreaks,  
    factorInfo = list(tension = list(sortLevels = TRUE)))  
rxGetVarInfo( recodedDF3 )  
  
# write data frame to .xdf file and perform similar recoding  
# but write the recoded factor to a new variable. Compare the  
# original with the recoded factor.  
inXDF <- file.path(tempdir(), "warpbreaks.xdf")  
outXDF <- file.path(tempdir(), "warpbreaksRecoded.xdf")  
rxDataStep(warpbreaks, outFile = inXDF, overwrite = TRUE)  
outDS <- rxFactors(inData = inXDF, outFile = outXDF, overwrite = TRUE,  
    factorInfo = list(recodedTension = list(sortLevels = TRUE,  
        varName = "tension")))  
DF <- rxDataStep(outDS)  
rxGetVarInfo( DF )  
  
# clean up  
if (file.exists(inXDF)) unlink(inXDF)  
if (file.exists(outXDF)) unlink(outXDF)  
  
###  
# Example 2: Recoding levels and indexes, saving recoding to a new factor variable  
###  
  
# Create an .xdf file with a factor variable named 'sex' with levels 'M' and 'F'  
set.seed(100)  
sex <- factor(sample(c("M","F"), size = 10, replace = TRUE), levels = c("M", "F"))  
DF <- data.frame(sex = sex, score = rnorm(10))  
DF[["sex"]]  
XDF <- file.path(tempdir(), "sex.xdf")  
XDF2 <- file.path(tempdir(), "newSex.xdf")  
rxDataStep(DF, outFile = XDF, overwrite = TRUE)  
rxDataStep(DF, outFile = XDF2, overwrite = TRUE)
```

```

rxDataStep(DF, outFile = XDF, overwrite = TRUE)

# Assume that we change our minds and now wish to
# rename the levels to "Female" and "Male"
# Let us do the recoding and store the result into a new
# variable named "Gender" keeping the old variable in place.
outDS <- rxFactors(inData = XDF, outFile = XDF2, overwrite = TRUE,
                     factorInfo = list(Gender = list(newLevels = c(Female = "F", Male = "M"),
                                                     varName = "sex")))
newDF <- rxDataStep(outDS)
print(newDF)

# clean up
if (file.exists(XDF)) unlink(XDF)
if (file.exists(XDF2)) unlink(XDF2)

####
# Example 3: Combining subsets of factor levels into single levels
####

# Create a data set that contains a factor variable 'Month'
# Note that the levels are not in alphabetical order.
set.seed(100)
DF <- data.frame(Month = factor(sample(month.name, size = 20, replace = TRUE),
                                 levels = rev(month.name)))

# Recode the months into quarters and store result into new variable named "Quarter"
recodedDF <- rxFactors(inData = DF,
                       factorInfo = list(Quarter = list(newLevels = list(Q1 = month.name[1:3],
                                                                     Q2 = month.name[4:6],
                                                                     Q3 = month.name[7:9],
                                                                     Q4 = month.name[10:12]),
                                         varName = "Month"))))

head(recodedDF)
recodedDF$Quarter

####
# Example 4: Coding and recoding combinations using a single factorInfo list
####

set.seed(100)
size <- 10
months <- factor(sample(month.name, size = size, replace = TRUE), levels = rev(month.name))
states <- factor(sample(state.name, size = size, replace = TRUE), levels = state.name)
animalFarm <- c("cow", "horse", "pig", "goat", "chicken", "dog", "cat")
animals <- factor(sample(animalFarm, size = size, replace = TRUE), levels = animalFarm)
values <- sample.int(100, size = size, replace = TRUE)
dblValues <- c(1, 2.1, 3.12, 4.123, 5.1234, 6.12345, 7.123456, 7.12346, 81234.56789, 91234567.8)
DF <- data.frame(Month = months, State = states, Animal = animals, VarInt = values,
                  VarDbl = dblValues, NotUsed1 = seq(size), NotUsed2 = rev(seq(size)))

factorInfo <- list()

# Convert months to quarters
Quarter = list(newLevels = list(Q1 = month.name[1:3], Q2 = month.name[4:6],
                                 Q3 = month.name[7:9], Q4 = month.name[10:12]),
               varName = "Month"),

# Sort animal levels
Animal = list(sortLevels = TRUE),

# Convert integer data to factor and do not sort levels
VarIntFac = list(varName = "VarInt", sortLevels = FALSE),

# Convert double data to factor; it will use a precision up to 6
VarDblFac = list(varName = "VarDbl"),

# In-place arbitrary grouping of state names using indexMap

```

```

StateSide = list(newLevels = c(LeftState = "1", RightState = "2"),
                indexMap = c(rep(1, 25), rep(2, 25)),
                varName = "State")
)

rxFactors(DF, factorInfo)

###  

# Example 5: Using 'newLevels' to rename, reorder, or collapse existing factor levels.  

#           All of these examples make use of the iris data set, which contains levels  

#           "setosa", "versicolor", and "virginica", in that order.  

###  

###  

# Renaming factor levels:  

#  

#   "setosa" to "Seto"  

#   "versicolor" to "Vers"  

#   "virginica" to "Virg"  

newLevels <- list(Seto = "setosa", Vers = "versicolor", Virg = "virginica")  

rxFactors(iris, factorInfo = list(Species = list(newLevels = newLevels)))$Species  

# Reordering:  

newLevels <- c("versicolor", "setosa", "virginica")  

rxFactors(iris, factorInfo = list(Species = list(newLevels = newLevels)))$Species  

# Collapsing: order does matter here, so the resulting order of the levels will  

# be "V" then "S". The 'sortLevels' argument is a quick means of alphabetically  

# sorting the resultant level names.  

newLevels <- list(V = "setosa", S = c("versicolor", "virginica"))  

rxFactors(iris, factorInfo = list(Species = list(newLevels = newLevels)))$Species  

rxFactors(iris, factorInfo = list(Species = list(newLevels = newLevels, sortLevels = TRUE)))$Species  

# Subset collapsing with renaming: accomplish with the use of 'otherLevel'  

newLevels <- list(S = "setosa")  

rxFactors(iris, factorInfo = list(Species = list(newLevels = newLevels, otherLevel =
"otherSpecies")))$Species  

# Superset specification: adding new species for a future study  

newLevels <- c("setosa", "versicolor", "virginica", "pumila", "narbuti", "camillae")  

rxFactors(iris, factorInfo = list(Species = list(newLevels = newLevels)))$Species

```

RxFileData-class: Class RxFileData

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

File-based data source connection class.

Extends

Class RxDataSource, directly.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxDataSource-class](#), [rxNewDataSource](#)

RxFileSystem: RevoScaleR File System object generator

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

This is the main generator for RxFileSystem S3 classes.

Usage

```
RxFileSystem( fileSystem, ...)

## S3 method for class `RxFileSystem':
print ( x, ... )
```

Arguments

`fileSystem`

character string specifying class name or file system type existing `RxFileSystem` object. Choices include: "RxNativeFileSystem" or "native", or "RxHdfsFileSystem" or "hdfs". Optional arguments `hostName` and `port` may be specified for HDFS file systems.

`x`

an RxFileSystem object.

`...`

other arguments are passed to the underlying function.

Details

This is a wrapper to specific generator functions for the RevoScaleR file system classes. For example, the RxHdfsFileSystem class uses function `RxHdfsFileSystem` as a generator. Therefore either `RxHdfsFileSystem()` or `RxFileSystem("hdfs")` will create an RxHdfsFileSystem object.

Value

A type of RxFileSystem file system object. This object may be used in `rxSetFileSystem`, `rxOptions`, `RxTextData`, or `RxXdfData` to set the file system.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxNativeFileSystem](#), [RxHdfsFileSystem](#), [rxSetFileSystem](#), [rxOptions](#), [RxXdfData](#), [RxTextData](#).

Examples

```
# Setup to run analyses to use HDFS file system
## Not run:

# Example 1
myHdfsFileSystem1 <- RxFileSystem(fileSystem = "hdfs")
rxSetFileSystem(fileSystem = myHdfsFileSystem1 )

# Example 2
myHdfsFileSystem2 <- RxFileSystem(fileSystem = "hdfs", hostName = "myHost", port = 8020)
rxSetFileSystem(fileSystem = myHdfsFileSystem2 )
## End(Not run)
```

rxFindFileInPath: Finds where in a given path a file is.

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Sequentially checks the entries in a delimited path string for a provided file name.

Usage

```
rxFindFileInPath( path, fileName )
```

Arguments

path

character vector, compute context or job object. This is a required parameter. If a compute context is provided, the `dataPath` from that context (assuming it has one) will be used; if a job object is provided, that job object's compute context's `dataPath` will be used. If a character vector is provided, each element of the vector should contain one directory path. If a character scalar is supplied, multiple directory paths can be contained by using the standard system delimiters (":" for Linux and ";") to separate the entries within the path. This allows system PATH's to be parsed using this function.

fileName

logical scalar. This is a required parameter. The name of the file being sought along the `path`.

Details

This function will sequentially check the locations (directories) provided in the `path`. Thus, if there exists more than one instance of a file with the same `fileName`, the first instance of a directory in the path containing the file will be the one returned.

Value

Returns NULL if the target file is not found in any of the possible locations, or the path to the file (not including the file name) if it is found.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

Examples

```
## Not run:

# using a Linux environment PATH
rxFindFileInPath(path=Sys.getenv("PATH"), fileName="Revoscript")

# using a compute context
rxFindFileInPath(path=myComputeContext, fileName="myData" )

# for Windows
rxFindFileInPath(path="C:\data\remember\to\escape\backslashes;D:\other\data", fileName="myData" )

# for Linux
rxFindFileInPath(path="/mnt/data:/home/myName/data", fileName="myData" )

# using a vector of paths
rxFindFileInPath( path=c("/dir1","/dir2",/dir3/dir4"), fileName="myData" )
## End(Not run)
```

rxFindPackage: Find Packages for Compute Context

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Find the path for one or more packages for a compute context.

Usage

```
rxFindPackage(package, computeContext = NULL, allNodes = FALSE, lib.loc = NULL, quiet = TRUE,  
verbose =getOption("verbose"))
```

Arguments

`package`

character vector of name(s) of package(es).

`computeContext`

an `RxComputeContext` or equivalent character string or `NULL`. If set to the default of `NULL`, the currently active compute context is used. Supported compute contexts are `RxInSqlServer` and `RxLocalSeq`.

`allNodes`

logical

`lib.loc`

a character vector describing the location of R library trees to search through, or `NULL`. The default value of `NULL` corresponds to checking the loaded namespace, then all libraries currently known in `.libPaths()`. In `RxInSqlServer` only `NULL` is supported.

`quiet`

logical. If `FALSE`, warnings or an error is given if the package is not found.

`verbose`

logical. If `TRUE`, additional diagnostics are printed if available.

Details

This is a wrapper for `find.package`. See the help file for additional details.

Value

A character vector of paths of package directories. If using a distributed compute context with the `allNodes` set to `TRUE`, a list of lists with a character vector of paths from each node will be returned.

Author(s)

See Also

[rxPackage](#), [find.package](#), [rxInstalledPackages](#), [rxInstallPackages](#),
[rxRemovePackages](#), [rxSyncPackages](#), [rxSqlLibPaths](#),
require

Examples

```
#  
# Find the paths for the RevoScaleR and lattice packages  
#  
packagePaths <- rxFindPackage(package = c("RevoScaleR", "lattice"))  
packagePaths  
  
## Not run:  
  
#  
# Find the path of the RevoScaleR package on a SQL Server compute context  
#  
sqlServerCompute <- RxInSqlServer(connectionString =  
    "Driver=SQL Server;Server=myServer;Database=TestDB;Uid=myID;Pwd=myPwd;")  
sqlPackagePaths <- rxFindPackage(package = "RevoScaleR", computeContext = sqlServerCompute)  
## End(Not run)
```

RxForeachDoPar-class: Class RxForeachDoPar

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Class for the RevoScaleR Compute Context using one of the foreach dopar back ends.

Generators

The targeted generator [RxForeachDoPar](#) as well as the general generator [RxComputeContext](#).

Extends

Class RxComputeContext, directly.

Methods

`show`

`signature(object = "RxForeachDoPar") : ...`

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxComputeContext](#), [RxLocalSeq](#), [RxLocalParallel](#), [RxSpark](#), [RxHadoopMR](#).

Examples

```
## Not run:

myComputeContext <- RxComputeContext("RxForeachDoPar")
is(myComputeContext, "RxComputeContext")
# [1] TRUE
is(myComputeContext, "RxForeachDoPar")
# [1] TRUE
## End(Not run)
```

RxForeachDoPar: Generate RxForeachDoPar Compute Context

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Creates a compute context object using the registered `foreach` parallel back end. This compute context can be used only to distribute computations via the `rxExec` function; it is ignored by Revolution HPA functions. This is the main generator for S4 class `RxForeachDoPar`.

Usage

```
RxForeachDoPar( object, dataPath = NULL, outDataPath = NULL )
```

Arguments

object

a compute context object. If `object` has slots for `dataPath` and/or `outDataPath`, they will be copied to the equivalent slots for the new `RxForeachDoPar` object. Explicit specifications of the `dataPath` and/or `outDataPath` arguments will override this.

dataPath

`NULL` or character vector defining the search path(s) for the input data source(s). If not `NULL`, it overrides any specification for `dataPath` in `rxOptions`

outDataPath

`NULL` or character vector defining the search path(s) for new output data file(s). If not `NULL`, this overrides any specification for `dataPath` in `rxOptions`

Details

A job is associated with the compute context in effect at the time the job was submitted. If the compute context subsequently changes, the compute context of the job is not affected.

Available parallel backends are system-dependent, but include `doRSR` and `doParallel`. These are separate packages that must be loaded and registered to accept `foreach` input.

Note that `dataPath` and `outDataPath` are only used by data sources used in **RevoScaleR** analyses. They do not alter the working directory for other R functions that read from or write to files.

Value

object of class `RxForeachDoPar`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

foreach, doParallel-package, registerDoParallel, doRSR-package, registerDoRSR, rxSetComputeContext, rxOptions, rxExec, RxComputeContext, RxLocalSeq, RxLocalParallel, RxForeachDoPar-class.

Examples

```
## Not run:

# Create a compute context using your registered foreach backend
doparContext <- RxForeachDoPar()

# Tell RevoScaleR to use the RxForeachDoPar compute context
rxSetComputeContext(doparContext)

## End(Not run)
```

rxFormula: Formula Syntax for RevoScaleR Analysis Functions

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Highlights of the similarities and differences in formulas between **RevoScaleR** and standard R functions.

Details

The formula syntax used by the **RevoScaleR** analysis functions is similar, but not identical, to regular R formula syntax. The most important differences are:

- * With the exception of `rxSummary`, dot (.) explanatory variable expansion is not supported.
- * Multiple column producing in-line variable transformations, e.g. `poly` and `bs`, are not supported.
- * The original order of the explanatory variables are maintained, i.e. the main effects are not forced to precede the interaction terms. (See `keep.order = TRUE` setting in `terms.formula` for more information.)

A formula typically consists of a *response*, which in most **RevoScaleR** functions can be a single variable or multiple variables combined using `cbind`, the `"~"` operator, and one or more *predictors*, typically separated by the `"+"` operator. The `rxSummary` function typically requires a formula with no response.

Interactions are indicated using the ":" operator. The interaction of two categorical variables results in a categorical variable containing the full set of combinations of the two categories and adds a coefficient to the model for each category. The interaction of two continuous variables is the same as the multiplication of the two variables. The interaction of a continuous and a categorical variable adds a coefficient for the continuous variable for each level of the categorical variable. The asterisk operator `"*"` between categorical variables adds all subsets of interactions of the variables to the model.

In **RevoScaleR**, predictors must be single-column variables.

RevoScaleR formulas support two formula functions for managing categorical variables:

- `F(x, low, high, exclude)` creates a categorical variable out of continuous variable `x`. Additional arguments `low`, `high`, and `exclude` can be included to specify the value of the lowest category, the highest category, and how to handle values outside the specified range. For each integer in the range from `low` to `high` inclusive, **RevoScaleR** creates a level and assigns values greater than or equal to an integer `n`, but less than `n+1`, to `n`'s level. If `x` is already a factor, `F(x, low, high, exclude)` can be used to limit the range of levels used; in this case `low` and `high` represent the indexes of the factor levels, and must be integers in the range from 1 to the number of levels.
- `N(x)` creates a continuous variable from categorical variable `x`. Note, however, that the value of this function is equivalent to the factor codes, and has no relation to any numeric values within the levels of the function. For this, use the construction `as.numeric(levels(x))[x]`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxTransform](#), [rxCrossTabs](#), [rxCube](#), [rxLinMod](#), [rxLogit](#), [rxSummary](#).

Examples

```
# These two lines are set up for the examples
sampleDataDir <- rxGetOption("sampleDataDir")
censusWorkers <- file.path(sampleDataDir, "CensusWorkers.xdf")

rxSummary(~ F(age) + sex, data = censusWorkers)
rxSummary(~ F(age, low = 30, high = 45, exclude = FALSE), data = censusWorkers)

rxCube(incwage ~ F(age) : sex, data = censusWorkers)
```

rxGetAvailableNodes: Gets a list of operational nodes on a cluster.

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Gets a list of operational nodes on a cluster. Note that this function will attempt to connect to the cluster when executed.

Usage

```
rxGetAvailableNodes(computeContext, makeRNodeNames = FALSE)
```

Arguments

`computeContext`

A distributed compute context (preferred, see [RxComputeContext](#)) or a `jobInfo` object

`makeRNodeNames`

logical. If `TRUE`, names of the nodes will be normalized for use as R variables. See [rxMakeRNodeNames](#) for details on name mangling.

Value

a character vector of node names, or `NULL`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxgetNodeInfo](#) [RxComputeContext](#) [rxGetJobs](#) [rxMakeRNodeNames](#)

Examples

```
## Not run:  
  
rxGetAvailableNodes( myCluster )  
## End(Not run)
```

rxGetEnableThreadPool: Get or Set Thread Pool State

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Gets or sets the current state of the thread pool (in a ready state or created ad hoc).

Usage

```
rxGetEnableThreadPool()  
rxSetEnableThreadPool( enable )
```

Arguments

`enable`

Logical scalar. If `TRUE`, the thread pool is instantiated and maintained in a ready state. If `FALSE`, threads are created in an ad hoc fashion; that is, they are created as needed.

Details

The `rxSetEnableThreadPool` function is used on Linux to turn the thread pool on and off. When the thread pool is on, it means that there exists a pool of threads that persist between calls, and are ready for processing.

When the thread pool is off, it means that threads will be created on an ad hoc basis when calls requiring threading are made. By default, the thread pool is turned off to allow R processes that have loaded RevoScaleR to fork successfully as a mechanism for spawning (for example, by the `multicore` package).

On the Windows platform, the thread pool always persists, regardless of any user settings.

There is a significant speed increase associated with having the thread pool ready and waiting to do work. If you are sure that you will not be using any spawning mechanism that uses fork, you may want to put the following at the end of your `Rprofile.site`. Make absolutely sure that it is after any lines that are used to load RevoScaleR:

```
invisible( rxSetEnableThreadPool( TRUE ) )
```

The following are possible cases where fork may be called. Obviously, this is not an all-inclusive list:

* `nohup` to launch jobs. This will cause a fork to be called.

* Using `multicore` and/or `doMC` to launch R processes with RevoScaleR

Note that when using `rxExec`, the default behavior for any worker node process on a Linux host will be to have the thread pool off (set to create threads in an ad hoc manner). If the function passed to `rxExec` is going to make multiple calls into RevoScaleR functions, you will probably want to include a call to `rxSetEnableThreadPool(TRUE)` as the first line of the function that you pass to `rxExec`.

For distributed HPA functions run on worker nodes, threading is always automatically handled for the user.

The `rxGetEnableThreadPool` function can be used to determine whether the thread pool is instantiated or not.

Note that on Windows, it will always be instantiated; on Linux, whether it is always instantiated or created on an

ad hoc basis is controlled by `rxSetEnableThreadPool`. At startup on Linux, the thread pool state will be off; that is, threads will be created on an ad hoc basis.

Value

`rxSetEnableThreadPool` returns the state of the thread pool prior to making the call (as opposed to the updated state). Thus, this function returns `TRUE` if the thread pool was instantiated and in a ready state prior to making the call, or `FALSE` if the thread pool was set to be created in an ad hoc fashion.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxOptions](#)

Examples

```
## Not run:  
  
rxGetEnableThreadPool()  
rxSetEnableThreadPool(TRUE)  
rxGetEnableThreadPool()  
## End(Not run)
```

rxGetFuzzyDist: Fuzzy distances for a character vector

7/12/2022 • 6 minutes to read • [Edit Online](#)

Description

EXPERIMENTAL: Get fuzzy distances for a character vector

Usage

```
rxGetFuzzyDist(stringsIn, data = NULL, outFile = NULL, varsToKeep = NULL,
  matchMethod = c("lv", "j", "jw", "exact", "none"),
  keyType = c("all", "alphanum", "alpha", "mphone3", "soundex", "soundexNum",
    "soundexAll", "soundexAm", "mphone3Vowels", "mphone3Exact"),
  ignoreCase = FALSE, ignoreSpaces = NULL, ignoreNumbers = TRUE, ignoreWords = NULL,
  minDistance = 0.7, dictionary = NULL, noMatchNA = FALSE,
  returnString = TRUE, returnIndex = NULL, overwrite = FALSE
)
```

Arguments

`stringsIn`

Character vector of strings to process or name of character variable.

`data`

`NULL` or data frame or RevoScaleR data source containing the variable to process.

`outFile`

`NULL` or RevoScaleR data source in which to put output.

`varsToKeep`

`NULL` or character vector of variables from the input 'data' to keep in the output data set. They will be replicated for multiple matches.

`matchMethod`

Method for matching to dictionary: 'none' for no matching, 'lv' for Levenshtein; 'j' for Jaro, 'jw' for JaroWinkler, 'exact' for exact matching

`keyType`

Transformation type in creating keys to be matched: 'all' to retain all characters, 'alphanum' for alphanumeric characters only, 'alpha' for letters only", 'mphone3' for Metaphone3 phonetic transformation, 'soundex' for Soundex phonetic transformation, 'mphone3Vowels' for Metaphone3 encoding more than initial vowels, 'mphone3Exact' for Metaphone3 with more exact consonants, 'soundexNum' for Soundex with numbers only, 'soundexAll' for Soundex not truncated at 4 characters, and 'soundexAm' for the American variant of Soundex.

`ignoreCase`

A logical specifying whether or not to ignore case when comparing strings to the dictionary

`ignoreSpaces`

A logical specifying whether or not to ignore spaces. If `FALSE`, for phonetic conversions each word in the string is processed separately and then concatenated together.

`ignoreNumbers`

A logical. If `FALSE`, numbers are converted to words before phonetic processing. If `TRUE`, numbers are ignored or removed

`ignoreWords`

An optional character vector containing words to ignore when matching.

`noMatchNA`

A logical. If `TRUE`, if a match is not found an empty string is returned. Only applies when dictionary is provided.

`minDistance`

Minimum distance required for a match; applies only to distance metric algorithms (Levenshtein, Jaro, JaroWinkler). One is a perfect match. Zero is no similarity.

`dictionary`

Character vector containing the dictionary for string comparisons. from strings before processing.

`returnString`

A logical specifying whether to return the string and dictionary values

`returnIndex`

`NULL` or logical specifying whether to return the string and dictionary indexes. If `NULL`, it is set to `!returnString`.

`overwrite`

A logical. If `TRUE` and the specified `outFile` exists, it will be overwritten.

Details

Four similarity distance measures are provided: Levenshtein, Jaro, Jaro-Winkler, and Bag of Words.

The Levenshtein edit-distance algorithm computes the least number of edit operations (number of insertions, deletions, and substitutions) that are necessary to modify one string to obtain another string.

The Jaro distance measure considers the number of matching characters and transpositions. Jaro-Winkler adds a prefix-weighting, giving higher match values to strings where prefixes match.

The Bag of Words measure looks at the number of matching words in a phrase, independent of order. In the implementation used in `rxGetFuzzyDist`, all measures are normalized to the same scale, where 0 is no match and 1 is a perfect match.

For information on phonetic conversions, see [rxGetFuzzyKeys](#).

Value

A data frame or data source containing the distances and either string and dictionary values or indexes.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxGetFuzzyKeys](#), [rxDataStep](#)

Examples

```
# Basic examples
myWords <- c("Allen", "Ellen", "Elena")
myDictionary <- c("Allen", "Helen", "Harry")
rxGetFuzzyDist(stringsIn = myWords, dictionary = myDictionary, minDistance = .5, returnString = FALSE)

rxGetFuzzyDist(stringsIn = c("Univ of Ontario"),
               dictionary = c("Univ of Michigan", "Univ of Ontario", "Univ Michigan",
                             "Michigan State", "Ontario Univ of"),
               matchMethod = "bag", ignoreWords = c("of"), minDistance = 0)

rxGetFuzzyDist(stringsIn = c("Univ of Ontario"),
               dictionary = c("Univ of Michigan", "Univ of Ontario", "Univ Michigan",
                             "Michigan State", "Ontario Univ of"),
               matchMethod = "bag", keyType = "soundex", ignoreSpaces = FALSE, minDistance = 0)

### The Jaro Measure
# To use the Jaro edit-distance, set the `matchType` to `j`.
# Create a small city dictionary
cityDictionary <- c("Portland", "Eugene", "Corvalis")

# Create a vector of city names to correct
cityNames <- c("Portland", "Portlande", "Eugen", "CORVALIS", "Corval")

# Large minimum distance should result in little change: final e's are dropped
cityKey1 <- rxGetFuzzyKeys(stringsIn = cityNames, matchMethod = "j", dictionary = cityDictionary,
                           minDistance = .9)
# Print results to console
cityKey1

# Smaller minimum distance should fix more
cityKey2 <- rxGetFuzzyKeys(stringsIn = cityNames, matchMethod = "j", dictionary = cityDictionary,
                           minDistance = .8)
# Print results to console
cityKey2

# Small minimum distance should fix everything
cityKey3 <- rxGetFuzzyKeys(stringsIn = cityNames, matchMethod = "j", dictionary = cityDictionary,
                           minDistance = .1)
# Print results to console
cityKey3

# Examine all of the distances greater than .1
distOut1 <- rxGetFuzzyDist(stringsIn = cityNames, matchMethod = "j", dictionary = cityDictionary,
                           minDistance = .1)
distOut1

# Use the ignoreCase argument: CORVALIS and Corvalis will now be a perfect match
distOut2 <- rxGetFuzzyDist(stringsIn = cityNames, matchMethod = "j", dictionary = cityDictionary,
                           ignoreCase = TRUE, minDistance = .1)
distOut2

### The Jaro-Winkler matchMethod
# Create a small city dictionary
cityDictionary <- c("Portland", "Eugene", "Corvalis")
# Create a vector of city names to correct
cityNames <- c("Portland", "Portlandia", "Eugenie", "Eugener", "Corvaliser", "Corving")
# Examine all of the distances greater than .5 for Jaro-Winkler
distOut1 <- rxGetFuzzyDist(stringsIn = cityNames, matchMethod = "jw", dictionary = cityDictionary,
                           minDistance = .5)
```

```

distOut1

# Compare with Jaro, which does not emphasize the matching prefix
distOut2 <- rxGetFuzzyDist(stringsIn = cityNames, matchMethod = "j", dictionary = cityDictionary,
minDistance = .5)
distOut2

### The Bag-of-Words matchMethod

institutionDictionary = c("University of Maryland", "Michigan State University", "University of Michigan",
"University of Washington", "Washington State University" )

origCol1 <- c("Univ of Maryland", "Michigan State", "Michigan State University", "Michigan University of",
"WASHINGTON UNIVERSITY OF", "Washington State Univ")

distOut <- rxGetFuzzyDist(stringsIn = origCol1,
  dictionary = institutionDictionary, matchMethod = "bag", minDistance = .2,
  ignoreCase = TRUE )
distOut

distOut <- rxGetFuzzyDist(stringsIn = origCol1,
  dictionary = institutionDictionary, matchMethod = "bag", minDistance = .2,
  ignoreWords = c("University", "Univ"), ignoreCase = TRUE )
distOut

### Using Distance Measures with Phonetic Transformations
# Distance measures can also be used with phonetic transformations of strings. For example,
# we can first convert the strings to Soundex, then perform the distance measure. In both
# cases shown, we will see a perfect match between `Univ of Ontario` and `Unive of Ontario`
# when we convert all of the words to Soundex.

uDictionary = c("Univ of Michigan", "Univ of Ontario", "Univ of Oklahoma", "Michigan State")

# Use a bag of words distance measure with a soundex conversion
outDist1 <- rxGetFuzzyDist(stringsIn = c("Unive of Ontario"),
  dictionary = uDictionary,
  matchMethod = "bag", keyType = "soundex", ignoreSpaces = FALSE, minDistance = 0)
outDist1

# Now use a Levenshtein distance measure with a soundex conversion
outDist2 <- rxGetFuzzyDist(stringsIn = c("Unive of Ontario"),
  dictionary = uDictionary,
  matchMethod = "lv", keyType = "soundex", ignoreSpaces = FALSE, minDistance = 0)
outDist2

## Use with xdf file
# Create temporary xdf file
inData <- data.frame(institution = c("Seattle Pacific U", "SEATTLE UNIV", "Seattle Central",
"U Washngtn", "UNIV WASH BOTHELL", "Puget Sound Univ",
"ANTIOC U SEATTLE", "North Seattle Univ", "North Seatle U",
"Seattle Inst Tech", "SeATLE College Nrth", "UNIV waSHINGTON",
"University Seattle", "Seattle Antioch U",
"Technology Institute Seattle", "Pgt Sound U",
"Seattle Central Univ", "Univ North Seattle",
"Bothell - Univ of Wash", "ANTIOCH SEATTLE"), stringsAsFactors = FALSE)

tempInFile <- tempfile(pattern = "fuzzyDistIn", fileext = ".xdf")
rxDataStep(inData = inData, outFile = tempInFile, rowsPerRead = 10)

uDictionary <- c("Seattle Pacific University", "Seattle University",
"University of Washington", "Seattle Central College",
"University of Washington, Bothell", "Puget Sound University",
"Antioch University, Seattle", "North Seattle College",
"Seattle Institute of Technology")

tempOutFile <- tempfile(pattern = "fuzzyDistOut", fileext = ".xdf")

outDataSource <- rxGetFuzzyDist(stringsIn = "institution",
  data = tempInFile, outFile = tempOutFile,

```

```
dictionary = uDictionary,
ignoreWords = c("University", "Univ", "U","of"),
ignoreCase = TRUE,
matchMethod = "bag",
ignoreSpaces = FALSE,
minDistance = .5,
keyType = "mphone3", overwrite = TRUE)
rxGetVarInfo(outDataSource)
# Read the new data set back into memory
newData <- rxDataStep(outDataSource)
oldWidth <- options(width = 120)
newData
options(oldWidth)
# Clean-up
file.remove(tempInFile)
file.remove(tempOutFile)
```

rxGetFuzzyKeys: Fuzzy keys for a character vector

7/12/2022 • 8 minutes to read • [Edit Online](#)

Description

EXPERIMENTAL: Get fuzzy keys for a character vector

Usage

```
rxGetFuzzyKeys(stringsIn, data = NULL, outFile = NULL, varsToKeep = NULL,
  matchMethod = c("lv", "j", "jw", "bag", "exact", "none"),
  keyType = c("all", "alphanum", "alpha", "mphone3", "soundex", "soundexNum",
    "soundexAll", "soundexAm", "mphone3Vowels", "mphone3Exact"),
  ignoreCase = FALSE, ignoreSpaces = NULL, ignoreNumbers = TRUE, ignoreWords = NULL,
  minDistance = 0.7, dictionary = NULL, keyVarName = NULL,
  costs = c(insert = 1, delete = 1, subst = 1),
  hasMatchVarName = NULL, matchDistVarName = NULL, numMatchVarName = NULL, noMatchNA = FALSE,
  overwrite = FALSE
)
```

Arguments

`stringsIn`

Character vector of strings to process or name of character variable.

`data`

`NULL` or data frame or RevoScaleR data source containing the variable to process.

`outFile`

`NULL` or RevoScaleR data source in which to put output.

`varsToKeep`

`NULL` or character vector of variables from the input 'data' to keep in the output data set. If `NULL`, all variables are kept. Ignored if data is `NULL`.

`matchMethod`

Method for matching to dictionary: 'none' for no matching, 'lv' for Levenshtein; 'j' for Jaro, 'jw' for JaroWinkler, 'bag' for bag of words, 'exact' for exact matching. The default matchMethod is 'lv'.

`keyType`

Transformation type in creating keys: 'all' to retain all characters, 'alphanum' for alphanumeric characters only, 'alpha' for letters only", 'mphone3' for Metaphone3 phonetic transformation, 'soundex' for Soundex phonetic transformation, 'mphone3Vowels' for Metaphone3 encoding more than initial vowels, 'mphone3Exact' for Metaphone3 with more exact consonants, 'soundexNum' for Soundex with numbers only, 'soundexAll' for Soundex not truncated at 4 characters, and 'soundexAm' for the American variant of Soundex.

`ignoreCase`

A logical specifying whether or not to ignore case when comparing strings to the dictionary

`ignoreSpaces`

A logical specifying whether or not to ignore spaces. If `FALSE`, for phonetic conversions each word in the string is processed separately and then concatenated together.

`ignoreNumbers`

A logical. If `FALSE`, numbers are converted to words before phonetic processing. If `TRUE`, numbers are ignored or removed.

`ignoreWords`

An optional character vector containing words to ignore when matching.

`noMatchNA`

A logical. If `TRUE`, if a match is not found an empty string is returned. Only applies when dictionary is provided.

`minDistance`

Minimum distance required for a match; applies only to distance metric algorithms (Levenshtein, Jaro, JaroWinkler). One is a perfect match. Zero is no similarity.

`dictionary`

Character vector containing the dictionary for string comparisons. Used for distance metric algorithms. from strings before processing.

`keyVarName`

`NULL` or a character string specifying the name to use for the newly created key variable. If `NULL`, the new variable name will be constructed using the `stringsIn` variable name and `.key`. Ignored if `data` is `NULL`.

`costs`

A named numeric vector with names `insert`, `delete`, and `subst` giving the respective costs for computing the Levenshtein distance. The default uses unit cost for all three. Ignored if Levenshtein distance not being used.

`hasMatchVarName`

`NULL` or a character string specifying the name to use for a logical variable indicating whether word was matched to dictionary or not. If `NULL`, no variable will be created.

`matchDistVarName`

`NULL` or a character string specifying the name to use for a numeric variable containing the distance of the match. If `NULL`, no variable will be created.

`numMatchVarName`

`NULL` or a character string specifying the name to use for an integer variable containing the number of alternative matches were found that satisfied `minDistance` criterion . If `NULL`, no variable will be created.

`overwrite`

A logical. If `TRUE` and the specified `outFile` exists, it will be overwritten.

Details

Two basic algorithms are provided, Soundex and Metaphone 3, with variations on both of them.

The `rxGetFuzzyKeys` function can process a character vector of data, a character column in a data frame, or a string variable in a RevoScaleR data source.

The simplified Soundex algorithm creates a 4-letter code: the first letter of the word as is, followed by 3 numbers representing consonants in the word, or zeros if necessary to fill out the four characters. Non-alphabetical

characters are ignored. The Soundex method is used widely, and is available, for example, on the RootsWeb web site <http://searches.rootsweb.ancestry.com/cgi-bin/Genea/soundex.sh>.

American Soundex differs slightly from standard simplified Soundex in its treatment of 'H' and 'W', which are treated differently when present between two other consonants. This is described by the U.S. National Archives <http://www.archives.gov/research/census/soundex.html>.

Another variant of Soundex is to code the first letter in the same way that all letters are coded, giving the first letter less importance.

Using `soundexNums` as the `keyType` provides this variant.

Another popular variant of Soundex is to continue coding all letters rather than stopping at 4.

Extra zeros are not added to fill out the code, so the result may be less than 4 characters. For this variant, use `soundexAll` as the `keyType`.

Note that in this case, spaces are treated as vowels in separating consonants.

The Metaphone 3 algorithm was developed by Lawrence Philips, who designed and developed the original Metaphone algorithm in 1990 and the Double Metaphone algorithm in 2000. Metaphone 3 is reported to increase the accuracy of phonetic encoding from the 89 against a database of the most common English words, and names and non-English words familiar in North America.

In Metaphone3, all vowels are encoded to the same value - 'A'. If the `mphone3` is used as the `keyType`, only initial vowels will be encoded at all. If `mphone3Vowels` is used as the `keyType`, 'A' will be encoded at all places in the word that any vowels are normally pronounced. 'W' as well as 'Y' are treated as vowels.

The `mphone3Exact` `keyType` is a variant of Metaphone3 that allows you to encode consonants as exactly as possible. This does not include 'S' vs. 'Z', since Americans will pronounce 'S' at the end of many words as 'Z', nor does it include 'CH' vs. 'SH'. It does cause a distinction to be made between 'B' and 'P', 'D' and 'T', 'G' and 'K', and 'V' and 'F'.

In phonetic matching, all non-alphabetic characters are ignored, so that any strings containing numbers may give misleading results. This is typically handled by removing numbers from strings and handling them separately, but occasionally it is convenient to have the numbers converted to their phonetic equivalent. This is accomplished by setting the `ignoreNumbers` parameter to FALSE.

Phonetic matching also typically treats the entire string as a single word. Setting the `ignoreSpaces` argument to `TRUE` results in each word being processed separately, with the result concatenated into a single (longer) code.

For information on similarity distance measures, see [rxGetFuzzyDist](#).

Value

A character vector containing the fuzzy keys or a data source containing the fuzzy keys in a new variable.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxGetFuzzyDist](#), [rxDataStep](#)

Examples

```
cityDictionary <- c("Aberdeen", "Anacortes", "Arlington", "Auburn",
  "Bellingham", "Blaine", "Bothell", "Bremerton", "Burlington", "Carnation")
```

```

    "Bellevue", "Bellingham", "Bothell", "Bremerton", "Bettell", "Camas",
    "Des Moines", "Edmonds", "Everett", "Federal Way", "Kennewick",
    "Marysville", "Olympia", "Pasco", "Pullman", "Redmond", "Renton", "Sammamish",
    "Seattle", "Shoreline", "Spokane", "Tacoma", "Vancouver", "Yakima")

cityNames <- c("Redmond", "Redmonde", "Edmondse", "REDMND", "EDMOnts")

# The string distance matchMethods require a dictionary
rxGetFuzzyKeys(stringsIn = cityNames, matchMethod = "lv", dictionary = cityDictionary)
rxGetFuzzyKeys(stringsIn = cityNames, matchMethod = "jw", dictionary = cityDictionary)

rxGetFuzzyKeys(stringsIn = cityNames, matchMethod = "lv", dictionary = cityDictionary, ignoreCase = TRUE)
rxGetFuzzyKeys(stringsIn = cityNames, matchMethod = "jw", dictionary = cityDictionary, ignoreCase = TRUE)

# Use mphone3 converstion before matching
rxGetFuzzyKeys(stringsIn = cityNames, matchMethod = "lv", keyType = "mphone3",
               dictionary = cityDictionary)

# Compare soundex and American soundex
origStr <- c("Ashcroft", "FLASHCARD")
soundexCode <- rxGetFuzzyKeys(stringsIn = origStr, keyType = "soundex")
soundexAmCode <- rxGetFuzzyKeys(stringsIn = origStr, keyType = "soundexAm")
# Show the original string, Soundex, and American Soundex codes
data.frame(origStr, soundexCode, soundexAmCode)

# Compare Metaphone3 with Metaphone3 with internal vowels
origStr <- c("Phorensics", "forensics", "Nicholas", "Nicolas", "Nikolas",
             "Knight", "Night", "Stephen", "Steven", "Matthew", "Matt", "Shuan", "Shawn",
             "McDonald", "MacDonald", "Schwarzenegger", "Shwardseneger", "Ellen", "Elena", "Allen")

mphone3Code <- rxGetFuzzyKeys(stringsIn = origStr, keyType = "mphone3")
mphone3VowelsCode <- rxGetFuzzyKeys(stringsIn = origStr, keyType = "mphone3Vowels")
data.frame(origStr, mphone3Code, mphone3VowelsCode)

# Compare Metaphone 3 with Metaphone3 with more exact consonants
origStr <- c("Phorensics", "forensics", "Nicholas", "Nicolas", "Nikolas",
             "Knight", "Night", "Stephen", "Steven", "Matthew", "Matt", "Shuan", "Shawn",
             "McDonald", "MacDonald", "Schwarzenegger", "Shwardseneger", "Ellen", "Elena", "Allen")

mphone3Code <- rxGetFuzzyKeys(stringsIn = origStr, keyType = "mphone3")
mphone3ExactCode <- rxGetFuzzyKeys(stringsIn = origStr, keyType = "mphone3Exact")
data.frame(origStr, mphone3Code, mphone3ExactCode)

# Including numbers and spaces
origStr <- c("10 Main Apt 410", "20 Main Apt 300", "20 Main Apt 302")
mphone3Code <- rxGetFuzzyKeys(stringsIn = origStr, keyType = "mphone3")
mphone3NumCode <- rxGetFuzzyKeys(stringsIn = origStr, keyType = "mphone3", ignoreNumbers = FALSE)
mphone3NumSpCode <- rxGetFuzzyKeys(stringsIn = origStr, keyType = "mphone3", ignoreNumbers = FALSE,
                                     ignoreSpaces = FALSE)
data.frame(origStr, mphone3Code, mphone3NumCode, mphone3NumSpCode)

# Non-phonetic key types
origStr <- c("10 Main St", "Main St. ", "15 Second Ave.", "Second 15 Ave.")
alphaCode <- rxGetFuzzyKeys(stringsIn = origStr, keyType = "alpha")
alphaNumCode <- rxGetFuzzyKeys(stringsIn = origStr, keyType = "alphaNum")
data.frame(origStr, alphaCode, alphaNumCode)

# Use minDistance to correct data
# Create a small city dictionary
cityDictionary <- c("Seattle", "Olympia", "Spokane")

# Create a vector of city names to correct
cityNames <- c("Seattle", "Seattlee", "Olympa", "SPOKANE", "OLYmpia")

# Large minimum distance should result in no change
cityKey1 <- rxGetFuzzyKeys(stringsIn = cityNames, matchMethod = "lv", dictionary = cityDictionary,
                           minDistance = .9)
# Print results to console
cityKey1

```

```

# Smaller minimum distance should fix matched cases, but not mixed case issues
cityKey2 <- rxGetFuzzyKeys(stringsIn = cityNames, matchMethod = "lv", dictionary = cityDictionary,
minDistance = .8)
# Print results to console
cityKey2

# Small minimum distance should fix everything, even mixed case issues
cityKey3 <- rxGetFuzzyKeys(stringsIn = cityNames, matchMethod = "lv", dictionary = cityDictionary,
minDistance = .1)
# Print results to console
cityKey3

## Use with xdf file
# Create temporary xdf file
inData <- data.frame(institution = c("Seattle Pacific U", "SEATTLE UNIV", "Seattle Central",
"U Washingt", "UNIV WASH BOTHELL", "Puget Sound Univ",
"ANTIOC U SEATTLE", "North Seattle Univ", "North Seatle U",
"Seattle Inst Tech", "SeATLE College Nrth", "UNIV waSHINGTON",
"University Seattle", "Seattle Antioch U",
"Technology Institute Seattle", "Pgt Sound U",
"Seattle Central Univ", "Univ North Seattle",
"Bothell - Univ of Wash", "ANTIOCH SEATTLE"), stringsAsFactors = FALSE)

tempInFile <- tempfile(pattern = "fuzzyDistIn", fileext = ".xdf")
rxDataStep(inData = inData, outFile = tempInFile, rowsPerRead = 10)

uDictionary <- c("Seattle Pacific University", "Seattle University",
"University of Washington", "Seattle Central College",
"University of Washington, Bothell", "Puget Sound University",
"Antioch University, Seattle", "North Seattle College",
"Seattle Institute of Technology")

tempOutFile <- tempfile(pattern = "fuzzyDistOut", fileext = ".xdf")

outDataSource <- rxGetFuzzyKeys(stringsIn = "institution",
  data = tempInFile, outFile = tempOutFile,
  dictionary = uDictionary,
  ignoreWords = c("University", "Univ",
    "of", "U"),
  ignoreCase = TRUE,
  matchMethod = "bag",
  ignoreSpaces = FALSE,
  minDistance = .4,
  keyType = "mphone3", overwrite = TRUE)
rxGetVarInfo(outDataSource)
# Read the new data set back into memory
newData <- rxDataStep(outDataSource)

# See the 'cleaned-up' institution names in the new institution.key variable
# Note that one input, Seattle Inst Tech, was not cleaned
newData

# Clean-up
file.remove(tempInFile)
file.remove(tempOutFile)

```

rxGetInfo: Get Data Source Information

7/12/2022 • 3 minutes to read • [Edit Online](#)

Description

Get basic information about an RevoScaleR data source or data frame

Usage

```
rxGetInfo(data, getVarInfo = FALSE, getBlockSizes = FALSE,  
          getValueLabels = NULL, varsToKeep = NULL, varsToDelete = NULL,  
          startRow = 1, numRows = 0, computeInfo = FALSE,  
          allNodes = TRUE, verbose = 0)
```

Arguments

data

a data frame, a character string specifying an .xdf, or an [RxDataSource](#) object. If a local compute context is being used, this argument may also be a list of data sources, in which case the output will be returned in a named list. See the details section for more information.

getVarInfo

logical value. If `TRUE`, variable information is returned.

getBlockSizes

logical value. If `TRUE`, block sizes are returned in the output for an .xdf file, and when printed the first 10 block sizes are shown.

getValueLabels

logical value. If `TRUE` and `getVarInfo` is `TRUE` or `NULL`, factor value labels are included in the output.

varsToKeep

character vector of variable names for which information is returned. If `NULL` or `getVarInfo` is `FALSE`, argument is ignored. Cannot be used with `varsToDelete`.

varsToDelete

character vector of variable names for which information is not returned. If `NULL` or `getVarInfo` is `FALSE`, argument is ignored. Cannot be used with `varsToKeep`.

startRow

starting row for retrieval of data if a data frame or .xdf file.

numRows

number of rows of data to retrieve if a data frame or .xdf file.

`computeInfo`

logical value. If `TRUE`, and `getVarInfo` is `TRUE`, variable information (e.g., high/low values) for non-xdf data sources will be computed by reading through the data set. If `TRUE`, and `getVarInfo` is `FALSE`, the number of variables will be gotten from from non-xdf data sources (but not the number of rows).

`allNodes`

logical value. Ignored if the active `RxComputeContext` compute context is local or `RxForeachDoPar`. Otherwise, if `TRUE`, a list containing the information for the data set on each node in the active compute context will be returned. If `FALSE`, only information on the data set on the master node will be returned. Note that the determination of the master node is not controlled by the end user. See the *RevoScaleR Distributed Computing Guide* for more information on master node computations.

`verbose`

integer value. If `0`, no additional output is printed. If `1`, additional summary information is printed for an .xdf file.

Details

If a local compute context is being used, the `data` and `file` arguments may be a list of data source objects, e.g., `data = list(iris, airquality, attitude)`, in which case a named list of results are returned. For `rxGetInfo`, a mix of supported data sources is allowed. Note that data frames should not be specified in quotes because, in that case, they will be interpreted as .xdf data paths.

If the `RxComputeContext` is distributed, `rxGetInfo` will request information from the compute context nodes.

Value

list containing the following possible elements:

`fileName`

character string containing the file name and path (if an .xdf file).

`objName`

character string containing object name (if not an .xdf file).

`class`

class of the object if an R object.

`length`

length of the object if it is not an .xdf file or data frame.

`numCompositeFiles`

number of composite data files(if a composite .xdf file).

`numRows`

number of rows in the data set.

`numVars`

number of variables in the data set.

`numBlocks`

number of blocks in the data set.

`varInfo`

list of variable information where each element is a list describing a variable. (See return value of [rxGetVarInfo](#) for more information.)

`rowsPerBlock`

integer vector containing number of rows in each block (if `getBlockSizes` is set to `TRUE`). Set to `NULL` if `data` is a data frame.

`data`

data frame containing the data (if `numRows > 0`)

If using a distributed compute context with the `allNodes` set to `TRUE`, a list of lists with information on the data from each node will be returned.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxDataStep](#), [rxGetVarInfo](#), [rxSetVarInfo](#).

Examples

```
# Summary information about a data frame
fileInfo <- rxGetInfo(data = iris, numRows = 5)
fileInfo

# Summary information about an .xdf file
mortFile <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers")
infoObj <- rxGetInfo(mortFile, getBlockSizes=TRUE)
numBlocks <- infoObj$numBlocks
aveRowsPerBlock <- infoObj$numRows/numBlocks
rowsPerBlock <- infoObj$rowsPerBlock
aveRowsPerBlock1 <- mean(rowsPerBlock)

# Obtain information on a variety of data sources
fourthGradersXDF <- file.path(rxGetOption("sampleDataDir"), "fourthgraders.xdf")
KyphosisDS <- RxXdfData(file.path(rxGetOption("sampleDataDir"), "kyphosis.xdf"))
claimsTxtDS <- RxTextData(file.path(rxGetOption("sampleDataDir"), "claims.txt"))
rxGetInfo(data = list(iris, fourthGradersXDF, KyphosisDS, claimsTxtDS))
```

rxGetJobInfo: Get Job Information from Distributed Computing Job

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Gets job information for a given distributed computing job.

Usage

```
rxGetJobInfo(object)
```

Arguments

`object`

an object containing `jobInfo` information, such as that returned from a non-waiting, distributed computation.

Details

The object returned from a non-waiting, distributed computation contains job information together with other information. The job information is used internally by such functions as `rxGetJobStatus`, `rxGetJobOutput`, and `rxGetJobResults`. It is sometimes useful to extract it for its own sake, as it contains complete information on the job's compute context as well as other information needed by the distributed computing resources.

For most users, the principal use of this function is to determine whether a given object actually contains job information. If the return value is not `NULL`, then the object contains job information. Note, however, that the structure of the job information is subject to change, so code that attempts to manipulate it directly is not guaranteed to be forward-compatible.

Value

the job information, if present, or `NULL`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxSpark](#), [RxHadoopMR](#), [RxInSqlServer](#), [rxGetJobStatus](#), [rxGetJobOutput](#), [rxGetJobResults](#)

Examples

```
## Not run:

# set up a non-waiting HPC Server compute context:
myCluster <- RxSpark(nameNode = "my-name-service-server", port = 8020, wait = FALSE)
rxOptions(computeContext=myCluster)

myJob <- rxExec(function(){ print( "Hello World"); return ( 1 ) })
# The job information consists of the job's compute context
# and other information needed to prepare and complete the job
rxGetJobInfo(myJob)
# The results object will be a list containing the value 1 for each node
rxGetJobResults(myJob)
# Get job results will remove the job object (by default)
# Another call to rxGetJobInfo(myJob) would return no output

## End(Not run)
```

rxGetJobOutput: Get Console Output from Distributed Computing Job

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Gets the console output from the various nodes in a non-waiting distributed computing job.

Usage

```
rxGetJobOutput(jobInfo)
```

Arguments

`jobInfo`

a job information object, such as that returned from a non-waiting, distributed computation, for example, the `rxgLastPendingJob` object, if available.

Details

During a job run, the state of the output is non-deterministic (that is, it may or may not be on disk, and what is on disk at any given point in time may not reflect the actual completion state of a job).

If `autoCleanup` has been set to `TRUE`, the console output will not persist after the job completes.

Unlike `rxGetJobResults`, this function does not remove any job information upon retrieval.

Value

This function is called for its side effect of printing console output; it does not have a useful return value.

See Also

[RxSpark](#), [RxHadoopMR](#), [RxInSqlServer](#), [rxGetJobs](#), [rxCleanupJobs](#), [rxGetJobResults](#), [rxExec](#).

Examples

```
## Not run:

# set up a non-waiting HPC Server compute context:
myCluster <- RxSpark(nameNode = "my-name-service-server", port = 8020, wait = FALSE)
rxOptions(computeContext=myCluster)

myJob <- rxExec(function(){ print( "Hello World"); return ( 1 ) })
# The job output will contain the printed text "Hello World" from each node
rxGetJobOutput(myJob)
# The results object will be a list containing the value 1 for each node
rxGetJobResults(myJob)
# Get job results will remove the job object (by default)
# Another call to rxGetJobOutput(myJob) would return no output

## End(Not run)
```

rxGetJobResults: Obtain Distributed Computing Job Status and Results

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Obtain distributed computing results and processing status.

Usage

```
rxGetJobResults(jobInfo, consoleOutput = NULL, autoCleanup = NULL)
rxGetJobStatus(jobInfo)
```

Arguments

jobInfo

a `jobInfo` object as returned by `rxExec` or a `RevoScaleRanalysis` function, for example, the `rxgLastPendingJob` object, if available.

consoleOutput

`NULL` or logical value. If `TRUE`, the console output from all of the processes is printed to the user console. If `FALSE`, no console output is displayed. Output can be retrieved with the function `rxGetJobOutput` for a non-waiting job. If not `NULL`, this flag overrides the value set in the compute context when the job was submitted. If `NULL`, the setting in the compute context will be used.

autoCleanup

`NULL` or logical value. If `TRUE`, the default behavior is to clean up any artifacts created by the distributed computing job. If `FALSE`, then the artifacts are not deleted, and the results may be acquired using `rxGetJobResults`, and the console output via `rxGetJobOutput` until the `rxCleanupJobs` is used to delete the artifacts. If not `NULL`, this flag overwrites the value set in the compute context when the job was submitted. If you routinely set `autoCleanup=FALSE`, you will eventually fill your hard disk with compute artifacts. If you set `autoCleanup=TRUE` and experience performance degradation on a Windows XP client, consider setting `autoCleanup=FALSE`.

Details

The possible job status strings returned by `rxGetJobStatus` are:

`"undetermined"`

information no longer retained by job scheduler. May still retrieve results and output when available.

`"finished"`

job has successfully completed.

`"failed"`

job has failed.

"canceled"

job has been canceled and the cancel process is completed.

"missing"

job is no longer available on the cluster.

"running"

job is either running or completing it's run (finishing, flushing buffers, etc.), or it is in the process of being canceled.

"queued"

job is in one of the following states: it is being configured, has been submitted but has not started, is being validated, or is in the job queue.

On LSF clusters, job information by default is held for only one hour (although this is configurable using the LSF parameter CLEAN_PERIOD); jobs older than the CLEAN_PERIOD setting will have status "undetermined".

Value

`rxGetJobResults` : either the results of the run (prepended with console output if the `consoleOutput` argument is set to `TRUE`) or a message saying that the results are not available because the job has not finished, has failed, or was deleted.

`rxGetJobStatus` : a character string denoting the job status.

See Also

[RxSpark](#), [RxHadoopMR](#), [RxInSqlServer](#), [rxGetJobs](#), [rxCleanupJobs](#), [rxGetJobStatus](#), [rxExec](#).

Examples

```
## Not run:

# set up the cluster object
myCluster <- RxSpark(nameNode = "my-name-service-server", port = 8020, wait = FALSE)
rxOptions(computeContext=myCluster)

func <- function(name)
{
  print("hi")
  cat(name)
  name
}

jobInfo <- rxExec(func, "dog", elemType = "cores", timesToRun = 5)
rxGetJobStatus(jobInfo)
rxGetJobResults(jobInfo)
## End(Not run)
```

rxGetJobs: Get Distributed Computing Jobs

7/12/2022 • 3 minutes to read • [Edit Online](#)

Description

Returns a list of job objects associated with the given compute context and matching the specified parameters.

Usage

```
rxGetJobs(computeContext, exactMatch = FALSE, startTime = NULL, endTime = NULL,  
states = NULL, verbose = TRUE)
```

Arguments

`computeContext`

A compute context object.

`exactMatch`

Determines if jobs are matched using the full compute context, or a simpler subset. If `TRUE`, only jobs which use the same context object are returned. If `FALSE`, all jobs which have the same `headNode` (if available) and `ShareDir` are returned.

`startTime`

A time, specified as a `POSIXct` object. If specified, only jobs created at or after `startTime` are returned. For non-RxHadoopMR contexts, this time should be specified in the user's local time; for RxHadoopMR contexts, the time should be specified in GMT. See below for more details.

`endTime`

A time, specified as a `POSIXct` object. If specified, only jobs created at or before `endTime` are returned. For non-RxHadoopMR contexts, this time should be specified in the user's local time; for RxHadoopMR contexts, the time should be specified in GMT. See below for more details.

`states`

If specified (as a character vector of states that can include `"none"`, `"finished"`, `"failed"`, `"canceled"`, `"undetermined"`, `"queued"` or `"running"`), only jobs in those states are returned. Otherwise, no filtering is performed on job state.

`verbose`

If `TRUE` (the default), a brief summary of each job is printed as it is found. This includes the current job status as returned by `rxGetJobStatus`, the modification time of the job, and the current job ID (this is used as the component name in the returned list of job information objects). If no job status is returned, the job status shows `none`.

Details

One common use of `rxGetJobs` is as input to the `rxCleanupJobs` function, which is used to clean up completed non-waiting jobs when `autoCleanup` is not specified.

If `exactMatch=FALSE`, only the shared directory `shareDir` and the cluster head name `headNode` (if available) are compared. Otherwise, all slots are compared. However, if the `nodes` slot in either compute context is `NULL`, that slot is also omitted from the comparison.

On LSF clusters, job information by default is held for only one hour (although this is configurable using the LSF parameter `CLEAN_PERIOD`); jobs older than the `CLEAN_PERIOD` setting will have status `"undetermined"`.

For non-RxHadoopMR cluster types, all time values are specified and displayed in the user's computer's local time settings, regardless of the time zone settings and differences between the user's computer and the cluster. Thus, start and end times for job filtering should be provided in local time, with the expectation that cluster time values will also be converted for the user into system local time. For RxHadoopMR, the job time and comparison times are stored and performed based on a GMT time.

Note also that when there are a large number of jobs on the cluster, you can improve performance by using the `startTime` and `endTime` parameters to narrow your search.

Value

Returns a `rxJobInfoList`, list of job information objects based on the compute context.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxCleanupJobs](#), [rxGetJobOutput](#), [rxGetJobResults](#), [rxGetJobStatus](#), [rxExec](#), [RxSpark](#), RxHadoopMR, RxInSqlServer, RxComputeContext

Examples

```
## Not run:

myCluster <- RxComputeContext("RxSpark",
  # Location of Revo64 on each node
  revoPath = file.path(defaultRNodePath, "bin", "x64"),
  nameNode = "cluster-head2",
  # User directory for read/write
  shareDir = "\\AllShare\\myName"
)

rxSetComputeContext(computeContext = myCluster )

# Get all jobs older than a week and newer than two weeks that are finished or canceled.
rxGetJobs(myCluster, startTime = Sys.time() - 3600 * 24 * 14, endTime = Sys.time() - 3600 * 24 * 7,
  exactMatch = FALSE, states = c( "finished", "canceled" ) )

# Get all jobs associated with myCluster compute context and then get job output and results
myJobs <- rxGetJobs(myCluster)
print(myJobs)
# returns
# rxJob_1461
myJobs$rxJob_1461
rxGetJobOutput(myJobs$rxJob_1461)
rxGetJobResults(myJobs$rxJob_1461)
## End(Not run)
```

rxGetNodeInfo: Provides information about all nodes on a cluster.

7/12/2022 • 3 minutes to read • [Edit Online](#)

Description

Provides information about the capabilities of the nodes on a cluster.

Usage

```
rxGetNodeInfo( computeContext = NULL, ..., namesOnly = FALSE, makeRNodeNames = FALSE, getWorkersOnly = TRUE )
```

Arguments

`computeContext`

A distributed compute context (preferred), a `jobInfo` object, or (deprecated) a character scalar containing the name of the Microsoft HPC cluster head node being queried. If you are interested in information about a particular node, be sure that node is included in the group specified in the compute context, or that the compute context has `groups=NULL` (HPC compute contexts). Setting `nodes=NULL` and `groups=NULL` or `queue="all"` in the compute context is the best way to ensure getting information on all nodes.

`...`

additional arguments for modifying the compute context before requesting the node information. These arguments must be in the compute context constructor.

`namesOnly`

logical. If `TRUE`, only a vector containing the names of the nodes will be returned.

`makeRNodeNames`

logical. If `TRUE`, names of the nodes will be normalized for use as R variables. See `rxMakeRNodeNames` for details on name mangling.

`getWorkersOnly`

logical. If `TRUE`, returns only those nodes within the cluster that are configured to actually execute jobs (where applicable; currently LSF only.).

Details

This function will return information on all the nodes on a given cluster if the `nodes` slot of the compute context is `NULL` and `groups` and `queue` are not specified in the compute context. If either `groups` or `queue` is specified, information is provided only for nodes within the specified groups or queue, using the normal rules described in the compute context for taking the unions of node sets and nodes in queues. The best way to get information on all nodes is to ensure that both the nodes and groups slots are set to `NULL` for HPC, and that nodes and queues are set to `NULL` and `"all"`, respectively, for LSF.

Note that unlike `rxGetAvailableNodes`, the return value of this function will contain all nodes in the compute context regardless of their state.

(`rxGetAvailableNodes` returns only online nodes).

If the return value of this function is used to populate node names elsewhere and `namesOnly` is `TRUE`, `makeRNodeNames` should be set to the default `FALSE`. If `namesOnly` is `FALSE`, the `nodeName` element of each list element should be used, not the names of the list element, given that the list element names will be mangled to be proper R names.

This operation is performed because names with a dash (-) are not permitted as variable names (the R interpreter interprets this as a minus operation). Thus, the mangling process replaces the dashes with an underscore _ to form the variable names. See [rxMakeRNodeNames](#) for details on name mangling.

Also, note that

node names are always forced to all capital letters (node names should be case agnostic, but in some cluster related software, it is necessary to force them to be upper case. In general, however, machine names are always treated as being case insensitive.

No other changes are made to node names during the mangling process.)

Value

If `namesOnly` is `TRUE`, a character vector containing the names of the nodes. If `makeRNodeNames` is `TRUE`, these names will be normalized for use as R variables.

If `namesOnly` is `FALSE`, a named list of lists, where each top level name is a node name on the cluster, normalized for use as an R variable. See [rxMakeRNodeNames](#) for more details.

Each named element in the list will contain some or all of the following:

`nodeName`

character scalar. The true name of the node (as opposed to the list item name).

`cpuSpeed`

float. The processor speed in MHz for Microsoft HPC, or the CPU factor for LSF.

`memSize`

integer. The amount of RAM in MB.

`numCores`

integer. The number of cores.

`numSockets`

integer. The number of CPU sockets.

`nodeState`

character scalar. May be either "online", "offline", "draining" (shutting down, but still with jobs in queue), or "unknown".

`isReachable`

logical. Determines if there is an operational network path between the head node and the given compute node.

`groupsOrQueues`

list of character scalars. The node groups (on MS HPC) or queues (under LSF) to which the node belongs.

Author(s)

See Also

[rxGetAvailableNodes](#), [rxMakeRNodeNames](#).

Examples

```
## Not run:

# Returns list of lists with information about each specified node
rxgetNodeInfo(myCluster, nodes = "comp1,comp2,comp3")
rxgetNodeInfo(myCluster, nodes = c("g1","g2","g3"))

# Returns a vector with the names of the nodes in the cluster, excluding the
# head node if computeOnHeadNode is set to FALSE in the compute context
rxgetNodeInfo(myCluster, namesOnly = TRUE)

# Node names in the returned vector are converted to valid R names
rxgetNodeInfo(myCluster, namesOnly = TRUE, makeRNodeNames = TRUE)

## End(Not run)
```

rxGetPartitions: Get Partitions of a Partitioned Xdf Data Source

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Get partitions enumeration of a partitioned Xdf data source.

Usage

```
rxGetPartitions(data)
```

Arguments

`data`

an existing partitioned data source object which was created by `RxXdfData` with `createPartitionSet = TRUE` and constructed by `rxPartition`.

Details

Value

Data frame with $(n+1)$ columns, the first n columns are partitioning columns specified by `varsToPartition` in `rxPartition` and the $(n+1)$ th column is a data source column where each row contains an Xdf data source object of one partition.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

`RXfdfData`, `rxPartition`, `rxExecBy`

Examples

```
# create an input Xdf data source
inFile <- "claims.xdf"
inFile <- file.path(dataPath = rxGetOption(opt = "sampleDataDir"), inFile)
inXdfDS <- RxXdfData(file = inFile)

# create an output partitioned Xdf data source
outFile <- file.path(tempdir(), "partitionedClaims.xdf")
outPartXdfDataSource <- RxXdfData(file = outFile, createPartitionSet = TRUE)

# construct and save the partitioned Xdf to disk
partDF <- rxPartition(inData = inXdfDS, outData = outPartXdfDataSource, varsToPartition = c("car.age"))

# enumerate partitions
partDF <- rxGetPartitions(outPartXdfDataSource)
rxGetInfo(partDF, getVarInfo = TRUE)
partDF$DataSource
```

rxGetSparklyrConnection: Get sparklyr connection from Spark compute context

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Get a Spark compute context with sparklyr interop. `rxGetSparklyrConnection` get sparklyr spark connection from created Spark compute context.

Usage

```
rxGetSparklyrConnection(  
    computeContext = rxGetOption("computeContext"))
```

Arguments

`computeContext`

Compute context get created by `rxSparkConnect`.

Value

object of sparklyr spark connection

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

Examples

```
## Not run:  
  
library("sparklyr")  
cc <- rxSparkConnect(interop = "sparklyr")  
sc <- rxGetSparklyrConnection(cc)  
iris_tbl <- copy_to(sc, iris)  
## End(Not run)
```

rxGetVarInfo: Get Variable Information for a Data Source.

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Get variable information for a RevoScaleR data source or data frame, including variable names, descriptions, and value labels

Usage

```
rxGetVarInfo(data, getValueLabels = TRUE, varsToKeep = NULL,  
             varsToDrop = NULL, computeInfo = FALSE, allNodes = TRUE)
```

Arguments

data

a data frame, a character string specifying the .xdf file, or an [RxDataSource](#) object. If a local compute context is being used, this argument may also be a list of data sources, in which case the output will be returned in a named list. See the details section for more information.

getValueLabels

logical value. If `TRUE`, value labels (including factor levels) are included in the output if present.

varsToKeep

character vector of variable names for which information is returned. If `NULL`, argument is ignored. Cannot be used with `varsToDrop`.

varsToDrop

character vector of variable names for which information is not returned. If `NULL`, argument is ignored. Cannot be used with `varsToKeep`.

computeInfo

logical value. If `TRUE`, variable information (e.g., high/low values) for non-xdf data sources will be computed by reading through the data set.

allNodes

logical value. Ignored if the active [RxComputeContext](#) compute context is local. Otherwise, if `TRUE`, a list containing the variable information for the data set on each node in the active compute context will be returned. If `FALSE`, only information on the data set on the master node will be returned.

Details

Will also give partial information for lists and matrices.

If a local compute context is being used, the `data` and `file` arguments may be a list of data source objects, e.g., `data = list(iris, airquality, attitude)`, in which case a named list of results are returned. For `rxGetVarInfo`, a mix of supported data sources is allowed.

If the `RxComputeContext` is distributed, `rxGetVarInfo` will request information from the compute context nodes.

Value

list with named elements corresponding to the variables in the data set. Each list element is also a list with with following possible elements:

`description`

character string specifying the variable description

`varType`

character string specifying the variable type

`storage`

character string specifying the storage type

`low`

numeric giving the low values, possibly generated through a temporary factor transformation `F()`

`high`

numeric giving the high values, possibly generated through a temporary factor transformation `F()`

`levels`

(factor only) a character vector containing the factor levels

`valueInfoCodes`

character vector of value codes, for informational purposes only

`valueInfoLabels`

character vector of value labels that is the same length as `valueInfoCodes`, used for informational purposes only

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxSetVarInfo](#), [rxDataStep](#).

Examples

```
# Specify name and location of sample data file
censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers")

# Read the variable information
varInfo <- rxGetVarInfo(censusWorkers)

# Print the variable information
varInfo

# Get variable information about a built-in data frame
rxGetVarInfo( iris )

# Obtain variable information on a variety of data sources
fourthGradersXDF <- file.path(rxGetOption("sampleDataDir"), "fourthgraders.xdf")
KyphosisDS <- RxXdfData(file.path(rxGetOption("sampleDataDir"), "kyphosis.xdf"))
rxGetVarInfo(data = list(iris, fourthGradersXDF, KyphosisDS))
```

rxGetVarInfo: Get Variable Information for a Data Source

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Get variable information for a RevoScaleR data source or data frame, including variable names, descriptions, and value labels

Usage

```
rxGetVarInfo(data, getValueLabels = TRUE, varsToKeep = NULL,  
             varsToDrop = NULL, computeInfo = FALSE, allNodes = TRUE)
```

Arguments

data

a data frame, a character string specifying the .xdf file, or an [RxDataSource](#) object. If a local compute context is being used, this argument may also be a list of data sources, in which case the output will be returned in a named list. See the details section for more information.

getValueLabels

logical value. If `TRUE`, value labels (including factor levels) are included in the output if present.

varsToKeep

character vector of variable names for which information is returned. If `NULL`, argument is ignored. Cannot be used with `varsToDrop`.

varsToDrop

character vector of variable names for which information is not returned. If `NULL`, argument is ignored. Cannot be used with `varsToKeep`.

computeInfo

logical value. If `TRUE`, variable information (e.g., high/low values) for non-xdf data sources will be computed by reading through the data set.

allNodes

logical value. Ignored if the active [RxComputeContext](#) compute context is local. Otherwise, if `TRUE`, a list containing the variable information for the data set on each node in the active compute context will be returned. If `FALSE`, only information on the data set on the master node will be returned.

Details

Will also give partial information for lists and matrices.

If a local compute context is being used, the `data` and `file` arguments may be a list of data source objects, e.g., `data = list(iris, airquality, attitude)`, in which case a named list of results are returned. For `rxGetVarInfo`, a mix of supported data sources is allowed.

If the `RxComputeContext` is distributed, `rxGetVarInfo` will request information from the compute context nodes.

Value

list with named elements corresponding to the variables in the data set. Each list element is also a list with with following possible elements:

`description`

character string specifying the variable description

`varType`

character string specifying the variable type

`storage`

character string specifying the storage type

`low`

numeric giving the low values, possibly generated through a temporary factor transformation `F()`

`high`

numeric giving the high values, possibly generated through a temporary factor transformation `F()`

`levels`

(factor only) a character vector containing the factor levels

`valueInfoCodes`

character vector of value codes, for informational purposes only

`valueInfoLabels`

character vector of value labels that is the same length as `valueInfoCodes`, used for informational purposes only

See Also

[rxSetVarInfo](#), [rxDataStep](#).

Examples

```
# Specify name and location of sample data file
censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers")

# Read the variable information
varInfo <- rxGetVarInfo(censusWorkers)

# Print the variable information
varInfo

# Get variable information about a built-in data frame
rxGetVarInfo( iris )

# Obtain variable information on a variety of data sources
fourthGradersXDF <- file.path(rxGetOption("sampleDataDir"), "fourthgraders.xdf")
KyphosisDS <- RxXdfData(file.path(rxGetOption("sampleDataDir"), "Kyphosis.xdf"))
rxGetVarInfo(data = list(iris, fourthGradersXDF, KyphosisDS))
```

rxGetVarNames: Variable names for a data source or data frame

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Read the variable names for data source or data frame

Usage

```
rxGetVarNames(data)
```

Arguments

`data`

an `RxDataSource` object, a character string specifying the .xdf file, or a data frame.

Details

If collInfo is used in the data source to specify new column names, these new names will be used in the return value.

Value

character vector containing the names of the variables in the data source or data frame.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxGetVarInfo](#), [rxSetVarInfo](#), [rxDataStep](#), [rxGetInfo](#).

Examples

```
# Specify name and location of sample data file
censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers")

# Get the variable names
varNames <- rxGetVarNames(censusWorkers)

# Print the variable names
varNames
```

rxGlm: Generalized Linear Models

7/12/2022 • 9 minutes to read • [Edit Online](#)

Description

Use `rxGlm` to fit generalized linear regression models for small or large data. Any valid `glm` family object can be used.

Usage

```
rxGlm(formula, data, family = gaussian(),
       pweights = NULL, fweights = NULL, offset = NULL,
       cube = FALSE, variableSelection = list(), rowSelection = NULL,
       transforms = NULL, transformObjects = NULL,
       transformFunc = NULL, transformVars = NULL,
       transformPackages = NULL, transformEnvir = NULL,
       dropFirst = FALSE, dropMain = rxGetOption("dropMain"),
       covCoef = FALSE, computeAIC = FALSE, initialValues = NA,
       coefLabelStyle = rxGetOption("coefLabelStyle"),
       blocksPerRead = rxGetOption("blocksPerRead"),
       maxIterations = 25, coeffTolerance = 1e-06,
       objectiveFunctionTolerance = 1e-08,
       reportProgress = rxGetOption("reportProgress"), verbose = 0,
       computeContext = rxGetOption("computeContext"),
       ...)
```

Arguments

formula

formula as described in [rxFormula](#).

data

either a data source object, a character string specifying a .xdf file, or a data frame object.

family

either an object of class `family` or a character string specifying the family. A family is a description of the error distribution and is associated with a link function to be used in the model. Any valid `family` object may be used. The following family/link combinations are implemented in C++: binomial/logit, gamma/log, poisson/log, and Tweedie. Other family/link combinations use a combination of C++ and R code. For the Tweedie distribution, use `family=rxTweedie(var.power, link.power)`. It is also possible to use `family=tweedie(var.power, link.power)` from the R package `tweedie`.

pweights

character string specifying the variable to use as probability weights for the observations.

fweights

character string specifying the variable to use as frequency weights for the observations.

offset

character string specifying a variable to use as an offset for the model (`offset="x"`). An offset may also be

specified as a term in the formula (`offset(x)`). An offset is a variable to be included as part of the linear predictor that requires no coefficient.

`cube`

logical flag. If `TRUE` and the first term of the predictor variables is categorical (a factor or an interaction of factors), the regression is performed by applying the Frisch-Waugh-Lovell Theorem, which uses a partitioned inverse to save on computation time and memory. See Details section below.

`variableSelection`

a list specifying various parameters that control aspects of stepwise regression. If it is an empty list (default), no stepwise model selection will be performed. If not, stepwise regression will be performed and `cube` must be `FALSE`. See [rxStepControl](#) for details.

`rowSelection`

name of a logical variable in the data set (in quotes) or a logical expression using variables in the data set to specify row selection. For example, `rowSelection = "old"` will use only observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` will use only observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

`transforms`

an expression of the form `list(name = expression, ...)` representing the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

`transformObjects`

a named list containing objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

`transformFunc`

variable transformation function. See [rxTransform](#) for details.

`transformVars`

character vector of input data set variables needed for the transformation function. See [rxTransform](#) for details.

`transformPackages`

character vector defining additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, e.g., those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.

`transformEnvir`

user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

`dropFirst`

logical flag. If `FALSE`, the last level is dropped in all sets of factor levels in a model. If that level has no observations (in any of the sets), or if the model as formed is otherwise determined to be singular, then an attempt is made to estimate the model by dropping the first level in all sets of factor levels. If `TRUE`, the starting

position is to drop the first level. Note that for cube regressions, the first set of factors is excluded from these rules and the intercept is dropped.

dropMain

logical value. If `TRUE`, main-effect terms are dropped before their interactions.

covCoef

logical flag. If `TRUE` and if `cube` is `FALSE`, the variance-covariance matrix of the regression coefficients is returned. Use the `rxCovCoef` function to obtain these data.

computeAIC

logical flag. If `TRUE`, the AIC of the fitted model is returned. The default is `FALSE`.

initialValues

Starting values for the Iteratively Reweighted Least Squares algorithm used to estimate the model coefficients. Supported values for this argument come in three forms:

- `NA` - The initial values are based upon the 'mustart' values generated by the `family$initialize` expression if an R family object is used, or the equivalent for a family coded in C++.
- `Scalar` - A numeric scalar that will be replicated once for each of the coefficients in the model to form the initial values. Zeros may be good starting values, depending upon the model.
- `Vector` - A numeric vector of length `k`, where `k` is the number of model coefficients, including the intercept if any and including any that might be dropped from sets of dummies. This argument is most useful when a set of estimated coefficients is available from a similar set of data. Note that `k` can be found by running the model on a small number of observations and obtaining the number of coefficients from the output, say `z`, via `length(z$coefficients)`.

coefLabelStyle

character string specifying the coefficient label style. The default is "Revo". If "R", R-compatible labels are created.

blocksPerRead

number of blocks to read for each chunk of data read from the data source.

maxIterations

maximum number of iterations.

coeffTolerance

convergence tolerance for coefficients. If the maximum absolute change in the coefficients (step), divided by the maximum absolute coefficient value, is less than or equal to this tolerance at the end of an iteration, the estimation is considered to have converged. To disable this test, set this value to 0.

objectiveFunctionTolerance

convergence tolerance for the objective function. If the absolute relative change in the deviance (-2.0 times log likelihood) is less than or equal to this tolerance at the end of an iteration, the estimation is considered to have converged. To disable this test, set this value to 0.

reportProgress

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`verbose`

integer value. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information are provided.

`computeContext`

a valid [RxComputeContext](#). The `RxSpark` and `RxHadoopMR` compute contexts distribute the computation among the nodes specified by the compute context; for other compute contexts, the computation is distributed if possible on the local computer.

`...`

additional arguments to be passed directly to the Revolution Compute Engine.

Details

The special function `F()` can be used in `formula` to force a variable to be interpreted as a factor.

When `cube` is `TRUE`, the Frisch-Waugh-Lovell (FWL) Theorem is applied to the model. The FWL approach parameterizes the model to include one coefficient for each category (a single factor level or combination of factor levels) instead of using an intercept in the model with contrasts for each of the factor combinations. Additionally when `cube` is `TRUE`, the output contains a `countDF` element representing the counts for each category.

Value

an `rxGlm` object containing the following elements:

`coefficients`

named vector of coefficients.

`covCoef`

variance-covariance matrix for the regression coefficient estimates.

`condition.number`

estimated reciprocal condition number of final weighted cross-product ($X'WX$) matrix.

`rank`

numeric rank of the fitted linear model.

`aliased`

logical vector specifying whether columns were dropped or not due to collinearity.

`coef.std.error`

standard errors of the coefficients.

`coef.t.value`

coefficients divided by their standard errors.

`coef.p.value`

p-values for `coef.t.value`; for the poisson and binomial families, the dispersion is taken to be 1, and the normal distribution is used ($\text{Pr}(>|z|)$); for other families the dispersion is estimated, and the t distribution is used ($\text{Pr}(>|t|)$).

`f.pvalue`

the p-value resulting from an F-test on the fitted model.

`df`

degrees of freedom, a 3-vector (p , $n-p$, p^*), the last being the number of non-aliased coefficients.

`fstatistics`

(for models including non-intercept terms) a 3-vector with the value of the F-statistic with its numerator and denominator degrees of freedom.

`params`

parameters sent to Microsoft R Services Compute Engine.

`formula`

the model formula

`call`

the matched call.

`nValidObs`

number of valid observations.

`nMissingObs`

number of missing observations.

`deviance`

minus twice the maximized log-likelihood (up to a constant)

`dispersion`

for the poisson and binomial families, the dispersion is 1; for other families the dispersion is estimated, and is the Pearson chi-squared statistic divided by the residual degrees of freedom.

Note

The generalized linear models are computed using the Iteratively Reweighted Least Squares (IRLS) algorithm.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

`glm`, `family`, [rxLinMod](#), [rxLogit](#), [rxTweedie](#), [rxTransform](#).

Examples

```

# Compare rxGlm and glm, which both can be used on small data sets
infertRxGlm <- rxGlm(case ~ age + parity + education + spontaneous + induced,
                      family = binomial(), dropFirst = TRUE, data = infert)
summary(infertRxGlm)

infertGlm <- glm(case ~ age + parity + education + spontaneous + induced,
                   data = infert, family = binomial())
summary(infertGlm)

# For the case of the binomial family with the 'logit' link family,
# the optimized rxLogit function can be used
infertRxLogit <- rxLogit(case ~ age + parity + education + spontaneous + induced,
                           data = infert, dropFirst = TRUE)
summary(infertRxLogit)

# Estimate a Gamma family model using sample data
claimsXdf <- file.path(rxGetOption("sampleDataDir"),"claims.xdf")
claimsGlm <- rxGlm(cost ~ age + car.age + type, family = Gamma,
                     dropFirst = TRUE, data = claimsXdf)
summary(claimsGlm)

# In the claims data, the cost is set to NA if no claim was made
# Convert NA to 0 for the cost, to prepare data for using
# the Tweedie family - which is appropriate for positive data
# that also contains exact zeros
# Read the transformed data into a data frame
claims <- rxDataStep(inData = claimsXdf,
                     transforms = list(cost = ifelse(is.na(cost), 0, cost)))

# Estimate using a Tweedie family
claimsTweedie <- rxGlm(cost ~ age + car.age + type ,
                        data=claims, family = rxTweedie(var.power =1.15))
summary(claimsTweedie)

# Illustrate the use of a Tweedie family with offset
TestData <- data.frame(
  Factor1 = as.factor(c(1,1,1,1,2,2,2,2)),
  Factor2 = as.factor(c(1,1,2,2,1,1,2,2)),
  Discount = c(1,2,1,2,1,2,1,2),
  Exposure = c(24000,40000,7000,14000,7500,15000,2000,5600),
  PurePrem = c(46,32,73,58,48,25,220,30)
)

rxGlmTweedieOffset <- rxGlm(PurePrem ~ Factor1 * Factor2 - 1 + offset(log(Discount)),
                            family = rxTweedie(var.power = 1.5, link.power = 0),
                            data = TestData, fweights = "Exposure", dropFirst = TRUE, dropMain = FALSE
)

## Not run:

require(statmod)
glmTweedieOffset <- glm(PurePrem ~ Factor1 * Factor2 - 1,
                        family = tweedie(var.power = 1.5, link.power = 0),
                        data = TestData, weights = Exposure, offset = log(Discount)
)
## End(Not run)

```

rxHadoopCommand: Execute Hadoop Commands

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Execute arbitrary Hadoop commands and perform standard file operations in Hadoop.

Usage

```
rxHadoopCommand(cmd, computeContext, sshUsername=NULL,  
                 sshHostname=NULL,  
                 sshSwitches=NULL,  
                 sshProfileScript=NULL, intern=FALSE)  
  
rxHadoopCopyFromLocal(source, dest, ...)  
  
rxHadoopCopyFromClient(source, nativeTarget="/tmp", hdfsDest,  
                       computeContext, sshUsername=NULL,  
                       sshHostname=NULL, sshSwitches=NULL, sshProfileScript=NULL)  
  
rxHadoopCopyToLocal(source, dest, ...)  
  
rxHadoopFileExists(path)  
  
rxHadoopListFiles(path="", recursive=FALSE, print, computeContext = rxGetComputeContext(), ...)  
  
rxHadoopMakeDir(path, ...)  
  
rxHadoopMove(source, dest, ...)  
  
rxHadoopCopy(source, dest, ...)  
  
rxHadoopRemove(path, skipTrash=FALSE, ...)  
  
rxHadoopRemoveDir(path, skipTrash=FALSE, ...)  
  
rxHadoopVersion()
```

Arguments

`cmd`

A character string containing a valid Hadoop command, that is, the `cmd` portion of `hadoop cmd`. Embedded quotes are not permitted.

`computeContext`

Run against this compute context. Default to the current compute context as returned by [rxGetComputeContext](#).

`sshUsername`

character string specifying the username for making an ssh connection to the Hadoop cluster.

`sshHostname`

character string specifying the hostname or IP address of the Hadoop cluster node or edge node that the client

will log into for launching Hadoop commands.

sshSwitches

character string specifying any switches needed for making an ssh connection to the Hadoop cluster.

sshProfileScript

Optional character string specifying the absolute path to a profile script that will exist on the `sshHostname` host. This is used when the target ssh host does not automatically read in a `.bash_profile`, `.profile` or other shell environment configuration file for the definition of requisite variables.

intern

logical (not `NA`) specifying whether to capture the output of a Hadoop command as an R character vector in a local compute context. (When using the `RxHadoopMR` compute context, any output is always returned as an R character vector.)

source

character vector specifying file(s) to be copied or moved.

dest

character string specifying the destination of a copy or move. If `source` includes more than one file, `dest` must be a directory.

nativeTarget

character string specifying a directory in the Hadoop cluster's native file system, to be used as an intermediate location for file(s) copied from a client machine.

hdfsDest

character string specifying a directory in the Hadoop Distributed File System.

path

character vector specifying location of one or more files or directories.

print

Deprecation Warning: the `print` argument in `rxHadoopListFiles` is now deprecated and is going to be removed in the next release. If `FALSE`, `rxHadoopListFiles` will return a `character` vector of paths; by default it prints paths to the console.

recursive

logical flag. If `TRUE`, directory listings are recursive.

skipTrash

logical flag. If `TRUE`, removal via `rxHadoopRemove` and `rxHadoopRemoveDir` bypasses the trash folder, if one has been set up.

...

additional arguments to be passed directly to the `rxHadoopCommand` function.

Details

`rxHadoopCommand` allows you to run basic Hadoop commands. `rxCopyFromClient` allows a file to be copied from a remote client to the Hadoop Distributed File System on the Hadoop cluster. `rxHadoopVersion` calls the Hadoop `version` command and extracts and returns the version number only. The remaining functions are wrappers for various Hadoop file system commands:

- * `rxHadoopCopyFromLocal` wraps the Hadoop `fs -copyFromLocal` command.
- * `rxHadoopCopyToLocal` wraps the Hadoop `fs -copyToLocal` command.
- * `rxHadoopListFiles` wraps the Hadoop `fs -ls` or `fs -lsr` command.
- * `rxHadoopRemove` wraps the Hadoop `fs -rm` command.
- * `rxHadoopCopy` wraps the Hadoop `fs -cp` command.
- * `rxHadoopMove` wraps the Hadoop `fs -mv` command.
- * `rxHadoopMakeDir` wraps the Hadoop `fs -mkdir` command.
- * `rxHadoopRemoveDir` wraps the Hadoop `fs -rm -r` command.

Value

These functions are executed for their side effects and typically return `NULL` invisibly.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

`RxHadoopMR`.

Examples

```
## Not run:

rxHadoopCommand("version") # should return version information
rxHadoopMakeDir("/user/RevoShare/newUser")
rxHadoopCopyFromLocal("/tmp/foo.txt", "/user/RevoShare/newUser")
rxHadoopRemoveDir("/user/RevoShare/newUser")

## End(Not run)
```

RxHadoopMR-class: Class RxHadoopMR

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

DEPRECATED: Hadoop Map Reduce Local (File) compute context class.

Generators

The targeted generator [RxHadoopMR](#) as well as the general generator [RxComputeContext](#).

Extends

Class RxComputeContext, directly.

Methods

[show](#)

```
signature(object = "RxHadoopMR") : ...
```

See Also

[RxHadoopMR](#)

RxHadoopMR: Generate Hadoop Map Reduce Compute Context

7/12/2022 • 7 minutes to read • [Edit Online](#)

Description

DEPRECATED: Creates a compute context for use with a Hadoop cluster.

RxHadoopMR compute context is deprecated. Please consider using RxSpark compute context instead.

Usage

```
RxHadoopMR(object,
  hdfsShareDir = paste( "/user/RevoShare", Sys.info()[["user"]], sep="/"),
  shareDir = paste( "/var/RevoShare", Sys.info()[["user"]], sep="/"),
  clientShareDir = rxGetDefaultTmpDirByOS(),
  hadoopRPath = rxGetOption("unixRPath"),
  hadoopSwitches = "",
  revoPath = rxGetOption("unixRPath"),
  sshUsername = Sys.info()[["user"]],
  sshHostname = NULL,
  sshSwitches = "",
  sshProfileScript = NULL,
  sshClientDir = "",
  usingRunAsUserMode = FALSE,
  nameNode = rxGetOption("hdfsHost"),
  jobTrackerURL = NULL,
  port = rxGetOption("hdfsPort"),
  onClusterNode = NULL,
  wait = TRUE,
  consoleOutput = FALSE,
  showOutputWhileWaiting = TRUE,
  autoCleanup = TRUE,
  workingDir = NULL,
  dataPath = NULL,
  outDataPath = NULL,
  fileSystem = NULL,
  packagesToLoad = NULL,
  resultsTimeout = 15,
  ... )
```

Arguments

`object`

object of class RxHadoopMR. This argument is optional. If supplied, the values of the other specified arguments are used to replace those of `object` and the modified object is returned.

`hdfsShareDir`

character string specifying the file sharing location within HDFS. You must have permissions to read and write to this location.

`shareDir`

character string specifying the directory on the master (perhaps edge) node that is shared among all the nodes

of the cluster and any client host. You must have permissions to read and write in this directory.

clientShareDir

character string specifying the absolute path of the temporary directory on the client. Defaults to /tmp for POSIX-compliant non-Windows clients. For Windows and non-compliant POSIX clients, defaults to the value of the TEMP environment variable if defined, else to the TMP environment variable if defined, else to `NULL`. If the default directory does not exist, defaults to NULL. UNC paths ("`\host\dir`") are not supported.

hadoopRPath

character string specifying the path to the directory on the cluster compute nodes containing the files R.exe and Rterm.exe. The invocation of R on each node must be identical.

revoPath

character string specifying the path to the directory on the master (perhaps edge) node containing the files R.exe and Rterm.exe.

hadoopSwitches

character string specifying optional generic Hadoop command line switches, for example `-conf myconf.xml`. See <http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/CommandsManual.html> for details on the Hadoop command line generic options.

sshUsername

character string specifying the username for making an ssh connection to the Hadoop cluster. This is not needed if you are running your R client directly on the cluster. Defaults to the username of the user running the R client (that is, the value of `Sys.info()[["user"]]`).

sshHostname

character string specifying the hostname or IP address of the Hadoop cluster node or edge node that the client will log into for launching Hadoop jobs and for copying files between the client machine and the Hadoop cluster. Defaults to the hostname of the machine running the R client (that is, the value of `Sys.info()[["nodename"]]`). This field is only used if `onClusterNode` is `NULL` or `FALSE`. If you are using PuTTY on a Windows system, this can be the name of a saved PuTTY session that can include the user name and authentication file to use.

sshSwitches

character string specifying any switches needed for making an ssh connection to the Hadoop cluster. This is not needed if one is running one's R client directly on the cluster.

sshProfileScript

Optional character string specifying the absolute path to a profile script that will exists on the sshHostname host. This is used when the target ssh host does not automatically read in a .bash_profile, .profile or other shell environment configuration file for the definition of requisite variables such as HADOOP_STREAMING.

sshClientDir

character string specifying the Windows directory where Cygwin's ssh.exe and scp.exe or PuTTY's plink.exe and pscp.exe executables can be found. Needed only for Windows. Not needed if these executables are on the Windows Path or if Cygwin's location can be found in the Windows Registry. Defaults to the empty string.

usingRunAsUserMode

logical scalar specifying whether run-as-user mode is being used on the Hadoop cluster. When using run-as-user mode, local R processes started by the map-reduce framework will run as the same user that started the job, and will have any allocated local permissions. When not using run-as-user mode (the default for many Hadoop systems), local R processes will run as user mapred. Note that when running as user mapred,

permissions for files and directories will have to be more open in order to allow hand-offs between the user and mapred. Run-as-user mode is controlled for the Hadoop map-reduce framework by the xml setting, `mapred.task.tracker.task-controller`, in the `mapred-site.xml` configuration file. If it is set to the value `org.apache.hadoop.mapred.LinuxTaskController`, then run-as-user mode is in use. If it is set to the value `org.apache.hadoop.mapred.DefaultTaskController`, then run-as-user mode is not in use.

`nameNode`

character string specifying the Hadoop name node hostname or IP address. Typically you can leave this at its default value. If set to a value other than "default" or the empty string (see below), this must be an address that can be resolved by the data nodes and used by them to contact the name node. Depending on your cluster, it may need to be set to a private network address such as `"master.local"`. If set to the empty string, "", then the master process will set this to the name of the node on which it is running, as returned by `Sys.info()[[["nodename"]]]`. This is likely to work when the `sshHostname` points to the name node or the `sshHostname` is not specified and the R client is running on the name node. Defaults to `rxGetOption("hdfsHost")`.

`jobTrackerURL`

character scalar specifying the full URL for the jobtracker web interface. This is used only for the purpose of loading the job tracker web page from the `rxLaunchClusterJobManager` convenience function. It is never used for job control, and its specification in the compute context is completely optional. See the [rxLaunchClusterJobManager](#) page for more information.

`port`

numeric scalar specifying the port used by the name node for hadoop jobs. Needs to be able to be cast to an integer. Defaults to `rxGetOption("hdfsPort")`.

`onClusterNode`

logical scalar or NULL specifying whether the user is initiating the job from a client that will connect to either an edge node or an actual cluster node, directly from either an edge node or node within the cluster. If set to `FALSE` or `NULL`, then `sshHostname` must be a valid host.

`wait`

logical value. If `TRUE`, the job will be blocking and will not return until it has completed or has failed. If `FALSE`, the job will be non-blocking return immediately, allowing you to continue running other R code. The object `rxgLastPendingJob` is created with the job information. You can pass this object to the `rxGetJobStatus` function to check on the processing status of the job. `rxWaitForJob` will change a non-waiting job to a waiting job. Conversely, pressing ESC changes a waiting job to a non-waiting job, provided that the HPC scheduler has accepted the job. If you press ESC before the job has been accepted, the job is canceled.

`consoleOutput`

logical scalar. If `TRUE`, causes the standard output of the R processes to be printed to the user console.

`showOutputWhileWaiting`

logical scalar. If `TRUE`, causes the standard output of the remote primary R and hadoop job process to be printed to the user console while waiting for (blocking on) a job.

`autoCleanup`

logical scalar. If `TRUE`, the default behavior is to clean up the temporary computational artifacts and delete the result objects upon retrieval. If `FALSE`, then the computational results are not deleted, and the results may be acquired using `rxGetJobResults`, and the output via `rxGetJobOutput` until the `rxCleanupJobs` is used to delete the results and other artifacts. Leaving this flag set to `FALSE` can result in accumulation of compute artifacts which you may eventually need to delete before they fill up your hard drive.

`workingDir`

character string specifying a working directory for the processes on the master node.

`dataPath`

NOT YET IMPLEMENTED. character vector defining the search path(s) for the data source(s).

`outDataPath`

NOT YET IMPLEMENTED. `NULL` or character vector defining the search path(s) for new output data file(s). If not `NULL`, this overrides any specification for `outDataPath` in `rxOptions`

`fileSystem`

`NULL` or an `RxHdfsFileSystem` to use as the default file system for data sources when created when this compute context is active.

`packagesToLoad`

optional character vector specifying additional packages to be loaded on the nodes when jobs are run in this compute context.

`resultsTimeout`

A numeric value indicating for how long attempts should be made to retrieve results from the cluster. Under normal conditions, results are available immediately. However, under certain high load conditions, the processes on the nodes have reported as completed, but the results have not been fully committed to disk by the operating system. Increase this parameter if results retrieval is failing on high load clusters.

`...`

additional arguments to be passed directly to the Microsoft R Services Compute Engine.

Details

This compute context is supported for Cloudera (CDH4 and CDH5), Hortonworks (HDP 1.3 and 2.x), and MapR (3.0.2, 3.0.3, 3.1.0, and 3.1.1) Hadoop distributions on Red Hat Enterprise Linux 5 and 6.

Value

object of class `RxHadoopMR`.

See Also

`rxGetJobStatus`, `rxGetJobOutput`, `rxGetJobResults`, `rxCleanupJobs`, `RxSpark`, `RxSqlServer`, `RxComputeContext`, `rxSetComputeContext`, `RxHadoopMR-class`.

Examples

```
## Not run:  
  
#####  
# Run hadoop on edge node  
#####  
  
hadoopCC <- RxHadoopMR()  
  
#####  
# Run hadoop from a Windows client  
# (requires Cygwin and/or PuTTY)  
#####  
mySshUsername <- "user1"  
mySshHostname <- "12.345.678.90" #public facing cluster IP address  
mySshSwitches <- "-i /home/yourName/user1.pem" #use .ppk file with PuTTY  
myShareDir <- paste("/var/RevoShare", mySshUsername, sep ="/")  
myHdfsShareDir <- paste("/user/RevoShare", mySshUsername, sep="/")  
myHadoopCluster <- RxHadoopMR(  
  hdfsShareDir = myHdfsShareDir,  
  shareDir = myShareDir,  
  sshUsername = mySshUsername,  
  sshHostname = mySshHostname,  
  sshSwitches = mySshSwitches)  
  
## End(Not run)
```

rxHdfsConnect: Establish a Connection to the Hadoop Distributed File System

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Establishes a connection from RevoScaleR to the Hadoop Distributed File System (HDFS).

Usage

```
rxHdfsConnect(hostName, portNumber)
```

Arguments

`hostName`

character string specifying the host name of your Hadoop name node.

`portNumber`

integer scalar specifying the port number of your Hadoop name node.

Details

If you accept the install time option to specify a default HDFS connection, you are prompted for a host name and port number for your Hadoop name node, which are stored as environment variables `REVOHADOOPHOST` and `REVOHADOOPPORT`. If these environment variables are set, this function is called by `Rprofile.site` on startup.

After establishing the connection, this function sets the `rxOptions` for `hdfsHost` and `hdfsPort`, and subsequent calls to `RxHdfsFileSystem` should be done using the default values of `hostName` and `port`.

It is important that this function be called before any functions that call into `rJava`, in particular before initializing `rhdfs`.

Value

invisibly, the return value of `rxOptions`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

`rxOptions`, `link{RxHdfsFileSystem}`

Examples

```
## Not run:  
  
rxHdfsConnect(hostName = "sandbox-01", port = 8020)  
  
myHDFS <- RxHdfsFileSystem()  
## End(Not run)
```

RxHdfsFileSystem: RevoScaleR HDFS File System object generator

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

This is the main generator for RxHdfsFileSystem S3 class.

Usage

```
RxHdfsFileSystem( object, hostName = rxGetOption("hdfsHost"), port = rxGetOption("hdfsPort"), useWebHdfs = FALSE, oAuthParameters = NULL, verbose = FALSE )

## S3 method for class `RxHdfsFileSystem':
print ( x, ... )
```

Arguments

`object`

object of class RxHdfsFileSystem. This argument is optional. If supplied, the values of the other arguments are used to replace those of `object` and the modified object is returned. If these arguments are not supplied, they will take their default values.

`hostName`

character string specifying name of host for HDFS file system.

`port`

integer specifying port number.

`useWebHdfs`

NOT YET IMPLEMENTED Optional Flag indicating whether this is a HDFS or a WebHdfs interface - default FALSE

`oAuthParameters`

NOT YET IMPLEMENTED Optional list of OAuth2 parameters created using rxOAuthParameters function (valid only if useWebHdfs is TRUE) - default NULL

`verbose`

Optional Flag indicating "verbose" mode for WebHdfs HTTP calls (valid only if useWebHdfs is TRUE) - default FALSE

`x`

an RxHdfsFileSystem object.

`...`

other arguments are passed to the underlying function.

Details

Writing to the HDFS file system can only be done using a RxHadoopMR compute context with an [RxXdfData](#) data source. The 'rxHadoop' commands, such as [rxHadoopCopy](#), can also be used to manipulate data sets in HDFS.

Value

An RxHdfsFileSystem file system object. This object may be used to in [rxSetFileSystem](#), [rxOptions](#), [RxTextData](#), or [RxXdfData](#) to set the file system.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxFileSystem](#), [RxNativeFileSystem](#), [rxSetFileSystem](#), [rxOptions](#), [RxXdfData](#), [RxTextData](#), [rxOAuthParameters](#).

Examples

```
# Setup to run analyses to use HDFS file system
## Not run:

myHdfsFileSystem <- RxHdfsFileSystem(hostName = "myHost", port = 8020)
rxSetFileSystem(fileSystem = myHdfsFileSystem )
## End(Not run)
```

rxHistogram: Histogram

7/12/2022 • 6 minutes to read • [Edit Online](#)

Description

Histogram plot for a variable in an .xdf file or data frame

Usage

```
rxHistogram(formula, data, pweights = NULL, fweights = NULL, numBreaks = NULL,
           startVal = NULL, endVal = NULL, levelsToDrop = NULL,
           levelsToKeep = NULL, rowSelection = NULL, transforms = NULL,
           transformObjects = NULL, transformFunc = NULL, transformVars = NULL,
           transformPackages = NULL, transformEnvir = NULL,
           blocksPerRead = rxGetOption("blocksPerRead"),
           histType = "Counts",
           title = NULL, subtitle = NULL, xTitle = NULL, yTitle = NULL,
           xNumTicks = NULL, yNumTicks = NULL, xAxisMinMax = NULL,
           yAxisMinMax = NULL, fillColor = "cyan", lineColor = "black",
           lineStyle = "solid", lineWidth = 1, plotAreaColor = "gray90",
           gridColor = "white", gridLineWidth = 1, gridLineStyle = "solid",
           maxNumPanels = 100, reportProgress = rxGetOption("reportProgress"),
           print = TRUE, ...)
```

Arguments

formula

formula describing the data to plot. It should take the form of `~x|g1 + g2` where `g1` and `g2` are optional conditioning factor variables and `x` is the name of a variable or an on-the-fly factorization `F(x)`. Other expressions of `x` are not supported.

data

either an RxXdfData object, a character string specifying the .xdf file, or a data frame containing the variable to plot.

pweights

character string specifying the variable to use as probability weights for the observations.

fweights

character string specifying the variable to use as frequency weights for the observations.

numBreaks

number of breaks to use to cut numeric data, including the upper and lower bounds.

startVal

low value used for cutting numeric data.

endVal

high value used for cutting numeric data.

`levelsToDrop`

levels to exclude if the histogram variable is a factor.

`levelsToKeep`

levels to keep if the histogram variable is a factor.

`rowSelection`

name of a logical variable in the data set (in quotes) or a logical expression using variables in the data set to specify row selection. For example, `rowSelection = "old"` will use only observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` will use only observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

`transforms`

an expression of the form `list(name = expression, ...)` representing the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

`transformObjects`

a named list containing objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

`transformFunc`

variable transformation function. See [rxTransform](#) for details.

`transformVars`

character vector of input data set variables needed for the transformation function. See [rxTransform](#) for details.

`transformPackages`

character vector defining additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, e.g., those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.

`transformEnvir`

user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

`blocksPerRead`

number of blocks to read for each chunk of data read from the data source.

`histType`

character string specifying `"Counts"` or `"Percent"`.

`title`

main title for the plot. Alternatively `main` can be used.

`subtitle`

subtitle (at the bottom) for the plot. Alternatively `sub` can be used.

`xTitle`

title for the X axis. Alternatively `xlab` can be used.

`yTitle`

title for the Y axis. Alternatively `ylab` can be used.

`xNumTicks`

number of tick marks on X axis (ignored for factor variables).

`yNumTicks`

number of tick marks on Y axis.

`xAxisMinMax`

numeric vector of length 2 containing a minimum and maximum value for the X axis. Alternatively `xlim` can be used.

`yAxisMinMax`

numeric vector of length 2 containing a minimum and maximum value for the Y axis. Alternatively `ylim` can be used.

`fillColor`

fill color for histogram. Use colors to see color names.

`lineColor`

line color for border of histogram.

`lineStyle`

line style for border of histogram: `"blank"`, `"solid"`, `"dashed"`, ```dotted"`, `"dotdash"`, `"longdash"`, or `"twodash"`.

`lineWidth`

line width for border of histogram. Alternatively `lwd` can be used.

`plotAreaColor`

background color for the plot area.

`gridColor`

color for grid lines.

`gridLineWidth`

line width for grid lines.

`gridLineStyle`

line style for grid lines.

`maxNumPanels`

integer specifying the maximum number of panels to plot. The number of panels is determined by the product of the number of levels of each conditioning variable. If the number of panels exceeds the `maxNumPanels` an error is given and the plot is not drawn. If `maxNumPanels` is `NULL`, it is ignored.

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`print`

logical. If `TRUE`, the plot is printed. If `FALSE`, and the `lattice` package is loaded, an `lattice` plot object is returned invisibly and can be printed later.

`...`

additional arguments to be passed directly to the underlying `barchart` or `xyplot` function.

Details

`rxHistogram` calls `rxCube` to perform computations and uses the `lattice` graphics package (`barchart` or `xyplot`) to create the plot. The `rxHistogram` function will attempt bin continuous data in reasonable intervals. For faster computation (using a bin for every integer value), use the `F()` function around the variable. Descriptive argument names are used to facilitate quick and easy plotting and self-documenting code for new R users.

Value

An object of class "trellis". It is automatically printed within the function.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

`rxLinePlot`, `rxCube`, `histogram`.

Examples

```

# Examples using airline data
airlineData <- file.path(rxGetOption("sampleDataDir"), "AirlineDemoSmall.xdf")
# Use the F() function to quickly compute bins for each integer level
rxHistogram(~F(CRSDepTime), data = airlineData)
# Specify the approximate number of breaks
rxHistogram(~CRSDepTime, numBreaks=11, data = airlineData)

# Examples using census data subsample
censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers")
# Create panels for each of the 3 states
rxHistogram(~ sex | state, data = censusWorkers)
# Repeat, printing x axis labels at an angle, and all panels in a row
rxHistogram(~ sex | state, scales = list(x = list(rot = 30)),
            data = censusWorkers, layout = c(3,1))
# Create panels for age for each sex for each state
rxHistogram(~ age | sex + state, data = censusWorkers)
# Specify how wage income should be broken into bins
rxHistogram(~ incwage | state + sex, title="Wage Income Up To 100,000",
            endVal = 100000, numBreaks=21, data = censusWorkers)

# Show panels for each state on a separate page
numCols <- 1
numRows <- 2
## Not run:

par(ask=TRUE) # Set ask to pause between each plot
## End(Not run)

rxHistogram(~ age | sex + state, data = censusWorkers, layout=c(numCols, numRows))

# Create a jpeg file for each page, named myplot001.jpeg, etc
## Not run:

jpeg(file="myplot
rxHistogram(~ age | sex + state, data = censusWorkers,
            blocksPerRead=6, layout=c(numCols, numRows))
dev.off()
## End(Not run)

```

RxHpcServer-class: Class RxHpcServer

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

DEPRECATED: HPC Server compute context class.

Generators

The targeted generator [RxHpcServer](#) as well as the general generator [RxComputeContext](#).

Extends

Class RxComputeContext, directly.

Methods

[show](#)

```
signature(object = "RxHpcServer") : ...
```

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

rxImport: Import Data to .xdf or data frame

7/12/2022 • 14 minutes to read • [Edit Online](#)

Description

Import data into an .xdf file or `data.frame`. `rxImport` is multi-threaded.

Usage

```
rxImport(inData, outFile = NULL, varsToKeep = NULL,
         varsToDrop = NULL, rowSelection = NULL,
         transforms = NULL, transformObjects = NULL,
         transformFunc = NULL, transformVars = NULL,
         transformPackages = NULL, transformEnvir = NULL,
         append = "none", overwrite = FALSE, numRows = -1,
         stringsAsFactors = NULL, colClasses = NULL, colInfo = NULL,
         rowsPerRead = NULL, type = "auto", maxRowsByCols = NULL,
         reportProgress = rxGetOption("reportProgress"),
         verbose = 0,
         xdfCompressionLevel = rxGetOption("xdfCompressionLevel"),
         createCompositeSet = NULL,
         blocksPerCompositeFile = 3,
         ...)
```

Arguments

`inData`

a character string with the path for the data to import (delimited, fixed format, SPSS, SAS, ODBC, or XDF).

Alternatively, a data source object representing the input data source can be specified. (See [RxTextData](#), [RxSasData](#), [RxSpssData](#), and [RxOdbcData](#).) If a character string is supplied and `type` is set to `"auto"`, the type of file is inferred from its extension, with the default being a text file. A `data.frame` can also be used for `inData`.

`outFile`

a character string representing the output .xdf file, a [RxHiveData](#) data source, a [RxParquetData](#) data source or a [RxXdfData](#) object. If `NULL`, a data frame will be returned in memory.

`varsToKeep`

character vector of variable names to include when reading from the input data file. If `NULL`, argument is ignored. Cannot be used with `varsToDrop`. Not supported for ODBC or fixed format text files.

`varsToDrop`

character vector of variable names to exclude when reading from the input data file. If `NULL`, argument is ignored. Cannot be used with `varsToKeep`. Not supported for ODBC or fixed format text files.

`rowSelection`

name of a logical variable in the data set (in quotes) or a logical expression using variables in the data set to specify row selection. For example, `rowSelection = "old"` will use only observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` will use only observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is

greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

`transforms`

an expression of the form `list(name = expression, ...)` representing the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

`transformObjects`

a named list containing objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

`transformFunc`

variable transformation function. See [rxTransform](#) for details.

`transformVars`

character vector of input data set variables needed for the transformation function. See [rxTransform](#) for details.

`transformPackages`

character vector defining additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, e.g., those explicitly defined in RevoScaleR functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.

`transformEnvir`

user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

`append`

either `"none"` to create a new .xdf file or `"rows"` to append rows to an existing .xdf file. If `outFile` exists and `append` is `"none"`, the `overwrite` argument must be set to `TRUE`. Ignored if a data frame is returned.

`overwrite`

logical value. If `TRUE`, the existing `outData` will be overwritten. Ignored if a data frame is returned.

`numRows`

integer value specifying the maximum number of rows to import. If set to -1, all rows will be imported.

`stringsAsFactors`

logical indicating whether or not to automatically convert strings to factors on import. This can be overridden by specifying `"character"` in `colClasses` and `colInfo`. If `TRUE`, the factor levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use `colInfo` with specified `"levels"`.

`colClasses`

character vector specifying the column types to use when converting the data. The element names for the vector are used to identify which column should be converted to which type.

- Allowable column types are:

- `"logical"` (stored as `uchar`),

- "integer" (stored as `int32`),
- "float32" (the default for floating point data for .xdf files),
- "numeric" (stored as `float64` as in R),
- "character" (stored as `string`),
- "factor" (stored as `uint32`),
- "ordered" (ordered factor stored as `uint32`. Ordered factors are treated the same as factors in RevoScaleR analysis functions.),
- "int16" (alternative to integer for smaller storage space),
- "uint16" (alternative to unsigned integer for smaller storage space),
- "Date" (stored as Date, i.e. `float64`. Not supported for import types `"textFast"`, `"fixedFast"`, or `"odbcFast"`.)
- "POSIXct" (stored as POSIXct, i.e. `float64`. Not supported for import types `"textFast"`, `"fixedFast"`, or `"odbcFast"`.)
- Note for `"factor"` and `"ordered"` types, the levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use `colInfo` with specified `"levels"`.
- Note that equivalent types share the same bullet in the list above; for some types we allow both 'R-friendly' type names, as well as our own, more specific type names for .xdf data.
- Note also that specifying the column as a "factor" type is currently equivalent to "string" - for the moment, if you wish to import a column as factor data you must use the `colInfo` argument, documented below.

`colInfo`

list of named variable information lists. Each variable information list contains one or more of the named elements given below. When importing fixed format data, either `colInfo` or an .sts schema file should be supplied. For fixed format text files, only the variables specified will be imported. For all text types, the information supplied for `colInfo` overrides that supplied for `colClasses`.

- Currently available properties for a column information list are:
- `type` - character string specifying the data type for the column. See `colClasses` argument description for the available types. If the `type` is not specified for fixed format data, it will be read as character data.
- `newName` - character string specifying a new name for the variable.
- `description` - character string specifying a description for the variable.
- `levels` - character vector containing the levels when `type = "factor"`. If the `levels` property is not provided, factor levels will be determined by the values in the source column. If levels are provided, any value that does not match a provided level will be converted to a missing value.
- `newLevels` - new or replacement levels specified for a column of type "factor". It must be used in conjunction with the `levels` argument. After reading in the original data, the labels for each level will be replaced with the `newLevels`.
- `low` - the minimum data value in the variable (used in computations using the `F()` function.)
- `high` - the maximum data value in the variable (used in computations using the `F()` function.)
- `start` - the left-most position, in bytes, for the column of a fixed format file respectively. When all elements of `colInfo` have `start`, the text file is designated as a fixed format file. When none of the elements have it, the text file is designated as a delimited file. Specification of `start` must always be accompanied by specification of `width`.
- `width` - the number of characters in a fixed-width character column or the column of a fixed format file. If `width` is specified for a character column, it will be imported as a fixed-width character variable. Any characters beyond the fixed width will be ignored. Specification of `width` is required for all columns of a

fixed format file.

- `decimalPlaces` - the number of decimal places.

`rowsPerRead`

number of rows to read at a time.

`type`

character string set specifying file type of `inData`. This is ignored if `inData` is a data source. Possible values are:

- `"auto"` : file type is automatically detected by looking at file extensions and argument values.
- `"textFast"` : delimited text import using faster, more limited import mode. By default variables containing the values `TRUE` and `FALSE` or `T` and `F` will be created as logical variables.
- `"text"` : delimited text import using enhanced, slower import mode (not supported with HDFS). This allows for importing Date and POSIXct data types, handling the delimiter character inside a quoted string, and specifying decimal character and thousands separator. (See [RxTextData](#).)
- `"fixedFast"` : fixed format text import using faster, more limited import mode. You must specify a .sts format file or collInfo specifications with `start` and `width` for each variable.
- `"fixed"` : fixed format text import using enhanced, slower import mode (not supported with HDFS). This allows for importing Date and POSIXct data types and specifying decimal character and thousands separator. You must specify a .sts format file or collInfo specifications with `start` and `width` for each variable.
- `"sas"` : SAS data files. (See [RxSasData](#).)
- `"spss"` : SPSS data files. (See [RxSpssData](#).)
- `"odbcFast"` : ODBC import using faster, more limited import mode.
- `"odbc"` : ODBC import using slower, enhanced import on Windows. (See [RxOdbcData](#).)

`maxRowsByCols`

the maximum size of a data frame that will be read in if `outData` is set to `NULL`, measured by the number of rows times the number of columns. If the number of rows times the number of columns being imported exceeds this, a warning will be reported and a smaller number of rows will be read in than requested. If `maxRowsByCols` is set to be too large, you may experience problems from loading a huge data frame into memory.

`reportProgress`

integer value with options:

- `0` : no progress is reported.
- `1` : the number of processed rows is printed and updated.
- `2` : rows processed and timings are reported.
- `3` : rows processed and all timings are reported.

`verbose`

integer value. If `0`, no additional output is printed. If `1`, information on the import type is printed if `type` is set to `auto`.

`xdfCompressionLevel`

integer in the range of -1 to 9. The higher the value, the greater the amount of compression - resulting in smaller files but a longer time to create them. If `xdfCompressionLevel` is set to 0, there will be no compression and files will be compatible with the 6.0 release of Revolution R Enterprise. If set to -1, a default level of compression will be used.

`createCompositeSet`

logical value or `NULL`. If `TRUE`, a composite set of files will be created instead of a single .xdf file. A directory will be created whose name is the same as the .xdf file that would otherwise be created, but with no extension. Subdirectories data and metadata will be created. In the data subdirectory, the data will be split across a set of .xdff files (see `blocksPerCompositeFile` below for determining how many blocks of data will be in each file). In the metadata subdirectory there is a single .xdfm file, which contains the meta data for all of the .xdff files in the data subdirectory. When the compute context is `RxHadoopMR` a composite set of files is always created.

`blocksPerCompositeFile`

integer value. If `createCompositeSet=TRUE`, and if the compute context is not `RxHadoopMR`, this will be the number of blocks put into each .xdff file in the composite set. When importing is being done on Hadoop using MapReduce, the number of rows per .xdff file is determined by the rows assigned to each MapReduce task, and the number of blocks per .xdff file is therefore determined by `rowsPerRead`. If the `outFile` is an `RxXdfData` object, set the value for `blocksPerCompositeFile` there instead.

...

additional arguments to be passed directly to the underlying data source objects to be imported. These argument values will override the existing values in an existing data source, if it is passed in as the `inData`. See [RxTextData](#), [RxSasData](#), [RxSpssData](#), and [RxOdbcData](#).

Details

If a data source is passed in as `inData`, argument values specified in the call to `rxImport` will override any existing specifications in the data source. Setting `type` to `"text"`, `"fixed"`, or `"odbc"` is equivalent to setting `useFastRead` to `FALSE` in an `RxTextData` or `RxOdbcData` input data source. Similarly, setting `type` to `"textFast"`, `"fixedFast"`, or `"odbcFast"` is equivalent to setting `useFastRead` to `TRUE`.

The input and output data sources are automatically opened and closed within the `rxImport` function.

For reasons of performance, the `textFast` 'type' does not properly handle text files that contain the delimiter character inside a quoted string (for example, the entry "Wade, John" inside a comma delimited file. Use the `text` 'type' if your data set contains character data with this characteristic.

For information on using a .sts schema file for fixed format text import, see the [RxTextData](#) help file.

Encoding Details

Some files or data sources contain character data encoded in a particular format (for example, Excel files will always use UTF-16). Others may contain encoding information inside the file itself. When not determined by the file type, or specified within the file, character data in the input file or data source must be encoded as ASCII or UTF-8 in order to be imported correctly. If the data contains UTF-8 multibyte (i.e., non-ASCII) characters, make sure the `type` parameter is set to `"text"` or `"fixed"` for text import and `"odbc"` for ODBC import as the 'fast' versions of these values may not handle extended UTF-8 characters correctly.

Value

If an `outFile` is not specified, an output data frame is returned. If an `outFile` is specified, an `RxXdfData` data source is returned that can be used in subsequent RevoScaleR analysis.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxDataSource-class](#), [rxDataStep](#), [RxTextData](#), [RxSasData](#), [RxSpssData](#), [RxOdbcData](#), [RxXdfData](#), [rxSplit](#), [rxTransform](#).

Examples

```
#####
# Import a comma-delimited text file into an .xdf file
#####
# Create names for input and output data files
sampleDataDir <- rxGetOption("sampleDataDir")
inputFile <- file.path(sampleDataDir, "claims.txt")
outputFile <- file.path(tempdir(), "claims.xdf")

# Import the data
rxImport( inData = inputFile, outFile = outputFile)

# Look at summary information
rxGetInfo( data = outputFile, getVarInfo = TRUE)

# Clean-up: remove data file
file.remove( outputFile )

#####
# Import a small SAS file into a data frame in memory
#####
# Specify a SAS file name
claimsSasFileName <- file.path(rxGetOption("sampleDataDir"), "claims.sas7bdat")

# Import the data into a data frame in memory
claimsIn <- rxImport(inData = claimsSasFileName)
head(claimsIn)

#####
# Import a fixed format text file into an .xdf file
#####
# Specify input and output file names

claimsFF <- file.path(rxGetOption("sampleDataDir"), "claims.dat")
claimsXdf <- file.path(tempdir(), "importedClaims.xdf")

# Specify column information about the input file
claimsFFColInfo <-
list(rownames = list(start = 1, width = 3),
     age = list(type = "factor", start = 4, width = 5),
     car.age = list(type = "factor", start = 9, width = 3),
     type = list(type = "factor", start = 12, width = 1),
     cost = list(type = "numeric", start = 13, width = 6),
     number = list(type = "integer", start = 19, width = 3))

# Import the data into the xdf file
rxImport(inData = claimsFF, outFile = claimsXdf,
        colInfo = claimsFFColInfo,
        rowsPerRead = 65, # rowsPerRead will determine block size
        overwrite = TRUE)

# Look at information about the new file
rxGetInfo( data = claimsXdf, getVarInfo = TRUE)

# Clean-up: delete the new file
file.remove( claimsXdf)

#####
# Import a fixed format text file using an RxTextData data source
#####
```

```

claimsFF <- file.path(rxGetOption("sampleDataDir"), "claims.dat")
claimsXdf <- file.path(tempdir(), "importedClaims.xdf")

# Create an RxTextData data source

claimsDS <- RxTextData( file = claimsFF,
  colInfo = list(
    rownames = list(start = 1, width = 3),
    age = list(type = "factor", start = 4, width = 5),
    car.age = list(type = "factor", start = 9, width = 3),
    type = list(type = "factor", start = 12, width = 1),
    cost = list(type = "numeric", start = 13, width = 6),
    number = list(type = "integer", start = 19, width = 3)),
    rowsPerRead = 50 # rowsPerRead will determine block size
  )

# Import the data specified in the data source into an xdf file

rxImport(inData = claimsDS, outFile = claimsXdf,
  rowsPerRead = 65, # this will override 'rowsPerRead' in the data source
  overwrite = TRUE)

# Clean-up: delete the new file
file.remove( claimsXdf)

#####
# Import a an SPSS file into an Xdf file
#####

# Specify SPSS file name
spssFileName <- file.path(rxGetOption("unitTestDataDir"), "claimsInt.sav")

# Create an XDF data file path
xdffileName <- file.path(tempdir(), "ClaimsInt.xdf")

# Import data; notice that factor variables are created for any
# SPSS variables with value labels
rxImport(inData = spssFileName, outFile = xdffileName, overwrite = TRUE, reportProgress = 0)
varInfo <- rxGetVarInfo( data = xdffileName )
varInfo

# Reimport SPSS data, storing some variables as integer instead of
# numeric

spssColClasses <- c(RowNum = "integer", cost = "integer",
  number = "integer")
rxImport(inData = spssFileName, outFile = xdffileName, colClasses = spssColClasses,
  overwrite = TRUE, reportProgress = 0)
varInfo <- rxGetVarInfo( data = xdffileName )
varInfo

# Reimport SPSS data, changing re-specifying the strings used
# for factor levels

spssColInfo <- list(
  RowNum = list(type="integer", newName = "RowNumber"),
  cost = list(type="integer", newName = "TotalCost", description = "Cost of Auto"),
  number = list(type="integer", newName = "Number", description = "Number of Claims"),
  type = list(
    levels = as.character(1:4),
    newLevels = c("Small", "Medium", "Large", "Huge")))
rxImport(inData = spssFileName, outFile = xdffileName,
  colInfo = spssColInfo, overwrite = TRUE, reportProgress = 0)
varInfo <- rxGetVarInfo( data = xdffileName )
varInfo

# Clean-up: delete the new file

```

```
file.remove( xdfFileName )
```

RxInSqlServer-class: Class RxInSqlServer

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Creates a compute context for running Microsoft R Server analyses inside Microsoft SQL Server.

Currently only supported in Windows.

Objects from the Class

Objects can be created by calls of the form .

Slots

`connectionString` :

Object of class `"character"` $\sim\sim$

`wait` :

Object of class `"logical"` $\sim\sim$

`consoleOutput` :

Object of class `"logical"` $\sim\sim$

`autoCleanup` :

Object of class `"logical"` $\sim\sim$

`dataPath` :

Object of class `"characterORNULL"` $\sim\sim$

`packagesToLoad` :

Object of class `"characterORNULL"` $\sim\sim$

`executionTimeout` :

Object of class `"numeric"` $\sim\sim$

`description` :

Object of class `"character"` $\sim\sim$

`version` :

Object of class `"character"` $\sim\sim$

Extends

Class `RxDistributedHpa`, directly. Class `RxComputeContext`, by class `"RxDistributedHpa"`, distance 2. Class `RxComputeContext`, by class `"RxLocalSeq"`, distance 1.

Methods

```
doPreJobValidation  
signature(object = "RxInSqlServer") : ...  
  
initialize  
signature(.Object = "RxInSqlServer") : ...  
  
show  
signature(object = "RxInSqlServer") : ...
```

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxInSqlServer](#), [RxSqlServerData](#)

Examples

```
showClass("RxInSqlServer")
```

RxInSqlServer: Generate SQL Server In-Database Compute Context

7/12/2022 • 4 minutes to read • [Edit Online](#)

Description

Creates a compute context for running RevoScaleR analyses inside Microsoft SQL Server.

Currently only supported in Windows.

Usage

```
RxInSqlServer(object, connectionString = "", numTasks = rxGetOption("numTasks"), autoCleanup = TRUE,  
  consoleOutput = FALSE, executionTimeoutSeconds = 0, wait = TRUE, packagesToLoad = NULL,  
  shareDir = NULL, server = NULL, databaseName = NULL, user = NULL, password = NULL, ... )
```

Arguments

`object`

An optional RxInSqlServer object.

`connectionString`

An ODBC connection string used to connect to the Microsoft SQL Server database.

`numTasks`

Number of tasks (processes) to run for each computation. This is the maximum number of tasks that will be used; SQL Server may start fewer processes if there is not enough data, if too many resources are already being used by other jobs, or if `numTasks` exceeds the MAXDOP (maximum degree of parallelism) configuration option in SQL Server. Each of the tasks is given data in parallel, and does computations in parallel, and so computation time may decrease as `numTasks` increases. However, that may not always be the case, and computation time may even increase if too many tasks are competing for machine resources. Note that

`rxOptions(numCoresToUse=n)` controls how many cores (actually, threads) are used in parallel within each process, and there is a trade-off between `numCoresToUse` and `numTasks` that depends upon the specific algorithm, the type of data, the hardware, and the other jobs that are running.

`wait`

logical value. If `TRUE`, the job will be blocking and will not return until it has completed or has failed. If `FALSE`, the job will be non-blocking and return immediately, allowing you to continue running other R code. The object `rxgLastPendingJob` is created with the job information. The client connection with SQL Server must be maintained while the job is running, even in non-blocking mode.

`consoleOutput`

logical scalar. If `TRUE`, causes the standard output of the R process started by SQL Server to be printed to the user console. This value may be overwritten by passing a non-`NULL` logical value to the `consoleOutput` argument provided in `rxExec` and `rxGetJobResults`.

`autoCleanup`

logical scalar. If `TRUE`, the default behavior is to clean up the temporary computational artifacts and delete the result objects upon retrieval. If `FALSE`, then the computational results are not deleted, and the results may be acquired using `rxGetJobResults`, and the output via `rxGetJobOutput` until the `rxCleanupJobs` is used to delete the results and other artifacts. Leaving this flag set to `FALSE` can result in accumulation of compute artifacts which you may eventually need to delete before they fill up your hard drive.

`executionTimeoutSeconds`

numeric scalar. Defaults to 0 which means infinite wait.

`packagesToLoad`

optional character vector specifying additional packages to be loaded on the nodes when jobs are run in this compute context.

`shareDir`

character string specifying the temporary directory on the client that is used to serialize the R objects back and forth. If not specified, a subdirectory under Absolute or paths relative to current directory can be specified.

`server`

Target SQL Server instance. Can also be specified in the connection string with the `Server` keyword.

`databaseName`

Database on the target SQL Server instance. Can also be specified in the connection string with the `Database` keyword.

`user`

SQL login to connect to the SQL Server instance. SQL login can also be specified in the connection string with the `uid` keyword.

`password`

Password associated with the SQL login. Can also be specified in the connection string with the `pwd` keyword.

`...`

additional arguments to be passed to the underlying function. Two useful additional arguments are `traceEnabled=TRUE` and `traceLevel=7`, which taken together enable run-time tracing of your in-SQL Server computations. `traceEnabled` and `traceLevel` are deprecated as of MRS 9.0.2 and will be removed from this compute context in the next major release. Please use `rxOptions(traceLevel=7)` to enable run-time tracing in-SQL Server.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxComputeContext](#), [RxInSqlServer-class](#), [RxSqlServerData](#), [rxOptions](#).

Examples

```

## Not run:

# In this example we connect to the database MyDatabase on the SQL Server instance MyServer using MyUser and
MyPassword as credentials.
# The query generates a rowset with 1000 rows and 2 columns:
# - The first column is all integers going from 1 to 1000.
# - The second column consists of random integers between 1 and 1000.

# Note: for improved security, read connection string from a file, such as
# connectionString <- readLines("connectionString.txt")

connectionString <- "Server=MyServer;Database=MyDatabase;UID=MyUser;PWD=MyPassword"
sqlQuery <- "WITH nb AS (SELECT 0 AS n UNION ALL SELECT n+1 FROM nb where n < 9) SELECT
n1.n+10*n2.n+100*n3.n+1 AS n, ABS(CHECKSUM(NewId()))
rxSummary(
  formula = ~ .,
  data = RxSqlServerData(sqlQuery = sqlQuery, connectionString = connectionString),
  computeContext = RxInSqlServer(connectionString = connectionString)
)

# Sample output:
# Number of valid observations: 1000
#
#   Name    Mean     StdDev    Min Max  ValidObs MissingObs
#   n      500.500  288.8194  1    1000  1000      0
#   value  504.582  295.5115  1    1000  1000      0
## End(Not run)

```

rxInstalledPackages: Installed Packages for Compute Context

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Find (or retrieve) details of installed packages for a compute context.

Usage

```
rxInstalledPackages(computeContext = NULL, allNodes = FALSE, lib.loc = NULL,  
                    priority = NULL, noCache = FALSE, fields = "Package",  
                    subarch = NULL)
```

Arguments

`computeContext`

an [RxComputeContext](#) or equivalent character string or `NULL`. If set to the default of `NULL`, the currently active compute context is used. Supported compute contexts are [RxInSqlServer](#) and [RxLocalSeq](#).

`allNodes`

logical.

`lib.loc`

a character vector describing the location of R library trees to search through, or `NULL`. The default value of `NULL` corresponds to checking the loaded namespace, then all libraries currently known in `.libPaths()`. In [RxInSqlServer](#) only `NULL` is supported.

`priority`

character vector or `NULL` (default). If non-null, used to select packages; `"high"` is equivalent to `c("base", "recommended")`. To select all packages without an assigned priority use `priority = "NA"`.

`noCache`

logical. If `TRUE`, do not use cached information, nor cache it.

`fields`

a character vector giving the fields to extract from each package's DESCRIPTION file, or `NULL`. If `NULL`, the following fields are used: `"Package"`, `"LibPath"`, `"Version"`, `"Priority"`, `"Depends"`, `"Imports"`, `"LinkingTo"`, `"Suggests"`, `"Enhances"`, `"License"`, `"License_is_FOSS"`, `"License_restricts_use"`, `"OS_type"`, `"MD5sum"`, `"NeedsCompilation"`, and `"Built"`. Unavailable fields result in `NA` values.

`subarch`

character string or `NULL`. If non-null and non-empty, used to select packages which are installed for that sub-architecture.

Details

This is a wrapper for `installed.packages`. See the help file for additional details. Note that `rxInstalledPackages` treats the `field` argument differently, only returning the `fields` specified in the argument.

Value

By default, a character vector of installed packages is returned. If `fields` is not set to `"Package"`, a matrix with one row per package is returned. The row names are the package names and the possible column names are `"Package"`, `"LibPath"`, `"Version"`, `"Priority"`, `"Depends"`, `"Imports"`, `"LinkingTo"`, `"Suggests"`, `"Enhances"`, `"License"`, `"License_is_FOSS"`, `"License_restricts_use"`, `"OS_type"`, `"MD5sum"`, `"NeedsCompilation"`, and `"Built"` (the R version the package was built under). Additional columns can be specified using the `fields` argument. If using a distributed compute context with the `allNodes` set to `TRUE`, a list of matrices from each node will be returned. In `RxInSqlServer` compute context multiple rows for a package will be returned if different versions of the same package is installed in different `"system"`, `"shared"` and `"private"` scopes.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxPackage](#), `installed.packages`, [rxFindPackage](#), [rxInstallPackages](#),
[rxRemovePackages](#), [rxSyncPackages](#), [rxSqlLibPaths](#),
require

Examples

```
#  
# Find the packages installed for the current compute context  
#  
myPackages <- rxInstalledPackages()  
myPackages  
  
#  
# Get full information about all the packages installed for the current compute context  
#  
myPackageInfo <- rxInstalledPackages(fields = NULL)  
myPackageInfo  
  
#  
# Get specific information about the installed packages  
#  
myPackageInfo <- rxInstalledPackages(fields = c("Package", "Version", "Built"))  
myPackageInfo  
  
## Not run:  
  
#  
# Find the packages installed on a SQL Server compute context  
#  
sqlServerCompute <- RxInSqlServer(connectionString =  
  "Driver=SQL Server;Server=myServer;Database=TestDB;Trusted_Connection=True;")  
sqlPackages <- rxInstalledPackages(computeContext = sqlServerCompute)  
sqlPackages  
## End(Not run)
```

rxInstallPackages: Install Packages for Compute Context

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Install R packages from repositories or install local files for the current session.

Usage

```
rxInstallPackages(pkgs, skipMissing = FALSE, repos =getOption("repos"), verbose =getOption("verbose"),
scope = "private", owner = '', computeContext = rxGetOption("computeContext"))
```

Arguments

pkgs

`character` vector of the names of packages whose current versions should be downloaded from the repositories. If `repos` = `NULL`, a character vector of file paths of .zip files containing binary builds of packages. (`http://` and `file://` URLs are also accepted and the files will be downloaded and installed from local copies. If you specify .zip files with `repos` = `NULL` with `RxComputeContext` compute context the .zip file paths should already be present on a folder.

skipMissing

`logical`. Applicable only for `RxInSqlServer` compute context. If `TRUE`, skips missing dependent packages for which otherwise an error is generated.

repos

`character` vector, the base URL(s) of the repositories to use. Can be `NULL` to install from local files, directories.

verbose

`logical`. If `TRUE`, "progress report" is given during installation of given packages.

scope

`character`. Applicable only for `RxInSqlServer` compute context. Should be either `"shared"` or `"private"`. `"shared"` installs the packages on per database shared location on SQL server which in turn can be used (referred) by multiple different users. `"private"` installs the packages on per database, per user private location on SQL server which is only accessible to the single user.

owner

`character`. Applicable only for `RxInSqlServer` compute context. This is generally empty `''` value. Should be either empty `''` or a valid SQL database user account name. Only users in `'db_owner'` role for a database can specify this value to install packages on behalf of other users.

computeContext

an `RxComputeContext` or equivalent character string or `NULL`. If set to the default of `NULL`, the currently active

compute context is used. Supported compute contexts are [RxInSqlServer](#), [RxLocalSeq](#).

Details

This is a simple wrapper for `install.packages`. For [RxInSqlServer](#) compute context the user specified as part of connection string is used for installing the packages if `owner` argument is empty. The user calling this function needs to be granted permissions by database owner by making them member of either '`rpkgs-shared`' or '`rpkgs-private`' database role. Users in '`rpkgs-shared`' role can install packages to '`"shared"`' location and '`"private"` location. Users in '`rpkgs-private`' role can only install packages '`"private"` location for their own use. To use the packages installed on the SQL server a user needs to be a member of at least the '`'rpkgs-users'`' role.

See the help file for additional details.

Value

Invisible `NULL`

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxPackage](#), `install.packages`, [rxFindPackage](#), [rxInstalledPackages](#), [rxRemovePackages](#), [rxSyncPackages](#),
[rxSqlLibPaths](#),
`require`

Examples

```
## Not run:

#
# create SQL compute context
#
sqlcc <- RxInSqlServer(connectionString = "Driver=SQL
Server;Server=myServer;Database=TestDB;Trusted_Connection=True;")

#
# list of packages to install
#
pkgs <- c("dplyr")

#
# Install a package and its dependencies into private scope
#
rxInstallPackages(pkgs = pkgs, verbose = TRUE, scope = "private", computeContext = sqlcc)

#
# Install a package and its dependencies into shared scope
#
rxInstallPackages(pkgs = pkgs, verbose = TRUE, scope = "shared", computeContext = sqlcc)

#
# Install a package and its dependencies into private scope for user1
#
rxInstallPackages(pkgs = pkgs, verbose = TRUE, scope = "private", owner = "user1", computeContext = sqlcc)

#
# Install a package and its dependencies into shared scope for user1
#
rxInstallPackages(pkgs = pkgs, verbose = TRUE, scope = "shared", owner = "user1", computeContext = sqlcc)
## End(Not run)
```

rxKmeans: K-Means Clustering

7/12/2022 • 8 minutes to read • [Edit Online](#)

Description

Perform k-means clustering on small or large data.

Usage

```
rxKmeans(formula, data,
          outFile = NULL, outColName = ".rxCluster",
          writeModelVars = FALSE, extraVarsToWrite = NULL,
          overwrite = FALSE, numClusters = NULL, centers = NULL,
          algorithm = "Lloyd", numStartRows = 0, maxIterations = 1000,
          numStarts = 1, rowSelection = NULL,
          transforms = NULL, transformObjects = NULL,
          transformFunc = NULL, transformVars = NULL,
          transformPackages = NULL, transformEnvir = NULL,
          blocksPerRead = rxGetOption("blocksPerRead"),
          reportProgress = rxGetOption("reportProgress"), verbose = 0,
          computeContext = rxGetOption("computeContext"),
          xdfCompressionLevel = rxGetOption("xdfCompressionLevel"), ...)

## S3 method for class `rxKmeans':
print (x, header = TRUE, ...)
```

Arguments

`formula`

formula as described in [rxFormula](#).

`data`

a data source object, a character string specifying a .xdf file, or a data frame object.

`outFile`

either an RxXdfData data source object or a character string specifying the .xdf file for storing the resulting cluster indexes. If `NULL`, then no cluster indexes are stored to disk. Note that in the case that the input data is a data frame, the cluster indexes are returned automatically. Note also that, if `rowSelection` is specified and not `NULL`, then `outFile` cannot be the same as the `data` since the resulting set of cluster indexes will generally not have the same number of rows as the original data source.

`outColName`

character string to be used as a column name for the resulting cluster indexes if `outFile` is not `NULL`. Note that make.names is used on `outColName` to ensure that the column name is valid. If the `outFile` is an `RxOdbcData` source, dots are first converted to underscores. Thus, the default `outColName` becomes `"X_rxCluster"`.

`writeModelVars`

logical value. If `TRUE`, and the output file is different from the input file, variables in the model will be written to the output file in addition to the cluster numbers. If variables from the input data set are transformed in the model, the transformed variables will also be written out.

`extraVarsToWrite`

`NULL` or character vector of additional variables names from the input data to include in the `outFile`. If `writeModelVars` is `TRUE`, model variables will be included as well.

`overwrite`

logical value. If `TRUE`, an existing `outFile` with an existing column named `outColName` will be overwritten.

`numClusters`

number of clusters `k` to create. If `NULL`, then the `centers` argument must be specified.

`centers`

a `k x p` numeric matrix containing a set of initial (distinct) cluster centers. If `NULL`, then the `numClusters` argument must be specified.

`algorithm`

character string defining algorithm to use in defining the clusters. Currently supported algorithms are `"Lloyd"`. This argument is case insensitive.

`numStartRows`

integer specifying the size of the sample used to choose initial centers. If 0, (the default), the size is chosen as the minimum of the number of observations or 10 times the number of clusters.

`maxIterations`

maximum number of iterations allowed.

`numStarts`

if `centers` is `NULL`, `k` rows are randomly selected from the data source for use as initial starting points. The `numStarts` argument defines the number of these random sets that are to be chosen and evaluated, and the best result is returned. If `numStarts` is 0, the first `k` rows in the data set are used. Random selection of rows is only supported for .xdf data sources using the native file system and data frames. If the .xdf file is compressed, the random sample is taken from a maximum of the first 5000 rows of data.

`rowSelection`

name of a logical variable in the data set (in quotes) or a logical expression using variables in the data set to specify row selection. For example, `rowSelection = "old"` will use only observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` will use only observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

`transforms`

an expression of the form `list(name = expression, ...)` representing the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

`transformObjects`

a named list containing objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

`transformFunc`

variable transformation function. See [rxTransform](#) for details.

transformVars

character vector of input data set variables needed for the transformation function. See [rxTransform](#) for details.

transformPackages

character vector defining additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, e.g., those explicitly defined in [RevoScaleR](#) functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.

transformEnvir

user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

blocksPerRead

number of blocks to read for each chunk of data read from the data source.

reportProgress

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

verbose

integer value. If `0`, no additional output is printed. If `1`, additional summary information is printed.

computeContext

a valid [RxComputeContext](#). The `RxSpark` and `RxHadoopMR` compute contexts distribute the computation among the nodes specified by the compute context; for other compute contexts, the computation is distributed if possible on the local computer.

xdfCompressionLevel

integer in the range of -1 to 9 indicating the compression level for the output data if written to an `.xdf` file. The higher the value, the greater the amount of compression - resulting in smaller files but a longer time to create them. If `xdfCompressionLevel` is set to 0, there will be no compression and files will be compatible with the 6.0 release of Revolution R Enterprise. If set to -1, a default level of compression will be used.

...

additional arguments to be passed directly to the Revolution Compute Engine.

x

object of class `rxKmeans`.

logical value. If `TRUE`, header information is printed.

Details

Performs scalable k-means clustering using the classical Lloyd algorithm.

For reproducibility when using random starting values, you can pass a random seed by specifying `seed= value`

as part of your call. See the Examples.

Value

An object of class "rxKmeans" which is a list with components:

`cluster`

A vector of integers indicating the cluster to which each point is allocated. This information is always returned if the data source is a data frame. If the data source is not a data frame and `outFile` is specified. i.e., not `NULL`, the cluster indexes are written/appended to the specified file with a column name as defined by `outColName`.

`centers`

matrix of cluster centers.

`withinss`

within-cluster sum of squares (relative to the center) for each cluster.

`totss`

total within-cluster sum of squares.

`tot.withinss`

sum of the `withinss` vector.

`betweenss`

between-cluster sum of squares.

`size`

number of points in each cluster.

`valid.obs`

number of valid observations.

`missing.obs`

number of missing observations.

`numIterations`

number iterations performed.

`params`

parameters sent to Microsoft R Services Compute Engine.

`formula`

formula as described in [rxFormula](#).

`call`

the matched call.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

Lloyd, S. P. (1957, 1982) Least squares quantization in PCM. Technical Note, Bell Laboratories. Published in 1982 in *IEEE Transactions on Information Theory* 28, 128-137.

See Also

kmeans.

Examples

```
# Create data
N <- 1000
sd1 <- 0.3
mean1 <- 0
sd2 <- 0.5
mean2 <- 1
set.seed(10)
data <- rbind(matrix(rnorm(N, sd = sd1, mean = mean1), ncol = 2),
               matrix(rnorm(N, mean = mean2, sd = sd2), ncol = 2))
colnames(data) <- c("x", "y")
DF <- data.frame(data)
XDF <- paste(tempfile(), "xdf", sep=".")
if (file.exists(XDF)) file.remove(XDF)
rxDataStep(inData = DF, outFile = XDF)

centers <- DF[sample.int(NROW(DF), 2, replace = TRUE),] # grab 2 random rows for starting

# Example using an XDF file as a data source
rxKmeans(~ x + y, data = XDF, centers = centers)

# Example using a local data frame file as a data source
z <- rxKmeans(~ x + y, data = DF, centers = centers)

# Show a plot of the results
# By design, the data in 2-space populates two groups of points centered about
# points (0,0) and (1,1). The spread about the mean is based on a random set of
# points drawn from a Gaussian distribution with standard deviations 0.3 and 0.5.
# As a visual, the resulting plot shows two circles drawn at the centers with radii
# equal to the corresponding standard deviations.
plot(DF, col = z$cluster, asp = 1,
      main = paste("Lloyd k-means Clustering: ", z$numIterations, "iterations"))
symbols(mean1, mean1, circle = sd1, inches = FALSE, add = TRUE, fg = "black", lwd = 2)
symbols(mean2, mean2, circle = sd2, inches = FALSE, add = TRUE, fg = "red", lwd = 2)
points(z$centers, col = 2:1, bg = 1:2, pch = 21, cex = 2) # big filled dots for centers

# Example using randomly selected rows from data source as initial centers
# but with seed set for reproducibility
## Not run:

z <- rxKmeans(~ x + y, data = DF, numClusters = 2, seed=18)
## End(Not run)

# Example using first rows from Spss data source as initial centers
spssFile <- file.path(rxGetOption("sampleDataDir"),"claims.sav")
spssDS <- RxSpssData(spssFile, colClasses = c(cost = "integer"))
resultSpss <- rxKmeans(~cost, data = spssDS, numClusters = 2, numStarts = 0)
```

rxLaunchClusterJobManager: Launches the job management UI for a compute context.

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Launches the job management UI (if available) for a compute context. Currently this is only available for RxHadoopMR and RxSpark compute contexts.

Usage

```
rxLaunchClusterJobManager(context=rxGetOption("computeContext"))
```

Arguments

`context`

The compute context used to determine which UI to launch.

Details

For RxHadoopMR, the job default tracker web page and dfs information page are launched for the cluster. Note that the dfs information page is generated from the `nameNode` slot of the `RxHadoopMR` compute context, and the `jobTrackerURL` slot of the `RxHadoopMR` is used as the URL for the job tracker. If one or both of these are not provided, a best attempt will be made based on the hostname provided in the `sshHostname` slot.

For other compute contexts, if compute context is not supplied, current compute context will be used. If no job management UI is available for the context, this function will report an error.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

Examples

```
## Not run:  
  
myCluster <- RxSpark(nameNode = "my-name-service-server", port = 8020)  
rxOptions(computeContext=myCluster)  
rxLaunchClusterJobManager(myCluster)  
## End(Not run)
```

rxLinePlot: Line Plot

7/12/2022 • 5 minutes to read • [Edit Online](#)

Description

Line or scatter plot using data from an .xdf file or data frame - a wrapper function for xyplot.

Usage

```
rxLinePlot(formula, data, rowSelection = NULL,  
          transforms = NULL, transformObjects = NULL,  
          transformFunc = NULL, transformVars = NULL,  
          transformPackages = NULL, transformEnvir = NULL,  
          blocksPerRead = rxGetOption("blocksPerRead"), type = c("l"),  
          title = NULL, subtitle = NULL, xTitle = NULL, yTitle = NULL,  
          xNumTicks = 10, yNumTicks = 10, legend = TRUE, lineColor = NULL,  
          lineStyle = "solid", lineWidth = 2, symbolColor = NULL,  
          symbolStyle = "solid circle", symbolSize = NULL,  
          plotAreaColor = "gray90", gridColor = "white", gridLineWidth = 1,  
          gridLineStyle = "solid",  
          reportProgress = rxGetOption("reportProgress"), print = TRUE, ...)
```

Arguments

formula

formula for plot, taking the form of `y~x|g1 + g2` where `g1` and `g2` are optional conditioning factor variables.

data

either an RxXdfData object, a character string specifying the .xdf file, or a data frame containing the data to plot.

rowSelection

name of a logical variable in the data set (in quotes) or a logical expression using variables in the data set to specify row selection. For example, `rowSelection = "old"` will use only observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` will use only observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the `expression` function.

transforms

an expression of the form `list(name = expression, ...)` representing variable transformations required for the formula or rowSelection. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the `expression` function.

transformObjects

a named list containing objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

transformFunc

variable transformation function. See [rxTransform](#) for details.

transformVars

character vector of input data set variables needed for the transformation function. See [rxTransform](#) for details.

transformPackages

character vector defining additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, e.g., those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.

transformEnvir

user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

blocksPerRead

number of blocks to read for each chunk of data read from the data source.

type

character vector specifying the type of plot. Use `"l"` for line, `"p"` for points, `"b"` for both, `"smooth"` for a loess fit, `"s"` for stair steps, or `"r"` for a regression line.

title

main title for the plot. Alternatively `main` can be used.

subtitle

subtitle (at the bottom) for the plot. Alternatively `sub` can be used.

xTitle

title for the X axis. Alternatively `xlab` can be used.

yTitle

title for the Y axis. Alternatively `ylab` can be used.

xNumTicks

number of tick marks for numeric X axis.

yNumTicks

number of tick marks for numeric Y axis.

legend

logical value. If `TRUE` and more than one line or set of symbols is plotted, a legend is created.

lineColor

character or integer vector specifying line colors for the plot. See [colors](#) for a list of available colors.

lineStyle

line style for line plot: `"blank"`, `"solid"`, `"dashed"`, `"dotted"`, `"dotdash"`, `"longdash"`, or `"twodash"`. Specify `"blank"` for no line, or set `type` to `"p"`.

lineWidth

a positive number specifying the line width for line plot. The interpretation is device-specific.

`symbolColor`

character or integer vector denoting the symbol colors. See `colors` for a list of available colors.

`symbolStyle`

character or integer vector denoting the symbol style(s): `"circle"`, `"solid circle"`, `"square"`, `"solid square"`, `"triangle"`, `"solid triangle"`, `"diamond"`, `"solid diamond"`, `"plus"` (3), `"cross"` (4), `"down triangle"` (6), `"square x"` (7), `"star"` (8), `"diamond +"` (9), `"circle +"` (10), `"up down triangle"` (11), `"square +"` (12), `"circle x"` (13), or any single character.

`symbolSize`

numerical value giving the amount by which symbols should be magnified.

`plotAreaColor`

background color for the plot area and legend.

`gridColor`

color for grid lines. See `colors` for a list of available colors.

`gridLineWidth`

line width for grid lines.

`gridLineStyle`

line style for grid lines: `"blank"`, `"solid"`, `"dashed"`, `"dotted"`, `"dotdash"`, `"longdash"`, or `"twodash"`.

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`print`

logical. If `TRUE`, the plot is printed. If `FALSE`, and the `lattice` package is loaded, an `xyplot` object is returned invisibly and can be printed later. Note that the printing of the legend or key will depend on the `trellis.par.getsettings` at the time of printing.

`...`

additional arguments to be passed directly to the underlying `xyplot` function if loaded.

Details

`rxLinePlot` is enhanced by the recommended `lattice` package.

Value

an `xyplot` object if the `lattice` is loaded. Otherwise `NULL`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[xyplot](#), [rxHistogram](#).

Examples

```
# Create a simple scatter plot using a data frame
rxLinePlot(Ozone ~ Temp, data=airquality, type="p")

# Create the file + path name for the sample data set
censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers")

# Compute the weighted counts for age and sex by state
ageSex <- rxCube(~ F(age) : sex : state, data = censusWorkers, pweights = "perwt")

# Convert the results to a data frame ready for plotting
ageSex <- rxResultsDF(ageSex)

# Draw a line plot of the results in 3 panels (with lattice package loaded)
rxLinePlot(Counts ~ age | state, groups = sex, data = ageSex,
           title = "2000 U.S. Workers by Age and Sex for 3 States")

# Use transformation expressions to scale the
# Counts and age variables
rxLinePlot(Counts ~ age | state, groups = sex, data = ageSex,
           transforms=list(Counts = log(Counts), age = age/10),
           xlab="age / 10", ylab="log(Counts)",
           title = "2000 U.S. Workers by Age and Sex for 3 States")

# Use both a line and symbols in the plot
rxLinePlot(Counts ~ age | state, groups = sex, data = ageSex, type = "b",
           title = "2000 U.S. Workers by Age and Sex for 3 States")

# Create a plot using a subselection of data from an .xdf file
# Look at Arrival Delay vs Departure Time for flights more
# than an hour late and less than 3 hours late on Tuesday
airlineData <- file.path(rxGetOption("sampleDataDir"), "AirlineDemoSmall")
rxLinePlot(ArrDelay~CRSDepTime, data=airlineData,
           rowSelection= ArrDelay > 60.0 & ArrDelay <= 180.00 & DayOfWeek == "Tuesday",
           type=c( "p", "smooth"), lineColor="red")

# Customize a plot with a panel function (using the lattice package)
if ("lattice" %in% .packages())
{
  rxLinePlot(Ozone ~ Temp, data = airquality, type = "p",
             panel = function(x, y, ...)
  {
    panel.spline(x,y, lwd = 3, col = 3)
  })
}
```

rxLinMod: Linear Models

7/12/2022 • 10 minutes to read • [Edit Online](#)

Description

Fit linear models on small or large data.

Usage

```
rxLinMod(formula, data, pweights = NULL, fweights = NULL, cube = FALSE,
         cubePredictions = FALSE, variableSelection = list(),
         rowSelection = NULL, transforms = NULL, transformObjects = NULL,
         transformFunc = NULL, transformVars = NULL,
         transformPackages = NULL, transformEnvir = NULL,
         dropFirst = FALSE, dropMain = rxGetOption("dropMain"),
         covCoef = FALSE, covData = FALSE,
         coefLabelStyle = rxGetOption("coefLabelStyle"),
         blocksPerRead = rxGetOption("blocksPerRead"),
         reportProgress = rxGetOption("reportProgress"), verbose = 0,
         computeContext = rxGetOption("computeContext"),
         ...)
```

Arguments

`formula`

formula as described in [rxFormula](#).

`data`

either a data source object, a character string specifying a .xdf file, or a data frame object.

`pweights`

character string specifying the variable to use as probability weights for the observations.

`fweights`

character string specifying the variable to use as frequency weights for the observations.

`cube`

logical flag. If `TRUE` and the first term of the predictor variables is categorical (a factor or an interaction of factors), the regression is performed by applying the Frisch-Waugh-Lovell Theorem, which uses a partitioned inverse to save on computation time and memory. See Details section below.

`cubePredictions`

logical flag. If `TRUE` and `cube` is `TRUE` the predicted values are computed and included in the `countDF` component of the returned value. This may be memory intensive. See Details section below.

`variableSelection`

a list specifying various parameters that control aspects of stepwise regression. If it is an empty list (default), no stepwise model selection will be performed. If not, stepwise regression will be performed and `cube` must be `FALSE`. See [rxStepControl](#) for details.

rowSelection

name of a logical variable in the data set (in quotes) or a logical expression using variables in the data set to specify row selection. For example, `rowSelection = "old"` will use only observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` will use only observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

transforms

an expression of the form `list(name = expression, ...)` representing the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

transformObjects

a named list containing objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

transformFunc

variable transformation function. The variables used in the transformation function must be specified in `transformVars` if they are not variables used in the model. See [rxTransform](#) for details.

transformVars

character vector of input data set variables needed for the transformation function. See [rxTransform](#) for details.

transformPackages

character vector defining additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, e.g., those explicitly defined in RevoScaleR functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.

transformEnvir

user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

dropFirst

logical flag. If `FALSE`, the last level is dropped in all sets of factor levels in a model. If that level has no observations (in any of the sets), or if the model as formed is otherwise determined to be singular, then an attempt is made to estimate the model by dropping the first level in all sets of factor levels. If `TRUE`, the starting position is to drop the first level. Note that for cube regressions, the first set of factors is excluded from these rules and the intercept is dropped.

dropMain

logical value. If `TRUE`, main-effect terms are dropped before their interactions.

covCoef

logical flag. If `TRUE` and if `cube` is `FALSE`, the variance-covariance matrix of the regression coefficients is returned. Use the [rxCovCoef](#) function to obtain these data.

covData

logical flag. If `TRUE` and if `cube` is `FALSE` and if constant term is included in the formula, then the variance-

covariance matrix of the data is returned. Use the `rxCovData` function to obtain these data.

`coefLabelStyle`

character string specifying the coefficient label style. The default is "Revo". If "R", R-compatible labels are created.

`blocksPerRead`

number of blocks to read for each chunk of data read from the data source.

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`verbose`

integer value. If `0`, no additional output is printed. If `1`, additional summary information is printed.

`computeContext`

a valid `RxComputeContext`. The and `RxHadoopMR` compute context distributes the computation among the nodes specified by the compute context; for other compute contexts, the computation is distributed if possible on the local computer.

`...`

additional arguments to be passed directly to the Revolution Compute Engine.

Details

The special function `F()` can be used in `formula` to force a variable to be interpreted as a factor.

When `cube` is `TRUE`, the Frisch-Waugh-Lovell (FWL) Theorem is applied to the model. The FWL approach parameterizes the model to include one coefficient for each category (a single factor level or combination of factor levels) instead of using an intercept in the model with contrasts for each of the factor combinations. Additionally when `cube` is `TRUE`, the output contains a `countDF` element representing the counts for each category. If `cubePredictions` is also `TRUE`, predicted values using means of conditional independent continuous variables and weighted coefficients of conditional independent categorical variables are also computed and included in `countDF`. This may be memory intensive. If there are no conditional independent variables (outside of the cube), the predicted values are equivalent to the coefficients and will be included in `countDF` whenever `cube` is `TRUE`. Regardless of the setting for `cube`, the null model for the F-test of global significance is always the intercept-only model.

The `dropFirst` and `dropMain` arguments are provided primarily as a convenience to users comparing `rxLinMod` results to those of `lm`. While `rxLinMod` will sometimes drop main effects while retaining interactions involving those terms, `lm` will not. Setting `dropMain=FALSE` will give results more akin to those of `lm`. Similarly, `lm` defaults to using treatment contrasts, which essentially drop the first level of each factor from the finished model. On the other hand, `rxLinMod` by default uses a set of contrasts that drop the last level of each factor. Setting `dropFirst=TRUE` will give results more akin to those of `lm`.

Value

Let P be the number of regression coefficients returned for each dependent variable, $y(n)$ for $n=1, \dots, N$, specified in the regression model. Let X be the linear regression design matrix. The `rxLinMod` function returns

an object of class rxLinMod, which is a list containing the following elements:

`coefficients`

P x N numeric matrix containing the regression coefficients.

`covCoef`

variance-covariance matrix for the regression coefficient estimates.

`covData`

variance-covariance matrix for the explanatory variables in the regression model.

`residual.squares`

the sum of the squares of the residuals.

`condition.number`

numeric scalar representing the estimated reciprocal condition number of $X'X$ (moment or crossprod) matrix.

`rank`

integer scalar denoting the numeric rank of the fitted linear model.

`aliased`

logical vector specifying whether columns were dropped or not due to collinearity.

`coef.std.error`

P x N numeric matrix containing the standard errors of the regression coefficients.

`coef.t.value`

P x N numeric matrix containing the t-statistics for the regression coefficients.

`coef.p.value`

P x N numeric matrix containing the p-values for the t-stats ($\text{Pr}(>|t|)$)

`total.squares`

N element numeric vector whose nth element is defined by $Y'(n)Y(n)$ for $n=1,\dots,N$.

`y.var`

N element numeric vector whose nth element is defined by $(Y(n) - E\{Y(n)\})'(Y(n) - E\{Y(n)\})$, i.e., the mean deviation of each dependent variable.

`sigma`

N element numeric vector of standard error of residuals.

`residual.variance`

the variance of the residuals.

`r.squared`

N element numeric vector containing r-squared, the fraction of variance explained by the model.

`f.pvalue`

the p-value resulting from an F-test on the fitted model.

`df`

degrees of freedom, a 3-element vector (p, n-p, p*), the last being the number of non-aliased coefficients.

`y.names`

N element character vector containing the names of the dependent variables in the specified model.

`partitions`

when `cube` is `TRUE`, partitioned results will also be returned.

`fstatistics`

(for models including non-intercept terms) a list containing the named elements: `value`: an N element numeric vector of F-statistic values, `numdf`: corresponding numerator degrees of freedom and `dendf`: corresponding denominator degrees of freedom.

`adj.r.squared`

R-squared statistic 'adjusted', penalizing for higher p.

`params`

parameters sent to Microsoft R Services Compute Engine.

`formula`

the model formula. For stepwise regression, this is the final model selected.

`call`

the matched call.

`countDF`

when `cube` is `TRUE`, a data frame containing counts information for each cube. If `cubePredictions` is also `TRUE`, predicted values for each group in the cube are included.

`nValidObs`

number of valid observations.

`nMissingObs`

number of missing observations.

`deviance`

minus twice the maximized log-likelihood (up to a constant)

`anova`

for stepwise regression, a data frame corresponding to the steps taken in the search.

`formulaBase`

for stepwise regression, the base model from which the search is started.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

Frisch, Ragnar; Waugh, Frederick V., Partial Time Regressions as Compared with Individual Trends, *Econometrica*, 1 (4) (Oct., 1933), pp. 387-401.

Lovell, M., 1963, Seasonal adjustment of economic time series, *Journal of the American Statistical Association*, 58, pp. 993-1010.

See Also

[lm](#), [rxLogit](#), [rxTransform](#).

Examples

```
# Compare rxLinMod and lm, which produce similar results when contr.SAS is used
form <- Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width + Species
irisLinMod <- rxLinMod(form, data = iris)
irisLinMod
summary(irisLinMod)
irisLM <- lm(form, data = iris, contrasts = list(Species = contr.SAS))
summary(irisLM)

# Instead of using the equivalent of contr.SAS, estimate the parameters
# for the categorical levels without contrasting against an intercept term.
# The null model for the global F-test remains the intercept-only model.
irisCubeLinMod <-
  rxLinMod(Sepal.Length ~ Species + Sepal.Width + Petal.Length + Petal.Width,
            data = iris, cube = TRUE)
summary(irisCubeLinMod)

# Use the Sample Census Data
censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers")
censusLinMod <- rxLinMod(wkswork1 ~ age:sex, data = censusWorkers,
                         pweights = "perwt")
censusLinMod
censusSubsetLinMod <- rxLinMod(wkswork1 ~ age:sex, data = censusWorkers,
                                 pweights = "perwt", rowSelection = age > 39)
censusSubsetLinMod

# Use the Sample Airline Data and report progress during computations
sampleDataDir <- rxGetOption("sampleDataDir")
airlineDemoSmall <- file.path(sampleDataDir, "AirlineDemoSmall.xdf")
airlineLinMod <- rxLinMod(ArrDelay ~ CRSDepTime, data = airlineDemoSmall,
                           reportProgress = 1)
airlineLinMod <- rxLinMod(ArrDelay ~ CRSDepTime, data = airlineDemoSmall,
                           reportProgress = 2)
airlineLinMod <- rxLinMod(ArrDelay ~ CRSDepTime, data = airlineDemoSmall,
                           reportProgress = 2, blocksPerRead = 3)
summary(airlineLinMod)

# Create a local data.frame and define a transformation
# function to be applied to the data prior to processing.
myDF <- data.frame(sex = c("Male", "Male", "Female", "Male"),
                    age = c(20, 20, 12, 15), score = 1.1:4.1, sport=c(1:3,2))

# define variable transformation list. Wrap with expression()
# so that it is not evaluated upon assignment.
transforms <- expression(list(
  revage = rev(age),
  division = factor(c("A","B","B","A")),
  sport = factor(sport, labels=c("tennis", "golf", "football"))))

# Both user-defined transform and arithmetic expressions in formula.
rxLinMod(score ~ sin(revage) + sex, data = myDF, transforms = transforms)

# User-defined transform only.
rxLinMod(revage ~ division + sport, data = myDF, transforms = transforms)

# Arithmetic formula expression only.
rxLinMod(log(score) ~ sin(age) + sex, data = myDF)
```

```
# No variable transformations.  
rxLinMod(score ~ age + sex, data = myDF)  
  
# use multiple dependent variables in model formula  
# print and summarize results for comparison  
sampleDataDir <- rxGetOption("sampleDataDir")  
airlineDemoSmall <- file.path(sampleDataDir, "AirlineDemoSmall")  
airlineLinMod1 <- rxLinMod(ArrDelay ~ DayOfWeek, data = airlineDemoSmall)  
airlineLinMod2 <- rxLinMod(cbind(ArrDelay, CRSDepTime) ~ DayOfWeek,  
                           data = airlineDemoSmall)  
airlineLinMod3 <-  
  rxLinMod(cbind(pow(ArrDelay, 2), ArrDelay, CRSDepTime) ~ DayOfWeek,  
            data = airlineDemoSmall)  
airlineLinMod4 <- rxLinMod(pow(ArrDelay, 2) ~ DayOfWeek,  
                           data = airlineDemoSmall)  
airlineLinMod1  
airlineLinMod2  
airlineLinMod3  
airlineLinMod4  
summary(airlineLinMod2)
```

RxLocalParallel-class: Class RxLocalParallel

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Class for the RevoScaleR Local Parallel Compute Context.

Generators

The targeted generator [RxLocalParallel](#) as well as the general generator [RxComputeContext](#).

Extends

Class RxComputeContext, directly.

Methods

`show`

`signature(object = "RxLocalParallel") : ...`

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxHadoopMR](#), [RxSpark](#), [RxInSqlServer](#), [RxLocalSeq](#).

Examples

```
## Not run:  
  
myComputeContext <- RxComputeContext("RxLocalParallel")  
is(myComputeContext, "RxComputeContext")  
# [1] TRUE  
is(myComputeContext, "RxLocalParallel")  
# [1] TRUE  
## End(Not run)
```

RxLocalParallel: Generate Local Parallel Compute Context

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Creates a local compute context object that uses the doParallel back-end for HPC computations performed using rxExec. This compute context can be used only to distribute computations via the `rxExec` function; it is ignored by Revolution HPA functions. This is the main generator for S4 class RxLocalParallel.

Usage

```
RxLocalParallel( object, dataPath = NULL, outDataPath = NULL )
```

Arguments

`object`

a compute context object. If `object` has slots for `dataPath` and/or `outDataPath`, they will be copied to the equivalent slots for the new `RxLocalParallel` object. Explicit specifications of the `dataPath` and/or `outDataPath` arguments will override this.

`dataPath`

`NULL` or character vector defining the search path(s) for the input data source(s). If not `NULL`, it overrides any specification for `dataPath` in `rxOptions`

`outDataPath`

`NULL` or character vector defining the search path(s) for new output data file(s). If not `NULL`, this overrides any specification for `dataPath` in `rxOptions`

Details

A job is associated with the compute context in effect at the time the job was submitted. If the compute context subsequently changes, the compute context of the job is not affected.

Note that `dataPath` and `outDataPath` are only utilized by data sources used in **RevoScaleR** analyses. They do not alter the working directory for other R functions that read from or write to files.

Value

object of class `RxLocalParallel`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

`rxSetComputeContext`, `rxExec`, `rxOptions`, `RxComputeContext`, `RxLocalSeq`, `RxForeachDoPar`, `RxLocalParallel-class`.

Examples

```
## Not run:

# Create a RxLocalParallel object to use with rxExec
parallelContext <- RxLocalParallel()

# Tell RevoScaleR to use parallelContext compute context
rxSetComputeContext( parallelContext )

# Subsequent calls to rxExec will use the doParallel package
# behind the scenes.

## End(Not run)
```

RxLocalSeq-class: Class RxLocalSeq

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Class for the RevoScaleR Local Compute Context.

Generators

The targeted generator [RxLocalSeq](#) as well as the general generator [RxComputeContext](#).

Extends

Class RxComputeContext, directly.

Methods

`show`

`signature(object = "RxLocalSeq") : ...`

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxSpark](#), [RxHadoopMR](#), [RxInSqlServer](#), [RxLocalParallel](#), [RxForeachDoPar](#), [RxComputeContext](#), [rxSetComputeContext](#).

Examples

```
## Not run:  
  
myComputeContext <- RxComputeContext("RxLocalSeq")  
is(myComputeContext, "RxComputeContext")  
# [1] TRUE  
is(myComputeContext, "RxLocalSeq")  
# [1] TRUE  
## End(Not run)
```

RxLocalSeq: Generate Local Compute Context

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Creates a local compute context object.

This is the main generator for S4 class RxLocalSeq. Computations using rxExec will be processed sequentially.

This is the default compute context.

Usage

```
RxLocalSeq( object, dataPath = NULL, outDataPath = NULL )
```

Arguments

`object`

a compute context object. If `object` has slots for `dataPath` and/or `outDataPath`, they will be copied to the equivalent slots for the new `RxLocalSeq` object. Explicit specifications of the `dataPath` and/or `outDataPath` arguments will override this.

`dataPath`

`NULL` or character vector defining the search path(s) for the input data source(s). If not `NULL`, it overrides any specification for `dataPath` in `rxOptions`

`outDataPath`

`NULL` or character vector defining the search path(s) for new output data file(s). If not `NULL`, this overrides any specification for `dataPath` in `rxOptions`

Details

A job is associated with the compute context in effect at the time the job was submitted. If the compute context subsequently changes, the compute context of the job is not affected.

Note that `dataPath` and `outDataPath` are only utilized by data sources used in **RevoScaleR** analyses. They do not alter the working directory for other R functions that read from or write to files.

Value

object of class RxLocalSeq.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxSetComputeContext](#), [rxExec](#), [rxOptions](#), [RxComputeContext](#), [RxLocalParallel](#), [RxLocalSeq-class](#).

Examples

```
## Not run:

# Create a local compute context object
localContext <- RxLocalSeq()

# Tell RevoScaleR to use localContext compute context
rxSetComputeContext( localContext )

## End(Not run)
```

rxLocateFile: Find File on a Given Data Path

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Obtain the normalized absolute path to the first occurrence of the specified input file in the specified set of paths.

Usage

```
rxLocateFile(file, pathsToSearch = NULL, fileSystem = NULL,  
            isOutFile = FALSE, defaultExt = ".xdf", verbose = 0)
```

Arguments

`file`

Character string defining path to a file or a data source containing a file name.

`pathsToSearch`

character vector of search paths. If `NULL`, the search path determined by `isOutFile` is used. Only the paths listed are searched; the list is not recursive.

`fileSystem`

`NULL`, character string or `RxFileSystem` object indicating type of file system; `"native"` or `RxNativeFileSystem` object can be used for the local operating system, or an `RxHdfsFileSystem` object for the Hadoop file system. If `NULL`, the active compute context will be checked for a valid file system. If not available, `rxGetOption("fileSystem")` will be used to obtain the file system. If `file` is a data source containing a file system, that file system will take precedent.

`isOutFile`

logical value. If `TRUE`, search using `outDataPath` instead of `dataPath` in `rxOptions` or in the local compute context (e.g.,`RxLocalSeq`).

`defaultExt`

character string containing default extension to use if extension is missing from `file`.

`verbose`

integer value. If `0`, no additional output is printed. If `1`, information on the file to be located is printed.

Details

The `file` path may be relative, for example, `"one/two/myfile.xdf"` or absolute, for example, `"C:/one/two/myfile.xdf"`. If `file` is an absolute path, the existence of the file is verified and, if found, the absolute path is returned in normalized form. If `file` is a relative path, the current working directory and then any paths specified in `pathsToSearch` are prepended to `file` and a search for the file along the concatenated paths is performed. The first path where the file is found is returned in normalized form. If the file does not exist an error is thrown.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

Examples

```
## Not run:  
rxLocateFile( "myFile.xdf" )  
## End(Not run)
```

rxLogit: Logistic Regression

7/12/2022 • 9 minutes to read • [Edit Online](#)

Description

Use `rxLogit` to fit logistic regression models for small or large data.

Usage

```
rxLogit(formula, data, pweights = NULL, fweights = NULL, cube = FALSE,
        cubePredictions = FALSE, variableSelection = list(), rowSelection = NULL,
        transforms = NULL, transformObjects = NULL,
        transformFunc = NULL, transformVars = NULL,
        transformPackages = NULL, transformEnvir = NULL,
        dropFirst = FALSE, dropMain = rxGetOption("dropMain"),
        covCoef = FALSE, covData = FALSE,
        initialValues = NULL,
        coefLabelStyle = rxGetOption("coefLabelStyle"),
        blocksPerRead = rxGetOption("blocksPerRead"),
        maxIterations = 25, coeffTolerance = 1e-06,
        gradientTolerance = 1e-06, objectiveFunctionTolerance = 1e-08,
        reportProgress = rxGetOption("reportProgress"), verbose = 0,
        computeContext = rxGetOption("computeContext"),
        ...)
```

Arguments

formula

formula as described in [rxFormula](#). Dependent variable must be binary. It can be a logical variable, a factor with only two categories, or a numeric variable with values in the range (0,1). In the latter case it will be converted to a logical.

data

either a data source object, a character string specifying a .xdf file, or a data frame object.

pweights

character string specifying the variable to use as probability weights for the observations.

fweights

character string specifying the variable to use as frequency weights for the observations.

cube

logical flag. If `TRUE` and the first term of the predictor variables is categorical (a factor or an interaction of factors), the regression is performed by applying the Frisch-Waugh-Lovell Theorem, which uses a partitioned inverse to save on computation time and memory. See Details section below.

cubePredictions

logical flag. If `TRUE` and `cube` is `TRUE` the estimated model is evaluated (predicted) for each cell in the cube, fixing the non-cube variables in the model at their mean values, and these predictions are included in the `countDF` component of the returned value. This may be time and memory intensive for large models. See

Details section below.

variableSelection

a list specifying various parameters that control aspects of stepwise regression. If it is an empty list (default), no stepwise model selection will be performed. If not, stepwise regression will be performed and `cube` must be `FALSE`. See [rxStepControl](#) for details.

rowSelection

name of a logical variable in the data set (in quotes) or a logical expression using variables in the data set to specify row selection. For example, `rowSelection = "old"` will use only observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` will use only observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

transforms

an expression of the form `list(name = expression, ...)` representing the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

transformObjects

a named list containing objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

transformFunc

variable transformation function. See [rxTransform](#) for details.

transformVars

character vector of input data set variables needed for the transformation function. See [rxTransform](#) for details.

transformPackages

character vector defining additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, e.g., those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.

transformEnvir

user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

dropFirst

logical flag. If `FALSE`, the last level is dropped in all sets of factor levels in a model. If that level has no observations (in any of the sets), or if the model as formed is otherwise determined to be singular, then an attempt is made to estimate the model by dropping the first level in all sets of factor levels. If `TRUE`, the starting position is to drop the first level. Note that for cube regressions, the first set of factors is excluded from these rules and the intercept is dropped.

dropMain

logical value. If `TRUE`, main-effect terms are dropped before their interactions.

`covCoef`

logical flag. If `TRUE` and if `cube` is `FALSE`, the variance-covariance matrix of the regression coefficients is returned. Use the `rxCovCoef` function to obtain these data.

`covData`

logical flag. If `TRUE` and if `cube` is `FALSE` and if constant term is included in the formula, then the variance-covariance matrix of the data is returned. Use the `rxCovData` function to obtain these data.

`initialValues`

Starting values for the Iteratively Reweighted Least Squares algorithm used to estimate the model coefficients. Supported values for this argument come in four forms:

- `NA` - Initial values of the parameters are computed by a weighted least squares step in which, if there is no user-specified weight, the expected value (`mu`) of the dependent variable is set to 0.75 if the binary dependent variable is 1, and to 0.25 if it is 0, and to a correspondingly weighted value otherwise. This is equivalent to the default option for the `glm` function's logistic regression initial values, and can sometimes lead to convergence when the default `NULL` option does not, though it may result in more iterations in other cases. If convergence fails with the default `NULL` option, estimation is restarted with `NA`, and vice versa.
- `NULL` - (Default) The initial values will be estimated based on a linear regression. This can speed up convergence significantly in many cases. If the model fails to converge using these values, estimation is automatically re-started using the `NA` option for the initial values.
- `Scalar` - A numeric scalar that will be replicated once for each of the coefficients in the model to form the initial values. Zeros are often good starting values.
- `Vector` - A numeric vector of length `k`, where `k` is the number of model coefficients, including the intercept if any and including any that might be dropped from sets of dummies. This argument is most useful when a set of estimated coefficients is available from a similar set of data. Note that `k` can be found by running the model on a small number of observations and obtaining the number of coefficients from the output, say `z`, via `length(z$coefficients)`.

`coefLabelStyle`

character string specifying the coefficient label style. The default is "Revo". If "R", R-compatible labels are created.

`blocksPerRead`

number of blocks to read for each chunk of data read from the data source.

`maxIterations`

maximum number of iterations.

`coeffTolerance`

convergence tolerance for coefficients. If the maximum absolute change in the coefficients (step), divided by the maximum absolute coefficient value, is less than or equal to this tolerance at the end of an iteration, the estimation is considered to have converged. To disable this test, set this value to 0.

`gradientTolerance`

This argument is deprecated. Use `objectiveFunctionTolerance` and `coeffTolerance` to control convergence.

`objectiveFunctionTolerance`

convergence tolerance for the objective function. If the absolute relative change in the deviance (-2.0 times log likelihood) is less than or equal to this tolerance at the end of an iteration, the estimation is considered to have converged. To disable this test, set this value to 0.

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`verbose`

integer value. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information are provided.

`computeContext`

a valid `RxComputeContext`. The `RxSpark` and `RxHadoopMR` compute contexts distribute the computation among the nodes specified by the compute context; for other compute contexts, the computation is distributed if possible on the local computer.

`...`

additional arguments to be passed directly to the Revolution Compute Engine.

Details

The special function `F()` can be used in `formula` to force a variable to be interpreted as a factor.

When `cube` is `TRUE`, the Frisch-Waugh-Lovell (FWL) Theorem is applied to the model. The FWL approach parameterizes the model to include one coefficient for each category (a single factor level or combination of factor levels) instead of using an intercept in the model with contrasts for each of the factor combinations.

Additionally when `cube` is `TRUE`, the output contains a `countDF` element representing the counts for each category. If `cubePredictions` is also `TRUE`, predicted values using means of conditional independent continuous variables and weighted coefficients of conditional independent categorical variables are also computed and included in `countDF`. This may be time and memory intensive for large models. If there are no conditional independent variables (outside of the cube), the predicted values are equivalent to the coefficients and will be included in `countDF` whenever `cube` is `TRUE`.

Value

an `rxLogit` object containing the following elements:

`coefficients`

named vector of coefficients.

`covCoef`

variance-covariance matrix for the regression coefficient estimates.

`covData`

variance-covariance matrix for the explanatory variables in the regression model.

`condition.number`

estimated reciprocal condition number of final weighted cross-product ($X'WX$) matrix.

`rank`

numeric rank of the fitted linear model.

`aliased`

logical vector specifying whether columns were dropped or not due to collinearity.

`coef.std.error`

standard errors of the coefficients.

`coef.t.value`

coefficients divided by their standard errors.

`coef.p.value`

p-values for `coef.t.values`, using the normal distribution ($\Pr(|z|)$)

`total.squares`

$Y'Y$ of raw Y's

`f.pvalue`

the p-value resulting from an F-test on the fitted model.

`df`

degrees of freedom, a 3-vector ($p, n-p, p^*$), the last being the number of non-aliased coefficients.

`y.names`

names for dependent variables.

`partitions`

when `cube` is `TRUE`, partitioned results will also be returned.

`fstatistics`

(for models including non-intercept terms) a 3-vector with the value of the F-statistic with its numerator and denominator degrees of freedom.

`params`

parameters sent to Microsoft R Services Compute Engine.

`formula`

the model formula

`call`

the matched call.

`countDF`

when `cube` is `TRUE`, a data frame containing category counts for the cube portion will also be returned. If `cubePredictions` is also `TRUE`, predicted values for each group in the cube are included.

`nValidObs`

number of valid observations.

`nMissingObs`

number of missing observations.

`deviance`

minus twice the maximized log-likelihood (up to a constant)

`dispersion`

for `rxLogit`, which estimates a generalized linear model with family `binomial`, the dispersion is always 1.

Note

Logistic regression is computed using the Iteratively Reweighted Least Squares (IRLS) algorithm, which is equivalent to full maximum likelihood.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

Frisch, Ragnar; Waugh, Frederick V., Partial Time Regressions as Compared with Individual Trends, *Econometrica*, 1 (4) (Oct., 1933), pp. 387-401.

Lovell, M., 1963, Seasonal adjustment of economic time series, *Journal of the American Statistical Association*, 58, pp. 993-1010.

Lovell, M., 2008, A Simple Proof of the FWL (Frisch,Waugh,Lovell) Theorem, *Journal of Economic Education*.

See Also

[rxLinMod](#), [rxTransform](#).

Examples

```
# Compare rxLogit and glm, which produce similar results when contr.SAS is used
infertLogit <- rxLogit(case ~ age + parity + education + spontaneous + induced,
                         data = infert)
infertLogit
summary(infertLogit)

infertGLM <- glm(case ~ age + parity + education + spontaneous + induced,
                  data = infert, family = binomial(), contrasts = list(education = contr.SAS))
summary(infertGLM)

# Instead of using the equivalent of contr.SAS, estimate the parameters
# for the categorical levels without contrasting against an intercept term.
infertCubeLogit <-
  rxLogit(case ~ education + age + parity + spontaneous + induced,
          data = infert, cube = TRUE)

# Define an ageGroup variable through a variable transformation list.
# Partition the age groups between 1 and 100 by 20 years.
rxLogit(case ~ education + ageGroup + parity + spontaneous + induced,
        transforms=list(ageGroup = cut(age, seq(1, 100, 20))),
        data = infert, cube = TRUE)

summary(infertCubeLogit)
```

rxLorenz: Lorenz Curve and Gini Coefficient

7/12/2022 • 3 minutes to read • [Edit Online](#)

Description

Compute and plot an empirical Lorenz curve from a variable in a data set, optionally specifying a separate variable from which to compute the y-values for the curve. Compute the Gini Coefficient from the Lorenz curve data. Appropriate for big data sets since data is binned with computations performed in one pass, rather than sorting the data as part of the computation process.

Usage

```
rxLorenz(orderVarName, valueVarName = orderVarName, data, numBreaks = 1000,
  pweights = NULL, fweights = NULL, blocksPerRead = 1,
  reportProgress = 1, verbose = 0)

## S3 method for class `rxLorenz':
rxGini ( x )

## S3 method for class `rxLorenz':
plot (x, title = NULL, subtitle = NULL,
  xTitle = NULL, yTitle = NULL, lineColor = NULL,
  lineStyle = "solid", lineWidth = 2, equalityGridLine = TRUE,
  equalityColor = "grey75", equalityStyle = NULL, equalityWidth = 2, ...)
```

Arguments

`orderVarName`

A character string with the name of the variable to use in computing approximate quantiles.

`valueVarName`

A character string with the name of the variable to use to compute the mean values per quantile. Can be the same as `orderVarName`.

`data`

data frame, character string containing an .xdf file name (with path), or [RxDataSource-class](#) object representing a data set containing the actual and observed variables.

`numBreaks`

integer specifying the number of breaks to use in comuting approximate quantiles.

`pweights`

character string specifying the variable to use as probability weights for the observations.

`fweights`

character string specifying the variable to use as frequency weights for the observations.

`blocksPerRead`

number of blocks to read for each chunk of data read from the data source.

reportProgress

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

verbose

integer value. If `0`, no additional output is printed. If `1`, additional information is printed as summary statistics are computed.

x

output object from rxLorenz function.

title

main title for the plot.

subtitle

subtitle (at the bottom) for the plot.

xTitle

title for the X axis.

yTitle

title for the Y axis.

lineColor

character or integer vector specifying line color for the Lorenz curve. See colors for a list of available colors.

lineStyle

line style for line plot: `"blank"`, `"solid"`, `"dashed"`, `"dotted"`, `"dotdash"`, `"longdash"`, or `"twodash"`. Specify `"blank"` for no line, or set `type` to `"p"`.

lineWidth

a positive number specifying the line width for line plot. The interpretation is device-specific.

equalityGridLine

logical value. If `TRUE`, a diagonal grid line will be drawn representing complete equality.

equalityColor

character or integer vector specifying line color for the equality grid line. If `NULL`, the color of other grid lines will be used.

equalityStyle

line style for the equality grid line: `"blank"`, `"solid"`, `"dashed"`, `"dotted"`, `"dotdash"`, `"longdash"`, or `"twodash"`. If `NULL`, the style of other grid lines will be used.

equalityWidth

a positive number specifying the line width for line plot. If `NULL`, the width of other grid lines will be used.

...

Additional arguments to be passed to `xyplot`.

Details

`rxLorenz` computes the cumulative percentage values of the variable specified in `valueVarName` for groups binned by the `orderVarname`. The size of the bins is determined by `numBreaks`.

When plotted, the cumulative percentage values are plotted against the quantile percentages.

The Gini coefficient is computed by estimating the ratio of the area between the line of equality and the Lorenz curve to the total area under the line of equality (using trapezoidal integration). The Gini coefficient can range from 0 to 1, with 0 representing perfect equality.

Precision can be increased by increasing `numBreaks`.

Value

`rxLorenz` returns a data frame of class `"rxLorenz"` containing two variables: `cumVals` and `percent`. It also may have a `"description"` attribute containing the value variable name or description.

`rxGini` returns a numeric vector of length one containing the approximate Gini coefficient.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxPredict](#), [rxLogit](#), [rxGlm](#), [rxLinePlot](#), [rxQuantile](#), [rxRoc](#).

Examples

```

#####
# Example using simple data frames for extreme distributions
#####
# Lorenz curve for complete equality
testData <- data.frame(income = rep(100, times=10))
lorenzOut1 <- rxLorenz("income", data = testData, numBreaks = 100)
plot(lorenzOut1)
rxGini(lorenzOut1)

# Extreme inequality
testData <- data.frame(income = c(rep(0, times=99), 100))
lorenzOut2 <- rxLorenz("income", data = testData, numBreaks = 100)
plot(lorenzOut2, equalityWidth = 3, equalityColor = "black")
rxGini(lorenzOut2)

#####
# Example using xdf file from sample data
#####

censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers")

# Compute Lorenz data using probability weights
lorenzOut <- rxLorenz(orderVarName = "incwage", data = censusWorkers,
  pweights = "perwt")

# Plot the Lorenz Curve
lorenzPlot <- plot(lorenzOut,
  title = "Lorenz Curve for Workers from Three States",
  subtitle = "Data Source: 5 Percent Sample of U.S. 2000 Census",
  lineWidth = 3, equalityColor = "black", equalityStyle = "longdash")

# Compute the Gini Coefficient
giniCoef <- rxGini(lorenzOut)

```

rxMakeRNodeNames: Converts valid computer names into valid R variable names.

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Converts valid computer names into valid R variable names. Should only be used when you want to guarantee that host names are usable as variable names.

Usage

```
rxMakeRNodeNames( nodeNames )
```

Arguments

`nodeNames`

character vector of node names to be converted.

Details

`rxMakeRNodeNames` will perform the following transformations on each element of the character vector passed:

1 Perform `toupper` on the name.

1 Remove all white space from the name.

1 If one exists, removes the first dot and all characters following it from the name. (This has the effect of stripping the domain name off, if one exists.)

1 Changes any '-' (dash) characters to '_' (underscore) characters so that node names used as variables do not have to be quoted.

The names returned by this function are valid R names, that is, symbols, but they may no longer be valid computer node names. Do **not** use these names, or this function, in any context where the name may be used to query or control the actual computer; use the original computer name for that. This function is intended to be used only to generate R variable names for processing or storing distributed computing results from the associated computer. Note also that once a host name has been converted into a guaranteed acceptable R variable name, it is impossible to guarantee the reverse conversion.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

Examples

```
## Not run:  
  
rxMakeRNodeNames(rxGetNodes(myCluster))  
rxMakeRNodeNames( c("cluster-head","worker1.foo.edu") )  
## End(Not run)
```

rxMarginals: Marginal Table Statistics

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Obtain marginal statistics on `rxCrossTabs` contingency tables.

Usage

```
## S3 method for class 'rxCrossTabs':  
rxMarginals (x, output = "sums", ...)
```

Arguments

`x`

object of class `rxCrossTabs`.

`output`

character string specifying the type of output to display. Choices are `"sums"`, `"counts"` and `"means"`.

`...`

additional arguments to be passed directly to the underlying print method for the output list object.

Details

Developed primarily as a method to access marginal statistics from `rxCrossTabs` generated cross-tabulations.

Value

list for each contingency table stored in an `rxCrossTabs` object. Each element of the list contains a list of the marginal means (row, column, and grand) if `output` is `"means"` or marginal sums otherwise.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxCrossTabs](#)

Examples

```
admissions <- as.data.frame(UCBAdmissions)
rxMarginals(rxCrossTabs(Freq ~ Gender : Admit, data = admissions, margin = TRUE,
                         means = FALSE))
rxMarginals(rxCrossTabs(Freq ~ Gender : Admit, data = admissions, margin = TRUE,
                         means = TRUE))
```

rxMerge: Merge two data sources

7/12/2022 • 8 minutes to read • [Edit Online](#)

Description

Merge (join) two data sources on one or more match variables. The rxMerge function is multi-threaded. In local compute context, the data sources may be sorted .xdf files or data frames. In RxSpark compute context, the data sources may be RxParquetData, RxHiveData, RxOrcData, RxXdfData or RxTextData.

Usage

```
rxMerge( inData1, inData2 = NULL, outFile = NULL, matchVars = NULL, type = "inner",
         missingsLow = TRUE, autoSort = TRUE, duplicateVarExt = NULL,
         varsToKeep1 = NULL, varsToDrop1 = NULL, newVarNames1 = NULL,
         varsToKeep2 = NULL, varsToDrop2 = NULL, newVarNames2 = NULL,
         rowsPerOutputBlock = -1, decreasing = FALSE, overwrite = FALSE,
         maxRowsByCols = 3000000, bufferLimit = -1,
         reportProgress = rxGetOption("reportProgress"), verbose = 0,
         xdfCompressionLevel = rxGetOption("xdfCompressionLevel"), ... )

rxMergeXdf( inFile1, inFile2, outFile, matchVars = NULL, type = "inner",
            missingsLow = TRUE,   duplicateVarExt = NULL,
            varsToKeep1 = NULL, varsToDrop1 = NULL, newVarNames1 = NULL,
            varsToKeep2 = NULL, varsToDrop2 = NULL, newVarNames2 = NULL,
            rowsPerOutputBlock = -1, decreasing = FALSE, overwrite = FALSE,
            bufferLimit = -1, reportProgress = rxGetOption("reportProgress"),
            verbose = 0, xdfCompressionLevel = rxGetOption("xdfCompressionLevel"), ... )
```

Arguments

`inData1`

the first data set to merge. In local compute context, a data frame, a character string denoting the path to an existing .xdf file, or an RxXdfData object. If a list of RxXdfData objects is provided, they will be merged sequentially. In RxSpark compute context, an RxParquetData, RxHiveData, RxOrcData, RxXdfData or RxTextData data source. If a list of data source objects is provided, they will be merged sequentially.

`inFile1`

the first data set to merge; either a character string denoting the path to an existing .xdf file or an RxXdfData object.

`inData2`

the second data set to merge. In local compute context, a data frame, a character string denoting the path to an existing .xdf file, or an RxXdfData object. Can be `NULL` if a list of RxXdfData objects is provided for `inData1`. In RxSpark compute context, an RxParquetData, RxHiveData, RxOrcData, RxXdfData or RxTextData data source. Can be `NULL` if a list of data source objects is provided for `inData1`.

`inFile2`

the second data set to merge; either a character string denoting the path to an existing .xdf file or an RxXdfData object.

outFile

in local compute context, an .xdf path to store the merged output. If the `outFile` already exists, `overwrite` must be set to `TRUE` to overwrite the file. If `NULL`, a data frame containing the merged data will be returned. In RxSpark compute context, an [RxParquetData](#), [RxHiveData](#), [RxOrcData](#) or [RxXdfData](#) data source.

matchVars

character vector containing the names of the variables to match for merging. In local compute context, the data sets MUST BE presorted in the same order by these variables, unless `autoSort` is set to `TRUE`. See [rxSort](#). Not required for `type` equal to `"union"` or `"oneToOne"`.

type

a character string defining the merge method to use:

- `"inner"` compares each row of `inData1` with each row of `inData2` to find all pairs of rows in which the values of the `matchVars` are the same.
- `"oneToOne"` appends columns from `inData1` to `inData2`. Not supported in RxSpark compute context.
- `"left"` includes all rows that match (as in `"inner"`) plus rows from `inData1` that do not have matches. `NA`'s will be used for the values for variables from `inData2` that are not matched.
- `"right"` includes all rows that match (as in `"inner"`) plus rows from `inData2` that do not have matches. `NA`'s will be used for the values for variables from `inData1` that are not matched.
- `"full"` is a combination of both `"left"` and `"right"` merge.
- `"union"` append rows from `inData2` to `inData1`. Not supported in RxSpark compute context. The two input files must have the same number of columns with the same data types.

missingsLow

a logical scalar for controlling the treatment of missing values. If `TRUE`, missing values in the data are treated as the lowest value; if `FALSE`, they are treated as the highest value. Not supported in RxSpark compute context.

autoSort

a logical scalar for controlling whether or not to sort the input data sets by the `matchVars` before merging. If `TRUE`, the data sets are sorted before merging; if `FALSE`, it is assumed that the data sets are already sorted by the `matchVars`. Not supported in RxSpark compute context.

duplicateVarExt

a character vector of length two containing the extensions to be used for handling duplicate variable names in the two input data sets. These extensions are not applied to matching variables. If `NULL`, file or data frame names will be used as the extension. If the names are the same, the extensions `1` and `2` will be used (unless there are other variables with those names.) For example, if `duplicateVarExt = c("One", "Two")` and `inData1` and `inData2` both have the variable `y`, the output data set will contain the variables `y.One` and `y.Two`.

varsToKeep1

character vector of variable names to include from the `inData1`. If `NULL`, argument is ignored. Cannot be used with `varsToDelete1`.

varsToDelete1

character vector of variable names to exclude from `inData1`. If `NULL`, argument is ignored. Cannot be used with `varsToKeep1`.

newVarNames1

a named character vector of new names for variables from `inData1` when writing them to `outData`. For example, specifying `c(x = "newx", y = "newy")` would give the input variables `x` and `y` the names `newx` and

`newy` in the output data.

`varsToKeep2`

character vector of variable names to include from the `inData2`. If `NULL`, argument is ignored. Cannot be used with `varsToDelete2`.

`varsToDelete2`

character vector of variable names to exclude from `inData2`. If `NULL`, argument is ignored. Cannot be used with `varsToKeep2`.

`newVarNames2`

a named character vector of new names for variables from `inData2` when writing them to `outData`. For example, specifying `c(x = "newx", y = "newy")` would give the input variables `x` and `y` the names `newx` and `newy` in the output data.

`rowsPerOutputBlock`

an integer specifying how many rows should be written out to each block in the output .xdf file. If set to -1, the smaller of the two average block sizes of the input data sets will be used. Ignored if `outData` is `NULL`. Not supported in [RxSpark](#) compute context.

`decreasing`

a logical scalar specifying whether or not the `matchVars` variables were sorted in decreasing or increasing order. The input data must be sorted in the same order.

`overwrite`

logical value. If `TRUE`, an existing `outFile` will be overwritten. Ignored if `outData` is `NULL`

`maxRowsByCols`

the maximum size of a data frame that will be returned if `outFile` is set to `NULL` and `inData` is an .xdf file, measured by the number of rows times the number of columns. If the number of rows times the number of columns being created from the .xdf file exceeds this, a warning will be reported and the number of rows in the returned data frame will be truncated. If `maxRowsByCols` is set to be too large, you may experience problems from loading a huge data frame into memory. Not supported in [RxSpark](#) compute context.

`bufferLimit`

integer specifying the maximum size of the memory buffer (in Mb) to use in merging. The default value of `bufferLimit = -1` will attempt to determine an appropriate buffer limit based on system memory. Not supported in [RxSpark](#) compute context.

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported. Not supported in [RxSpark](#) compute context.

`verbose`

integer value. If `0`, no additional output is printed. If `1`, additional summary information is printed.

`xdfCompressionLevel`

integer in the range of -1 to 9. The higher the value, the greater the amount of compression for the output file -

resulting in smaller files but a longer time to create them. If `xdfCompressionLevel` is set to 0, there will be no compression and the output file will be compatible with the 6.0 release of Revolution R Enterprise. If set to -1, a default level of compression will be used.

...

additional arguments to be passed directly to the Microsoft R Server Compute Engine.

Details

The arguments `varsToKeep1` (or alternatively `varsToDrop1`) and `varsToKeep2` (or alternatively `varsToDrop2`) are used to define the set of variables from the input files that will be stored in the specified merged `outFile` file. The `matchVars` must be included in the variables read from the input files. A single copy of the `matchVars` variables will be saved in the `outFile` file.

Value

For `rxMerge` : If an `outFile` is not specified, a data frame with the merged data is returned. If an `outFile` is specified, a data source object is returned that can be used in subsequent RevoScaleR analysis. For `rxMergeXdf` : If merging is successful, `TRUE` is returned; otherwise `FALSE` is returned.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

`sort`

Examples

```

####

# Small data frame example
####

x <- 1:20
y <- 20:1
df1 <- data.frame(x=x, y=y)

x <- 20:1
y <- 1:20
df2 <- data.frame(x=x, y=y)

# Merge the two data frames into an .xdf file, matching on the variable x
outXDF <- file.path(tempdir(), ".rxTempOut.xdf")
rxMerge(inData1 = df1, inData2 = df2, outFile=outXDF, matchVars = "x", overwrite=TRUE)

# Read the data from the .xdf file into a data frame. y.df1 and y.df2 should be the same
df3 <- rxDataStep(inData = outXDF )
df3

if(file.exists(outXDF)) file.remove(outXDF)

####

# Merge two data frames, matching y1 from the first with y2 from the second
# Notice that the match variable is sorted in decreasing order.
x1 <- 1:20
y1 <- 20:1
df1 <- data.frame(x1=x1, y1=y1)

x2 <- 1:20
y2 <- 25:6
df2 <- data.frame(x2=x2, y2=y2)

dfOut <- rxMerge(inData1 = df1, inData2 = df2, matchVars = "y2", decreasing=TRUE,
newVarNames1 = c(y1 = "y2"))

# Look at the resulting merged data
rxGetInfo( dfOut,numRows=10,getVarInfo=TRUE )

# Merge an .xdf file and a data frame into a new .xdf file

# .xdf file names
indXDF <- file.path(tempdir(), ".rxTempIn.xdf")
outXDF <- file.path(tempdir(), ".rxTempOut.xdf")

indData <- data.frame(id = 1:12, state = rep(c("CA","OR", "WA"), times = 4))
# Put individual-level data frame in .xdf file
rxDataStep(inData = indData, outFile = indXDF, overwrite=TRUE)

# Create state-level data frame
stateData <- data.frame(state=c("CA","OR", "WA"), stateVal = c(1000, 400, 500))

# Merge individual-level .xdf file with state-level data frame
rxMerge(inData1 = indXDF, inData2 = stateData, outFile = outXDF,
matchVars = "state")

# Re-sort data by id
rxSort(inData = outXDF, outFile = outXDF, sortByVars = "id", overwrite = TRUE)

# Look at merged, sorted data
df4 <- rxDataStep(inData = outXDF)
df4

```

rxMultiTest: Multiple Variable Independence Tests

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Collects a list of tests for variable independence into a table.

Usage

```
rxMultiTest(tests, title = NULL)

## S3 method for class `rxMultiTest':
print  (x, ...)
```

Arguments

`tests`

a list of objects of class `htest`.

`title`

a title for the multiTest table. If `NULL`, the title is determined from the method slot of the `htest`.

`x`

object of class `rxMultiTest`.

`...`

additional arguments to the `print` method.

Value

an object of class `rxMultiTest`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

`rxChiSquaredTest`, `rxFisherTest`, `rxKendallCor`, `rxPairwiseCrossTab`, `rxRiskRatio`, `rxOddsRatio`.

Examples

```
data(mtcars)
tests <- list(t.test(mpg ~ vs, data = mtcars),
              t.test(hp ~ vs, data = mtcars))
rxMultiTest(tests)
```

rxNaiveBayes: Parallel External Memory Algorithm for Naive Bayes Classifiers

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Fit Naive Bayes Classifiers on an .xdf file or data frame for small or large data using parallel external memory algorithm.

Usage

```
rxNaiveBayes(formula, data, smoothingFactor = 0, ... )
```

Arguments

`formula`

formula as described in [rxFormula](#).

`data`

either a data source object, a character string specifying a .xdf file, or a data frame object.

`smoothingFactor`

a positive smoothing factor to account for cases not present in the training data. It avoids modeling issues by preventing zero conditional probability estimates.

`...`

additional arguments to be passed directly to [rxSummary](#) such as `byTerm`, `rowSelection`, `pweights`, `fweights`, `transforms`, `transformObjects`, `transformFunc`, `transformVars`, `transformPackages`, `transformEnvir`, `useSparseCube`, `removeZeroCounts`, `blocksPerRead`, `reportProgress`, `verbose`, `xdfCompressionLevel`.

Value

an `"rxNaiveBayes"` object containing the following components:

- `apriori` - a vector of prior class probabilities for the dependent variable.
- `tables` - a list of tables, one for each predictor variable.
 - For a categorical variable, the table contains the conditional probabilities of the variable given the target class.
 - For a numeric variable, the table contains the mean and standard deviation of the variable given the target class.
- `levels` - the levels of the dependent variable.
- `call` - the matched call.

Author(s)

References

Naive Bayes classifier https://en.wikipedia.org/wiki/Naive_Bayes_classifier .

See Also

[rxPredict.rxNaiveBayes](#).

Examples

```
# multi-class classification with an .xdf file
claimsXdf <- file.path(rxGetOption("sampleDataDir"), "claims.xdf")
claims.nb <- rxNaiveBayes(type ~ age + cost, data = claimsXdf)
claims.nb

# prediction
claims.nb.pred <- rxPredict(claims.nb, claimsXdf)
claims.nb.pred
table(claims.nb.pred[["type_Pred"]], rxDataStep(claimsXdf)[["type"]])
```

RxNativeFileSystem: RevoScaleR Native File System Object Generator

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

This is the main generator for RxNativeFileSystem S3 class.

Usage

```
RxNativeFileSystem()
```

Value

An RxNativeFileSystem file system object. This object may be used in [rxSetFileSystem](#), [rxOptions](#), [RxTextData](#), or [RxXdfData](#) to set the file system.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxFileSystem](#), [RxHdfsFileSystem](#), [rxSetFileSystem](#), [rxOptions](#), [RxXdfData](#), [RxTextData](#).

Examples

```
# Setup to run analyses to use HDFS file system, then native
## Not run:

myHdfsFileSystem <- RxHdfsFileSystem(hostName = "myHost", port = 8020)
rxSetFileSystem(fileSystem = myHdfsFileSystem )
# Perform other tasks
# Reset to native file system
rxSetFileSystem(fileSystem = RxNativeFileSystem())
## End(Not run)
```

rxNewDataSource: RevoScaleR Data Sources: Class Generator

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

This is the main generator for RxDataSource S4 classes.

Usage

```
## S4 method for class 'character':  
rxNewDataSource (name, ...)
```

Arguments

`name`

character name of the specific class to instantiate.

`...`

any other arguments are passed to the class generator `name`.

Details

This is a wrapper to specific class generator functions for the RCE data source classes. For example, the RxXdfData class uses function `RxXdfData` as a generator. Therefore either `RxXdfData(...)` or

```
rxNewDataSource("RxXdfData", ...)
```

Value

RxDataSource data source object. This object may be used to open and close the actual RevoScaleR data sources.

Side Effects

This function creates the data source instance in the Microsoft R Services Compute Engine, but does not actually open the data. The methods `rxOpen` and `rxClose` will open and close the data.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxXdfData](#), [RxTextData](#), [RxSasData](#), [RxSpssData](#), [RxOdbcData](#), [RxTeradata](#), [rxOpen](#), [rxReadNext](#).

Examples

```
fileName <- file.path(rxGetOption("sampleDataDir"), "claims.xdf")
ds <- rxNewDataSource("RxXdfData", fileName)
rxOpen(ds)
myData <- rxReadNext(ds)
rxClose(ds)
```

rxOAuthParameters: OAuth2 Token request

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Method to create parameter list to be used for getting an OAuth2 token.

Usage

```
rxOAuthParameters(authUri = NULL, tenantId = NULL, clientId = NULL, resource = NULL, username = NULL,  
password = NULL, authToken= NULL, useWindowsAuth = FALSE)
```

Arguments

`authUri`

Optional string containing OAuth Authentication URI - default NULL

`tenantId`

Optional string containing OAuth Tenant ID - default NULL

`clientId`

Optional string containing OAuth ClientID - default NULL

`resource`

Optional string containing OAuth Resource - default NULL

`username`

Optional string containing OAuth Username - default NULL

`password`

Optional string containing OAuth Password - default NULL

`authToken`

Optional string containing a valid OAuth token to be used for WebHdfs requests - default NULL

`useWindowsAuth`

Optional Flag indicating if Windows Authentication should be used for obtaining the OAuth token (applicable only on Windows) - default NULL

Details

Reading from HDFS file system via WebHdfs can only be done by first obtaining a OAuth2 token. This function allows the specification of parameters that can be set to retrieve a token.

Value

An rxOAuthParameters list object. This object may be used in [RxHdfsFileSystem](#) to set the OAuth request

method for WebHdfs usage.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxHdfsFileSystem](#)

Examples

```
# Setup to run analyses to use HDFS with access via WebHdfs and OAuth2
## Not run:

oAuth <- rxOAuthParameters(authUri = "https://login.windows.net/",
                            tenantId = "mytest.onmicrosoft.com",
                            clientId = "872cd9fa-d31f-45e0-9eab-6e460a02d1e2",
                            resource = "https://KonaCompute.net/",
                            username = "me@mytest.onmicrosoft.com",
                            password = "password")

myHdfsFileSystem <- RxHdfsFileSystem(hostName = "myHost", port = 443, useWebHdfs = TRUE, oAuthParameters =
oAuth)
rxSetFileSystem(fileSystem = myHdfsFileSystem )
## End(Not run)
```

RxOdbcData-class: Class RxOdbcData

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

ODBC data source connection class.

Generators

The targeted generator [RxOdbcData](#) as well as the general generator [rxNewDataSource](#).

Extends

Class RxDataSource, directly.

Methods

[show](#)

```
signature(object = "RxOdbcData") : ...
```

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxDataSource-class](#), [RxOdbcData](#), [rxNewDataSource](#)

RxOdbcData: Generate ODBC Data Source Object

7/12/2022 • 5 minutes to read • [Edit Online](#)

Description

This is the main generator for S4 class RxOdbcData, which extends RxDataSource.

Usage

```
RxOdbcData(table = NULL, sqlQuery = NULL, dbmsName = NULL, databaseName = NULL,
           useFastRead = NULL, trimSpace = NULL, connectionString = NULL,
           rowBuffering = TRUE, returnDataFrame = TRUE, stringsAsFactors = FALSE,
           colClasses = NULL, colInfo = NULL, rowsPerRead = 500000,
           verbose = 0, writeFactorsAsIndexes, ...)

## S3 method for class `RxOdbcData':
head (x, n = 6L, reportProgress = 0L, ...)
## S3 method for class `RxOdbcData':
tail (x, n = 6L, addrownums = TRUE, reportProgress = 0L, ...)
```

Arguments

`table`

`NULL` or character string specifying the table name. Cannot be used with `sqlQuery`.

`sqlQuery`

`NULL` or character string specifying a valid SQL select query. Cannot be used with `table`.

`dbmsName`

`NULL` or character string specifying the Database Management System (DBMS) name.

`databaseName`

`NULL` or character string specifying the name of the database.

`useFastRead`

logical specifying whether or not to use a direct ODBC connection. On Linux systems, this is the only ODBC connection available. Note: `useFastRead = FALSE` is currently not supported in writing to ODBC data source.

`trimSpace`

logical specifying whether or not to trim the white character of string data for reading.

`connectionString`

`NULL` or character string specifying the connection string.

`rowBuffering`

logical specifying whether or not to buffer rows on read from the database. If you are having problems with your ODBC driver, try setting this to `FALSE`.

`returnDataFrame`

logical indicating whether or not to convert the result from a list to a data frame (for use in `rxReadNext` only). If `FALSE`, a list is returned.

`stringsAsFactors`

logical indicating whether or not to automatically convert strings to factors on import. This can be overridden by specifying `"character"` in `colClasses` and `colInfo`. If `TRUE`, the factor levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use `colInfo` with specified `"levels"`.

`colClasses`

character vector specifying the column types to use when converting the data. The element names for the vector are used to identify which column should be converted to which type.

- Allowable column types are:

- `"logical"` (stored as `uchar`),
- `"integer"` (stored as `int32`),
- `"float32"` (the default for floating point data for .xdf files),
- `"numeric"` (stored as `float64` as in R),
- `"character"` (stored as `string`),
- `"factor"` (stored as `uint32`),
- `"int16"` (alternative to integer for smaller storage space),
- `"uint16"` (alternative to unsigned integer for smaller storage space),
- `"Date"` (stored as Date, i.e. `float64`)

- Note for `"factor"` type, the levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use `colInfo` with specified `"levels"`.
- Note that equivalent types share the same bullet in the list above; for some types we allow both 'R-friendly' type names, as well as our own, more specific type names for .xdf data.
- Note also that specifying the column as a "factor" type is currently equivalent to "string" - for the moment, if you wish to import a column as factor data you must use the `colInfo` argument, documented below.

`colInfo`

list of named variable information lists. Each variable information list contains one or more of the named elements given below. The information supplied for `colInfo` overrides that supplied for `colClasses`.

- Currently available properties for a column information list are:
- `type` - character string specifying the data type for the column. See `colClasses` argument description for the available types. Specify `"factorIndex"` as the `type` for 0-based factor indexes. `levels` must also be specified.
- `newName` - character string specifying a new name for the variable.
- `description` - character string specifying a description for the variable.
- `levels` - character vector containing the levels when `type = "factor"`. If the `levels` property is not provided, factor levels will be determined by the values in the source column. If levels are provided, any value that does not match a provided level will be converted to a missing value.
- `newLevels` - new or replacement levels specified for a column of type "factor". It must be used in conjunction with the `levels` argument. After reading in the original data, the labels for each level will be replaced with the `newLevels`.

- `low` - the minimum data value in the variable (used in computations using the `F()` function).
- `high` - the maximum data value in the variable (used in computations using the `F()` function).

`rowsPerRead`

number of rows to read at a time.

`verbose`

integer value. If `0`, no additional output is printed. If `1`, information on the odbc data source type (`odbc` or `odbcFast`) is printed.

`writeFactorsAsIndexes`

logical. If `TRUE`, when writing to an `RxOdbcData` data source, underlying factor indexes will be written instead of the string representations.

`...`

additional arguments to be passed directly to the underlying functions.

`x`

an `RxOdbcData` object

`n`

positive integer. Number of rows of the data set to extract.

`addrownums`

logical. If `TRUE`, row numbers will be created to match the original data set.

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

Details

The `tail` method is not functional for this data source type and will report an error.

If you are using `RxOdbcData` with a Teradata database and experience difficulty, try the `RxTeradata` data source instead. The `RxTeradata` data source permits finer control of the underlying Teradata options.

Value

object of class `RxOdbcData`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxOdbcData-class](#), [rxNewDataSource](#), [rxImport](#), [RxTeradata](#).

Examples

```
## Not run:

# Create an ODBC data source for a SQLite database
claimsSQLiteFileName <- file.path(rxGetOption("sampleDataDir"), "claims.sqlite")
connectionString <-
  paste("Driver={SQLite3 ODBC Driver};Database=", claimsSQLiteFileName,
       sep = "")
claimsOdbcSource <-
  RxOdbcData(sqlQuery = "SELECT * FROM claims",
             connectionString = connectionString)

# Create an xdf file name
claimsXdfFileName <- file.path(tempdir(), "importedClaims.xdf")

# Import the data into the xdf file
rxImport(claimsOdbcSource, claimsXdfFileName, overwrite = TRUE)

# Read xdf file into a data frame
claimsIn <- rxDataStep(inData = claimsXdfFileName)
head(claimsIn)

# Clean-up: delete the new file
file.remove(claimsXdfFileName)

# Create an ODBC data source for storing a data frame
irisOdbcSource <- RxOdbcData(table = "irisTable", connectionString = connectionString)

# Store the data frame in the database
rxWriteObject(irisOdbcSource, "irisData", iris)

# Retrieve the data frame from the database
irisData <- rxReadObject(irisOdbcSource, "irisData")
identical(irisData, iris) # TRUE

# Create an ODBC data source for storing a sample XDF
airlineOdbcSource <- RxOdbcData(table = "airlineTable", connectionString = connectionString)

# Store a sample XDF in the database
airlineXdf <- rxReadXdf( file.path(rxGetOption("sampleDataDir"), "AirlineDemoSmall.xdf") )
rxWriteObject(airlineOdbcSource, "airlineData", airlineXdf)

# Retrieve the XDF object from the database
airlineData <- rxReadObject(airlineOdbcSource, "airlineData")
identical(airlineData, airlineXdf) # TRUE
## End(Not run)
```

rxOpen-methods: Managing RevoScaleR Data Source Objects

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

These functions manage RevoScaleR data source objects.

Usage

```
rxOpen(src, mode = "r")
rxClose(src, mode = "r")
rxIsOpen(src, mode = "r")
rxReadNext(src)
rxWriteNext(from, to, ...)
```

Arguments

`from`

data frame object.

`src`

RxDataSource object.

`to`

RxDataSource object.

`mode`

character string specifying the mode (`r` or `w`) to open the file.

`...`

any other arguments to be passed on.

Value

For `rxOpen` and `rxClose`, a logical indicating whether the operation was successful. For `rxIsOpen`, a logical indicating whether or not the RxDataSource is open for the specified `mode`. For `rxReadNext`, either a data frame or a list depending upon the value of the `returnDataFrame` property within `src`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxNewDataSource](#), [RxXdfData](#).

Examples

```
ds <- RxXdfData(file.path(rxGetOption("sampleDataDir"), "claims.xdf"))
# ds contains only one block of data
rxOpen(ds) # must open the file before rxReadNext
rxReadNext(ds) # get the first block
rxReadNext(ds)
rxClose(ds)

# Use a data source to compute means by processing the data in chunks
# Data processing functions: for each chunk, compute sums of columns and
# number of rows, then update the results computed from previous chunks
processData <- function(dframe)
  list(sumCols = colSums(dframe), numRows = nrow(dframe))
updateResults <- function(x, y)
  list(sumCols = x$sumCols + y$sumCols, numRows = x$numRows + y$numRows)

# Create data source
censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers.xdf")
ds <- RxXdfData(censusWorkers, varsToKeep = c("age", "incwage"),
                blocksPerRead = 2)

# Process data and update results
rxOpen(ds)
resList <- processData(rxReadNext(ds))
while(TRUE)
{
  df <- rxReadNext(ds)
  if (nrow(df) == 0)
    break
  resList <- updateResults(resList, processData(df))
}
rxClose(ds)

# Compute the means of the variables from the accumulated results
varMeans <- resList$sumCols / resList$numRows
varMeans
```

rxOptions: Global Options for RevoScaleR

7/12/2022 • 7 minutes to read • [Edit Online](#)

Description

Functions to specify and retrieve options needed for **RevoScaleR** computations. These need to be set only once to carry out multiple computations.

Usage

```
rxOptions(initialize = FALSE,
          libDir,
          linkDllName = ifelse(.Platform$OS.type == "windows", "RxLink.dll", "libRxLink.so.2"),
          cintSysDir = setCintSysDir(),
          includeDir = setRxIncludeDir(),
          unitTestDir = system.file("unitTests", package = "RevoScaleR"),
          unitTestDataDir = system.file("unitTestData", package = "RevoScaleR"),
          sampleDataDir = system.file("SampleData", package = "RevoScaleR"),
          demoScriptsDir = system.file("demoScripts", package = "RevoScaleR"),
          blocksPerRead = 1,
          reportProgress = 2,
          rowDisplayMax = -1,
          memStatsReset = 0,
          memStatsDiff = 0,
          numCoresToUse = 4,
          numDigits = options()$digits,
          showTransformFn = FALSE,
          defaultDecimalColType = "float32",
          defaultMissingColType = "float32",
          computeContext = RxLocalSeq(),
          dataPath = ".",
          outDataPath = ".",
          transformPackages = c("RevoScaleR", "utils", "stats", "methods"),
          xdfCompressionLevel = 1,
          fileSystem = "native",
          useDoSMP = NULL,
          useSparseCube = FALSE,
          rngBufferSize = 1,
          dropMain = TRUE,
          coefLabelStyle = "Revo",
          numTasks = 1,
          hdfsHost = Sys.getenv("REVOHADOOPHOST"),
          hdfsPort = as.integer(Sys.getenv("REVOHADOOPPORT")),
          unixRPath = "/usr/bin/Revo64-8",
          mrsHadoopPath = "/usr/bin/mrs-hadoop",
          spark.executorCores = 2,
          spark.executorMem = "4g",
          spark.executorOverheadMem = "4g",
          spark.numExecutors = 65535,
          traceLevel = 0,
          ...)

rxgetOption(opt, default = NULL)
rxIsExpressEdition()
```

Arguments

`initialize`

logical value. If `TRUE`, `rxOptions` resets all **RevoScaleR** options to their default value.

`libDir`

character string specifying path to **RevoScaleR**'s lib directory. For 32-bit versions, this defaults to the `libs` directory; for 64-bit versions, this defaults to the `libs/x64` directory.

`linkDllName`

character string specifying name of the **RevoScaleR**'s DLL (Windows) or shared object (Linux).

`cintSysDir`

character string specifying path to **RevoScaleR**'s C/C++ interpreter (CINT) directory.

`includeDir`

character string specifying path to **RevoScaleR**'s include directory.

`unitTestDir`

character string specifying path to **RevoScaleR**'s RUnit-based test directory.

`unitTestDataDir`

character string specifying path to **RevoScaleR**'s RUnit-based test data directory.

`sampleDataDir`

character string specifying path to **RevoScaleR**'s sample data directory.

`demoScriptsDir`

character string specifying path to **RevoScaleR**'s demo script directory.

`blocksPerRead`

default value to use for `blocksPerRead` argument for many **RevoScaleR** functions. Represents the number of blocks to read within each read chunk.

`reportProgress`

default value to use for `reportProgress` argument for many **RevoScaleR** functions. Options are:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`rowDisplayMax`

scalar integer specifying the maximum number of rows to display when using the `verbose` argument in **RevoScaleR** functions. The default of `-1` displays all available rows.

`memStatsReset`

boolean integer. If `1`, reset memory status

`memStatsDiff`

boolean integer. If `1`, the change of memory status is shown.

`numCoresToUse`

scalar integer specifying the number of cores to use. If set to a value higher than the number of available cores,

the number of available cores will be used. If set to `-1`, the number of available cores will be used. Increasing the number of cores to use will also increase the amount of memory required for **RevoScaleR** analysis functions.

`numDigits`

controls the number of digits to use when converting numeric data to or from strings, such as when printing numeric values or importing numeric data as strings. The default is the current value of `options()$digits`, which defaults to 7. Beyond fifteen digits, however, results are likely to be unreliable.

`showTransformFn`

logical value. If `TRUE`, the transform function is shown.

`defaultDecimalColType`

Used to specify a column's data type when only decimal values (possibly mixed with missing (`NA`) values) are encountered upon first read of the data and the column's type information is not specified via `colInfo` or `colClasses`. Supported types are "float32" and "numeric", for 32-bit floating point and 64-bit floating point values, respectively.

`defaultMissingColType`

Used to specify a given column's data type when only missings (`NA`s) or blanks are encountered upon first read of the data and the column's type information is not specified via `colInfo` or `colClasses`. Supported types are "float32", "numeric", and "character" for 32-bit floating point, 64-bit floating point and string values, respectively.

`computeContext`

an [RxComputeContext](#) object representing the computational environment.

- [RxLocalSeq](#): compute locally, using sequential processing with [rxExec](#) High Performance Computing.
- [RxLocalParallel](#): compute locally, using the `'parallel'` package for processing with [rxExec](#) High Performance Computing.
- [RxForeachDoPar](#): use the currently registered parallel backend for 'foreach' for processing with [rxExec](#) High Performance Computing.
- [RxHadoopMR](#): use a Hadoop cluster for both High Performance Analytics for [rxExec](#) High Performance Computing.
- [RxSpark](#): use a Spark cluster for both High Performance Analytics and for [rxExec](#) High Performance Computing.

`dataPath`

character vector containing paths to search for local data sources. The default is to search just the current working directory. This will be ignored if `dataPath` is specified in the active compute context. See the Details section for more information regarding the path format.

`outDataPath`

character vector containing paths for writing new output data files. New data files will be written to the first path that exists. The default is to write to the current working directory. This will be ignored if `outDataPath` is specified in the active compute context.

`transformPackages`

character vector defining default set of R packages to be made available and preloaded for use in variable transformation functions.

`xdfCompressionLevel`

integer in the range of -1 to 9. The higher the value, the greater the amount of compression - resulting in smaller files but a longer time to create them. If `xdfCompressionLevel` is set to 0, there will be no compression and files will be compatible with the 6.0 release of Revolution R Enterprise. If set to -1, a default level of compression will be used.

`fileSystem`

character string or `RxFileSystem` object indicating type of file system; `"native"` or `RxNativeFileSystem` object can be used for the local operating system, or an `RxHdfsFileSystem` object for the Hadoop file system.

`useDoSMP`

`NULL`. Deprecated. Use a `RxLocalParallel` compute context.

`opt`

character string specifying the `RevoScaleR` option to obtain. A `NULL` is returned if the option does not exist.

`useSparseCube`

logical value. If `TRUE`, sparse cube is used.

`rngBufferSize`

a positive integer scalar specifying the buffer size for the Parallel Random Number Generators (RNGs) in MKL.

`dropMain`

logical value. If `TRUE`, main-effect terms are dropped before their interactions.

`coefLabelStyle`

character string specifying the coefficient label style. The default is "Revo". If "R", R-compatible labels are created.

`numTasks`

integer value. The default `numTasks` use in `RxInSqlServer`.

`hdfsHost`

character string specifying the host name of your Hadoop nameNode. Defaults to `Sys.getenv("REVOHADOOPHOST")`, or `"default"` if no REVOHADOOPHOST environment variable is set.

`hdfsPort`

integer scalar specifying the port number of your Hadoop nameNode, or a character string that can be coerced to numeric. Defaults to `as.integer(Sys.getenv("REVOHADOOPPORT"))`, or `0` if no REVOHADOOPPORT environment variable is set.

`unixRPath`

The path to R executable on a Unix/Linux node. By default it points to a path corresponding to this client's version.

`mrsHadoopPath`

Points to entry point to Hadoop MR which is deployed on every cluster node when MRS for Hadoop is installed. This script implements logic that determines which hadoop command should be called.

`traceLevel`

Specifies the traceLevel that MRS will run with. This parameter controls MRS Logging features as well as Runtime Tracing of ScaleR functions. Levels are inclusive, (i.e. level `3:INFO` includes levels `2:WARN` and `1:ERROR` log messages). The options are:

- `0 : DISABLED` - Tracing/Logging disabled.

- 1 : `ERROR` - `ERROR` coded trace points are logged to MRS log files
- 2 : `WARN` - `WARN` and `ERROR` coded trace points are logged to MRS log files.
- 3 : `INFO` - `INFO`, `WARN`, and `ERROR` coded trace points are logged to MRS log files.
- 4 : `DEBUG` - All trace points are logged to MRS log files.
- 5 : `RESERVED` - If set, will log at `DEBUG` granularity
- 6 : `RESERVED` - If set, will log at `DEBUG` granularity
- 7 : `TRACE` - ScaleR functions Runtime Tracing is activated and MRS log level is set to `DEBUG` granularity.

...

additional arguments to be passed through.

`default`

default value for an option that is returned if option is not found

Details

A full set of **RevoScaleR** options is set on load. Use `rxGetOption` to obtain the value of a single option.

The `dataPath` argument is a character vector of well formed directory paths, either in UNC ("`\host\dir`") or DOS ("`C:\dir`"). When specifying the paths, you must double the number of backslashes since R requires that backslashes be escaped. Updating the previous examples gives "`\\\host\\dir`" for UNC-type paths and "`C:\\dir`" for DOS-type paths. Alternatively, you could specify a DOS-type path with single forward slashes such as the output from `system.file` (e.g. "

`C:/Revolution/R-Enterprise-Node-7.4/R-3.1.3/library/RevoScaleR/SampleData`"). For Windows operating systems, the arguments that define a path must be in long format and not DOS 8.3 (short) format, e.g., "`"C:\\Program Files\\RRO\\R-3.1.3\\bin\\x64"` or "`C:/Program Files/RRO/R-3.1.3/bin/x64`" are correct formats while "`"C:/PROGRA~1/RRO/R-31~1.3/bin/x64"` is not.

`rxIsExpressEdition` is not currently functional.

Value

For `rxOptions`, a list containing the original `rxOptions` is returned. If there is no argument specified, the list is returned explicitly, otherwise the list is returned as an invisible object. For `rxGetOption`, the current value of the requested option is returned.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RevoScaleR](#), [RxLocalSeq](#), [RxLocalParallel](#), [RxForeachDoPar](#), [RxHadoopMR](#), [RxSpark](#), [RxInSqlServer](#).

Examples

```
# Get the location of the sample data sets for RevoScaleR
dataDir <- rxGetOption("sampleDataDir")
# See the current settings for options
rxOptions()
## Not run:

rxOptions(reportProgress = 0) # by default, don't report progress
rxOptions()$reportProgress # show value
rxOptions(TRUE) # reset all options
rxOptions()$reportProgress # 2

# Setup to run analyses on HPC cluster
myCluster <- RxSpark(nameNode = "my-name-service-server", port = 8020)

rxOptions( computeContext = myCluster )
## End(Not run)
```

rxPackage: R Package Management in SQL Server

7/12/2022 • 5 minutes to read • [Edit Online](#)

Description

This article explains how to enable and disable R package management on SQL Server Machine Learning Services (in-database), as well as installation, usage, and removal of individual packages. **RevoScaleR** provides the necessary rx functions for these tasks.

Details

SQL Server Machine Learning Services and the previous version, SQL Server R Services, support install and uninstall of R packages on SQL Server. A database administrator (in `db_owner` role) must grant permissions to allow access to package functions at both the database and instance level.

R package management functions are part of the base distribution. These functions allows packages to be installed from a local or remote repository into a local library (folder). R provides the following core functions for managing local libraries:

- * `available.packages()` - enumerate packages available in a repository for installation
- * `installed.packages()` - enumerate installed packages in a library
- * `install.packages()` - install packages, including dependency resolution, from a repository into a library
- * `remove.packages()` - remove installed packages from a library
- * `library()` - load the installed package and access its functions

RevoScaleR also provides package management functions, which is especially useful for managing packages in a SQL Server compute context in a client session. Packages in the database are reflected back on the file system, in a secure location. Access is controlled through database roles, which determine whether you can install packages from a local or remote repository into a database.

RevoScaleR provides the following core client functions for managing libraries in a database on a remote SQL server:

- * `rxInstalledPackages()` - enumerate installed packages in a database on SQL Server
 - * `rxInstallPackages()` - install packages, including dependency resolution, from a repository onto a library in a database. Simultaneously install the same packages on the file system using per-database and per-user profile locations.
 - * `rxRemovePackages()` - remove installed packages from a library in a database and further uninstall the packages from secured per-database, per-user location on SQL server
 - * `rxSqlLibPaths()` - get secured library paths for the given user to refer to the installed packages for SQL Server to then use it in `.libPaths()` to refer to the packages
 - * `library()` - same as R equivalent; used to load the installed package and access its functions
 - * `rxSyncPackages()` - synchronize packages from a database on SQL Server to the file system used by R
- There are two scopes for installation and usage of packages in a particular database in SQL Server:
- * `Shared scope` - Share the packages with other users of the same database.

- * `Private scope` - Isolate a package in a per-user private location, accessible only to the user installing the package.

Both scopes can be used to design different secure deployment models of packages. For example, using a `shared` scope, data scientist department heads can be granted permissions to install packages, which can then be used by all other data scientists in the group. In another deployment model, the data scientists can be granted `private` scope permissions to install and use their private packages without affecting other data scientists using the same server.

Database roles are used to secure package installation and access:

- * `rpkgs-users` - allows users to use shared packages installed by users belong to `rpkgs-shared` role
- * `rpkgs-private` - allows all permissions as `rpkgs-users` role and also allows users to install, remove and use private packages
- * `rpkgs-shared` - allows all permissions as `rpkgs-private` role and also allows users to install, remove shared packages
- * `db_owner` - allows all permissions as `rpkgs-shared` role and also allows users to install & remove shared and private packages for other users for management

Enable R package management on SQL Server

By default, R package management is turned off at the instance level. To use this functionality, the administrator must do the following:

- * Enable package management on the SQL Server instance.
- * Enable package management on the SQL database.

`RegisterRExt.exe` command line utility, which ships with RevoScaleR, allows administrators to enable package management for instances and specific databases. You can find `RegisterRExt.exe` at
`<SQLInstancePath>\R_SERVICES\library\RevoScaleR\rxLibs\x64\RegisterRExt.exe`.

To enable R package management at instance level, open an elevated command prompt and run the following command:

- * `RegisterRExt.exe /installpkgmgmt [/instance:name] [/user:username] [/password:*|password]`

This command creates some package-related, instance-level artifacts on the SQL Server machine.

Next, enable R package management at database level using an elevated command prompt and the following command:

- *
`RegisterRExt.exe /installpkgmgmt /database:dbname [/instance:name] [/user:username] [/password:*|password]`

This command creates database artifacts, including `rpkgs-users`, `rpkgs-private` and `rpkgs-shared` database roles to control user permissions who can install, uninstall, and use packages.

Disable R package management on a SQL Server

To disable R package management on a SQL server, the administrator must do the following:

- * Disable package management on the database.
- * Disable package management on the SQL Server instance.

Use the `RegisterRExt.exe` command line utility located at

```
<SQLInstancePath>\R_SERVICES\library\RevoScaleR\rxLibs\x64\RegisterRExt.exe .
```

To disable R package management at the database level, open an elevated command prompt and run the following command:

```
*
```

```
RegisterRExt.exe /uninstallpkgmgmt /database:databasefilename [/instance:name] [/user:username]  
[/password:*|password]
```

This command removes the package-related database artifacts from the database, as well as the packages in the secured file system location.

After disabling package management at the database level, disable package management at instance level by running the following command at an elevated command prompt:

```
* RegisterRExt.exe /uninstallpkgmgmt [/instance:name] [/user:username] [/password:*|password]
```

This command removes the package-related, per-instance artifacts from the SQL Server.

See Also

[rxSqlLibPaths](#), [rxInstalledPackages](#), [rxInstallPackages](#),
[rxRemovePackages](#), [rxFindPackage](#), library require

Examples

```

## Not run:

#
# install and remove packages from client
#
sqlcc <- RxInSqlServer(connectionString = "Driver=SQL
Server;Server=myServer;Database=TestDB;Trusted_Connection=True;")

pkgs <- c("dplyr")
rxInstallPackages(pkgs = pkgs, verbose = TRUE, scope = "private", computeContext = sqlcc)
rxRemovePackages(pkgs = pkgs, verbose = TRUE, scope = "private", computeContext = sqlcc)

#
# use the installed R package from rx function like rxExec(...)
#
usePackageRxFunction <- function()
{
  library(dplyr)

  # returns list of functions contained in dplyr
  ls(pos="package:dplyr")

  #
  # more code to use dplyr functionality
  #
  # ...
  #

}

rxSetComputeContext(sqlcc)
rxExec(usePackageRxFunction)

#
# use the installed R packages in an R function call running on SQL server using T-SQL
#
declare @instance_name nvarchar(100) = @@SERVERNAME, @database_name nvarchar(128) = db_name();
exec sp_execute_external_script
@language = N'R',
@script = N'
#
# setup the lib paths to private and shared libraries
#
connStr <- paste("Driver=SQL Server;Server=", instance_name, ";Database=", database_name,
";Trusted_Connection=true;", sep="");
.libPaths(rxSqlLibPaths(connStr));

#
# use the installed R package from rx function like rxExec(...)
#
library("dplyr");

#
# more code to use dplyr functionality
#
# ...
#
',
@input_data_1 = N'',
@params = N'@instance_name nvarchar(100), @database_name nvarchar(128)',
@instance_name = @instance_name,
@database_name = @database_name;
## End(Not run)

```

rxPairwiseCrossTab: Apply Function to Pairwise Combinations of an xtabs Object

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Apply a function 'FUN' to all pairwise combinations of the rows and columns of an xtabs object, stratifying by higher dimensions.

Usage

```
rxPairwiseCrossTab(x, pooled = TRUE, FUN = rxRiskRatio, ...)
## S3 method for class `rxPairwiseCrossTabList':
print (x, ...)
## S3 method for class `rxPairwiseCrossTabTable':
print (x, digits = 3, scientific = FALSE, p.value.digits = 3, ...)
```

Arguments

`x`

an object of class xtabs, rxCrossTabs, or rxCube for `rxPairwiseCrossTab`. An object of class rxPairwiseCrossTabList or rxPairwiseCrossTabTable for the print methods of rxPairwiseCrossTabList and rxPairwiseCrossTabTable objects.

`pooled`

logical value. If `TRUE`, the comparison groups are pooled. Otherwise all pairwise comparisons are included.

`FUN`

function that takes a two by two table and returns an object of class htest.

`...`

additional arguments.

`digits`

number of digits to use in printing numeric values.

`scientific`

logical value. If `TRUE`, scientific notation is used to print numeric values.

`p.value.digits`

number of digits to use in printing p-values.

Value

a rxPairwiseCrossTabList object.

Author(s)

See Also

[rxChiSquaredTest](#), [rxFisherTest](#), [rxKendallCor](#), [rxMultiTest](#), [rxRiskRatio](#), [rxOddsRatio](#).

Examples

```
mtcarsDF <- rxFactors(datasets::mtcars, factorInfo = c("gear", "cyl", "vs"),
                        varsToKeep = c("gear", "cyl", "vs"), sortLevels = TRUE)
xtabObj <- rxCrossTabs(~ gear : cyl : vs, data = mtcarsDF, returnXtabs = TRUE)

rxPairwiseCrossTab(xtabObj, FUN = rxRiskRatio)
rxPairwiseCrossTab(xtabObj, pooled = FALSE, FUN = rxOddsRatio)
```

rxPartition: Partition Data by Key Values and Save the results to a Partitioned .Xdf

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Partition input data sources by key values and save the results to a partitioned Xdf on disk.

Usage

```
rxPartition(inData, outData, varsToPartition, append = "rows", overwrite = FALSE, ...)
```

Arguments

`inData`

either a data source object, a character string specifying a .xdf file, or a data frame object.

`outData`

a partitioned data source object created by `RxXdfData` with `createPartitionSet = TRUE`.

`varsToPartition`

character vector of variable names to specify the values in those variables to be used for partitioning

`append`

either "none" to create a new files or "rows" to append rows to an existing file. If `outData` exists and `append` is "none", the `overwrite` argument must be set to `TRUE`.

`overwrite`

logical value. If `TRUE`, an existing `outData` will be overwritten. `overwrite` is ignored if `append = "rows"`.

`...`

additional arguments to be passed directly to the Revolution Compute Engine.

Value

a data frame of partitioning values and data sources, each row in the data frame represents one partition and the data source in the last variable holds the data of a specific partition.

Note

In the current version, this function is single threaded.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxExecBy](#), [RxXdfData](#)

Examples

```
#####
# Construct a partitioned Xdf
#####

# create an input Xdf data source
inFile <- "claims.xdf"
inFile <- file.path(dataPath = rxGetOption(opt = "sampleDataDir"), inFile)
inXdfDS <- RxXdfData(file = inFile)

# create an output partitioned Xdf data source
outFile <- file.path(tempdir(), "partitionedClaims.xdf")
outPartXdfDataSource <- RxXdfData(file = outFile, createPartitionSet = TRUE)

# construct and save the partitioned Xdf to disk
partDF <- rxPartition(inData = inXdfDS, outData = outPartXdfDataSource, varsToPartition = c("car.age"))

#####
# Append new data to an existing partitioned Xdf
#####

# create two sets of data frames from Xdf data source
inFile <- "claims.xdf"
inFile <- file.path(dataPath = rxGetOption(opt = "sampleDataDir"), inFile)
inXdfDS <- RxXdfData(file = inFile)
inDF <- rxImport(inData = inXdfDS)

df1 <- inDF[1:50,]
df2 <- inDF[51:nrow(inDF),]

# create an output partitioned Xdf data source
outFile <- file.path(tempdir(), "partitionedClaims.xdf")
outPartXdfDataSource <- RxXdfData(file = outFile, createPartitionSet = TRUE)

# construct the partitioned Xdf from the first data set df1 and save to disk
partDF1 <- rxPartition(inData = df1, outData = outPartXdfDataSource, varsToPartition = c("car.age",
"type"), append = "none", overwrite = TRUE)

# append data from the second data set to the existing partitioned Xdf
partDF2 <- rxPartition(inData = df2, outData = outPartXdfDataSource, varsToPartition = c("car.age",
"type"))

# overwrite an existing partitioned Xdf
partDF2 <- rxPartition(inData = inXdfDS, outData = outPartXdfDataSource, varsToPartition = c("car.age"),
append = "none", overwrite = TRUE)
```

rxPingNodes: Simple Test of Compute Cluster Nodes

7/12/2022 • 4 minutes to read • [Edit Online](#)

Description

This function provides a simple test of the compute context's ability to perform a round trip through one or more computation nodes.

Usage

```
rxPingNodes(computeContext = rxGetOption("computeContext"), timeout = 0, filter = NULL)
```

Arguments

`computeContext`

A distributed compute context. [RxSpark](#) context is supported. See the details section for more information.

`timeout`

A non-negative integer value. Real valued numbers will be cast to integers. This parameter sets a total (real clock) time to attempt to perform a ping. The timer is not started until the system has first determined which nodes are unavailable (meaning down, unreachable or not usable for jobs, such as scheduler only nodes on LSF). At least one attempt to complete a ping is performed regardless of the setting of `timeout`. If the default value of `0` is used, there is no timeout.

`filter`

`NULL`, or a character vector containing one or more of the ping states. If `NULL`, no filtering is performed. If a character vector of one or more of the ping states is provided, then only those nodes determined to be in the states enumerated will be returned.

Details

This function provides an application level "ping" to one or more nodes in a cluster or cloud. To this end, a trivial job is launched for each node to be pinged. A successful ping means that communication between the end user and the scheduler and communication between the end user and the shared data directory was successful; that R was launched and ran a trivial function successfully on the host being pinged; and that the results were returned successfully.

While this function does not test certain aspects of the messaging required for HPA functions (e.g., `rxSummary`), it does allow the user to easily test the majority of the end-to-end job functionality within a supported cloud or cluster.

The compute context provided is used to determine the cluster or queue that is to be pinged. Furthermore, the nodes to be pinged will be determined in the usual fashion; that is, `NULL` in the `nodes` indicates use of all nodes; values in the `groups` or `queue` fields will cause the set of nodes to be checked to be the intersection between all the nodes in the `groups` or `queue` specified, and the set of nodes specifically specified in the `nodes` parameter,

and so forth. Note that for clusters and clouds that do have a head node, `computeOnHeadNode` is respected. For more information, see the compute context constructors or the `rxGetNodeInfo` for more information.

Most other values in the compute context are respected when determining how a ping will be sent. The following fields in particular are of note when using this tool:

`priority`

May be used to allow the ping jobs to run sooner than other longer running jobs.

`exclusive`

Should usually be avoided

`autoCleanup`

Should almost always be set to `TRUE`; however, may be of use to a system administrator diagnosing a problem.

Value

An object of type `rxPingResults`. This is essentially a list in which component is named using an `rxMakeRNodeNames` translated node name in the same manner and for the same reasons described for `rxGetNodeInfo`, with the `getWorkersOnly` parameter set to FALSE.

Each element of this list contains two elements: `nodeName` which holds the true, unmangled name of the node, and `status`, which contains a character scalar with one of the following values:

`unavail`

The node failed its scheduler level check prior to an attempt to actually ping the node. This does not necessarily mean that the node is not functional; rather, it only means that it cannot support having a job run on it.

`success`

The round trip job was a success.

`failedJob`

The scheduler failed the job. This could be due to permissions, corrupt libraries, or a problem relating to the GUID directory.

`failedSession`

The R process on the worker host was started, but failed.

`timeout`

The ping was sent, but a response was never received. This could be due to a problem with the installation, or other long running jobs being queued ahead of the ping job, or a system failure.

An `as.vector` method is provided for the `rxPingResults` object which returns a character vector of the non-mangled (`rxMakeRNodeNames` translated) node names for use in another compute context, filtered by the `filter` parameter originally provided.

Finally, the `rxPingResults` object has a `logical` attribute associated with it: `allok`. This attribute is set to `TRUE` if all of the pinged nodes' states (after filtering) were set to `"success"`. Otherwise, this attribute is set to `FALSE`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

`rxGetAvailableNodes`, `rxMakeRNodeNames`, `rxgetNodeInfo`, `rxGetAvailableNodes`.

Examples

```
## Not run:

# Attempts to ping all the nodes for the current compute context.
rxPingNodes()

# Pings all the nodes from myCluster, returning values only for those that are
# currently not operational
rxPingNodes( myCluster, filter=c("unavail","failedJob","failedSession") )

# Pings all the nodes from myCluster; times out after 2 minutes
rxPingNodes( myCluster, timeout=120 )

# Extract the allOk attribute from the return value
attr(rxPingNodes(), "allOk")
## End(Not run)
```

rxPredict: Predicted values and residuals for model objects built using RevoScaleR

7/12/2022 • 8 minutes to read • [Edit Online](#)

Description

Compute predicted values and residuals using the following objects: [rxLinMod](#), [rxGlm](#), [rxLogit](#), [rxDTTree](#), [rxBTrees](#), and [rxDForest](#).

Usage

```
rxPredict(modelObject, data = NULL, ...)
## S3 method for class `default':
rxPredict (modelObject, data = NULL, outData = NULL,
           computeStdErrors = FALSE, interval = "none", confLevel = 0.95,
           computeResiduals = FALSE, type = c("response", "link"),
           writeModelVars = FALSE, extraVarsToWrite = NULL, removeMissings = FALSE,
           append = c("none", "rows"), overwrite = FALSE, checkFactorLevels = TRUE,
           predVarNames = NULL, residVarNames = NULL,
           intervalVarNames = NULL, stdErrorsVarNames = NULL, predNames = NULL,
           blocksPerRead = rxGetOption("blocksPerRead"),
           reportProgress = rxGetOption("reportProgress"), verbose = 0,
           xdfCompressionLevel = rxGetOption("xdfCompressionLevel"), ...)
```

Arguments

modelObject

object returned from a RevoScaleR model fitting function. Valid values include [rxLinMod](#), [rxLogit](#), [rxGlm](#), [rxDTTree](#), [rxBTrees](#), and [rxDForest](#). Objects with multiple dependent variables are not supported in `rxPredict`.

data

An [RxXdfData](#) data source object to be used for predictions. If not using a distributed compute context such as RxHadoopMR, a data frame, or a character string specifying the input .xdf file can also be used.

outData

file or existing data frame to store predictions; can be same as the input file or `NULL`. If not `NULL`, a character string specifying the output `.xdf` file, a RxXdfData object, a RxOdbcData data source, or a RxSqlServerData data source. `outData` can also be a delimited [RxTextData](#) data source if using a native file system and not appending.

computeStdErrors

logical value. If `TRUE`, the standard errors for each dependent variable are calculated.

interval

character string defining the type of interval calculation to perform. Supported values are `"none"`, `"confidence"`, and `"prediction"`.

confLevel

numeric value representing the confidence level on the interval [0, 1].

computeResiduals

logical value. If `TRUE`, residuals are computed.

type

Applies to `rxGlm` and `rxLogit`, used to set the type of prediction. Valid values are `"response"` and `"link"`. If `type = "response"`, the predictions are on the scale of the response variable. For instance, for the binomial model, the predictions are in the range (0,1). If `type = "link"`, the predictions are on the scale of the linear predictors. Thus for the binomial model, the predictions are of log-odds.

writeModelVars

logical value. If `TRUE`, and the output data set is different from the input data set, variables in the model will be written to the output data set in addition to the predictions (and residuals, standard errors, and confidence bounds, if requested). If variables from the input data set are transformed in the model, the transformed variables will also be included.

extraVarsToWrite

`NULL` or character vector of additional variables names from the input data or transforms to include in the `outData`. If `writeModelVars` is `TRUE`, model variables will be included as well.

removeMissings

logical value. If `TRUE`, rows with missing values are removed.

append

either `"none"` to create a new file or `"rows"` to append rows to an existing file. If `outData` exists and `append` is `"none"`, the `overwrite` argument must be set to `TRUE`. You can append only to `RxTeradata` data source. Ignored for data frames.

overwrite

logical value. If `TRUE`, an existing `outData` will be overwritten. `overwrite` is ignored if appending rows. Ignored for data frames.

checkFactorLevels

logical value. If `TRUE`, up to 1000 factor levels for the data will be verified against factor levels in the model. Setting to `FALSE` can speed up computations if using lots of factors.

predVarNames

character vector specifying name(s) to give to the prediction results

residVarNames

character vector specifying name(s) to give to the residual results.

intervalVarNames

`NULL` or a character vector defining low and high confidence interval variable names, respectively. If `NULL`, the strings `"_Lower"` and `"_Upper"` are appended to the dependent variable names to form the confidence interval variable names.

stdErrorsVarNames

`NULL` or a character vector defining variable names corresponding to the standard errors, if calculated. If `NULL`, the string `"_StdErr"` is appended to the dependent variable names to form the standard errors variable names.

predNames

character vector specifying name(s) to give to the prediction and residual results; if length is 2, the second name is used for residuals. This argument is deprecated and `predVarNames` and `residVarNames` should be used instead.

`blocksPerRead`

number of blocks to read for each chunk of data read from the data source. If the `data` and `outData` are the same file, `blocksPerRead` must be 1.

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`verbose`

integer value. If `0`, no additional output is printed. If `1`, additional summary information is printed.

`xdfCompressionLevel`

integer in the range of -1 to 9 indicating the compression level for the output data if written to an `.xdf` file. The higher the value, the greater the amount of compression - resulting in smaller files but a longer time to create them. If `xdfCompressionLevel` is set to 0, there will be no compression and files will be compatible with the 6.0 release of Revolution R Enterprise. If set to -1, a default level of compression will be used.

...

additional arguments to be passed directly to the Revolution Compute Engine.

Details

`rxPredict` computes predicted values and/or residuals from an existing model type. The most common way to call `rxPredict` is `rxPredict(modelObject, data, outData)`. Typically, all the other arguments are left at their defaults.

For `rxLogit`, the residuals are equivalent to those computed for `glm` with `type` set to `"response"`, e.g.,
`residuals(glmObj, type="response")`.

If the `data` is the same data used to create the `modelObject`, the predicted values are the fitted values for the original model.

If the `data` specified is an `.xdf` file, the `outData` must be `NULL` or an `.xdf` file. If `outData` is an `.xdf` file, the computed data will be appended as columns. If `outData` is `NULL`, the computed columns will be appended to the `data` `.xdf` file.

If the `data` specified is a data frame, the `outData` must be `NULL` or a data frame. If `outData` is a data frame, a copy of the data frame with the new columns appended will be returned. If `outData` is `NULL`, a vector or list of the computed values will be returned.

If a transformation function is being used for the model estimation, the information variable `.rxIsPrediction` can be used to exclude computations for the dependent variable when running `rxPredict`. See `rxTransform` for an example.

Value

If a data frame is specified as the input `data`, a data frame is returned. If a data frame is specified as the

`outData`, variables containing the results are added to the data frame and it is returned. If `outData` is `NULL`, a data frame containing the predicted values (and residuals and standard errors, if requested) is returned.

If an .xdf file is specified as the input `data`, an `RxXdfData` data source object is returned that can be used in subsequent RevoScaleR analyses. If `outData` is an .xdf file, the `RxXdfData` data source represents the `outData` file. If `outData` is `NULL`, the predicted values (and, if requested, residuals) are appended to the original `data` file. The returned `RxXdfData` object represents this file.

Computing Standard Errors of Predicted Values

Use `computeStdErrors` to control whether or not prediction standard errors are computed.

Use `interval` to control whether confidence or prediction (tolerance) intervals are computed at the specified level (`confLevel`). These are sometimes referred to as *narrow* and *wide* intervals, respectively.

Use `stdErrorsVarNames` to name the standard errors output variable and `intervalVarNames` to specify the output variable names of the lower and upper confidence/tolerance intervals.

In calculating the prediction standard errors, keep the following in mind:

* Prediction standard errors are available for both `rxLinMod` and `rxLogit` models.

* Standard errors are computationally intensive for large models, i.e., those involving a large number of model parameters.

* `rxLinMod` and `rxLogit` must be called with `covCoef = TRUE` because the variance-covariance matrix of the coefficients must be available.

* Cube regressions are not supported (`cube = TRUE`).

* Multiple dependent variables are currently not supported.

* For `rxLogit`, `interval = "confidence"` is supported (unlike `predict.glm`, which does not support confidence bounds), but `interval = "prediction"` is not supported.

* If residuals are requested, and if there are missing values in the dependent variable, then all computed values (prediction, standard errors, confidence levels) will be assigned the value missing, and will be removed if `removeMissings = TRUE`. If no residuals are requested, then missings in the dependent variable (which need not exist in the data) have no effect.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxLinMod](#), [rxLogit](#), [rxGlm](#), [rxPredict.rxDTree](#), [rxPredict.rxDForest](#), [rxPredict.rxNaiveBayes](#).

Examples

```

# Load the built-in iris data set and predict sepal length
myIris <- iris
myIris[1:5,]
form <- Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width + Species
irisLinMod <- rxLinMod(form, data = myIris)
myIrisPred <- rxPredict(modelObject = irisLinMod, data = myIris)
myIris$SepalLengthPred <- myIrisPred$Sepal.Length_Pred
myIris[1:5,]
irisResiduals<- rxPredict(modelObject = irisLinMod, data = myIris, computeResiduals = TRUE)
names(irisResiduals)

# Use sample data to compare lm and glm results with rxPredict
sampleDataDir <- rxGetOption("sampleDataDir")
mortFile <- file.path(sampleDataDir, "mortDefaultSmall.xdf")
linModPredictFile <- file.path(tempdir(), "mortPredictLinMod.xdf")
logitPredictFile <- file.path(tempdir(), "mortPredictLogit.xdf")
mortDF <- rxDataStep(inData = mortFile)

# Compare residuals from rxLinMod with lm
linMod <- rxLinMod(creditScore ~ yearsEmploy, data = mortFile)
rxPredict(modelObject = linMod, data = mortFile, outData = linModPredictFile,
          computeResiduals = TRUE)
residDF <- rxDataStep(inData = linModPredictFile)
mortLM <- lm(creditScore ~ yearsEmploy, data = mortDF)
# Sum of differences should be very small
sum(mortLM$residuals - residDF$creditScore_Resid)

# Create logit model object and compute predictions and residuals
logitModObj <- rxLogit(default ~ creditScore, data = mortFile)
rxPredict(modelObject = logitModObj, data = mortFile,
          outData = logitPredictFile, computeResiduals = TRUE)
residDF <- rxDataStep(inData = logitPredictFile)
mortGLM <- glm(default ~ creditScore, data = mortDF, family = binomial())

# maximum differences should be very small
max(abs(mortGLM$fitted.values - residDF$default_Pred))
max(abs(residuals(mortGLM, type = "response") - residDF$default_Resid))

```

rxPredict.rxDForest: Prediction for Large Data Classification and Regression Forests

7/12/2022 • 4 minutes to read • [Edit Online](#)

Description

Calculate predicted or fitted values for a data set from an object inheriting from class `rxDForest`.

Usage

```
## S3 method for class `rxDForest':  
rxPredict (modelObject, data = NULL,  
           outData = NULL, predVarNames = NULL, writeModelVars = FALSE, extraVarsToWrite = NULL,  
           append = c("none", "rows"), overwrite = FALSE,  
           type = c("response", "prob", "vote"), cutoff = NULL, removeMissings = FALSE,  
           computeResiduals = FALSE, residType = c("usual", "pearson", "deviance"), residVarNames = NULL,  
  
           blocksPerRead = rxGetOption("blocksPerRead"), reportProgress = rxGetOption("reportProgress"),  
           verbose = 0, xdfCompressionLevel = rxGetOption("xdfCompressionLevel"),  
           ... )
```

Arguments

`modelObject`

object inheriting from class `rxDForest`.

`data`

either a data source object, a character string specifying a .xdf file, or a data frame object.

`outData`

file or existing data frame to store predictions; can be same as the input file or `NULL`. If not `NULL`, must be an .xdf file if `data` is an .xdf file or a data frame if `data` is a data frame.

`predVarNames`

character vector specifying name(s) to give to the prediction results.

`writeModelVars`

logical value. If `TRUE`, and the output file is different from the input file, variables in the model will be written to the output file in addition to the predictions. If variables from the input data set are transformed in the model, the transformed variables will also be written out.

`extraVarsToWrite`

`NULL` or character vector of additional variables names from the input data or transforms to include in the `outData`. If `writeModelVars` is `TRUE`, model variables will be included as well.

`append`

either "none" to create a new file or "rows" to append rows to an existing file. If `outData` exists and `append` is "none", the `overwrite` argument must be set to `TRUE`. You can append only to `RxTeradata` data source. Ignored for data frames.

overwrite

logical value. If `TRUE`, an existing `outData` will be overwritten. `overwrite` is ignored if appending rows. Ignored for data frames.

type

character string specifying the type of predicted values to be returned. Supported choices for an object of class `rxDForest` are "response", "prob", and "vote".

- "response" - a vector of predicted values for a regression forest and predicted classes (with majority vote) for a classification forest.
- "prob" - (Classification only) a matrix of predicted class probabilities whose columns are the probability of the first, second, etc. class. It essentially sums up the probability predictions for each class over all the trees and thus may give different class predictions from those obtained with "response" or "vote".
- "vote" - (Classification only) a matrix of predicted vote counts whose columns are the vote counts of the first, second, etc. class.
"class" is also allowed but automatically converted to "response". Supported choices for an object of class `rxBTrees` are "response" and "link".
- "response" - the predictions are on the scale of the response variable.
- "link" - the predictions are on the scale of the linear predictors.

cutoff

(Classification only) a vector of length equal to the number of classes specifying the dividing factors for the class votes. The default is the one used when the decision forest is built.

removeMissing

logical value. If `TRUE`, rows with missing values are removed and will not be included in the output data.

computeResiduals

logical value. If `TRUE`, residuals are computed.

residType

see `residuals.rpart` for details.

residVarNames

character vector specifying name(s) to give to the residual results.

blocksPerRead

number of blocks to read for each chunk of data read from the data source.

reportProgress

integer value with options:

- 0 : no progress is reported.
- 1 : the number of processed rows is printed and updated.
- 2 : rows processed and timings are reported.
- 3 : rows processed and all timings are reported.

verbose

integer value. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information are provided.

`xdfCompressionLevel`

integer in the range of -1 to 9 indicating the compression level for the output data if written to an `.xdf` file. The higher the value, the greater the amount of compression - resulting in smaller files but a longer time to create them. If `xdfCompressionLevel` is set to 0, there will be no compression and files will be compatible with the 6.0 release of Revolution R Enterprise. If set to -1, a default level of compression will be used.

`...`

additional arguments to be passed directly to the Microsoft R Services Compute Engine.

Details

Prediction for large data models requires both a fitted model object and a data set, either the original data (to obtain fitted values and residuals) or a new data set containing the same set of variables as the original fitted model. Notice that this is different from the behavior of `predict`, which can usually work on the original data simply by referencing the fitted model.

Value

Depending on the form of `data`, this function variously returns a data frame or a data source representing a `.xdf` file.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

Breiman, L. (2001) Random Forests. *Machine Learning* **45**(1), 5--32.

Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. (1984) *Classification and Regression Trees*. Wadsworth.

Therneau, T. M. and Atkinson, E. J. (2011) *An Introduction to Recursive Partitioning Using the RPART Routines*.

Yael Ben-Haim and Elad Tom-Tov (2010) A streaming parallel decision tree algorithm. *Journal of Machine Learning Research* **11**, 849--872.

See Also

`rpart`, `rxDForest`, `rxBTrees`.

Examples

```
set.seed(1234)

# classification
iris.sub <- c(sample(1:50, 25), sample(51:100, 25), sample(101:150, 25))
iris.dforest <- rxDForest(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
  data = iris[iris.sub, ], cp = 0.01)
iris.dforest

table(rxPredict(iris.dforest, iris[-iris.sub, ], type = "class")[[1]],
  iris[-iris.sub, "Species"])

# regression
infert.nrow <- nrow(infert)
infert.sub <- sample(infert.nrow, infert.nrow / 2)
infert.dforest <- rxDForest(case ~ age + parity + education + spontaneous + induced,
  data = infert[infert.sub, ], cp = 0.01)
infert.dforest

hist(rxPredict(infert.dforest, infert[-infert.sub, ])[[1]] -
  infert[-infert.sub, "case"])
```

rxPredict.rxDTree: Prediction for Large Data Classification and Regression Trees

7/12/2022 • 3 minutes to read • [Edit Online](#)

Description

Calculate predicted or fitted values for a data set from an rxDTre object.

Usage

```
## S3 method for class 'rxDTre':
rxPredict (modelObject, data = NULL, outData = NULL,
           predVarNames = NULL, writeModelVars = FALSE, extraVarsToWrite = NULL,
           append = c("none", "rows"), overwrite = FALSE,
           type = c("vector", "prob", "class", "matrix"), removeMissings = FALSE,
           computeResiduals = FALSE, residType = c("usual", "pearson", "deviance"), residVarNames = NULL,
           blocksPerRead = rxGetOption("blocksPerRead"), reportProgress = rxGetOption("reportProgress"),
           verbose = 0, xdfCompressionLevel = rxGetOption("xdfCompressionLevel"),
           ... )
```

Arguments

`modelObject`

object returned from a call to `rxDTre`.

`data`

either a data source object, a character string specifying a .xdf file, or a data frame object.

`outData`

file or existing data frame to store predictions; can be same as the input file or `NULL`. If not `NULL`, must be an .xdf file if `data` is an .xdf file or a data frame if `data` is a data frame.

`predVarNames`

character vector specifying name(s) to give to the prediction results.

`writeModelVars`

logical value. If `TRUE`, and the output file is different from the input file, variables in the model will be written to the output file in addition to the predictions. If variables from the input data set are transformed in the model, the transformed variables will also be written out.

`extraVarsToWrite`

`NULL` or character vector of additional variables names from the input data or transforms to include in the `outData`. If `writeModelVars` is `TRUE`, model variables will be included as well.

`append`

either `"none"` to create a new files or `"rows"` to append rows to an existing file. If `outData` exists and `append` is

"none" , the `overwrite` argument must be set to `TRUE`. You can append only to RxTeradata data source. Ignored for data frames.

`overwrite`

logical value. If `TRUE` , an existing `outData` will be overwritten. `overwrite` is ignored if appending rows. Ignored for data frames.

`type`

see predict.rpart for details.

`removeMissings`

logical value. If `TRUE` , rows with missing values are removed and will not be included in the output data.

`computeResiduals`

logical value. If `TRUE` , residuals are computed.

`residType`

see residuals.rpart for details.

`residVarNames`

character vector specifying name(s) to give to the residual results.

`blocksPerRead`

number of blocks to read for each chunk of data read from the data source.

`reportProgress`

integer value with options:

- `0` : no progress is reported.
- `1` : the number of processed rows is printed and updated.
- `2` : rows processed and timings are reported.
- `3` : rows processed and all timings are reported.

`verbose`

integer value. If `0` , no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information are provided.

`xdfCompressionLevel`

integer in the range of -1 to 9 indicating the compression level for the output data if written to an `.xdf` file. The higher the value, the greater the amount of compression - resulting in smaller files but a longer time to create them. If `xdfCompressionLevel` is set to 0, there will be no compression and files will be compatible with the 6.0 release of Revolution R Enterprise. If set to -1, a default level of compression will be used.

`...`

additional arguments to be passed directly to the Microsoft R Services Compute Engine.

Details

Prediction for large data models requires both a fitted model object and a data set, either the original data (to obtain fitted values and residuals) or a new data set containing the same set of variables as the original fitted model. Notice that this is different from the behavior of `predict` , which can usually work on the original data simply by referencing the fitted model.

Value

Depending on the form of `data`, this function variously returns a data frame or a data source representing a .xdf file.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. (1984) *Classification and Regression Trees*. Wadsworth.

Therneau, T. M. and Atkinson, E. J. (2011) *An Introduction to Recursive Partitioning Using the RPART Routines*.

Yael Ben-Haim and Elad Tom-Tov (2010) A streaming parallel decision tree algorithm. *Journal of Machine Learning Research* 11, 849--872.

See Also

`rpart`, `rpart.control`, `rpart.object`, [rxDTree](#).

Examples

```
set.seed(1234)

# classification
iris.sub <- c(sample(1:50, 25), sample(51:100, 25), sample(101:150, 25))
iris.dtree <- rxDTree(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
  data = iris[iris.sub, ], cp = 0.01)
iris.dtree

table(rxPredict(iris.dtree, iris[-iris.sub, ], type = "class")[[1]],
  iris[-iris.sub, "Species"])

# regression
infert.nrow <- nrow(infert)
infert.sub <- sample(infert.nrow, infert.nrow / 2)
infert.dtree <- rxDTree(case ~ age + parity + education + spontaneous + induced,
  data = infert[infert.sub, ], cp = 0.01)
infert.dtree

hist(rxPredict(infert.dtree, infert[-infert.sub, ])[[1]] -
  infert[-infert.sub, "case"])
```

rxPredict.rxNaiveBayes: Prediction for Large Data Naive Bayes Classifiers

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Calculate predicted or fitted values for a data set from an rxNaiveBayes object.

Usage

```
## S3 method for class 'rxNaiveBayes':  
rxPredict (modelObject, data = NULL, outData = NULL, type = c("class", "prob"), prior = NULL,  
predVarNames = NULL, writeModelVars = FALSE, extraVarsToWrite = NULL, checkFactorLevels = TRUE,  
... )
```

Arguments

`modelObject`

object returned from a call to [rxNaiveBayes](#).

`data`

either a data source object, a character string specifying a .xdf file, or a data frame object.

`outData`

file or existing data frame to store predictions; can be same as the input file or `NULL`. If not `NULL`, must be an .xdf file if `data` is an .xdf file or a data frame if `data` is a data frame.

`type`

character string specifying the type of predicted values to be returned. Supported choices are

- `"class"` - a vector of predicted classes.
- `"prob"` - a matrix of predicted class probabilities whose columns are the probability of the first, second, etc. class.

`prior`

a vector of prior probabilities. If unspecified, the class proportions of the data counts in the training set are used. If present, they should be specified in the order of the factor levels of the response and they must be all non-negative and sum to 1.

`predVarNames`

character vector specifying name(s) to give to the prediction results.

`writeModelVars`

logical value. If `TRUE`, and the output file is different from the input file, variables in the model will be written to the output file in addition to the predictions. If variables from the input data set are transformed in the model, the transformed variables will also be written out.

`extraVarsToWrite`

`NULL` or character vector of additional variables names from the input data to include in the `outData`. If

`writeModelVars` is `TRUE`, model variables will be included as well.

`checkFactorLevels`

logical value. If `TRUE`, the factor levels for the data will be verified against factor levels in the model. Setting to `FALSE` can speed up computations if using lots of factors.

...

additional arguments to be passed directly to `rxDataStep` such as `removeMissingsOnRead`, `overwrite`,
`blocksPerRead`, `reportProgress`, `xdfCompressionLevel`.

Details

Prediction for large data models requires both a fitted model object and a data set, either the original data (to obtain fitted values and residuals) or a new data set containing the same set of variables as the original fitted model. Notice that this is different from the behavior of `predict`, which can usually work on the original data simply by referencing the fitted model.

Value

Depending on the form of `data`, this function variously returns a data frame or a data source representing a `.xdf` file.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

Naive Bayes classifier https://en.wikipedia.org/wiki/Naive_Bayes_classifier.

See Also

`rxNaiveBayes`.

Examples

```
# multi-class classification with a data.frame
iris.nb <- rxNaiveBayes(Species ~ ., data = iris)
iris.nb

# prediction
iris.nb.pred <- rxPredict(iris.nb, iris)
iris.nb.pred
table(iris.nb.pred[["Species_Pred"]], iris[["Species"]])
```

rxPrivacyControl: Changes the opt in state for anonymous data usage collection.

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Sets the state to be opted-in or out for anonymous usage collection.

Usage

```
rxPrivacyControl(optIn)
```

Arguments

`optIn`

A logical value that specifies to opt in `TRUE` or to opt out `FALSE` with anonymous data usage collection. If not specified, the value is returned.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

rxQuantile: Approximate Quantiles for .xdf Files and Data Frames

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Quickly computes approximate quantiles (without sorting)

Usage

```
rxQuantile(varName, data, pweights = NULL, fweights = NULL,
probs = seq(0, 1, 0.25), names = TRUE,
maxIntegerBins = 500000, multiple = NA,
numericBins = FALSE, numNumericBreaks = 1000,
blocksPerRead = rxGetOption("blocksPerRead"),
reportProgress = rxGetOption("reportProgress"), verbose = 0)
```

Arguments

`varName`

A character string containing the name of the numeric variable for which to compute the quantiles.

`data`

data frame, character string containing an .xdf file name (with path), or [RxDataSource-class](#) object representing the data set.

`pweights`

character string specifying the variable to use as probability weights for the observations.

`fweights`

character string specifying the variable to use as frequency weights for the observations.

`probs`

numeric vector of probabilities with values in the [0,1] range.

`names`

logical; if `TRUE`, the result has a `names` attribute.

`maxIntegerBins`

integer. The maximum number of integer bins to use for integer data. For exact results, this should be larger than the range of data. However, larger values may increase memory requirements and computational time.

`multiple`

numeric value to multiply data values by before computing integer bins.

`numericBins`

logical. If `TRUE`, do not use integer approximations for bins.

`numNumericBreaks`

integer. The number of breaks to use in computing numeric bins. Ignored if `numericBins` is `FALSE`.

`blocksPerRead`

number of blocks to read for each chunk of data read from an `.xdf` data source.

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`verbose`

integer value. If `0`, no additional output is printed. If `1`, additional computational information may be printed.

Details

`rxQuantiles` computes approximate quantiles by counting binned data, then computing a linear interpolation of the empirical cdf for continuous data or the inverse of empirical distribution function for integer data.

If the binned data are integers, or can be converted to integers by multiplication, the computation is exact when integral bins are used. The size of the bins can be controlled by using the `multiple` function if desired.

Missing values are removed before computing the quantiles.

Value

A vector the length of `probs` is returned; if `names = TRUE`, it has a names attribute.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

`quantile`, `rxCube`.

Examples

```
# Estimate a GLM model and compute quantiles for the predictions
claimsXdf <- file.path(rxGetOption("sampleDataDir"),"claims.xdf")
claimsPred <- tempfile(pattern = "claimsPred", fileext = ".xdf")

claimsGlm <- rxGlm(cost ~ age + car.age + type, family = Gamma,
                     dropFirst = TRUE, data = claimsXdf)

rxPredict(claimsGlm, data = claimsXdf, outData = claimsPred,
          writeModelVars = TRUE, overwrite = TRUE)

predBreaks <- rxQuantile(data = claimsPred, varName = "cost_Pred",
                          probs = seq(from = 0, to = 1, by = .1))

predBreaks

# Compare with the quantile function
claimsPredDF <- rxDataStep(inData = claimsPred)
quantile(claimsPredDF$cost_Pred, probs = seq(0, 1, by = .1), type = 4)

file.remove(claimsPred)
```

rxReadXdf: Read .xdf File

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Read data from an .xdf file into a data frame.

Usage

```
rxReadXdf(file, varsToKeep = NULL, varsToDrop = NULL, rowVarName = NULL,
          startRow = 1, numRows = -1, returnDataFrame = TRUE,
          stringsAsFactors = FALSE, maxRowsByCols = NULL,
          reportProgress = rxGetOption("reportProgress"), readByBlock = FALSE,
          cppInterp = NULL)
```

Arguments

`file`

either an RxXdfData object or a character string specifying the .xdf file.

`varsToKeep`

character vector of variable names to include when reading from the input data file. If `NULL`, argument is ignored. Cannot be used with `varsToDrop`.

`varsToDrop`

character vector of variable names to exclude when reading from the input data file. If `NULL`, argument is ignored. Cannot be used with `varsToKeep`.

`rowVarName`

optional character string specifying the variable in the data file to use as row names for the output data frame.

`startRow`

starting row for retrieval.

`numRows`

number of rows of data to retrieve. If -1, all are read.

`returnDataFrame`

logical indicating whether or not to create a data frame. If `FALSE`, a list is returned.

`stringsAsFactors`

logical indicating whether or not to convert strings into factors in R.

`maxRowsByCols`

the maximum size of a data frame that will be read in, measured by the number of rows times the number of columns. If the number of rows times the number of columns being extracted from the .xdf file exceeds this, a warning will be reported and a smaller number of rows will be read in than requested. If `maxRowsByCols` is set to be too large, you may experience problems from loading a huge data frame into memory. To extract a subset of

rows and/or columns from an .xdf file, use [rxDataStep](#).

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`readByBlock`

read data by blocks. This argument is deprecated.

`cppInterp`

list of information sent to C++ interpreter.

Value

if `returnDataFrame` is `TRUE`, a data frame is returned. Otherwise a list is returned.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxImport](#), [rxDataStep](#).

Examples

```
# Example reading the first 10 rows from the CensusWorkers xdf file
censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers")
myCensusDF <- rxReadXdf(censusWorkers, numRows = 10)
myCensusDF
```

rxRealTimeScoring: Real-time scoring in SQL Server ML Services

7/12/2022 • 6 minutes to read • [Edit Online](#)

Description

Real-time scoring brings the `rxPredict` functionality available in **RevoScaleR**/**revoscalepy** and **MicrosoftML** packages to Microsoft ML Server and SQL Server platforms with near real-time performance.

This functionality is available on SQL Server 2017+. On SQL Server 2016, you can take advantage of this functionality by upgrading your in-database R Services to the latest Microsoft ML Server using the information in the following [link](#).

NOTE: This document contains information regarding **Real-time scoring in SQL Server ML Services**. For information regarding **Real-time scoring in ML Server**, please refer to the [publishService](#) documentation.

Details

With Microsoft R Server 9.1 version, we are introducing this feature which allows models trained using **RevoScaleR**(**revoscalepy**) and **MicrosoftML** packages to be used for high performance scoring in SQL Server. These models can be published and scored without using R interpreter, which reduces the overhead of multiple process interactions. Hence it allows for faster prediction performance.

The following is the list of models that are currently supported in real-time scoring:

* **RevoScaleR**

* `rxLogit`

* `rxLinMod`

* `rxBTrees`

* `rxDTree`

* `rxDForest`

* **MicrosoftML**

* `rxFastTrees`

* `rxFastForest`

* `rxLogisticRegression`

* `rxOneClassSvm`

* `rxNeuralNet`

* `rxFastLinear`

Workflow for Real-time scoring in SQL Server

The high-level workflow for real-time scoring in SQL Server is as follows:

- 1 Serialize model for real-time scoring

1 Publish model to SQL Server

1 Enable real-time scoring functionality in SQL Server ML Services using RegisterRExt tool

1 Call real-time scoring stored procedure using T-SQL

The following sections describe each of these steps in details. These steps are illustrated with a sample in the Example section below.

1. Serialize model for Real-time scoring

To enable this functionality in SQL Server, we are adding support for serializing the model trained in R in a specific `raw` format that allows the model to be scored in an efficient manner. The serialized model can then be stored in a SQL Server database table for doing real-time scoring. The following new APIs are introduced to allow this serialization functionality

- * `rxSerializeModel()` - Serialize a **RevoScaleR/MicrosoftML** model in `raw` format to enable saving the model to a database. This enables the model to be loaded into SQL Server for real-time scoring.
- * `rxUnserializeModel()` - Retrieve the original R model object from the serialized raw model.

2. Publish model to SQL Server

The serialized models can be published to the target SQL Server Database table in `VARBINARY` format for real-time scoring. You can use the existing `rxWriteObject` API to publish the model to SQL Server. Alternatively, you can also save the `raw` type to a file and load into SQL Server.

- * `rxWriteObject()` - Store/retrieve R objects to/from ODBC data sources like SQL Server. The API is modeled after a simple key value store.

NOTE: Since the model is already serialized via `rxSerializeModel` there is no need to additionally serialize in `rxWriteObject`, so the `serialize` flag need to set to `FALSE`.

3. Enable Real-time scoring functionality in SQL Server ML Services

By default, real-time scoring functionality is disabled on SQL Server ML Services and it needs to be enabled for the instance and particular SQL database. To use this functionality, the server administrator needs use the RegisterRExt.exe tool.

`RegisterRExt.exe` is the command line utility which ships with RevoScaleR package and allows administrators to enable real-time scoring feature in the SQL server instance and database. You can find RegisterRExt.exe at either of the following locations:

R Services - `<SQLInstancePath>\R_SERVICES\library\RevoScaleR\rxLibs\x64\RegisterRExt.exe`.

Python Services - `<SQLInstancePath>\PYTHON_SERVICES\Lib\site-packages\RevoScalePy\rxLibs\RegisterRExt.exe`.

3a. Enable real-time scoring for SQL Server instance

Open an elevated command prompt and run the following command:

* `RegisterRExt.exe /installRts [/instance:name] [/python]`

To disable real-time scoring for SQL Server instance, open an elevated command prompt and run the following command:

* `RegisterRExt.exe /uninstallrts [/instance:name] [/python]`

3b. Enable real-time scoring for a particular database

Open an elevated command prompt and run the following command:

```
* RegisterRExt.exe /installRts [/instance:name] /database:[databasename] [/python]
```

To disable real-time scoring for a particular database, open an elevated command prompt and run the following command:

```
* RegisterRExt.exe /uninstallrts /database:databasename [/instance:name] [/python]
```

The /instance flag is optional if the database is part of the default SQL Server instance. This command will create assemblies and stored procedure on the SQL Server database to allow real-time scoring. Additionally, it creates a database role `rxpredict_users` to help database admins to grant permission to use the real-time scoring functionality.

/python flag is required only if using PYTHON SERVICES version of registerext.exe

NOTE: In SQL Server 2016, RegisterRExt will enable SQL CLR functionality in the instance and the specific database will be marked as `TRUSTWORTHY`. Please carefully consider additional security implications of doing this. In SQL Server 2017 and higher, SQL CLR will be enabled and specific CLR assemblies are trusted using `sp_addtrustedassembly`.

4. Call real-time scoring stored procedure using T-SQL

The functionality once installed, will be available in the particular SQL Server database via a stored procedure called `sp_rxPredict`. Here is the syntax of the `sp_rxPredict` stored procedure

Syntax

```
sp_rxPredict  
@model [VARBINARY](max) ,  
@inputData [NVARCHAR](max)
```

where

- * `@model` - Real-time model to be used for scoring in `VARBINARY` format
- * `@inputData` - SQL query to be used to get the input data for scoring

Limitations:

1 Real-time scoring does not use an R/python interpreter for scoring hence any functionality requiring R interpreter is not supported. Here are a few unsupported scenarios:

- * Models of `rxGlm` and `rxNaiveBayes` algorithms are not currently supported
- * **RevoScaleR** Models with R transform function or transform based formula (e.g `"A ~ log(B)"`) are not supported in real-time scoring. Such transforms to input data may need to be handled before calling real-time scoring.

1 Arguments other than `modelObject` / `data` available in `rxPredict` are not supported in real-time scoring

Known Issues:

1 `sp_rxPredict` with `RevoScaleR` model only supports only following .NET column types are double, float, short, ushort, long, ulong and string. You may need to filter out unsupported types in your input data before using it for real-time scoring. For corresponding SQL type information refer [SQL-CLR Type Mapping](#)

1 `sp_rxPredict` is optimized for fast predictions on smaller data (from a few rows to a few 1000 rows), the performance may start degrading on larger data sets.

1 `sp_rxPredict` returns inaccurate error message when NULL value is passed as model

`System.Data.SqlTypes.SqlNullValueException: Data in Null`

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxSerializeModel](#), [rxWriteObject](#), [publishService](#), [rxPredict](#)

Examples

```

## Not run:

form <- Sepal.Length ~ Sepal.Width + Species
model <- rxLinMod(form, data = iris)
library(RevoScaleR)
# 1. Serialize model for real-time scoring
serializedModel <- rxSerializeModel(model)

# 2: Publish model to database
server <- ".\SQL2016"
database <- "revotestdb"
modelTable <- "modelTable"

# Create a table for storing the real-time scoring model
connectionString <- paste0("Driver=SQL Server;Server=", server, ";Database=", database,
";Trusted_Connection=True")
odbcDS <- RxOdbcData(table = modelTable, connectionString = connectionString)
if (!rxSqlServerTableExists(table = modelTable, connectionString = connectionString))
{
  ddlCreateTable <- paste(" create table [", modelTable, "] (",
    "      [id] varchar(200) not null, ",
    "      [value] varbinary(max), ",
    "      constraint unique_id unique (id))",
    sep = "")
  rxExecuteSQLDDL(odbcDS, ddlCreateTable)
}

modelId <- "linModModel" # Any key to identify the model
rxWriteObject(dest = odbcDS, key = modelId, value = serializedModel, serialize = FALSE, overwrite=TRUE) # serialize is FALSE since the model is already serialized into raw in Step 1

# 3: Enable real-time scoring functionality using RegisterRExt (see above)

# 4: Call real-time scoring stored procedure (sp_rxPredict) using T-SQL
query <- paste0("
  DECLARE @model VARBINARY(max);
  SELECT @model = value FROM ", modelTable, " WHERE id = ''", modelId, "';
  EXEC sp_rxPredict @model = @model, @inputData =
  SELECT
  CAST(3.1 AS FLOAT) AS [Sepal.Width],
  ''virginica'' AS Species", """")
# Use CAST to map numeric types to SQL Float

# T-SQL Example above uses inline data in the query. Alternatively, a data could come from a table in a SQL
# database

cat(query) # TSQL to be executed in SQL Server
# The above command displays the T-SQL that can be executed in a SQL Server for real-time scoring

# To test this from R you can execute the following test method
rtsResult <- RevoScaleR:::rxExecuteSQL(connectionString = connectionString, query = query)

## End(Not run)

```

rxRemoteCall: Remote calling mechanism for distributed computing.

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Function that is executed on the master node in a distributed computing environment. This function should **not** be called directly by the user and, rather, serves as a necessary and convenient mechanism for performing remote R calls on the master node in a distributed computing environment.

Usage

```
##Arguments
```

```
rxCommArgsList
```

list of arguments

```
##Author(s)
```

Microsoft Corporation [Microsoft Technical Support](#)

rxRemoteFilePath: Merge path using the remote platform's separator.

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

This is a utility function to automatically build a path according to file path rules of the remote compute context platform.

Usage

##Arguments

...

character strings containing parts of the path to be merged.

computeContext

An [RxComputeContext](#) context object specifying the platform information.

##Author(s) Microsoft Corporation [Microsoft Technical Support](#)

##See Also

[RxComputeContext](#), [RxHadoopMR](#), [RxSpark](#).

##Examples

```
## Not run:  
  
# myHadoopCC is an RxHadoopMR compute context  
myFilePath <- rxRemoteFilePath("/var/RevoShare", "testDir", myHadoopCC)  
## End(Not run)
```

rxRemoteGetId: Provides a unique identifier for the process.

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Provides a unique identifier for the process in the range [1:N] where N is the number of processes.

Usage

```
rxRemoteGetId()
```

Details

This function is intended for use in calls to `rxExec` to identify the process performing a given computation.

You can also use `rxRemoteGetId` to obtain separate random number seeds. See the examples.

Value

In a distributed compute context, this returns the parametric ID for each node or core that executes the function. If the compute context is local, this always returns `-1`.

Examples

```
## Not run:

rxExec(rxRemoteGetId)

# set a seed on each node before generating random numbers
myNorm <- function(x)
{
  set.seed(rxRemoteGetId())
  rnorm(x)
}
rxExec(myNorm, ARGS=as.list(1:5))
## End(Not run)
```

rxRemoteHadoopMRCall: Remote calling mechanism for distributed computing on Hadoop MapReduce clusters.

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Function that is executed by a master process in a Hadoop MapReduce cluster. This function should **not** be called directly by the user. It serves as an internal mechanism for performing remote and distributed R calls on a Hadoop MapReduce cluster.

Usage

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

rxRemovePackages: Remove Packages from Compute Context

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Removes installed packages from a compute context.

Usage

```
rxRemovePackages(pkgs, lib, dependencies = TRUE, checkReferences = TRUE,
  verbose = getOption("verbose"), scope = "private", owner = '',
  computeContext = rxGetOption("computeContext"))
```

Arguments

`pkgs`

a `character` vector of names of the packages to be removed.

`lib`

a `character` vector identifying library path from where the package needs to be removed. This argument is not supported in `RxInSqlServer` compute context. Only valid for a local compute context.

`dependencies`

logical. Applicable only for `RxInSqlServer` compute context. If `TRUE`, does dependency resolution of the packages being removed and removes the dependent packages also if the dependent packages aren't referenced by other packages outside the dependency closure.

`checkReferences`

logical. Applicable only for `RxInSqlServer` compute context. If `TRUE`, verifies there are no references to the dependent packages by other packages outside the dependency closure.

`verbose`

logical. If `TRUE`, "progress report" is given during removal of given packages.

`scope`

character. Applicable only for `RxInSqlServer` compute context. Should be either `"shared"` or `"private"`. `"shared"` removes the packages from a per-database shared location on SQL Server which in turn could have been used (referred) by multiple different users. `"private"` removes the packages from a per-database, per-user private location on SQL Server which is only accessible to the single user.

`owner`

character. Applicable only for `RxInSqlServer` compute context. This is generally empty `''` value. Should be either empty `''` or a valid SQL database user account name. Only users in `'db_owner'` role for a database can specify this value to remove packages on behalf of other users.

`computeContext`

an [RxComputeContext](#) or equivalent character string or `NULL`. If set to the default of `NULL`, the currently active compute context is used. Supported compute contexts are [RxInSqlServer](#), [RxLocalSeq](#).

Details

This is a simple wrapper for remove.packages. For [RxInSqlServer](#) compute context, the user specified as part of connection string is used for removing the packages if `owner` argument is empty. The user calling this function needs to be granted permissions by database owner by making them member of either `'rpkgs-shared'` or `'rpkgs-private'` database role. Users in `'rpkgs-shared'` role can remove packages from `"shared"` location and their own `"private"` location. Users in `'rpkgs-private'` role can only remove packages from their own `"private"` location.

Value

Invisible `NULL`

See Also

[rxPackage](#), [remove.packages](#), [rxFindPackage](#), [rxInstalledPackages](#), [rxInstallPackages](#),

[rxSyncPackages](#), [rxSqlLibPaths](#),

`require`

Examples

```

## Not run:

#
# create SQL compute context
#
sqlcc <- RxInSqlServer(connectionString =
  "Driver=SQL Server;Server=myServer;Database=TestDB;Trusted_Connection=True;")

pkgs <- c("dplyr")

#
# Remove a package and its dependencies after checking references from private scope
#
rxRemovePackages(pkgs = pkgs, verbose = TRUE, scope = "private", computeContext = sqlcc)

#
# Remove a package and its dependencies after checking references from shared scope
#
rxRemovePackages(pkgs = pkgs, verbose = TRUE, scope = "shared", computeContext = sqlcc)

#
# Forcefully remove a package and its dependencies without checking references from
# private scope
#
rxRemovePackages(pkgs = pkgs, checkReferences = FALSE, verbose = TRUE,
  scope = "private", computeContext = sqlcc)

#
# Force fully remove a package without its dependencies and without
# any references checking from shared scope private scope
#
rxRemovePackages(pkgs = pkgs, dependencies = FALSE, checkReferences = FALSE,
  verbose = TRUE, scope = "shared", computeContext = sqlcc)

#
# Remove a package and its dependencies from private scope for user1
#
rxRemovePackages(pkgs = pkgs, verbose = TRUE, scope = "private", owner = "user1", computeContext = sqlcc)

#
# Remove a package and its dependencies from shared scope for user1
#
rxRemovePackages(pkgs = pkgs, verbose = TRUE, scope = "shared", owner = "user1", computeContext = sqlcc)
## End(Not run)

```

rxResultsDF: Crosstab Counts or Sums Data Frame

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Obtain a counts, sums, or means data frame from an analysis object.

Usage

```
## S3 method for class `rxCrossTabs':  
rxResultsDF (object, output = "counts", element = 1,  
    integerLevels = NULL, integerCounts = TRUE, ...)  
## S3 method for class `rxCube':  
rxResultsDF (object, output = "counts",  
    integerLevels = NULL, integerCounts = TRUE, ...)  
## S3 method for class `rxLinMod':  
rxResultsDF (object, output = "counts",  
    integerLevels = NULL, integerCounts = TRUE, ...)  
## S3 method for class `rxLogit':  
rxResultsDF (object, output = "counts",  
    integerLevels = NULL, integerCounts = TRUE, ...)  
## S3 method for class `rxSummary':  
rxResultsDF (object, output = "stats", element = 1,  
    integerLevels = NULL, integerCounts = TRUE, useRowNames = TRUE, ...)
```

Arguments

`object`

object of class `rxCrossTabs`, `rxCube`, `rxLinMod`, `rxLogit`, or `rxSummary`.

`output`

character string specifying the type of output to display. Typically this is `"counts"`, or for `rxSummary` `"stats"`. If there is a dependent variable in the `rxCrossTabs` formula, `"sums"` and `"means"` can be used.

`element`

integer specifying the element number from the object list to extract. Currently only `1` is supported.

`integerLevels`

logical scalar or `NULL`. If `TRUE`, the first column of the returned data frame will be converted to integers. If `FALSE`, it will be a factor column. If `NULL`, it will be returned as an integer column if it was wrapped in an `F()` in the formula.

`integerCounts`

logical scalar. If `TRUE`, the counts or sums in the returned data frame will be converted to integers. If `FALSE`, they will be numeric doubles.

`useRowNames`

logical scalar. If `TRUE`, the names of the variables for which there are results will be put into the row names for the data frame rather than in a separate `Names` variable.

...

additional arguments to be passed directly to the underlying print method for the output list object.

Details

Method to access counts, sums, and means data frames from RevoScaleR analysis objects.

Value

a data frame containing the computed counts, sums, or means.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxCrossTabs](#) [rxCube](#) [rxLinMod](#) [rxLogit](#) [rxSummary](#) [rxMarginals](#)

Examples

```
admissions <- as.data.frame(UCBAdmissions)
myCrossTab1 <- rxCrossTabs(~Gender : Admit, data = admissions)
countsDF1 <- rxResultsDF(myCrossTab1)

# If a dependent variable is used, "sums" and "means" can be used
myCrossTab2 <- rxCrossTabs(Freq ~ Gender : Admit, data = admissions)
countsDF <- rxResultsDF(myCrossTab2)
sumsDF <- rxResultsDF(myCrossTab2, output = "sums")
meansDF <- rxResultsDF(myCrossTab2, output = "means")
```

rxRiskRatio: Relative Risk Ratio and Odds Ratio

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Calculate the relative risk and odds ratio on a two-by-two table.

Usage

```
rxRiskRatio(tbl, conf.level = 0.95, alternative = "two.sided")
rxOddsRatio(tbl, conf.level = 0.95, alternative = "two.sided")
```

Arguments

`tbl`

two-by-two table.

`conf.level`

level of the confidence interval.

`alternative`

character string defining alternative hypothesis. Supported values are `"two.sided"`, `"less"`, or `"greater"`.

Value

an object of class `htest`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxChiSquaredTest](#), [rxFisherTest](#), [rxKendallCor](#), [rxMultiTest](#), [rxPairwiseCrossTab](#)

Examples

```
mtcarsDF <- rxFactors(datasets::mtcars, factorInfo = c("gear", "cyl", "vs"),
                        varsToKeep = c("gear", "cyl", "vs"), sortLevels = TRUE)
gearVsXtabs <- rxCrossTabs(~ gear : vs, data = mtcarsDF, returnXtabs = TRUE)
rxRiskRatio(gearVsXtabs[1:2,])
rxOddsRatio(gearVsXtabs[1:2,])
```

rxRngNewStream: Parallel Random Number Generation

7/12/2022 • 3 minutes to read • [Edit Online](#)

Description

R interface to Parallel Random Number Generators (RNGs) in MKL.

Usage

```
rxRngNewStream(kind = "MT2203", subStream = 0, seed = NULL, normalKind = NULL)
rxRngGetStream()
rxRngSetStream(streamState)
rxRngDelStream(kind = "default", normalKind = "default")
```

Arguments

kind

a character string specifying the desired kind of random number generator. The available kinds are "MCG31", "R250", "MRG32K3A", "MCG59", "MT19937", "MT2203", "SFMT19937", "SOBOL". While "MT2203" allows creating up to 6024 independent random streams, the other kinds allow only 1.

subStream

an integer value identifying the index of the independent random stream. The valid value is from 0 to 6023 for "MT2203", and 0 for the other kinds.

seed

an integer that will be used to initialize the random number generator.

normalKind

a character string specifying the method of Normal generation. The default `NULL` means no change, and generally will allow random normals to be generated by means of inverting the cumulative distribution function. Other allowable values are `"BOXMULLER"`, `"BOXMULLER2"`, (both similar to the standard R `"Box-Muller"` option described in `Random`), and `"ICDF"`, an alternate implementation of the standard R `"Inversion"`.

streamState

an integer vector containing the state of the random number generator.

Details

`rxRngNewStream` allocates memory for storing the RNG state and `rxRngDelStream` needs to be called to release the memory. These functions set or modify the current random number state and are implemented as "user-defined" random number generators. See `Random` and `Random.user` for more details.

Most of the basic random number generators are *pseudo-random* number generators; random number streams are deterministically generated from a given seed, but give the appearance of randomness. The `"SOBOL"`

random number generator is a *quasi*-random number generator; it is simply a sequence that cover a given hypercube in roughly the same way as would truly random points from a uniform distribution. The quasi-random number generators always return the same sequences of numbers and are thus perfectly correlated. They can be useful, however, in fields such as Monte Carlo integration.

Value

`rxRngNewStream`

returns invisibly a two-element character vector of the RNG and normal kinds used *before* the call. This is the same as the object returned by `RNGkind`; see `Random` for details. If the previous random number state was generated by a call to `rxRngNewStream`, the first element will be `"user-defined"`, otherwise one of the strings listed in `Random`.

`rxRngGetStream`

returns an integer vector containing the current RNG state.

`rxRngSetStream`

returns invisibly an integer vector containing the current RNG state and replaces the current active RNG state with the new one.

`rxRngDelStream`

returns invisibly an integer vector containing the current RNG state and delete the current active RNG.

Note

In addition to the independent streams provided by "MT2203", independent streams can also be obtained from some of the other kinds using block-splitting and/or leapfrogging methods.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

Intel Math Kernel Library, Reference Manual.

Intel Math Kernel Library, Vector Statistical Library Notes.

http://software.intel.com/sites/products/documentation/doclib/mkl_sa/11/vslnotes/vslnotes.pdf

Vector Statistical Library (VSL) Performance Data.

http://software.intel.com/sites/products/documentation/doclib/mkl_sa/11/vsl/vsl_performance_data.htm.

See Also

`Random`, `Random.user`.

Examples

```

## Not run:

## initialize, save, load, and delete
num <- 5
kinds <- c("MCG31", "R250", "MRG32K3A", "MCG59",
"MT19937", "MT2203", "SFMT19937", "SOBOL")

for (kind in kinds) {
  kind.old <- rxRngNewStream (kind = kind)
  a <- runif (num)
  saved <- rxRngGetStream ()
  a1 <- runif (num)
  rxRngSetStream (saved)
  a2 <- runif (num)
  rxRngDelStream(kind.old[1], kind.old[2])

  stopifnot( all (a1 == a2) )
}

## parallel random number generation
rxOptions(numCoresToUse=4)
oldcc <- rxSetComputeContext("localpar")
"ParallelRNG" <- function( RngKind = "MT2203", subStream = 0, seed = NULL, RnGenerator = "runif", length = 1
)
{
  rxRngNewStream( kind = RngKind, subStream = subStream, seed = seed )
  do.call( RnGenerator, list( length ) )
}
# generates 5 uniform random numbers on each of 4 workers, giving the same streams on workers
# 1 and 3 and on 2 and 4, respectively
rxExec( ParallelRNG, RngKind = "MT2203", subStream = rxElemArg( c( 0, 1, 0, 1 ) ),
       seed = 17, RnGenerator = "runif", length = 5 )
rxSetComputeContext(oldcc)
## End(Not run)

```

rxRoc: Receiver Operating Characteristic (ROC) computations and plot

7/12/2022 • 5 minutes to read • [Edit Online](#)

Description

Compute and plot an ROC curve using actual and predicted values from binary classifier system

Usage

```
rxRoc(actualVarName, predVarNames, data, numBreaks = 100,
       removeDups = TRUE, blocksPerRead = 1, reportProgress = 0)

rxRocCurve(actualVarName, predVarNames, data, numBreaks = 100,
           blocksPerRead = 1, reportProgress = 0, computeAuc = TRUE, title = NULL,
           subtitle = NULL, xTitle = NULL, yTitle = NULL, legend = NULL,
           chanceGridLine = TRUE, ...)

## S3 method for class `rxRoc':
as.data.frame ( x, ..., var = NULL)

## S3 method for class `rxRoc':
rxAuc ( x )

## S3 method for class `rxRoc':
plot (x, computeAuc = TRUE, title = NULL, subtitle,
       xTitle = NULL, yTitle = NULL, legend = NULL, chanceGridLine = TRUE, ...)
```

Arguments

`actualVarName`

A character string with the name of the variable containing actual (observed) binary values.

`predVarNames`

A character string or vector of character strings with the name(s) of the variable containing predicted values in the [0,1] interval.

`data`

data frame, character string containing an .xdf file name (with path), or `RxXdfData` object representing an .xdf file containing the actual and observed variables.

`numBreaks`

integer specifying the number of breaks to use to determine thresholds for computing the true and false positive rates.

`removeDups`

logical; if `TRUE`, rows containing duplicate entries for sensitivity and specificity will be removed from the returned data frame. If performing computations for more than one prediction variable, this implies that there may be a different number of rows for each prediction variable.

`blocksPerRead`

number of blocks to read for each chunk of data read from the data source.

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`computeAuc`

logical value. If `TRUE`, the AUC is computed for each prediction variable and printed in the subtitle or legend text.

`title`

main title for the plot. Alternatively `main` can be used. If `NULL` a default title will be created.

`subtitle`

subtitle (at the bottom) for the plot. If `NULL` and `computeAuc` is `TRUE`, the AUC for a single prediction variable will be computed and printed in the subtitle.

`xTitle`

title for the X axis. Alternatively `xlab` can be used. If `NULL`, a default X axis title will be used.

`yTitle`

title for the Y axis. Alternatively `ylab` can be used. If `NULL`, a default Y axis title will be used.

`legend`

logical value. If `TRUE` and more than one prediction variable is specified, a legend is created. If `computeAuc` is `TRUE`, the AUC is computed for each prediction variable and printed in the legend text.

`chanceGridLine`

logical value. If `TRUE`, a grid line from (0,0) to (1,1) is added to represent a pure chance model.

`x`

an rxRoc object.

`var`

an integer or character string specifying the prediction variable for which to extract data frame containing the ROC computations. If an integer is specified, it will use that as an index to an alphabetized list of `predictionVarNames`. If `NULL`, all of the computed data will be returned in a data frame.

`...`

additional arguments to be passed directly to an underlying function. For plotting functions, these are passed to the `xyplot` function.

Details

`rxRoc` computes the sensitivity (true positive rate) and specificity (true negative rate) using a variable containing actual (observed) zero and one values and a variable containing predicted values in the unit interval as the discrimination threshold is varied. The thresholds are determined by the `numBreaks` argument. The

computations are done on chunks of data, so that they can be performed on very large data sets. If more than one prediction variable is specified, the computations will be performed for each prediction variable. Observations that have a missing value for the actual value or any of the prediction values are removed before computations are performed.

The `rxRocCurve` and the S3 `plot` method for an `rxRoc` object plot the computed sensitivity (true positive rate) versus 1 - specificity (false positive rate). ROC curves were first used during World War II for detecting enemy objects in battle fields.

Value

`rxRoc` returns a data frame of class `"rxRoc"` containing four variables: `threshold`, `sensitivity`, `specificity`, and `predVarName` (a factor variable containing the prediction variable name).

The `rxAuc` S3 method for an `rxRoc` object returns the AUC (area under the curve) summary statistic.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxPredict](#), [rxLogit](#), [rxGlm](#), [rxLinePlot](#).

Examples

```
#####
# Example using simple created actual and prediction data
#####
# Create a data frame with made-up actual and predicted values
sampleDF <- data.frame(actual = c(0,0,0,0,0, 1,1,1,1,1))
sampleDF$prediction <- c(.6, .5, .4, .3, .2, .8, .7, .6, .5, .4)
# Add predictions that are all wrong and all right
sampleDF$wrongPrediction <- c(.99, .99, .99, .99, .99, .01, .01, .01, .01, .01)
sampleDF$rightPrediction <- c( .01, .01, .01, .01,.99, .99, .99, .99, .99)
# Compute the ROC information for all three prediction variables
rocOut <- rxRoc(actualVarName = "actual", predVarNames =
  c("prediction", "wrongPrediction", "rightPrediction"),
  data = sampleDF, numBreaks = 10)
# View the computed sensitivity and specificity
rocOut
# Plot the results
plot(rocOut, title = "ROC Curve for Simple Data",
  lineStyle = c("solid", "twodash", "dashed"))
#####
# Example using data frame with one predicted variable
#####
# Estimate a logistic regression model using the internal 'infert' data
rxLogitOut <- rxLogit(case ~ spontaneous + induced, data=infert )

# Compute predictions for the model, creating a new data frame with
# predictions and the original data used to estimate the model
rxPredOut <- rxPredict(modelObject = rxLogitOut, data = infert,
  writeModelVars = TRUE, predVarNames = "casePred1")

# Compute the ROC data for the default number of thresholds
rxRocObject <- rxRoc(actualVarName = "case", predVarNames = c("casePred1"),
  data = rxPredOut)

# Draw the ROC curve
plot(rxRocObject)
```

```

#####
# Example using a data frame with two predicted variables and rxRocCurve
#####

# As in first example, estimate a logistic regression model and
# compute predictions

logitOut1 <- rxLogit(case ~ spontaneous + induced, data=infert )

predOut <- rxPredict(modelObject = logitOut1, data = infert,
  writeModelVars = TRUE, predVarNames = "Model1")

# Estimate another model, and add predictions to prediction data frame
logitOut2 <- rxLogit(case ~ spontaneous + induced + parity, data=infert )
predOut <- rxPredict(modelObject = logitOut2, data = infert,
  outData = predOut, predVarNames = "Model2")

# Do computations and plot ROC curve
rxRocCurve(actualVarName = "case", predVarNames = c("Model1", "Model2"),
  data = predOut,
  title = "ROC Curves for 'case', including 'parity' in Model2")

#####
# Example using xdf files
#####
mortXdf <- file.path(rxGetOption("sampleDataDir"), "mortDefaultSmall")

logitOut1 <- rxLogit(default ~ creditScore + yearsEmploy + ccDebt,
  data = mortXdf, blocksPerRead = 5)

predFile <- tempfile(pattern = ".rxPred", fileext = ".xdf")

# predOutXdf will be a data source object representing the
# prediction xdf file (predFile)
predOutXdf <- rxPredict(modelObject = logitOut1, data = mortXdf,
  writeModelVars = TRUE, predVarNames = "Model1", outData = predFile)

# Estimate a second model without ccDebt
logitOut2 <- rxLogit(default ~ creditScore + yearsEmploy,
  data = predOutXdf, blocksPerRead = 5)

# Add predictions to prediction data file
predOutXdf <- rxPredict(modelObject = logitOut2, data = predOutXdf,
  predVarNames = "Model2")

rxRocCurve(actualVarName = "default",
  predVarNames = c("Model1", "Model2"),
  data = predOutXdf)

# Remove temporary file storing predictions
file.remove(predFile)

```

RxSasData-class: Class RxSasData

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

SAS data source connection class.

Generators

The targeted generator [RxSasData](#) as well as the general generator [rxNewDataSource](#).

Extends

Class RxFileData, directly. Class RxDataSource, by class RxFileData.

Methods

[show](#)

```
signature(object = "RxSasData") : ...
```

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxDataSource-class](#), [RxSasData](#), [rxNewDataSource](#)

RxSasData: Generate SAS Data Source Object

7/12/2022 • 5 minutes to read • [Edit Online](#)

Description

Generate an RxSasData object that contains information about a SAS data set to be imported or analyzed. RxSasData is an S4 class, which extends RxDataSource.

Usage

```
RxSasData(file, stringsAsFactors = FALSE, colClasses = NULL, colInfo = NULL,
          rowsPerRead = 500000, formatFile = NULL, labelsAsLevels = TRUE,
          labelsAsInfo = TRUE, mapMissingCodes = "all",
          varsToKeep = NULL, varsToDrop = NULL,
          checkVarsToKeep = FALSE)

## S3 method for class `RxSasData':
head  (x, n = 6L, reportProgress = 0L, ...)
## S3 method for class `RxSasData':
tail  (x, n = 6L, addrownums = TRUE, reportProgress = 0L, ...)
```

Arguments

`file`

character string specifying a SAS data file of type .sas7bdat (.sd7).

`formatFile`

character string specifying a .sas7cat file containing value labels for the variables stored in `file`.

`stringsAsFactors`

logical indicating whether or not to automatically convert strings to factors on import. This can be overridden by specifying "character" in `colClasses` and `colInfo`. If `TRUE`, the factor levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use `colInfo` with specified "levels".

`colClasses`

character vector specifying the column types to use when converting the data. The element names for the vector are used to identify which column should be converted to which type.

- Allowable column types are:

- "logical" (stored as `uchar`),
- "integer" (stored as `int32`),
- "float32" (the default for floating point data for .xdf files),
- "numeric" (stored as `float64` as in R),
- "character" (stored as `string`),
- "factor" (stored as `uint32`),
- "int16" (alternative to integer for smaller storage space),

- "uint16" (alternative to unsigned integer for smaller storage space)
- "Date" (stored as Date, i.e. float64)
- Note for "factor" type, the levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use colInfo with specified "levels".
- Note that equivalent types share the same bullet in the list above; for some types we allow both 'R-friendly' type names, as well as our own, more specific type names for .xdf data.
- Note also that specifying the column as a "factor" type is currently equivalent to "string" - for the moment, if you wish to import a column as factor data you must use the colInfo argument, documented below.

colInfo

list of named variable information lists. Each variable information list contains one or more of the named elements given below. The information supplied for colInfo overrides that supplied for colClasses .

- Currently available properties for a column information list are:
- type - character string specifying the data type for the column. See colClasses argument description for the available types.
- newName - character string specifying a new name for the variable.
- description - character string specifying a description for the variable.
- levels - character vector containing the levels when "type" = "factor" . If the levels property is not provided, factor levels will be determined by the values in the source column. If levels are provided, any value that does not match a provided level will be converted to a missing value.
- newLevels - new or replacement levels specified for a column of type "factor". It must be used in conjunction with the levels argument. After reading in the original data, the labels for each level will be replaced with the newLevels .
- low - the minimum data value in the variable (used in computations using the F() function).
- high - the maximum data value in the variable (used in computations using the F() function).

rowsPerRead

number of rows to read at a time.

labelsAsLevels

logical. If TRUE , variables containing value labels in the SAS format file will be converted to factors, using the value labels as factor levels.

labelsAsInfo

logical. If TRUE , variables containing value labels in the SAS format file that are not converted to factors will retain the information as valueInfoCodes and valueInfoLabels in the .xdf file. This information can be obtained using rxGetVarInfo. This information will also be returned as attributes for the columns in a dataframe when using rxDataStep.

mapMissingCodes

character string specifying how to handle SAS variables with multiple missing value codes. If "a11" , all of the values set as missing in SAS will be treated as NA . If "none" , the missing value specification in SAS will be ignored and the original values will be imported. If "first" , the values equal to the first missing value code will be imported as NA , while any other missing value codes will be treated as the original values.

varsToKeep

character vector of variable names to include when reading from the input data file. If NULL , argument is

ignored. Cannot be used with `varsToDrop`.

`varsToDrop`

character vector of variable names to exclude when reading from the input data file. If `NULL`, argument is ignored. Cannot be used with `varsToKeep`.

`checkVarsToKeep`

logical value. If `TRUE` variable names specified in `varsToKeep` will be checked against variables in the data set to make sure they exist. An error will be reported if not found. Ignored if more than 500 variables in the data set.

`x`

an `RxSasData` object

`n`

positive integer. Number of rows of the data set to extract.

`addrownums`

logical. If `TRUE`, row numbers will be created to match the original data set.

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`...`

arguments to be passed to underlying functions

Details

The `tail` method is not functional for this data source type and will report an error.

Value

object of class `RxSasData`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxSasData-class](#), [rxNewDataSource](#), [rxImport](#).

Examples

```

#####
# Importing a SAS file
#####
# SAS file name
claimsSasFileName <- file.path(rxGetOption("sampleDataDir"), "claims.sas7bdat")

# Import the SAS data into a data frame
claimsDF <- rxImport(claimsSasFileName)
rxGetInfo(claimsDF, getVarInfo = TRUE, numRows = 5)

# XDF file name
claimsXdfFileName <- file.path(tempdir(), "importedClaims.xdf")

# Import the data into the xdf file
rxImport(claimsSasFileName, claimsXdfFileName, overwrite = TRUE)

# Instead, import SAS file into a data frame
claimsIn <- rxImport(claimsSasFileName)
rxGetInfo(claimsIn, getVarInfo = TRUE, numRows = 5)

# Clean up
file.remove(claimsXdfFileName)

#####
# Using a SAS data source
#####
sasDataFile <- file.path(rxGetOption("sampleDataDir"), "claims.sas7bdat")

# Create a SAS data source with information about variables and
# rows to read in each chunk
sasDS <- RxSasData(sasDataFile, stringsAsFactors = TRUE,
  colClasses = c(RowNum = "integer"),
  rowsPerRead = 50)

# Compute and draw a histogram directly from the SAS file
rxHistogram(~cost|type, data = sasDS)

# Compute summary statistics
rxSummary(~., data = sasDS)

# Estimate a linear model
linModObj <- rxLinMod(cost~age + car_age + type, data = sasDS)
summary(linModObj)

# Import a subset into a data frame for further inspection
subData <- rxImport(inData = sasDS, rowSelection = cost > 400,
  varsToKeep = c("cost", "age", "type"))
subData

```

rxSerializeModel: RevoScaleR Model Serialization and Unserialization

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Serialize a **RevoScaleR/MicrosoftML** model in raw format to enable saving the model. This allows model to be loaded into SQL Server for real-time scoring.

Usage

```
rxSerializeModel(model, metadata = NULL, realtimeScoringOnly = FALSE, ...)  
  
rxUnserializeModel(serializedModel, ...)
```

Arguments

`model`

`RevoScaleR / MicrosoftML` model to be serialized

`metadata`

Arbitrary metadata of `raw` type to be stored with the serialized model. Metadata will be returned when unserialized.

`realtimeScoringOnly`

Drops fields not required for real-time scoring. NOTE: Setting this flag could reduce the model size but `rxUnserializeModel` can no longer retrieve the RevoScaleR model

`serializedModel`

Serialized model to be unserialized

Details

`rxSerializeModel` converts models into `raw` bytes to allow them to be saved and used for real-time scoring.

The following is the list of models that are currently supported in real-time scoring:

* **RevoScaleR**

* `rxLogit`

* `rxLinMod`

* `rxBTrees`

* `rxDTree`

* `rxDForest`

- * MicrosoftML
- * rxFastTrees
- * rxFastForest
- * rxLogisticRegression
- * rxOneClassSvm
- * rxNeuralNet
- * rxFastLinear

RevoScaleR models containing R transformations or transform based formula (e.g "A ~ log(B)") are not supported in real-time scoring. Such transforms to input data may need to be handled before calling real-time scoring.

`rxUnserializeModel` method is used to retrieve the original R model object and metadata from the serialized raw model.

Value

`rxSerializeModel` returns a serialized model.

`rxUnserializeModel` returns original R model object. If metadata is also present returns a list containing the original model object and metadata.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

Examples

```
myIris <- iris
myIris[1:5,]
form <- Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width + Species
irisLinMod <- rxLinMod(form, data = myIris)

# Serialize model for scoring
serializedModel <- rxSerializeModel(irisLinMod)

# Save model to file or SQL Server (use rxWriteObject)
# serialized model can now be used for real-time scoring

unserializedModel <- rxUnserializeModel(serializedModel)
```

rxSetComputeContext: Get and Set the compute context

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Get or set the active compute context for RevoScaleR computations

Usage

```
rxSetComputeContext(computeContext, ...)  
rxGetComputeContext()
```

Arguments

`computeContext`

character string specifying class name or description of the specific class to instantiate, or an existing `RxComputeContext` object. Choices include: `"RxLocalSeq"` or `"local"`, `"RxLocalParallel"` or `"localpar"`, `"RxSpark"` or `"spark"`, `"RxHadoopMR"` or `"hadoopmr"`, and `"RxForeachDoPar"` or `"dopar"`.

`...`

any other arguments are passed to the class generator determined from `computeContext`.

Value

`rxSetComputeContext` returns the previously active compute context invisibly. `rxGetComputeContext` returns the active compute context.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxComputeContext](#), [rxOptions](#), [rxGetOption](#) [rxExec](#).

Examples

```
## Not run:

origComputeContext <- rxSetComputeContext("localpar")
x <- 1:10
rxExec(print, x, elemType = "cores", timesToRun = 10)
rxSetComputeContext( origComputeContext )
## End(Not run)
```

rxSetFileSystem: Set and Get RevoScaleR File System

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Set and get the default file system for RevoScaleR operations.

Usage

```
rxSetFileSystem( fileSystem, ...)  
rxGetFileSystem(x = NULL)
```

Arguments

`fileSystem`

character string specifying class name, file system type, or existing `RxFileSystem` object. Choices include: "RxNativeFileSystem" or "native", or "RxHdfsFileSystem" or "hdfs". Optional arguments `hostName` and `port` may be specified for HDFS file systems.

`...`

other arguments are passed to the underlying class generator.

`x`

optional `RxXdfData` or `RxTextData` object from which to retrieve the file system object.

Value

If `x` is a file system or is a data source object that contains one, that `RxFileSystem` object is returned. Next, if the compute context contains a file system, that `RxFileSystem` object is returned. If no file system object has been found, the previously set `RxFileSystem` file system object is returned. The file system object is returned invisibly.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxFileSystem](#), [RxNativeFileSystem](#), [RxHdfsFileSystem](#), [rxOptions](#), [RxXdfData](#), [RxTextData](#).

Examples

```
# Setup to run analyses to use HDFS file system
## Not run:

# Example 1
myHdfsFileSystem1 <- RxFileSystem(fileSystem = "hdfs")
rxSetFileSystem(fileSystem = myHdfsFileSystem1 )

# Example 2
myHdfsFileSystem2 <- RxHdfsFileSystem( hostName = "default", port = 0)
rxSetFileSystem( fileSystem = myHdfsFileSystem2)

# Example 3
rxSetFileSystem(fileSystem = "hdfs", hostName = "myHost", port = 8020)

## End(Not run)

# Specify a file system in a text data source object
hdfsFS1 <- RxHdfsFileSystem()
textDS <- RxTextData(file = "myfile.txt", fileSystem = hdfsFS1)

# Retrieve the file system information
rxGetFileSystem(textDS)
```

rxSetInfo: Set info such as a description

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Set .xdf file or data frame information, such as a description

Usage

```
rxSetInfo(data, description = "")  
rxSetInfoXdf(file, description = "")
```

Arguments

`data`

An .xdf file name, an [RxXdfData](#) object, or a data frame.

`file`

An .xdf file name or an [RxXdfData](#) object

`description`

character string containing the file or data frame description

Value

An [RxXdfData](#) object representing the .xdf file, or a new data frame with the `.rxDescription` attribute set to the specified `description`.

See Also

[rxGetInfo](#).

Examples

```
# Create a sample data frame and .xdf file
outFile <- tempfile(pattern= ".rxTempFile", fileext = ".xdf")
if (file.exists(outFile)) file.remove(outFile)

origData <- data.frame(x=1:10)
rxDataStep(inData = origData, outFile = outFile, rowsPerRead = 5)

# Set a description in the data file
myDescription <- "Hello Data File!"
rxSetInfo( data = outFile, description = myDescription)
rxGetInfo(outFile)

# Read a data frame out of the output file
myData <- rxDataStep(outFile )
# Look at its attribute
attr(myData, ".rxDescription")

file.remove(outFile)

myNewDescription = "Good-by data."
myData <- rxSetInfo(data=myData, description = myNewDescription)
rxGetInfo(myData)
```

rxSetVarInfo: Set Variable Information for .xdf File or Data Frame

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Set the variable information for an .xdf file, including variable names, descriptions, and value labels, or set attributes for variables in a data frame

Usage

```
rxSetVarInfo(varInfo, data)
rxSetVarInfoXdf(varInfo, file)
```

Arguments

`varInfo`

list containing lists of variable information for variables in the XDF data source or data frame.

`data`

a data frame, a character string specifying the .xdf file, or an `RxXdfData` object.

`file`

character string specifying the .xdf file or an `RxXdfData` object.

Details

The list for each variable contained in `varInfo` can have the following elements:

- * `position` : integer indicating the position of the variable in the data set. If present, will be used instead of the list name.
- * `newName` : character string giving a new name for the variable.
- * `description` : character string giving a description of the variable.
- * `low` : numeric value specifying the low value used in *on the fly* factor conversions using the `F()` transformation. Ignored for data frames.
- * `high` : numeric value specifying the high value used in *on the fly* factor conversions using the `F()` transformation. Ignored for data frames.
- * `levels` : character vector of factor levels. This will re-label an existing factor, but not change the underlying values.
- * `valueInfoCodes` : character vector of value codes for informational purposes only.
- * `valueInfoLabels` : character vector of value labels the same length as `valueInfoCodes`, used for informational purposes only.

* `tzone`: character string giving `tzone` attribute for a POSIXct variable.

Value

If the input data is a data frame, a data frame is returned containing variables with the new attributes. If the input data represents an .xdf file or composite file, an [RxXdfData](#) object representing the modified file is returned.

Note

`rxSetVarInfoXdf` and `rxSetVarInfo` do not change the underlying data so the user cannot set the variable type and storage type using this function, or recode the levels of a factor variable. To recode factors, use [rxFactors](#).
`rxSetVarInfo` is not supported for single .xdf files in HDFS.

See Also

[rxDataStep](#), [rxFactors](#),

Examples

```

# Rename the factor levels for a variable

# Create a sample data set
# We will change the names of the factor levels, after the fact
set.seed(100)
myData1 <- data.frame(y = rnorm(30),
                      month = factor(c(0:11, as.integer(runif(18, 0, 12)))))

# Plot the data
rxLinePlot(y ~ month, type = "p", data = myData1)

# Set the labels for the month variable
monthLabels <- c("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep",
                 "Oct", "Nov", "Dec")

# Get the variable information for the data frame
varInfo <- rxGetVarInfo(myData1)

# Reset the names of the factor levels
varInfo$month$levels <- monthLabels
myData2 <- rxSetVarInfo(varInfo, myData1)

# Plot the new data
rxLinePlot(y ~ month, type = "p", data = myData2)

# Redo, this time putting the data in an .xdf file
# Specify name of output file
tempFile <- file.path(tempdir(), "rxTempTestFile.xdf")

# Convert the original data to an XDF file
rxDataStep(myData1, outFile = tempFile, overwrite = TRUE)

# Get the varInfo of the new XDF file
varInfo <- rxGetVarInfo(tempFile)

varInfo$month$levels <- monthLabels

# Write the modified varInfo back to the file
rxSetVarInfo(varInfo, tempFile)

# Read the data and plot it, showing the new factor levels
myData2 <- rxDataStep(inData = tempFile)
rxLinePlot(y ~ month, type = "p", data = myData2)

## Change variable names and add descriptions
varInfo <- list(y = list(newName = "Rainfall",
                        description = "Rainfall per Month"),
                  list(position = 2, description = "Month of Year"))
rxSetVarInfo(varInfo, tempFile)
rxGetVarInfo(tempFile)
file.remove(tempFile)

```

rxSort: Variable sorting of an .xdf file or data frame.

7/12/2022 • 6 minutes to read • [Edit Online](#)

Description

Efficient multi-key sorting of the variables in an .xdf file or data frame in a local compute context.

Usage

```
rxSort(inData, outFile = NULL, sortByVars, decreasing = FALSE,
       type = "auto", missingsLow = TRUE, caseSensitive = FALSE,
       removeDupKeys = FALSE, varsToKeep = NULL, varsToDrop = NULL,
       dupFreqVar = NULL, overwrite = FALSE, maxRowsByCols = 3000000,
       bufferLimit = -1, reportProgress = rxGetOption("reportProgress"),
       verbose = 0, xdfCompressionLevel = rxGetOption("xdfCompressionLevel"),
       ...)
```

Arguments

inData

a data frame, a character string denoting the path to an existing .xdf file, or an RxXdfData object.

inFile

either an RxXdfData object or a character string denoting the path to an existing .xdf file.

outFile

an .xdf path to store the sorted output. If `NULL`, a data frame with the sorted output will be returned from `rxSort`.

sortByVars

character vector containing the names of the variables to use for sorting. If multiple variables are used, the first `sortByVars` variable is sorted and common values are grouped. The second variable is then sorted within the first variable groups. The third variable is then sorted within groupings formed by the first two variables, and so on.

decreasing

a logical scalar or vector defining the whether or not the `sortByVars` variables are to be sorted in decreasing or increasing order. If a vector, the length `decreasing` must be that of `sortByVars`. If a logical scalar, the value of `decreasing` is replicated to the length `sortByVars`.

type

a character string defining the sorting method to use. Type `"auto"` automatically determines the sort method based on the amount of memory required for sorting. If possible, all of the data will be sorted in memory. Type `"mergeSort"` uses a merge sort method, where chunks of data are pre-sorted, then merged together. Type `"varByVar"` uses a variable-by-variable sort method, which assumes that the `sortByVars` variables and the calculated sorted index variable can be held in memory simultaneously. If `type="varByVar"`, the variables in the sorted data are re-ordered so that the variables named in `sortByVars` come first, followed by any remaining

variables.

missingsLow

a logical scalar for controlling the treatment of missing values. If `TRUE`, missing values in the data are treated as the lowest value; if `FALSE`, they are treated as the highest value.

caseSensitive

a logical scalar. If `TRUE`, case sensitive sorting is performed for character data.

removeDupKeys

logical scalar. If `TRUE`, only the first observation will be kept when duplicate values of the key (`sortByVars`) are encountered. The sort `type` must be set to `"auto"` or `"mergeSort"`.

varsToKeep

character vector defining variables to keep when writing the output to file. If `NULL`, this argument is ignored. This argument takes precedence over `varsToDelete` if both are specified, i.e., both are not `NULL`. If both `varsToKeep` and `varsToDelete` are `NULL` then all variables are written to the data source.

varsToDelete

character vector of variable names to exclude when writing the output data file. If `NULL`, this argument is ignored. If both `varsToKeep` and `varsToDelete` are `NULL` then all variables are written to the data source.

dupFreqVar

character string denoting name of new variable for frequency counts if `removeDupKeys` is set to `TRUE`. Ignored if `removeDupKeys` is set to `FALSE`. If `NULL`, a new variable for frequency counts will not be created.

overwrite

logical value. If `TRUE`, an existing `outFile` will be overwritten.

maxRowsByCols

the maximum size of a data frame that will be returned if `outFile` is set to `NULL` and `inData` is an .xdf file, measured by the number of rows times the number of columns. If the number of rows times the number of columns being created from the .xdf file exceeds this, a warning will be reported and the number of rows in the returned data frame will be truncated. If `maxRowsByCols` is set to be too large, you may experience problems from loading a huge data frame into memory.

bufferLimit

integer specifying the maximum size of the memory buffer (in Mb) to use in sorting when `type` is set to `"auto"`. The default value of `bufferLimit = -1` will attempt to determine an appropriate buffer limit based on system memory.

reportProgress

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

verbose

integer value. If `0`, no additional output is printed. If `1`, additional summary information is printed.

`xdfCompressionLevel`

integer in the range of -1 to 9. The higher the value, the greater the amount of compression for the output file - resulting in smaller files but a longer time to create them. If `xdfCompressionLevel` is set to 0, there will be no compression and the output file will be compatible with the 6.0 release of Revolution R Enterprise. If set to -1, a default level of compression will be used.

`blocksPerRead`

deprecated argument. It will be ignored.

...

additional arguments to be passed directly to the Revolution Compute Engine.

Details

Using `sortbyVars`, multiple keys can be specified to perform an iterative, within-category sorting of the variables in the `inData` or `inFile` .xdf file. The argument `varsToKeep` (or alternatively `varsToDelete`) is used to define the set of sorted variables to store in the specified `outFile` file or the returned data frame if `outData` is `NULL`. The `sortbyVars` variables are automatically prepended to the set of output variables defined by `varsToKeep` or `varsToDelete`.

Value

If an `outFile` is not specified, a data frame with the sorted data is returned. If an `outFile` is specified, an `RxXdfData` data source is returned that can be used in subsequent RevoScaleR analysis. If sorting is unsuccessful, `FALSE` is returned.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

`sort`

Examples

```

###  

# Small data example  

###  

# sort data frame, decreasing by one column and increasing by the other  

sortByVars <- c("Sepal.Length", "Sepal.Width")  

decreasing <- c(TRUE, FALSE)  

sortedIris <- rxSort( inData = iris, sortByVars = sortByVars,  

    decreasing = decreasing)  

# define file names and locations  

inXDF <- file.path(tempdir(), ".rxInFileTemp.xdf")  

outXDF1 <- file.path(tempdir(), ".rxOutFileTemp1.xdf")  

# Create xdf file for iris data  

rxDataStep(inData = iris, outFile = inXDF, overwrite = TRUE)  

rxGetInfo(inXDF)  

# sort the iris data set, first by sepal length in decreasing order  

# and then by sepal width in increasing order  

sortByVars <- c("Sepal.Length", "Sepal.Width")  

decreasing <- c(TRUE, FALSE)  

rxSort(inData = inXDF, outFile = outXDF1, sortByVars = sortByVars,  

    decreasing = decreasing)  

z1 <- rxDataStep(inData = outXDF1)  

print(head(z1,10))  

# clean up  

if (file.exists(inXDF)) file.remove(inXDF)  

if (file.exists(outXDF1)) file.remove(outXDF1)  

###  

# larger data example  

###  

sampleDataDir <- rxGetOption("sampleDataDir")  

CensusPath <- file.path(sampleDataDir, "CensusWorkers.xdf")  

outXDF <- file.path(tempdir(), ".rxCensusSorted.xdf")  

# sort census data by 'age' and then 'incwage' in increasing  

# and decreasing order, respectively. drop the 'perwt' and 'wkswork1'  

# variables from the output.  

rxSort(inData = CensusPath, outFile = outXDF, sortByVars = c("age", "incwage"),  

    decreasing = c(FALSE, TRUE), varsToDrop = c("perwt", "wkswork1"))  

z <- rxDataStep(outXDF)  

print(head(z, 10))  

if (file.exists(outXDF)) file.remove(outXDF)  

###  

# example removing duplicates and creating duplicate counts variable  

###  

sampleDataDir <- rxGetOption("sampleDataDir")  

airDemo <- file.path(sampleDataDir, "AirlineDemoSmall.xdf")  

airDedup <- file.path(tempdir(), ".rxAirDedup.xdf")  

rxSort(inData = airDemo, outFile = airDedup,  

    sortByVars = c("DayOfWeek", "CRSDepTime", "ArrDelay"),  

    removeDupKeys = TRUE, dupFreqVar = "FreqWt")  

# Use the duplicate frequency as frequency weights in a regression  

linModObj <- rxLinMod(ArrDelay~CRSDepTime + DayOfWeek,  

    data = airDedup, fweights = "FreqWt")  

summary(linModObj)  

if (file.exists(airDedup)) file.remove(airDedup)

```

RxSpark-class: Class RxSpark

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Spark compute context class.

Generators

The targeted generator [RxSpark](#) as well as the general generator [RxComputeContext](#).

Extends

Class RxComputeContext, directly.

Methods

`show`

`signature(object = "RxSpark") : ...`

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxSpark](#)

RxSpark: Create Spark compute context, connect and disconnect a Spark application

7/12/2022 • 10 minutes to read • [Edit Online](#)

Description

`RxSpark` creates a Spark compute context. `rxSparkConnect` creates the compute context object with `RxSpark` and then immediately starts the remote Spark application.

`rxSparkDisconnect` shuts down the remote Spark application with `rxStopEngine` and switches to a local compute context. All `rx*` function calls after this will run in a local compute context.

Usage

```
RxSpark(  
  object,  
  hdfsShareDir = paste( "/user/RevoShare", Sys.info()[[ "user" ]], sep="/" ),  
  shareDir = paste( "/var/RevoShare", Sys.info()[[ "user" ]], sep="/" ),  
  clientShareDir = rxGetDefaultTmpDirByOS(),  
  sshUsername = Sys.info()[[ "user" ]],  
  sshHostname = NULL,  
  sshSwitches = "",  
  sshProfileScript = NULL,  
  sshClientDir = "",  
  nameNode = rxGetOption("hdfsHost"),  
  master = "yarn",  
  jobTrackerURL = NULL,  
  port = rxGetOption("hdfsPort"),  
  onClusterNode = NULL,  
  wait = TRUE,  
  numExecutors = rxGetOption("spark.numExecutors"),  
  executorCores = rxGetOption("spark.executorCores"),  
  executorMem = rxGetOption("spark.executorMem"),  
  driverMem = "4g",  
  executorOverheadMem = rxGetOption("spark.executorOverheadMem"),  
  extraSparkConfig = "",  
  persistentRun = FALSE,  
  sparkReduceMethod = "auto",  
  idleTimeout = 3600,  
  suppressWarning = TRUE,  
  consoleOutput = FALSE,  
  showOutputWhileWaiting = TRUE,  
  autoCleanup = TRUE,  
  workingDir = NULL,  
  dataPath = NULL,  
  outDataPath = NULL,  
  fileSystem = NULL,  
  packagesToLoad = NULL,  
  resultsTimeout = 15,  
  tmpFSWorkDir = NULL,  
  ... )  
  
rxSparkConnect(  
  hdfsShareDir      = paste( "/user/RevoShare", Sys.info()[[ "user" ]], sep="/" ),  
  shareDir          = paste( "/var/RevoShare", Sys.info()[[ "user" ]], sep="/" ),  
  clientShareDir    = rxGetDefaultTmpDirByOS(),  
  sshUsername       = Sys.info()[[ "user" ]],  
  sshHostname       = NULL,  
  sshSwitches       = ""
```

```

sshSwitches      = " ",
sshProfileScript = NULL,
sshClientDir    = "",
nameNode         = rxGetOption("hdfsHost"),
master          = "yarn",
jobTrackerURL   = NULL,
port             = rxGetOption("hdfsPort"),
onClusterNode   = NULL,
numExecutors     = rxGetOption("spark.numExecutors"),
executorCores   = rxGetOption("spark.executorCores"),
executorMem     = rxGetOption("spark.executorMem"),
driverMem       = "4g",
executorOverheadMem = rxGetOption("spark.executorOverheadMem"),
extraSparkConfig = "",
sparkReduceMethod = "auto",
idleTimeout      = 10000,
suppressWarning  = TRUE,
consoleOutput    = FALSE,
showOutputWhileWaiting = TRUE,
autoCleanup      = TRUE,
workingDir       = NULL,
dataPath         = NULL,
outDataPath      = NULL,
fileSystem       = NULL,
packagesToLoad   = NULL,
resultsTimeout   = 15,
reset            = FALSE,
interop          = NULL,
tmpFSWorkDir    = NULL,
...  )

```

rxSparkDisconnect(computeContext = rxGetOption("computeContext"))

Arguments

`object`

object of class RxSpark. This argument is optional. If supplied, the values of the other specified arguments are used to replace those of `object` and the modified object is returned.

`hdfsShareDir`

character string specifying the file sharing location within HDFS. You must have permissions to read and write to this location. When you are running in Spark local mode, this parameter will be ignored and it will be forced to be equal to the parameter shareDir.

`shareDir`

character string specifying the directory on the master (perhaps edge) node that is shared among all the nodes of the cluster and any client host. You must have permissions to read and write in this directory.

`clientShareDir`

character string specifying the absolute path of the temporary directory on the client. Defaults to /tmp for POSIX-compliant non-Windows clients. For Windows and non-compliant POSIX clients, defaults to the value of the TEMP environment variable if defined, else to the TMP environment variable if defined, else to `NULL`. If the default directory does not exist, defaults to NULL. UNC paths ("\\host\dir") are not supported.

`sshUsername`

character string specifying the username for making an ssh connection to the Spark cluster. This is not needed if you are running your R client directly on the cluster. Defaults to the username of the user running the R client (that is, the value of `Sys.info()["user"]`).

sshHostname

character string specifying the hostname or IP address of the Spark cluster node or edge node that the client will log into for launching Spark jobs and for copying files between the client machine and the Spark cluster.

Defaults to the hostname of the machine running the R client (that is, the value of `sys.info()["nodename"]`)

This field is only used if `onClusterNode` is `NULL` or `FALSE`. If you are using PuTTY on a Windows system, this can be the name of a saved PuTTY session that can include the user name and authentication file to use.

sshSwitches

character string specifying any switches needed for making an ssh connection to the Spark cluster. This is not needed if one is running one's R client directly on the cluster.

sshProfileScript

Optional character string specifying the absolute path to a profile script on the sshHostname host. This is used when the target ssh host does not automatically read in a .bash_profile, .profile or other shell environment configuration file for the definition of requisite variables.

sshClientDir

character string specifying the Windows directory where Cygwin's ssh.exe and scp.exe or PuTTY's plink.exe and pscp.exe executables can be found. Needed only for Windows. Not needed if these executables are on the Windows Path or if Cygwin's location can be found in the Windows Registry. Defaults to the empty string.

nameNode

character string specifying the Spark name node hostname or IP address. Typically you can leave this at its default value. If set to a value other than "default" or the empty string (see below), this must be an address that can be resolved by the data nodes and used by them to contact the name node. Depending on your cluster, it may need to be set to a private network address such as `"master.local"`. If set to the empty string, "", then the master process will set this to the name of the node on which it is running, as returned by `Sys.info()["nodename"]`. This is likely to work when the sshHostname points to the name node or the sshHostname is not specified and the R client is running on the name node. If you are running in Spark local mode, this parameter defaults to "file:///"; otherwise it defaults to `rxgetOption("hdfsHost")`.

master

Character string specifying the master URL passed to Spark. The value of this parameter could be "yarn", "standalone" and "local". The formats of Spark's master URL (except for mesos) specified in this website <https://spark.apache.org/docs/latest/submitting-applications.html> is also supported.

jobTrackerURL

character scalar specifying the full URL for the jobtracker web interface. This is used only for the purpose of loading the job tracker web page from the `rxLaunchClusterJobManager` convenience function. It is never used for job control, and its specification in the compute context is completely optional. See the [rxLaunchClusterJobManager](#) page for more information.

port

numeric scalar specifying the port used by the name node for HDFS. Needs to be able to be cast to an integer. Defaults to `rxgetOption("hdfsPort")`.

onClusterNode

logical scalar or `NULL` specifying whether the user is initiating the job from a client that will connect to either an edge node or an actual cluster node, directly from either an edge node or node within the cluster. If set to `FALSE` or `NULL`, then `sshHostname` must be a valid host.

wait

logical scalar. If `TRUE` or if `persistentRun` is `TRUE`, the job will be blocking and the invoking function will not return until the job has completed or has failed. Otherwise, the job will be non-blocking and the invoking function will return, allowing you to continue running other R code. The object `rxgLastPendingJob` is created with the job information. You can pass this object to the `rxGetJobStatus` function to check on the processing status of the job. `rxWaitForJob` will change a non-waiting job to a waiting job. Conversely, pressing ESC changes a waiting job to a non-waiting job, provided that the scheduler has accepted the job. If you press ESC before the job has been accepted, the job is canceled.

`numExecutors`

numeric scalar specifying the number of executors in Spark, which is equivalent to parameter `--num-executors` in spark-submit app. If not specified, the default behavior is to launch as many executors as possible, which may use up all resources and prevent other users from sharing the cluster.

`executorCores`

numeric scalar specifying the number of cores per executor, which is equivalent to parameter `--executor-cores` in spark-submit app.

`executorMem`

character string specifying memory per executor (e.g., 1000M, 2G), which is equivalent to parameter `--executor-memory` in spark-submit app.

`driverMem`

character string specifying memory for driver (e.g., 1000M, 2G), which is equivalent to parameter `--driver-memory` in spark-submit app.

`executorOverheadMem`

character string specifying memory overhead per executor (e.g. 1000M, 2G), which is equivalent to setting `spark.yarn.executor.memoryOverhead` in YARN. Increasing this value will allocate more memory for the R process and the ScaleR engine process in the YARN executors, so it may help resolve job failure or executor lost issues.

`extraSparkConfig`

character string specifying extra arbitrary Spark properties (e.g.,

```
"--conf spark.speculation=true --conf spark.yarn.queue=aQueue")
```

, which is equivalent to additional parameters passed into spark-submit app.

`persistentRun`

EXPERIMENTAL. logical scalar. If `TRUE`, the Spark application (and associated processes) will persist across jobs until the `idleTimeout` is reached or the `rxStopEngine` function is called explicitly. This avoids the overhead of launching a new Spark application for each job. If `FALSE`, a new Spark application will be launched when a job starts and will be terminated when the job completes.

`sparkReduceMethod`

Spark job collects all parallel tasks' results to reduce as final result. This parameter decides reduce strategy: `oneStage/twoStage/auto`. `oneStage`: reduce parallel tasks to one result with one reduce function; `twoStage`: reduce parallel tasks to square root size with the first reduce function, then reduce to final result with the second reduce function; `auto`: spark will smartly select `oneStage` or `twoStage`.

`idleTimeout`

numeric scalar specifying the number of seconds of being idle (i.e., not running any Spark job) before system kills the Spark process. Setting a value greater than 600 is recommended. Setting a negative value will not timeout. sparklyr interoperation will have no timeout.

`suppressWarning`

logical scalar. If `TRUE`, suppress warning message regarding missing Spark application parameters.

`consoleOutput`

logical scalar. If `TRUE`, causes the standard output of the R processes to be printed to the user console.

`showOutputWhileWaiting`

logical scalar. If `TRUE`, causes the standard output of the remote primary R and Spark job process to be printed to the user console while waiting for (blocking on) a job.

`autoCleanup`

logical scalar. If `TRUE`, the default behavior is to clean up the temporary computational artifacts and delete the result objects upon retrieval. If `FALSE`, then the computational results are not deleted, and the results may be acquired using `rxGetJobResults`, and the output via `rxGetJobOutput` until the `rxCleanupJobs` is used to delete the results and other artifacts. Leaving this flag set to `FALSE` can result in accumulation of compute artifacts which you may eventually need to delete before they fill up your hard drive.

`workingDir`

character string specifying a working directory for the processes on the master node.

`dataPath`

NOT YET IMPLEMENTED. character vector defining the search path(s) for the data source(s).

`outDataPath`

NOT YET IMPLEMENTED. `NULL` or character vector defining the search path(s) for new output data file(s). If not `NULL`, this overrides any specification for `outDataPath` in `rxOptions`

`fileSystem`

`NULL` or an `RxHdfsFileSystem` to use as the default file system for data sources when created when this compute context is active.

`packagesToLoad`

optional character vector specifying additional packages to be loaded on the nodes when jobs are run in this compute context.

`resultsTimeout`

A numeric value indicating for how long attempts should be made to retrieve results from the cluster. Under normal conditions, results are available immediately. However, under certain high load conditions, the processes on the nodes have reported as completed, but the results have not been fully committed to disk by the operating system. Increase this parameter if results retrieval is failing on high load clusters.

`tmpFSWorkDir`

character string specifying the temporary working directory in worker nodes. It defaults to `/dev/shm` to utilize in-memory temporary file system for performance gain, when the size of `/dev/shm` is less than 1G, the default value would switch to `/tmp` for guarantee sufficiency. You can specify a different location if the size of `/dev/shm` is insufficient. Please make sure that YARN run-as user has permission to read and write to this location

`reset`

if `TRUE` all cached Spark Data Frames will be freed and all existing Spark applications that belong to the current user will be shut down.

`interop`

`NULL` or a character string or vector of package names for RevoScaleR interoperation. Currently, the "sparklyr" package is supported.

`computeContext`

Spark compute context to be terminated by `rxSparkDisconnect`.

`...`

additional arguments to be passed directly to the Microsoft R Services Compute Engine.

Details

This compute context is supported for Cloudera, Hortonworks, and MapR Hadoop distributions.

Value

object of class RxSpark.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxGetJobStatus](#), [rxGetJobOutput](#), [rxGetJobResults](#), [rxCleanupJobs](#), [RxHadoopMR](#), [RxInSqlServer](#), [RxComputeContext](#), [rxSetComputeContext](#), [RxSpark-class](#).

Examples

```

## Not run:

#####
# Running client on edge node
#####

hadoopCC <- RxSpark()

#####
# Running from a Windows client
# (requires Cygwin and/or PuTTY)
#####

mySshUsername <- "user1"
mySshHostname <- "12.345.678.90" #public facing cluster IP address
mySshSwitches <- "-i /home/yourName/user1.pem" #use .ppk file with PuTTY
myShareDir <- paste("/var/RevoShare", mySshUsername, sep ="/")
myHdfsShareDir <- paste("/user/RevoShare", mySshUsername, sep="/")
mySparkCluster <- RxSpark(
  hdfsShareDir = myHdfsShareDir,
  shareDir = myShareDir,
  sshUsername = mySshUsername,
  sshHostname = mySshHostname,
  sshSwitches = mySshSwitches)

#####
## rxSparkConnect example
myHadoopCluster <- rxSparkConnect()

##rxSparkDisconnect example
rxSparkDisconnect(myHadoopCluster)

#####
## sparklyr interoperation example
library("sparklyr")
cc <- rxSparkConnect(interop = "sparklyr")
sc <- rxGetSparklyrConnection(cc)
mtcars_tbl <- copy_to(sc, mtcars)
hive_in_data <- RxHiveData(table = "mtcars")
rxSummary(~., data = hive_in_data)
## End(Not run)

```

rxSparkCacheData {RevoScaleR}: Set the Cache Flag in Spark Compute Context

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Use this function to set the cache flag to control whether data objects should be cached in Spark memory system, applicable for `RxXdfData`, `RxHiveData`, `RxParquetData`, and `RxOrcData`.

Usage

```
rxSparkCacheData(data, cache = TRUE)
```

Arguments

`data`

a data source that can be RxXdfData, RxHiveData, RxParquetData, or RxOrcData. RxTextData is currently not supported.

`cache`

logical value controlling whether the data should be cached or not. If `TRUE` the data will be cached in the Spark memory system after the first use for performance enhancement.

Details

Note that `RxHiveData` with `saveAsTempTable=TRUE` is always cached in memory regardless of the `cache` parameter setting in this function.

Value

data object with new cache flag.

See Also

[rxSparkListData](#), [rxSparkRemoveData](#), [RxXdfData](#), [RxHiveData](#), [RxParquetData](#), [RxOrcData](#).

Examples

```
## Not run:

hiveData <- RxHiveData(table = "mytable")

## this does not work. The cache flag of the input hiveData is not changed, so it is still FALSE.
rxSparkCacheData(hiveData)
rxImport(hiveData) # data is not cached

## this works. The cache flag of hiveData is now set to TRUE.
hiveData <- rxSparkCacheData(hiveData)
rxImport(hiveData) # data is now cached

## set cache value to FALSE
hiveData <- rxSparkCacheData(hiveData, FALSE)
## End(Not run)
```

RxHiveData, RxParquetData, RxOrcData

{RevoScaleR}: Generate Hive, Parquet or ORC Data Source Object

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

These are constructors for Hive, Parquet and ORC data sources which extend `RxDatasource`. These three data sources can be used only in `RxSpark` compute context.

Usage

```
RxHiveData(query = NULL, table = NULL, colInfo = NULL, saveAsTempTable = FALSE, cache = FALSE,  
writeFactorsAsIndexes = FALSE)  
  
RxParquetData(file, colInfo = NULL, fileSystem = "hdfs", cache = FALSE, writeFactorsAsIndexes = FALSE)  
  
RxOrcData(file, colInfo = NULL, fileSystem = "hdfs", cache = FALSE, writeFactorsAsIndexes = FALSE)
```

Arguments

query

character string specifying a Hive query, e.g. `"select * from sample_table"`. Cannot be used with 'table'.

table

character string specifying the name of a Hive table, e.g. `"sample_table"`. Cannot be used with 'query'.

colInfo

list of named variable information lists. Each variable information list contains one or more of the named elements given below (see `rxCreateCollInfo` for more details):

- Currently available properties for a column information list are:

- `type` character string specifying the data type for the column. Supported types are:
 - `"logical"` (stored as `uchar`)
 - `"integer"` (stored as `int32`)
 - `"int16"` (alternative to integer for smaller storage space)
 - `"float32"` (stored as `FloatType`)
 - `"numeric"` (stored as `float64`)
 - `"character"` (stored as `string`)
 - `"factor"` (stored as `uint32`)
 - `"Date"` (stored as `Date`, i.e. `float64`)
 - `"POSIXct"` (stored as `POSIXct`, i.e. `float64`)
- `levels` character vector containing the levels when `type = "factor"`. If the `"levels"` property is not provided, factor levels will be determined by the values in the source column. If levels are

provided, any value that does not match a provided level will be converted to a missing value.

saveAsTempTable

logical. Only applicable when using as output with `table` parameter. If `TRUE` register a temporary Hive table in Spark memory system otherwise generate a persistent Hive table. The temporary Hive table is always cached in Spark memory system.

file

character string specifying a file path, e.g. `"/tmp/AirlineDemoSmall.parquet"` or `"/tmp/AirlineDemoSmall.orc"`.

fileSystem

character string `"hdfs"` or `RxFileSystem` object indicating type of file system. It supports native HDFS and other HDFS compatible systems, e.g., Azure Blob and Azure Data Lake. Local file system is not supported.

cache

[Deprecated] logical. If `TRUE` data will be cached in the Spark application's memory system after the first use.

writeFactorsAsIndexes

logical. If `TRUE`, when writing to an output data source, underlying factor indexes will be written instead of the string representations.

Value

object of RxHiveData, RxParquetData or RxOrcData.

Examples

```
## Not run:

myHadoopCluster <- rxSparkConnect()

colInfo = list(DayOfWeek = list(type = "factor"))

### import from a parquet file
ds1 <- RxParquetData(file = "/tmp/AirlineDemoSmall.parquet",
  colInfo = colInfo)
rxImport(ds1)

### import from an orc file
ds2 <- RxOrcData(file = "/tmp/AirlineDemoSmall.orc",
  colInfo = colInfo)
rxImport(ds2)

### import from a Hive query
ds3 <- RxHiveData(query = "select * from hivesampletable")
rxImport(ds3)

### import from a Hive persistent table
ds4 <- RxHiveData(table = "AirlineDemo")
rxImport(ds4)

### output to a orc file
out1 <- RxOrcData(file = "/tmp/AirlineDemoSmall.orc",
  colInfo = colInfo)
rxDataStep(inData = ds1, outFile=out1, overwrite=TRUE)

### output to a Hive temporary table
out2 <- RxHiveData(table = "AirlineDemoSmall", saveAsTempTable=TRUE)
rxDataStep(inData = ds1, outFile=out2)
## End(Not run)
```

rxSparkListData, rxSparkRemoveData {RevoScaleR}: Manage Cached Data in Spark

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Use these functions to manage the objects cached in the Spark memory system. These functions are only applicable when using [RxSpark](#) compute context.

Usage

```
rxSparkListData(showDescription=TRUE,  
                computeContext = rxGetOption("computeContext"))  
  
rxSparkRemoveData(list,  
                  computeContext = rxGetOption("computeContext"))
```

Arguments

`list`

list of cached objects need to be deleted.

`showDescription`

logical indicating whether or not to print out the detail to console.

`computeContext`

`RxSpark` compute context object.

Value

List of all objects cached in Spark memory system for `rxSparkListData`.

No return values for `rxSparkRemoveData`.

Examples

```
## Not run:

cc <- rxSparkConnect()

colInfo = list( DayOfWeek = list(type = "factor"))
df <- RxParquetData(file = "/tmp/AirlineDemoSmall.parquet", colInfo = colInfo)

### example for rxSparkListData

## No object in list, because no algorithm has been run
rxSparkListData()

rxLogit((ArrDelay>0) ~ CRSDepTime + DayOfWeek, data = df)

## After the first run, a Spark data object is added into the list
rxSparkListData()

### example for rxSparkRemoveData

## remove an object
rxSparkRemoveData(df)

## remove all cached objs
rxSparkRemoveData(rxSparkListData())

rxSparkDisconnect(cc)
## End(Not run)
```

rxSplitXdf: Split a Single Data Set into Multiple Sets

7/12/2022 • 12 minutes to read • [Edit Online](#)

Description

Split an input .xdf file or data frame into multiple .xdf files or a list of data frames.

Usage

```
rxSplit(inData, outFilesBase = NULL, outFileSuffixes = NULL,
        numOut = NULL, splitBy = "rows", splitByFactor = NULL,
        varsToKeep = NULL, varsToDrop = NULL, rowSelection = NULL,
        transforms = NULL, transformObjects = NULL,
        transformFunc = NULL, transformVars = NULL,
        transformPackages = NULL, transformEnvir = NULL,
        overwrite = FALSE, removeMissings = FALSE, rowsPerRead = -1,
        blocksPerRead = 1, updateLowHigh = FALSE, maxRowsByCols = NULL,
        reportProgress = rxGetOption("reportProgress"), verbose = 0,
        xdfCompressionLevel = rxGetOption("xdfCompressionLevel"), ...)

rxSplitXdf(inFile, outFilesBase = basename(rxXdffFileName(inFile)), outFileSuffixes = NULL,
           numOutFiles = NULL, splitBy = "rows", splitByFactor = NULL,
           varsToKeep = NULL, varsToDrop = NULL, rowSelection = NULL,
           transforms = NULL, transformObjects = NULL,
           transformFunc = NULL, transformVars = NULL,
           transformPackages = NULL, transformEnvir = NULL,
           overwrite = FALSE, removeMissings = FALSE,
           rowsPerRead = -1, blocksPerRead = 1, updateLowHigh = FALSE,
           reportProgress = rxGetOption("reportProgress"), verbose = 0,
           xdfCompressionLevel = rxGetOption("xdfCompressionLevel"), ...)
```

Arguments

inData

a data frame, a character string defining the path to the input .xdf file, or an [RxXdfData](#) object

inFile

a character string defining the path to the input .xdf file, or an [RxXdfData](#) object

outFilesBase

a character string or vector defining the file names/paths to use in forming the the output .xdf files. These names/paths will be embellished with any specified `outFileSuffixes`. For `rxSplit`, `outFilesBase = NULL` means that the default value for .xdf and [RxXdfData](#) `inData` objects will be `basename(rxXdffFileName(inData))` while it will be a blank string for `inData` a data frame.

outFileSuffixes

a vector containing the suffixes to append to the base name(s)/path(s) specified in `outFilesBase`. Before appending, `outFileSuffixes` is converted to class character via the `as.character` function. If `NULL`, `seq(numOutFiles)` is used in its place if `numOutFiles` is not `NULL`.

numOut

number of outputs to create, e.g., the number of output files or number of data frames returned in the output list.

`numOutFiles`

number of files to create. This argument is used only when `outFilesBase` and `outFileSuffixes` do not provide sufficient information to form unique paths to multiple output files. See the Examples section for its use.

`splitBy`

a character string denoting the method to use in partitioning the `inFile` .xdf file. With `numFiles` defined as the number of output files (formed by the combination of `outFilesBase`, `outFileSuffixes`, and `numOutFiles` arguments), the supported values for `splitBy` are:

- `"rows"` - To the extent possible, a uniform partition of the number of *rows* in the `inFile` .xdf file is performed. The minimum number of rows in each output file is defined by `floor(numRows/numFiles)`, where `numRows` is the number of rows in `inFile`. Additional rows will be distributed uniformly amongst the last set of files.
- `"blocks"` - To the extent possible, a uniform partition of the number of *blocks* in the `inFile` .xdf file is performed. If `blocksPerRead = 1`, the minimum number of blocks in each output file is defined by `floor(numBlocks/numFiles)`, where `numBlocks` is the number of blocks in `inFile`. Additional blocks will be distributed uniformly amongst the last set of files. For `blocksPerRead > 1`, the number of blocks in each output file is reduced accordingly since multiple blocks read at one time are collapsed to a single block upon write. This argument is ignored if `splitByFactor` is a valid factor variable name.

`splitByFactor`

character string identifying the name of the factor to use in splitting the `inFile` .xdf data such that each file contains the data corresponding to a single level of the factor. The `splitBy` argument is ignored if `splitByFactor` is a valid factor variable name. If `splitByFactor = NULL`, the `splitBy` argument is used to define the split method.

`varsToKeep`

character vector of variable names to include when reading from the input data file. If `NULL`, argument is ignored. Cannot be used with `varsToDrop`.

`varsToDrop`

character vector of variable names to exclude when reading from the input data file. If `NULL`, argument is ignored. Cannot be used with `varsToKeep`.

`rowSelection`

name of a logical variable in the data set (in quotes) or a logical expression using variables in the data set to specify row selection. For example, `rowSelection = "old"` will use only observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` will use only observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

`transforms`

an expression of the form `list(name = expression, ...)` representing the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

`transformObjects`

a named list containing objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`. Ignored for data frames.

`transformFunc`

variable transformation function. See [rxTransform](#) for details.

`transformVars`

character vector of input data set variables needed for the transformation function. See [rxTransform](#) for details.

`transformPackages`

character vector defining additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, e.g., those explicitly defined in RevoScaleR functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.

`transformEnvir`

user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

`overwrite`

logical value. If `TRUE`, an existing `outFile` will be overwritten, or if appending columns existing columns with the same name will be overwritten. `overwrite` is ignored if appending rows. Ignored for data frames.

`removeMissings`

logical value. If `TRUE`, rows with missing values will not be included in the output data.

`rowsPerRead`

number of rows to read for each chunk of data read from the input data source. Use this argument for finer control of the number of rows per block in the output data source. If greater than 0, `blocksPerRead` is ignored. Cannot be used if `inFile` is the same as `outFile`.

`blocksPerRead`

number of blocks to read for each chunk of data read from the data source. Ignored for data frames or if `rowsPerRead` is positive.

`updateLowHigh`

logical value. If `FALSE`, the low and high values for each variable in the `inFile` .xdf file will be copied to the header of each output file so that all output files reflect the global range of the data. If `TRUE`, each variable's low and high values are updated to reflect the range of values in the corresponding file, likely resulting in different low and high values for the same variable between output files. Note that when using `rxSplit` and the output is a list of data frames, the `updateLowHigh` argument has no effect.

`maxRowsByCols`

argument sent directly to the `rxDataStep` function behind the scenes when converting the output .xdf files (back) into a list of data frames. This parameter is provided primarily for the case where `rxSplit` is being used to split a .xdf file or `RxxdfData` data source into portions that are then read back into R as a list of data frames. In this case, `maxRowsByCols` provides a mechanism for the user to control the maximum number of elements read from the output .xdf file in an effort to limit the amount of memory needed for storing each partition as a data frame object in R.

reportProgress

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

verbose

integer value. If `0`, no additional output is printed. If `1`, additional summary information is printed.

xdfCompressionLevel

integer in the range of -1 to 9. The higher the value, the greater the amount of compression - resulting in smaller files but a longer time to create them. If `xdfCompressionLevel` is set to 0, there will be no compression and files will be compatible with the 6.0 release of Revolution R Enterprise. If set to -1, a default level of compression will be used.

...

additional arguments to be passed directly to the function used to partition the data. For example,

- `Uniform Partition` - a `fill` argument with supported values are `"left"`, `"center"`, or `"right"`, can be used to place the remaining rows/blocks in the set of output files. For example, if `sortBy = "blocks"`, `inFile` has `15` blocks, and the number of output files is `6`, the distribution of blocks for the set of `6` output files will be:
 - `fill = "left"` - 3 3 3 2 2 2
 - `fill = "center"` - 2 3 3 3 2 2
 - `fill = "right"` - 2 2 2 3 3 3

Details

rxSplit: Use `rxSplit` as a general function to partition a data frame or .xdf file. Behind the scenes, the `rxSplitXdf` function is called by `rxsplit` after converting the `inData` data source into a (temporary) .xdf file. If both `outFilesBase` and `outFileSuffixes` are `NULL` (the default), then the type of output is determined by the type of `inData`: if the `inData` is an .xdf file name or an `RxXdfData` object, and list of `RxXdfData` data sources representing the new split .xdf files is returned. If the `inData` is a data frame, then the output is returned as a list of data frames. If `outFilesBase` is an empty character string and `outFileSuffixes` is `NULL`, a list of data frames is always returned. In the case that a list of data frames is returned, the names of the list are in `shortFileName.splitType.NumberOrFactor` format, e.g., `iris.Species.versicolor` or `myXdfFile.rows.3`.

rxSplitXdf: Use `rxSplitXdf` to partition an .xdf file. The `inFile` .xdf file is uniformly partitioned (to the extent possible) and each partition is written to a distinct user-specified file. As an example, if `inFile` contains a million rows, `splitBy = "rows"`, and the number of output files we specify to create is five, then each output file will contain two hundred thousand observations, assuming none were dropped via a `rowSelection` argument specification. The number of output files is controlled either directly or indirectly through a combination of arguments: `inFile`, `outFilesBase`, `outFileSuffixes`, and `numOutFiles`.

In general, the values of these arguments are pasted together to form a character vector of output file paths, with scalar values auto-expanded to vectors (of an appropriate size) prior to pasting. This scheme allows the user to either specify directly the full paths to the output files they wish to form or do so implicitly (and perhaps more conveniently) using various combinations of these arguments. We illustrate the use of these combinations in the examples below, which are assumed to be run under Windows.

COL 1	COL 2	COL 3
INPUT ARGUMENTS		OUTPUT FILE VECTOR
<code>inFile = "foo.xdf", numOutFiles = 3</code>		<code>"foo1.xdf", "foo2.xdf", "foo3.xdf"</code>
<code>outFilesBase = "out", numOutFiles = 4</code>		<code>"out1.xdf", "out2.xdf", "out3.xdf", "out4.xdf"</code>
<code>outFilesBase = "golf", outFileSuffixes = c("club", "tee", "score")</code>		<code>"golfclub.xdf", "golftee.xdf", "golfscore.xdf"</code>
<code>outFilesBase = "C:\\\\myDir\\\\", outFileSuffixes = c("a\\\\same", "b\\\\same", "c\\\\same")</code>		<code>"C:\\\\myDir\\\\a\\\\same.xdf", "C:\\\\myDir\\\\b\\\\same.xdf", "C:\\\\myDir\\\\c\\\\same.xdf"</code>
<code>outFilesBase = c("C:\\\\here\\\\file1.xdf", "D:\\\\there\\\\file2.xdf", "E:\\\\everywhere\\\\file3.xdf")</code>		<code>"C:\\\\here\\\\file1.xdf", "D:\\\\there\\\\file2.xdf", "E:\\\\everywhere\\\\file3.xdf"</code>

Value

a list of data frames or an invisible list of `RxXdfData` data source objects corresponding to the created output files.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxDataStep](#), [rxImport](#), [rxTransform](#)

Examples

```
#####
# rxSplit Examples

# DF -> DF : Data frame input to list of data frames
IrisDFList <- rxSplit(iris, numOut = 3, reportProgress = 0, verbose = 0)
names(IrisDFList)
head(IrisDFList[[3]])

IrisDFList <- rxSplit(iris, splitByFactor = "Species", reportProgress = 0, verbose = 0)
names(IrisDFList)
head(IrisDFList[[1]])

# DF -> XDF : Data frame input to .xdf outputs (list of RxXdfData data sources)
irisXDFs <- rxSplit(iris, splitByFactor = "Species", outFilesBase = file.path(tempdir(),"iris"),
                      reportProgress = 0, verbose = 0, overwrite = TRUE)
print(irisXDFs)
invisible(sapply(irisXDFs, function(x) if (file.exists(x@file)) unlink(x@file)))

# XDF -> DF : .xdf file to a list of data frames
# Return a list of data frames instead of creating .xdf files by specifying
# outFilesBase = ""
```

```

XDF <- file.path(tempdir(), "iris.xdf")
rxDataStep(iris, XDF, overwrite = TRUE)
IrisDFList <- rxSplit(XDF, splitByFactor = "Species", outFilesBase = "",
  reportProgress = 0, verbose = 0, overwrite = TRUE)
names(IrisDFList)
head(IrisDFList[[1]])
if (file.exists(XDF)) file.remove(XDF)

# Split the fourth graders data by gender and use row selection
# to collect information only on blue eyed children
fourthGradersXDF <- file.path(rxGetOption("sampleDataDir"), "fourthgraders.xdf")
rxSplit(fourthGradersXDF, splitByFactor = "male", outFilesBase = "",
  rowSelection = (eyecolor == "Blue"))

# XDF -> XDF : .xdf file into multiple .xdf files
XDF <- tempfile(pattern = "iris", fileext = ".xdf")
outXDF1 <- tempfile(pattern = "irisOut", fileext = ".xdf")
outXDF2 <- tempfile(pattern = "irisOut", fileext = ".xdf")
rxDataStep(iris, XDF)
outFiles <- rxSplit(XDF, outFilesBase = tempdir(), outFileSuffixes = basename(c(outXDF1, outXDF2)),
  reportProgress = 0, verbose = 0, overwrite = TRUE)
print(outFiles)

# Remove created files
invisible(sapply(outFiles, function(x) if (file.exists(x@file)) unlink(x@file)))
if (file.exists(XDF)) file.remove(XDF)
if (file.exists(outXDF1)) file.remove(outXDF1)
if (file.exists(outXDF2)) file.remove(outXDF2)

#####
# rxSplitXdf Examples

# Split CensusWorkers.xdf data into five files
# with a (nearly) uniform distribution of rows across the files.
# Put the files in a temporary directory and use "Census" as the
# basename for the files. Append each file with "-byRows" and
# an index indicating the file number in the set.
inFile <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers")
outFilesBase <- file.path(tempdir(), "byRowsDir", "Census-byRows")
rxSplitXdf(inFile, outFilesBase = outFilesBase,
  numOutFiles = 5, splitBy = "rows", verbose = 1)

# Obtain information from each of the resulting output files
# and remove the files along with the parent directory.
byRowFiles <- list.files(dirname(outFilesBase), full = TRUE)
infoByRows <- sapply(byRowFiles, rxGetInfo, simplify = FALSE)
unlink(dirname(outFilesBase), recursive = TRUE)

# Perform a similar split by blocks
outFilesBase <- file.path(tempdir(), "byBlocksDir", "Census-byBlocks")
rxSplitXdf(inFile, outFilesBase = outFilesBase,
  numOutFiles = 5, splitBy = "blocks", verbose = 1)

# Obtain information from each of the resulting output files
# and remove the files along with the parent directory.
byBlockFiles <- list.files(dirname(outFilesBase), full = TRUE)
infoByBlocks <- sapply(byBlockFiles, rxGetInfo, simplify = FALSE)
unlink(dirname(outFilesBase), recursive = TRUE)

# Create a barplot comparing the two methods on the resulting
# number of rows in each output file. The barplot of the
# "rows" split case is uniform while the "blocks" split is
# highly non-uniform.
rows <- sapply(infoByRows, "[[", "numRows")
blocks <- sapply(infoByBlocks, "[[", "numRows")
numRowsData <- cbind(rows, blocks)
barplot(numRowsData, beside = TRUE,
  main = "CensusWorkers Partition: Row Distribution",
  xlab = "sortBy Method",

```

```
ylab = "Number of Rows",
legend.text = basename(rownames(numRowsData)),
args.legend = list(x = "topright"),
ylim = c(0, 1.5 * max(numRowsData))

# Create a similar plot comparing the number of resulting blocks
# in the output .xdf files.
rows <- sapply(infoByRows, "[[", "numBlocks")
blocks <- sapply(infoByBlocks, "[[", "numBlocks")
numBlocksData <- cbind(rows, blocks)
barplot(numBlocksData, beside = TRUE,
        main = "CensusWorkers Partition: Block Distribution",
        xlab = "sortBy Method",
        ylab = "Number of Blocks",
        legend.text = basename(rownames(numBlocksData)),
        args.legend = list(x = "topright"),
        ylim = c(0, 1.5 * max(numBlocksData)))
```

RxSpssData-class: Class RxSpssData

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

SPSS data source connection class.

Generators

The targeted generator [RxSpssData](#) as well as the general generator [rxNewDataSource](#).

Extends

Class RxFileData, directly. Class RxDataSource, by class RxFileData.

Methods

[show](#)

`signature(object = "RxSpssData") : ...`

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxDataSource-class](#), [RxSpssData](#), [rxNewDataSource](#)

RxSpssData: Generate SPSS Data Source Object

7/12/2022 • 4 minutes to read • [Edit Online](#)

Description

Generate an RxSpssData object that contains information about an SPSS data set to be imported or analyzed. RxSpssData is an S4 class, which extends RxDataSource.

Usage

```
RxSpssData(file, stringsAsFactors = FALSE, colClasses = NULL, colInfo = NULL,
           rowsPerRead = 500000, labelsAsLevels = TRUE, labelsAsInfo = TRUE,
           mapMissingCodes = "all", varsToKeep = NULL, varsToDrop = NULL,
           checkVarsToKeep = FALSE)

## S3 method for class `RxSpssData':
head  (x, n = 6L, reportProgress = 0L, ...)
## S3 method for class `RxSpssData':
tail  (x, n = 6L, addrownums = TRUE, reportProgress = 0L, ...)
```

Arguments

`file`

character string specifying an SPSS data file of type .sav.

`stringsAsFactors`

logical indicating whether or not to automatically convert strings to factors on import. This can be overridden by specifying `"character"` in `colClasses` and `colInfo`. If `TRUE`, the factor levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use `colInfo` with specified `"levels"`.

`colClasses`

character vector specifying the column types to use when converting the data. The element names for the vector are used to identify which column should be converted to which type.

- Allowable column types are:

- `"logical"` (stored as `uchar`),
- `"integer"` (stored as `int32`),
- `"float32"` (the default for floating point data for .xdf files),
- `"numeric"` (stored as `float64` as in R),
- `"character"` (stored as `string`),
- `"factor"` (stored as `uint32`),
- `"int16"` (alternative to integer for smaller storage space),
- `"uint16"` (alternative to unsigned integer for smaller storage space)
- `"Date"` (stored as Date, i.e. `float64`)

- Note for "factor" type, the levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use `colInfo` with specified "levels".
- Note that equivalent types share the same bullet in the list above; for some types we allow both 'R-friendly' type names, as well as our own, more specific type names for .xdf data.
- Note also that specifying the column as a "factor" type is currently equivalent to "string" - for the moment, if you wish to import a column as factor data you must use the `colInfo` argument, documented below.

`colInfo`

list of named variable information lists. Each variable information list contains one or more of the named elements given below. The information supplied for `colInfo` overrides that supplied for `colClasses`.

- Currently available properties for a column information list are:
- `type` - character string specifying the data type for the column. See `colClasses` argument description for the available types.
- `newName` - character string specifying a new name for the variable.
- `description` - character string specifying a description for the variable.
- `levels` - character vector containing the levels when `type = "factor"`. If the levels property is not provided, factor levels will be determined by the values in the source column. If levels are provided, any value that does not match a provided level will be converted to a missing value.
- `newLevels` - new or replacement levels specified for a column of type "factor". It must be used in conjunction with the `levels` argument. After reading in the original data, the labels for each level will be replaced with the `newLevels`.
- `low` - the minimum data value in the variable (used in computations using the `F()` function).
- `high` - the maximum data value in the variable (used in computations using the `F()` function).

`rowsPerRead`

number of rows to read at a time. This will determine the size of a block in the .xdf file if using `rxImport`.

`labelsAsLevels`

logical. If `TRUE`, variables containing value labels in the SPSS file will be converted to factors, using the value labels as factor levels.

`labelsAsInfo`

logical. If `TRUE`, variables containing value labels in the SPSS file that are not converted to factors will retain the information as `valueInfoCodes` and `valueInfoLabels` in the .xdf file. This information can be obtained using `rxGetVarInfo`. This information will also be returned as attributes for the columns in a dataframe when using `rxDataStep`.

`mapMissingCodes`

character string specifying how to handle SPSS variables with multiple missing value codes. If `"all"`, all of the values set as missing in SPSS will be treated as `NA`. If `"none"`, the missing value specification in SAS will be ignored and the original values will be imported. If `"first"`, the values equal to the first missing value code will be imported as `NA`, while any other missing value codes will be treated as the original values.

`varsToKeep`

character vector of variable names to include when reading from the input data file. If `NULL`, argument is ignored. Cannot be used with `varsToDelete`.

`varsToDrop`

character vector of variable names to exclude when reading from the input data file. If `NULL`, argument is ignored. Cannot be used with `varsToKeep`.

`checkVarsToKeep`

logical value. If `TRUE` variable names specified in `varsToKeep` will be checked against variables in the data set to make sure they exist. An error will be reported if not found. Ignored if more than 500 variables in the data set.

`x`

an `RxSpssData` object

`n`

positive integer. Number of rows of the data set to extract.

`addrownums`

logical. If `TRUE`, row numbers will be created to match the original data set.

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`...`

arguments to be passed to underlying functions

Details

The `tail` method is not functional for this data source type and will report an error.

Value

object of class `RxSpssData`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxSpssData-class](#), [rxNewDataSource](#), [rxiImport](#).

Examples

```
# Create a SPSS data source
claimsSpssFileName <- file.path(rxGetOption("sampleDataDir"), "claims.sav")
claimsSpssSource <- RxSpssData(claimsSpssFileName)

# Specify an xdf data source
claimsXdffFileName <- file.path(tempdir(), "importedClaims.xdf")

# Import the data into the xdf file
myXdfDataSource <- rxImport(claimsSpssSource, claimsXdffFileName, overwrite = TRUE)

# Instead, import the (small) data set into a data frame
claimsIn <- rxImport(claimsSpssSource)
head(claimsIn)

# Clean up
file.remove(claimsXdffFileName)
```

rxSqlLibPaths: Search Paths for Packages in SQL compute context

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Gets the search path for the library trees for packages while executing inside the SQL Server, using [RxInSqlServer](#) compute context or using T-SQL script with `sp_execute_external_script` stored procedure with embedded R script.

Usage

```
rxSqlLibPaths(connectionString)
```

Arguments

`connectionString`

a `character` connection string for the SQL Server. This should be a local connection string (external connection strings are not supported while executing on a SQL Server). You can also specify [RxInSqlServer](#) compute context object for input, from which the connection string will be extracted and used.

Details

For [RxInSqlServer](#) compute context, a user specified on the connection string must be a member of one of the following roles `'db_owner'`, `'rpkgs-shared'`, `'rpkgs-private'` or `'rpkgs-private'` in the database. When `rxExec()` function is called from a client machine with [RxInSqlServer](#) compute context to execute the `rx` function on SQL Server, the `.libPaths()` is automatically updated to include the library paths returned by this `rxSqlLibPaths` function.

Value

A character vector of the library paths containing both `"shared"` or `"private"` scope of the packages if the user specified in the connection string is allowed access. On access denied, it returns an empty character vector.

See Also

[rxPackage](#), `.libPaths`, [rxFindPackage](#), [rxInstalledPackages](#), [rxInstallPackages](#),
[rxRemovePackages](#), [rxSyncPackages](#), `require`

Examples

```
## Not run:

#
# An example sp_execute_external_script T-SQL code using rxSqlLibPaths()
#
declare @instance_name nvarchar(100) = @@SERVERNAME, @database_name nvarchar(128) = db_name();
exec sp_execute_external_script
@language = N'R',
@script = N'
connStr <- paste("Driver=SQL Server;Server=", instance_name, ";Database=", database_name,
";Trusted_Connection=true;", sep="");
.libPaths(rxSqlLibPaths(connStr));
print(.libPaths());
',
@input_data_1 = N'',
@params = N'@instance_name nvarchar(100), @database_name nvarchar(128)',
@instance_name = @instance_name,
@database_name = @database_name;

## End(Not run)
```

RxSqlServerData-class: Class RxSqlServerData

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Microsoft SQL Server data source connection class.

Generators

The targeted generator [RxSqlServerData](#) as well as the general generator [rxNewDataSource](#).

Extends

Class RxDataSource, directly.

Methods

[show](#)

```
signature(object = "RxSqlServerData") : ...
```

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxDataSource-class](#), [RxSqlServerData](#), [rxNewDataSource](#)

RxSqlServerData: GenerateSqlServer Data Source Object

7/12/2022 • 5 minutes to read • [Edit Online](#)

Description

This is the main generator for S4 class RxSqlServerData, which extends RxDataSource.

Usage

```
RxSqlServerData(table = NULL, sqlQuery = NULL, connectionString = NULL,
  rowBuffering = TRUE, returnDataFrame = TRUE, stringsAsFactors = FALSE,
  colClasses = NULL, colInfo = NULL, rowsPerRead = 50000, verbose = 0,
  useFastRead = TRUE, server = NULL, databaseName = NULL,
  user = NULL, password = NULL, writeFactorsAsIndexes = FALSE)

## S3 method for class `RxSqlServerData':
head  (x, n = 6L, reportProgress = 0L,    ...  )
## S3 method for class `RxSqlServerData':
tail  (x, n = 6L, addrownums = TRUE, reportProgress = 0L,    ...  )
```

Arguments

`table`

`NULL` or character string specifying the table name. Cannot be used with `sqlQuery`.

`sqlQuery`

`NULL` or character string specifying a valid SQL select query. Cannot contain hidden characters such as tabs or newlines. Cannot be used with `table`. If you want to use `TABLESAMPLE` clause in `sqlQuery`, set `rowBuffering` argument to `FALSE`.

`connectionString`

`NULL` or character string specifying the connection string.

`rowBuffering`

logical specifying whether or not to buffer rows on read from the database. If you are having problems with your ODBC driver, try setting this to `FALSE`.

`returnDataFrame`

logical indicating whether or not to convert the result from a list to a data frame (for use in `rxReadNext` only). If `FALSE`, a list is returned.

`stringsAsFactors`

logical indicating whether or not to automatically convert strings to factors on import. This can be overridden by specifying "character" in `colClasses` and `colInfo`. If `TRUE`, the factor levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use `colInfo` with specified "levels".

colClasses

character vector specifying the column types to use when converting the data. The element names for the vector are used to identify which column should be converted to which type.

- Allowable column types are:

- "logical" (stored as `uchar`),
- "integer" (stored as `int32`),
- "float32" (the default for floating point data for .xdf files),
- "numeric" (stored as `float64` as in R),
- "character" (stored as `string`),
- "factor" (stored as `uint32`),
- "int16" (alternative to integer for smaller storage space),
- "uint16" (alternative to unsigned integer for smaller storage space),
- "Date" (stored as Date, i.e. `float64`)

- Note for "factor" type, the levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use `colInfo` with specified "levels".

- Note that equivalent types share the same bullet in the list above; for some types we allow both 'R-friendly' type names, as well as our own, more specific type names for .xdf data.
- Note also that specifying the column as a "factor" type is currently equivalent to "string" - for the moment, if you wish to import a column as factor data you must use the `colInfo` argument, documented below.

colInfo

list of named variable information lists. Each variable information list contains one or more of the named elements given below. The information supplied for `colInfo` overrides that supplied for `colClasses`.

- Currently available properties for a column information list are:
- `type` - character string specifying the data type for the column. See `colClasses` argument description for the available types. Specify "factorIndex" as the `type` for 0-based factor indexes. `levels` must also be specified.
- `newName` - character string specifying a new name for the variable.
- `description` - character string specifying a description for the variable.
- `levels` - character vector containing the levels when `type = "factor"`. If the levels property is not provided, factor levels will be determined by the values in the source column. If levels are provided, any value that does not match a provided level will be converted to a missing value.
- `newLevels` - new or replacement levels specified for a column of type "factor". It must be used in conjunction with the `levels` argument. After reading in the original data, the labels for each level will be replaced with the `newLevels`.
- `low` - the minimum data value in the variable (used in computations using the `F()` function).
- `high` - the maximum data value in the variable (used in computations using the `F()` function).

rowsPerRead

number of rows to read at a time.

verbose

integer value. If `0`, no additional output is printed. If `1`, information on the odbc data source type (`odbc` or `odbcFast`) is printed.

...

additional arguments to be passed directly to the underlying functions.

x

an `RxSqlServerData` object

n

positive integer. Number of rows of the data set to extract.

`addrownums`

logical. If `TRUE`, row numbers will be created to match the original data set.

`reportProgress`

integer value with options:

- `0` : no progress is reported.
- `1` : the number of processed rows is printed and updated.
- `2` : rows processed and timings are reported.
- `3` : rows processed and all timings are reported.

`useFastRead`

logical specifying whether or not to use a direct ODBC connection. On Linux systems, this is the only ODBC connection available. Note: `useFastRead = FALSE` is currently not supported in writing to SQL Server data source.

`server`

Target SQL Server instance. Can also be specified in the connection string with the `Server` keyword.

`databaseName`

Database on the target SQL Server instance. Can also be specified in the connection string with the `Database` keyword.

`user`

SQL login to connect to the SQL Server instance. SQL login can also be specified in the connection string with the `uid` keyword.

`password`

Password associated with the SQL login. Can also be specified in the connection string with the `pwd` keyword.

`writeFactorsAsIndexes`

logical. If `TRUE`, when writing to an `RxOdbcData` data source, underlying factor indexes will be written instead of the string representations.

Details

The `tail` method is not functional for this data source type and will report an error.

The `user` and `password` arguments take precedence over equivalent information provided within the `connectionString` argument. If, for example, you provide the user name in both the `connectionString` and `user` arguments, the one in `connectionString` is ignored.

Value

object of class RxSqlServerData.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxSqlServerData-class](#), [rxNewDataSource](#), [rxImport](#).

Examples

```
## Not run:

# Create an RxSqlServerData data source

# Note: for improved security, read connection string from a file, such as
# sqlServerConnString <- readLines("sqlServerConnString.txt")

sqlServerConnString <- "SERVER=hostname;DATABASE=RevoTestDB;UID=DBUser;PWD=Password;"

sqlServerDataDS <- RxSqlServerData(sqlQuery = "SELECT * FROM claims",
                                    connectionString = sqlServerConnString)

# Create an xdf file name
claimsXdfFileName <- file.path(tempdir(), "importedClaims.xdf")

# Import the data into the xdf file
rxImport(sqlServerDataDS, claimsXdfFileName, overwrite = TRUE)

# Read xdf file into a data frame
claimsIn <- rxDataStep(claimsXdfFileName)
head(claimsIn)
## End(Not run)
```

rxSqlServerDropTable: rxSqlServerDropTable

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Execute a SQL statement that drops a table or checks for existence.

Usage

```
rxSqlServerTableExists(table, connectionString = NULL)
rxSqlServerDropTable(table, connectionString = NULL)
```

Arguments

`table`

character string specifying a table name or an `RxSqlServerData` data source that has the `table` specified.

`connectionString`

`NULL` or character string specifying the connection string. If `NULL`, the connection string from the currently active compute context will be used if available.

`...`

Additional arguments to be passed through.

Details

An SQL query is passed to the ODBC driver.

Value

`rxSqlServerTableExists` returns `TRUE` if the table exists, `FALSE` otherwise. `rxSqlServerDropTable` returns `TRUE` if the table is successfully dropped, `FALSE` otherwise (for example, if the table did not exist).

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxInSqlServer](#), [RxSqlServerData](#).

Examples

```
## Not run:

# With an RxInSqlServer active compute context
tempTable <- "rxTempTest"
if (rxSqlServerTableExists(tempTable)) rxSqlServerDropTable(tempTable)
## End(Not run)
```

rxStepControl: Control for Stepwise Regression

7/12/2022 • 4 minutes to read • [Edit Online](#)

Description

Various parameters that control aspects of stepwise regression.

Usage

```
rxStepControl(method = "stepwise", scope = NULL,  
             maxSteps = 1000, stepCriterion = "AIC",  
             maxSigLevelToAdd = NULL, minSigLevelToDrop = NULL,  
             refitEachStep = NULL, keepStepCoefs = FALSE,  
             scale = 0, k = 2, test = NULL, ... )
```

Arguments

method

a character string specifying the method of stepwise search:

- "stepwise": bi-directional search.
- "backward": backward elimination.
- "forward": forward selection.

Default is "stepwise" if the scope argument is not missing, otherwise "backward".

scope

either a single formula, or a named list containing components upper and lower, both formulae, defining the range of models to be examined in the stepwise search.

maxSteps

an integer specifying the maximum number of steps to be considered, typically used to stop the process early and the default is 1000.

stepCriterion

a character string specifying the variable selection criterion:

- "AIC": Akaike's information criterion.
- "SigLevel": significance level, the traditional stepwise approach in SAS. This argument is similar to the SELECT option of the GLMSELECT procedure in SAS. Default is "AIC".

maxSigLevelToAdd

a numeric scalar specifying the significance level for adding a variable to the model. This argument is used only when stepCriterion = "SigLevel" and is similar to the SLENTRY option of the GLMSELECT procedure in SAS. The defaults are 0.50 for "forward" and 0.15 for "stepwise".

minSigLevelToDrop

a numeric scalar specifying the significance level for dropping a variable from the model. This argument is used only when stepCriterion = "SigLevel" and is similar to the SLSTAY option of the GLMSELECT procedure in SAS.

The defaults are 0.10 for "backward" and 0.15 for "stepwise".

refitEachStep

a logical flag specifying whether or not to refit the model at each step. The default is `NULL`, indicating to refit the model at each step for `rxLogit` and `rxGlm` but not for `rxLinMod`.

keepStepCoefs

a logical flag specifying whether or not to keep the model coefficients at each step. If `TRUE`, a data.frame `stepCoefs` will be returned with the fitted model with rows corresponding to the coefficients and columns corresponding to the iterations. Additional computation may be required to generate the coefficients at each step. Those stepwise coefficients can be visualized by plotting the fitted model with `rxStepPlot`.

scale

optional numeric scalar specifying the scale parameter of the model. It is used in computing the AIC statistics for selecting the models. The default 0 indicates it should be estimated by maximum likelihood. See "scale" in step for details.

k

optional numeric scalar specifying the weight of the number of equivalent degrees of freedom in computing AIC for the penalty. See "k" in step for details.

test

a character string specifying the test statistic to be included in the results, either "F" or "Chisq". Both test statistics are relative to the original model.

...

additional arguments to be passed directly to the Microsoft R Services Compute Engine.

Details

Stepwise models must be computed on the same dataset in order to be compared so rows with missing values in any of the variables in the upper model are removed before the model fitting starts. Consequently, the stepwise models might be different from the corresponding models fitted with only the selected variables if there are missing values in the data set.

When computing stepwise models with `rxLogit` or `rxGlm`, you can sometimes improve the speed and quality of the fitting by setting `returnAlways=TRUE` in the initial `rxLogit` or `rxGlm` call. When `returnAlways=TRUE`, `rxLogit` and `rxGlm` always return the solution tried so far that has the minimum deviance.

Value

A list containing the options.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S*. New York: Springer (4th ed).

Chambers, J. M. and Hastie, T. J. eds (1992) *Statistical Models in S*. Wadsworth & Brooks/Cole.

Goodnight, J. H. (1979) A Tutorial on the SWEEP Operator. *The American Statistician* Vol. 33 No. 3, 149--158.

See Also

[step](#), [rxStepPlot](#).

Examples

```
## setup
form <- Sepal.Length ~ Sepal.Width + Petal.Length
scope <- list(
  lower = ~ Sepal.Width,
  upper = ~ Sepal.Width + Petal.Length + Petal.Width * Species)

## lm/step
## We need to specify the contrasts for the factor variable Species,
## even though this is not part of the original model. This will
## generate a warning, so we suppress that warning here.
suppressWarnings(rlm.obj <- lm(form, data = iris, contrasts = list(Species = contr.SAS)))
rlm.step <- step(rlm.obj, direction = "both", scope = scope, trace = 1)

## rxLinMod/variableSelection
varsel <- rxStepControl(method = "stepwise", scope = scope)
rxlm.step <- rxLinMod(form, data = iris, variableSelection = varsel,
  verbose = 1, dropMain = FALSE, coefLabelStyle = "R")

## compare lm/step and rxLinMod/variableSelection
rlm.step$anova
rxlm.step$anova

as.matrix(coef(rlm.step))
as.matrix(coef(rxlm.step))

## rxLinMod/variableSelection with keepStepCoefs = TRUE
varsel <- rxStepControl(method = "stepwise", scope = scope, keepStepCoefs = TRUE)
rxlm.step <- rxLinMod(form, data = iris, variableSelection = varsel,
  verbose = 1, dropMain = FALSE, coefLabelStyle = "R")
rxStepPlot(rxlm.step)
```

rxStepPlot: Step Plot

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Plot stepwise coefficients for `rxLinMod`, `rxLogit` and `rxGlm` objects.

Usage

```
rxStepPlot(x, plotStepBreaks = TRUE, omitZeroCoefs = TRUE, eps = .Machine$double.eps,
           main = deparse(substitute(x)), xlab = "Step", ylab = "Stepwise Coefficients",
           type = "b", pch = "*",
           ... )
```

Arguments

`x`

a stepwise regression object of class `rxLinMod`, `rxLogit`, or `rxGlm` with a non-empty `stepCoefs` component, which can be generated by setting the component `keepStepCoefs` of the `variableSelection` argument to `TRUE` when the model is fitted.

`plotStepBreaks`

logical value. If `TRUE`, vertical lines are drawn at each step in the stepwise coefficient paths.

`omitZeroCoefs`

logical flag. If `TRUE`, coefficients with sum of absolute values less than `eps` will be omitted for plotting.

`eps`

the smallest positive floating-point number that is treated as nonzero.

`main, xlab, ylab, type, pch, ...`

additional graphical arguments to be passed directly to the underlying matplot function.

Value

the nonzero stepwise coefficients are returned invisibly.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxStepControl](#), [rxLinMod](#), [rxLogit](#), [rxGlm](#).

Examples

```
## setup
form <- Sepal.Length ~ Sepal.Width + Petal.Length
scope <- list(
  lower = ~ Sepal.Width,
  upper = ~ Sepal.Width + Petal.Length + Petal.Width * Species)

## rxLinMod/variableSelection with keepStepCoefs = TRUE
varsel <- rxStepControl(method = "stepwise", scope = scope, keepStepCoefs = TRUE)
rxlm.step <- rxLinMod(form, data = iris, variableSelection = varsel,
  verbose = 1, dropMain = FALSE, coefLabelStyle = "R")
rxStepPlot(rxlm.step)
```

rxStopEngine: Stop Distributed Computing Engine

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

`rxStopEngine` stops the remote Spark application.

Usage

```
rxStopEngine(computeContext, ... )
rxStopEngine(computeContext, scope = "session")
```

Arguments

`computeContext`

a valid [RxDistributedHpa-class](#). Currently only [RxSpark](#) is supported.

`scope`

only used in `rxStopEngine` for `RxSpark`; a single `character` that takes the value of either:

- `"session"` : stop engine applications running in the current R session.
- `"user"` : stop engine applications running by the current user.

Details

This function stops distributed computing engine applications with scope set to either "session" or "user".

Specifically, for the [RxSpark](#) compute context it stops the remote Spark application(s).

Value

No useful return value.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxSpark](#)

Examples

```
## Not run:

rxStopEngine( computeContext )
rxStopEngine( computeContext , scope = "user" )
## End(Not run)
```

rxSummary: Object Summaries

7/12/2022 • 6 minutes to read • [Edit Online](#)

Description

Produce univariate summaries of objects in RevoScaleR.

Usage

```
rxSummary(formula, data, byGroupOutFile = NULL,
          summaryStats = c("Mean", "StdDev", "Min", "Max", "ValidObs", "MissingObs"),
          byTerm = TRUE, pweights = NULL, fweights = NULL, rowSelection = NULL,
          transforms = NULL, transformObjects = NULL,
          transformFunc = NULL, transformVars = NULL,
          transformPackages = NULL, transformEnvir = NULL,
          overwrite = FALSE,
          useSparseCube = rxGetOption("useSparseCube"),
          removeZeroCounts = useSparseCube,
          blocksPerRead = rxGetOption("blocksPerRead"),
          rowsPerBlock = 100000,
          reportProgress = rxGetOption("reportProgress"), verbose = 0,
          computeContext = rxGetOption("computeContext"), ...)
```

Arguments

formula

formula, as described in [rxFormula](#). The formula typically does not contain a response variable, i.e. it should be of the form `~ terms`. If `~.` is used as the formula, summary statistics will be computed for all non-character variables. If a numeric variable is interacted with a factor variable, summary statistics will be computed for each category of the factor.

data

either a data source object, a character string specifying a .xdf file, or a data frame object to summarize.

byGroupOutFile

NULL, a character string or vector of character strings specifying .xdf file names(s), or an RxXdfData object or list of RxXdfData objects. If not NULL, and the formula includes computations by factor, the by-group summary results will be written out to one or more .xdf files. If more than one `.xdf` file is created and a single character string is specified, an integer will be appended to the base `byGroupOutFile` name for additional file names. The resulting RxXdfData objects will be listed in the `categorical` component of the output object. `byGroupOutFile` is not supported when using distributed compute contexts such as RxHadoopMR.

summaryStats

a character vector containing one or more of the following values: "Mean", "StdDev", "Min", "Max", "ValidObs", "MissingObs", "Sum".

byTerm

logical variable. If `TRUE`, missings will be removed by term (by variable or by interaction expression) before computing summary statistics. If `FALSE`, observations with missings in any term will be removed before

computations.

pweights

character string specifying the variable to use as probability weights for the observations.

fweights

character string specifying the variable to use as frequency weights for the observations.

rowSelection

name of a logical variable in the data set (in quotes) or a logical expression using variables in the data set to specify row selection. For example, `rowSelection = "old"` will use only observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` will use only observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

transforms

an expression of the form `list(name = expression, ...)` representing the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

transformObjects

a named list containing objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

transformFunc

variable transformation function. See [rxTransform](#) for details.

transformVars

character vector of input data set variables needed for the transformation function. See [rxTransform](#) for details.

transformPackages

character vector defining additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, e.g., those explicitly defined in RevoScaleR functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.

transformEnvir

user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

overwrite

logical value. If `TRUE`, an existing `byGroupOutFile` will be overwritten. `overwrite` is ignored if `byGroupOutFile` is `NULL`.

useSparseCube

logical value. If `TRUE`, sparse cube is used.

removeZeroCounts

logical flag. If `TRUE`, rows with no observations will be removed from the output for counts of categorical data.

By default, it has the same value as `useSparseCube`. For large summary computation, this should be set to `TRUE`, otherwise R may run out of memory even if the internal C++ computation succeeds.

`blocksPerRead`

number of blocks to read for each chunk of data read from the data source.

`rowsPerBlock`

maximum number of rows to write to each block in the `byGroupOutFile` (if it is not `NULL`).

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`verbose`

integer value. If `0`, no additional output is printed. If `1`, additional summary information is printed.

`computeContext`

a valid `RxComputeContext`. The `RxSpark`, and `RxHadoopMR` compute contexts distribute the computation among the nodes specified by the compute context; for other compute contexts, the computation is distributed if possible on the local computer.

`...`

additional arguments to be passed directly to the Revolution Compute Engine.

Details

Special function `F()` can be used in formula to force a variable to be interpreted as factors.

If the formula contains a single dependent or response variable, summary statistics are computed for the interaction between that variable and the first term of the independent variables. (Multiple response variables are not permitted.) For example, using the formula `y ~ xfac` will give the same results as using the formula `~y:xfac`, where `y` is a continuous variable and `xfac` is a factor. Summary statistics for `y` are computed for each factor level of `x`. This facilitates using the same formula in `rxSummary` as in, for example, `rxCube` or `rxLinMod`.

Value

an `rxSummary` object containing the following elements:

`nobs.valid`

number of valid observations.

`nobs.missing`

number of missing observations.

`sDataFrame`

data frame containing summaries for continuous variables.

`categorical`

list of summaries for categorical variables.

categorical.type

types of categorical summaries: can be "counts", or "cube" (for crosstab counts) or "none" (if there is no categorical summaries).

formula

formula used to obtain the summary.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxTransform](#)

Examples

```
# Create a local data frame
DF <- data.frame(sex = c("Male", "Male", "Female", "Male"),
                 age = c(20, 20, 12, 10), score = 1.1:4.1)

# get summary of sex variable
rxSummary(~ sex, DF)

# obtain within sex-category statistics of the score variable
rxSummary(score ~ sex, DF)

# use transforms to create a factor variable and compute
# summary statistics by each factor level
rxSummary(~score:ageGroup, data=DF,
           transforms = list(ageGroup = cut(age, seq(0, 30, 10)))) 

# the following will give the same results
rxSummary(score~ageGroup, data=DF,
           transforms = list(ageGroup = cut(age, seq(0, 30, 10)))) 

# the same formula can be used in rxCube
rxCube(score~ageGroup, data=DF, transforms=list(ageGroup = cut(age, seq(0,30,10)))) 

# Write summary statistics by group to an .xdf file. Here the groups
# are defined as year of age by sex by state (3 states in CensusWorkers file),
# so summary statistics for 46 x 2 x 3 groups are computed. The first term
# will just compute the Counts for each group, while the second two will
# compute by-group Means and ValidObs for incwage and wkswork1

censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers.xdf")
sumOutFile <- tempfile(pattern = ".rxTempSumOut", fileext = ".xdf")

sumOut <- rxSummary(~F(age):sex:state + incwage:F(age):sex:state + wkswork1:F(age):sex:state,
                     data = censusWorkers, blocksPerRead = 3,
                     byGroupOutFile = sumOutFile, rowsPerBlock = 10, summaryStats = c("Mean", "ValidObs"))

rxGetVarInfo(sumOutFile)

file.remove(sumOutFile)
```

rxSyncPackages: Synchronize Packages for Compute Context

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Synchronizes all R packages installed in a database with the packages on the file system. The packages contained in the database are also installed on the file system so they can be loaded by R. You might need to use rxSyncPackages if you rebuilt a server instance or restored SQL Server databases on a new machine, or if you think an R package on the file system is corrupted.

Usage

```
rxSyncPackages(computeContext = rxGetOption("computeContext"),
               scope = c("shared", "private"),
               owner = c(),
               verbose = getOption("verbose"))
```

Arguments

computeContext

an [RxComputeContext](#) or equivalent character string or `NULL`. If set to the default of `NULL`, the currently active compute context is used. Supported compute contexts are [RxInSqlServer](#).

scope

character vector containing either `"shared"` or `"private"` or both. `"shared"` synchronizes the packages on a per-database shared location on SQL Server, which in turn can be used (referred) by multiple different users. `"private"` synchronizes the packages on per-database, per-user private location on SQL Server which is only accessible to the single user. By default both `"shared"` and `"private"` are set, which will synchronize the entire table of packages for all scopes and users.

owner

character vector. Should be either empty `''` or a vector of valid SQL database user account names. Only users in `'db_owner'` role for a database can specify this value to install packages on behalf of other users.

verbose

logical. If `TRUE`, "progress report" is given during installation of given packages.

Details

For a [RxInSqlServer](#) compute context, the user specified as part of connection string is considered the package owner if the `owner` argument is empty. To call this function, a user must be granted permissions by a database owner, making the user a member of either `'rpkgs-shared'` or `'rpkgs-private'` database role. Members of the `'rpkgs-shared'` role can install packages to `"shared"` location and `"private"` location. Members of the `'rpkgs-private'` role can only install packages in a `"private"` location for their own use. To use the packages installed on the SQL Server, a user must be at least a member of `'rpkgs-users'` role.

See the help file for additional details.

Value

Invisible `NULL`

See Also

[rxInstallPackages](#), [rxPackage](#), [install.packages](#), [rxFindPackage](#), [rxInstalledPackages](#), [rxRemovePackages](#),
[rxSqlLibPaths](#),
[require](#)

Examples

```
## Not run:

#
# create SQL compute context
#
connectionString <- "Driver=SQL Server;Server=myServer;Database=TestDB;Trusted_Connection=True;"
computeContext <- RxInSqlServer(connectionString = connectionString )

#
# Synchronizes all packages listed in the database for all scopes and users
#
rxSyncPackages(computeContext=computeContext, verbose=TRUE)

#
# Synchronizes all packages for shared scope
#
rxSyncPackages(computeContext=computeContext, scope="shared", verbose=TRUE)

#
# Synchronizes all packages for private scope
#
rxSyncPackages(computeContext=computeContext, scope="private", verbose=TRUE)

#
# Synchronizes all packages for a private scope for user1
#
rxSyncPackages(pcomputeContext=computeContext, scope="private", owner = "user1", verbose=TRUE))

## End(Not run)
```

RxTeradata-class: Class RxTeradata

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Teradata data source connection class.

Generators

The targeted generator [RxTeradata](#) as well as the general generator [rxNewDataSource](#).

Extends

Class RxDataSource, directly.

Methods

[show](#)

```
signature(object = "RxTeradata") : ...
```

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxDataSource-class](#), [RxTeradata](#), [rxNewDataSource](#)

RxTeradata: Generate Teradata Data Source Object

7/12/2022 • 6 minutes to read • [Edit Online](#)

Description

This is the main generator for S4 class RxTeradata, which extends RxDataSource.

Usage

```
RxTeradata(table = NULL, sqlQuery = NULL, dbmsName = NULL, databaseName = NULL,
           connectionString = NULL, user = NULL, password = NULL, teradataId = NULL,
           rowBuffering = TRUE, trimSpace = NULL, returnDataFrame = TRUE, stringsAsFactors = FALSE,
           colClasses = NULL, colInfo = NULL, rowsPerRead = 500000, verbose = 0,
           tableOpClause = NULL, writeFactorsAsIndexes = FALSE,
           ... )

## S3 method for class `RxTeradata':
head (x, n = 6L, reportProgress = 0L, ...)
## S3 method for class `RxTeradata':
tail (x, n = 6L, addrownums = TRUE, reportProgress = 0L, ...)
```

Arguments

`table`

`NULL` or character string specifying the table name. Cannot be used with `sqlQuery`.

`sqlQuery`

`NULL` or character string specifying a valid SQL select query. Cannot contain hidden characters such as tabs or newlines. Cannot be used with `table`.

`dbmsName`

`NULL` or character string specifying the Database Management System (DBMS) name.

`databaseName`

`NULL` or character string specifying the name of the database.

`connectionString`

`NULL` or character string specifying the connection string.

`user`

`NULL` or character string specifying the user name.

`password`

`NULL` or character string specifying the password.

`teradataId`

`NULL` or character string specifying the teradatald.

`rowBuffering`

logical specifying whether or not to buffer rows on read from the database. If you are having problems with your ODBC driver, try setting this to `FALSE`.

`trimSpace`

logical specifying whether or not to trim the white character of string data for reading.

`returnDataFrame`

logical indicating whether or not to convert the result from a list to a data frame (for use in `rxReadNext` only). If `FALSE`, a list is returned.

`stringsAsFactors`

logical indicating whether or not to automatically convert strings to factors on import. This can be overridden by specifying `"character"` in `colClasses` and `colInfo`. If `TRUE`, the factor levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use `colInfo` with specified `"levels"`.

`colClasses`

character vector specifying the column types to use when converting the data. The element names for the vector are used to identify which column should be converted to which type.

- Allowable column types are:

- `"logical"` (stored as `uchar`),
- `"integer"` (stored as `int32`),
- `"float32"` (the default for floating point data for .xdf files),
- `"numeric"` (stored as `float64` as in R),
- `"character"` (stored as `string`),
- `"factor"` (stored as `uint32`),
- `"int16"` (alternative to integer for smaller storage space),
- `"uint16"` (alternative to unsigned integer for smaller storage space),
- `"Date"` (stored as Date, i.e. `float64`)

- Note for `"factor"` type, the levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use `colInfo` with specified `"levels"`.

- Note that equivalent types share the same bullet in the list above; for some types we allow both 'R-friendly' type names, as well as our own, more specific type names for .xdf data.
- Note also that specifying the column as a "factor" type is currently equivalent to "string" - for the moment, if you wish to import a column as factor data you must use the `colInfo` argument, documented below.

`colInfo`

list of named variable information lists. Each variable information list contains one or more of the named elements given below. The information supplied for `colInfo` overrides that supplied for `colClasses`.

- Currently available properties for a column information list are:
- `type` - character string specifying the data type for the column. See `colClasses` argument description for the available types.
- `newName` - character string specifying a new name for the variable.
- `description` - character string specifying a description for the variable.
- `levels` - character vector containing the levels when `type = "factor"`. If the levels property is not provided,

factor levels will be determined by the values in the source column. If levels are provided, any value that does not match a provided level will be converted to a missing value.

- `newLevels` - new or replacement levels specified for a column of type "factor". It must be used in conjunction with the `levels` argument. After reading in the original data, the labels for each level will be replaced with the `newLevels`.
- `low` - the minimum data value in the variable (used in computations using the `F()` function).
- `high` - the maximum data value in the variable (used in computations using the `F()` function).

`rowsPerRead`

number of rows to read at a time.

`verbose`

integer value. If `0`, no additional output is printed. If `1`, information on the odbc data source type (`odbc` or `odbcFast`) is printed.

`writeFactorsAsIndexes`

logical. If `TRUE`, when writing to an `RxOdbcData` data source, underlying factor indexes will be written instead of the string representations.

`...`

additional arguments to be passed directly to the underlying functions, including the Teradata Export driver. One important attribute that can be passed is the `TD_SPOOLMODE` attribute, which RevoScaleR sets to `"NoSpool"` for efficiency. If you encounter difficulties while extracting data, you might want to specify `TD_SPOOLMODE="Spool"`. To tune performance, use the `TD_MIN_SESSIONS` and `TD_MAX_SESSIONS` arguments to specify the minimum and maximum number of sessions. If these are not specified, the default values of 1 and 4, respectively, are used.

`TD_TRACE_LEVEL` specifies the types of diagnostic messages written by each instance of the driver to an external log file. Setting it to `1` turns off tracing (the default), `2` activates the tracing function for driver specific activities, `3` activates the tracing function for interaction with the Teradata Database, `4` activates the tracing function for activities related to the Notify feature, `5` activates the tracing function for activities involving the opcommon library, and `7` activates tracing for all of the above activities. `TD_TRACE_OUTPUT` specifies the name of the external file used for tracing messages. If a file with the specified name already exists, that file will be overwritten. `TD_OUTLIMIT` limits the number of rows that the Export driver exports. `TD_MAX_DECIMAL_DIGITS` specifies the maximum number of decimal digits (a value for the maximum precision) to be returned by the database. The default value is 18, and values above that are not supported. See Chapter 6 of the *Teradata Parallel Transporter Application Programming Interface Programmer Guide* for details on allowable attributes.

`x`

an `RxTeradata` object

`n`

positive integer. Number of rows of the data set to extract.

`addrownums`

logical. If `TRUE`, row numbers will be created to match the original data set.

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.

- `3` : rows processed and all timings are reported.

`tableOpClause`

Deprecated.

Details

The `tail` method is not functional for this data source type and will report an error.

The `RxTeradata` data source behaves differently depending upon the compute context currently in effect. In a local compute context, `RxTeradata` uses the Teradata Parallel Transporter's Export driver to read large quantities of data quickly from Teradata.

To use the Teradata connector, you need Teradata ODBC drivers and the Teradata Parallel Transporter installed on your client using the Teradata 14.10 or 15.00 client installer, and you need a high-speed connection (100 Mbps or above) to a Teradata appliance running version 14.0 or later.

The `user`, `password`, and `teradataId` arguments take precedence over equivalent information provided within the `connectionString` argument. If, for example, you provide the user name in both the `connectionString` and `user` arguments, the one in `connectionString` is ignored.

Value

object of class RxTeradata.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

Teradata Corporation (2012). *SQL Fundamentals*.

Teradata Corporation (2012). *Teradata Parallel Transporter Application Programming Interface Programmer Guide*

See Also

[RxTeradata-class](#), [rxNewDataSource](#), [rxImport](#).

Examples

```
## Not run:

# Create a Teradata data source
# Note: for improved security, read connection string from a file, such as
# teradataConnString <- readLines("tdConnString.txt")

teradataConnString <- "DRIVER=Teradata;DBNAME=hostname;DATABASE=RevoTestDB;UID=DBUser;PWD=Password;"

rxTeradataDS <- RxTeradata(sqlQuery = "SELECT * FROM RevoTestDB.claims",
                           connectionString = teradataConnString)

# Create an xdf file name
claimsXdffFileName <- file.path(tempdir(), "importedClaims.xdf")

# Import the data into the xdf file
rxImport(rxTeradataDS, claimsXdffFileName, overwrite = TRUE)

# Read xdf file into a data frame
claimsIn <- rxDataStep(inData = claimsXdffFileName)
head(claimsIn)
## End(Not run)
```

rxTeradataSql: rxTeradataSql

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Execute an arbitrary SQL statement that does not return data in a Teradata data base.

Usage

```
rxTeradataSql(sqlStatement, connectionString = NULL, ...)
rxTeradataTableExists(table, connectionString = NULL, ...)
rxTeradataDropTable(table, connectionString = NULL, ...)
```

Arguments

`sqlStatement`

character string specifying valid SQL statement that does not return data

`table`

One of the following:

- a character string specifying a table name, in the form `"tablename"` or `"database.tablename"`. (In Teradata, each user is a database; the user database can be specified as `"username.tablename"`.)
- an `RxTeradata` data source that has the `table` specified
- an `RxDynamicData` data source that has the `table` specified.

`connectionString`

`NULL` or character string specifying the connection string. If `NULL`, the connection string from the currently active compute context will be used if available.

`...`

Additional arguments to be passed through.

Details

An SQL query is passed to the Teradata ODBC driver.

Value

`rxTeradataSql` is executed for the side effects and returns `NULL` invisibly. `rxTeradataTableExists` returns `TRUE` if the table exists, `FALSE` otherwise. The database searched for the table is determined as follows:

* If the `table` argument is a character string and specifies a database, that is, if the `table` argument is specified as `"database.tablename"`, the specified database is searched for the table. If the containing database does not exist, an error is returned.

* If the `table` argument is a character string and does not specify a database, that is, if the `table` argument is specified as `"tablename"`, the database specified in the `connectionString` is searched. If `connectionString` is

missing, the connection string in the current compute context object is used.

* If the `table` argument is an `RxTeradata` or `RxOdbcData` data source, the table name specified in the data source is searched for in the database specified in the data source.

`rxTeradataDropTable` returns `TRUE` if the table is successfully dropped, `FALSE` otherwise (for example, if the table did not exist).

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxTeradata](#)

Examples

```
## Not run:

# Copy a table
sqlStatement <- "CREATE TABLE YourDataBase.rxClaimsCopy AS YourDataBase.claims WITH DATA"
rxTeradataSql(sqlStatement = sqlStatement)

# Use the RxOdbcData data source and its sqlQuery argument to
# get data back as a data frame:
rxTeradataQuery <- function(query, connectionString=NULL, maxRowsByCols=NULL)
{
  if (is.null(connectionString))
  {
    currentComputeContext <- rxGetComputeContext()
    if (.hasSlot(currentComputeContext, "connectionString"))
    {
      connectionString <- currentComputeContext@connectionString
    }
    else
    {
      stop("A 'connectionString' must be specified.")
    }
  }
  internalDS <- RxOdbcData(sqlQuery=query, connectionString(connectionString))
  rxImport(internalDS, maxRowsByCols=maxRowsByCols)
}

## Use the above to show the tables in database 'dbtest':
tableQuery <- paste("SELECT TableName FROM dbc.tables ",
                    "WHERE tablekind = 'T' and databasename='dbtest'", sep="")
rxTeradataQuery(query=tableQuery)
## End(Not run)
```

RxTextData-class: Class RxTextData

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Text data source connection class.

Generators

The targeted generator [RxTextData](#) as well as the general generator [rxNewDataSource](#).

Extends

Class RxFileData, directly. Class RxDataSource, by class RxFileData.

Methods

[show](#)

`signature(object = "RxTextData") : ...`

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxDataSource-class](#), [RxTextData](#), [rxNewDataSource](#)

RxTextData: Generate Text Data Source Object

7/12/2022 • 15 minutes to read • [Edit Online](#)

Description

This is the main generator for S4 class RxTextData, which extends RxDataSource.

Usage

```
RxTextData(file, stringsAsFactors = FALSE,
           colClasses = NULL, colInfo = NULL,
           varsToKeep = NULL, varsToDrop = NULL, missingValueString = "NA",
           rowsPerRead = 500000, delimiter = NULL, combineDelimiters = FALSE,
           quoteMark = "\"", decimalPoint = ".", thousandsSeparator = NULL,
           readDateFormat = "[%y[-] [/] %m[-] [/] %d]",
           readPOSIXctFormat = "%y[-] [/] %m[-] [/] %d [%H:%M[:%S]][%p]",
           centuryCutoff = 20, firstRowIsColNames = NULL, rowsToSniff = 10000,
           rowsToSkip = 0, returnDataFrame = TRUE,
           defaultReadBufferSize = 10000,
           defaultDecimalColType = rxGetOption("defaultDecimalColType"),
           defaultMissingColType = rxGetOption("defaultMissingColType"),
           writePrecision = 7, stripZeros = FALSE, quotedDelimiters = FALSE,
           isFixedFormat = NULL, useFastRead = NULL,
           createFileSet = NULL, rowsPerOutFile = NULL,
           verbose = 0, checkVarsToKeep = FALSE,
           fileSystem = NULL, inputEncoding = "utf-8", writeFactorsAsIndexes = FALSE)
```

Arguments

`file`

character string specifying a text file. If it has an .stsextension, it is interpreted as a fixed format schema file. If the

`colInfo` argument contains `start` and `width` information, it is interpreted as a fixed format data file.

Otherwise, it is treated as a delimited text data file. See the Details section for more information on using .sts files.

`returnDataFrame`

logical indicating whether or not to convert the result from a list to a data frame (for use in `rxReadNext` only). If `FALSE`, a list is returned.

`stringsAsFactors`

logical indicating whether or not to automatically convert strings to factors on import. This can be overridden by specifying "character" in `colClasses` and `colInfo`. If `TRUE`, the factor levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use `colInfo` with specified "levels".

`colClasses`

character vector specifying the column types to use when converting the data. The element names for the vector are used to identify which column should be converted to which type.

- Allowable column types are:

- "logical" (stored as `uchar`),
- "integer" (stored as `int32`),
- "float32" (the default for floating point data for .xdf files),
- "numeric" (stored as `float64` as in R),
- "character" (stored as `string`),
- "factor" (stored as `uint32`),
- "ordered" (ordered factor stored as `uint32`. Ordered factors are treated the same as factors in RevoScaleR analysis functions.),
- "int16" (alternative to integer for smaller storage space),
- "uint16" (alternative to unsigned integer for smaller storage space),
- "Date" (stored as Date, i.e. `float64`. Not supported if `useFastRead` = `TRUE`.)
- "POSIXct" (stored as POSIXct, i.e. `float64`. Not supported if `useFastRead` = `TRUE`.)

- Note for "factor" and "ordered" types, the levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use `colInfo` with specified "levels".

`colInfo`

list of named variable information lists. Each variable information list contains one or more of the named elements given below. When importing fixed format data, either `colInfo` or an .sts schema file should be supplied. For such fixed format data, only the variables specified will be imported. For all text types, the information supplied for `colInfo` overrides that supplied for `colclasses`.

- Currently available properties for a column information list are:
- `type` - character string specifying the data type for the column. See `colclasses` argument description for the available types. If the `type` is not specified for fixed format data, it will be read as character data.
- `newName` - character string specifying a new name for the variable.
- `description` - character string specifying a description for the variable.
- `levels` - character vector containing the levels when `type = "factor"`. If the `levels` property is not provided, factor levels will be determined by the values in the source column. If levels are provided, any value that does not match a provided level will be converted to a missing value.
- `newLevels` - new or replacement levels specified for a column of type "factor". It must be used in conjunction with the `levels` argument. After reading in the original data, the labels for each level will be replaced with the `newLevels`.
- `low` - the minimum data value in the variable (used in computations using the `F()` function).
- `high` - the maximum data value in the variable (used in computations using the `F()` function).
- `start` - the left-most position, in bytes, for the column of a fixed format file respectively. When all elements of `colInfo` have "start", the text file is designated as a fixed format file. When none of the elements have it, the text file is designated as a delimited file. Specification of `start` must always be accompanied by specification of `width`.
- `width` - the number of characters in a fixed-width character column or the column of a fixed format file. If `width` is specified for a character column, it will be imported as a fixed-width character variable. Any characters beyond the fixed width will be ignored. Specification of `width` is required for all columns of a fixed format file (if not provided in an .sts file).
- `decimalPlaces` - the number of decimal places.
- `index` - column index in the original delimited text data file. It is used as an alternative to naming the variable information list if the original delimited text file does not contain column names. Ignored if a name for the list is specified. Should not be used with fixed format files.

`varsToKeep`

character vector of variable names to include when reading from the input data file. If `NULL`, argument is ignored. Cannot be used with `varsToDrop`.

`varsToDrop`

character vector of variable names to exclude when reading from the input data file. If `NULL`, argument is ignored. Cannot be used with `varsToKeep`.

`missingValueString`

character string containing the missing value code. It can be an empty string: `""`.

`rowsPerRead`

number of rows to read at a time.

`delimiter`

character string containing the character to use as the separator between variables. If `NULL` and the `colInfo` argument does not contain `"start"` and `"width"` information (which implies a fixed-formatted file), the delimiter is auto-sensed from the list `","`, `\t`, `";"`, and `" "`.

`combineDelimiters`

logical indicating whether or not to treat consecutive non-space (`" "`) delimiters as a single delimiter. Space `" "` delimiters are always combined.

`quoteMark`

character string containing the quotation mark. It can be an empty string: `""`.

`decimalPoint`

character string containing the decimal point. Not supported when `useFastRead` is set to `TRUE`.

`thousandsSeparator`

character string containing the thousands separator. Not supported when `useFastRead` is set to `TRUE`.

`readDateFormat`

character string containing the time date format to use during read operations. Not supported when `useFastRead` is set to `TRUE`. Valid formats are:

- `%c` Skip a single character (see also `%w`).
- `%Nc` Skip `N` characters.
- `%%c` Skip the rest of the input string.
- `%d` Day of the month as integer (01-31).
- `%m` Month as integer (01-12) or as character string.
- `%w` Skip a whitespace delimited word (see also `%c`).
- `%y` Year. If less than 100, `centuryCutoff` is used to determine the actual year.
- `%Y` Year as found in the input string.
- `%, [% , %]` input the `%`, `[`, and `]` characters from the input string.
- `[...]` square brackets within format specifications indicate optional components; if present, they are used, but they need not be there.

`readPOSIXctFormat`

character string containing the time date format to use during read operations. Not supported when `useFastRead` is set to `TRUE`. Valid formats are:

- `%c` Skip a single character (see also `%w`).
- `%Nc` Skip `N` characters.
- `%%c` Skip the rest of the input string.
- `%d` Day of the month as integer (01-31).
- `%H` Hour as integer (00-23).
- `%m` Month as integer (01-12) or as character string.
- `%M` Minute as integer (00-59).
- `%n` Milliseconds as integer (00-999).
- `%N` Milliseconds or tenths or hundredths of second.
- `%p` Character string defining 'am'/'pm'.
- `%S` Second as integer (00-59).
- `%w` Skip a whitespace delimited word (see also `%c`).
- `%y` Year. If less than 100, `centuryCutoff` is used to determine the actual year.
- `%Y` Year as found in the input string.
- `%%`, `%[`, `%]` input the `%`, `[`, and `]` characters from the input string.
- `[...]` square brackets within format specifications indicate optional components; if present, they are used, but they need not be there.

`centuryCutoff`

integer specifying the changeover year between the twentieth and twenty-first century if two-digit years are read. Values less than `centuryCutoff` are prefixed by 20 and those greater than or equal to `centuryCutoff` are prefixed by 19. If you specify 0, all two digit dates are interpreted as years in the twentieth century. Not supported when `useFastRead` is set to `TRUE`.

`firstRowIsColNames`

logical indicating if the first row represents column names for reading text. If `firstRowIsColNames` is `NULL`, then column names are auto-detected. The logic for auto-detection is: if the first row contains all values that are interpreted as character and the second row contains at least one value that is interpreted as numeric, then the first row is considered to be column names; otherwise the first row is considered to be the first data row. Not relevant for fixed format data. As for writing, it specifies to write column names as the first row. If `firstRowIsColNames` is `NULL`, the default is to write the column names as the first row.

`rowsToSniff`

number of rows to use in determining column type.

`rowsToSkip`

integer indicating number of leading rows to ignore. Only supported for `useFastRead = TRUE`.

`defaultReadBufferSize`

number of rows to read into a temporary buffer. This value could affect the speed of import.

`defaultDecimalColType`

Used to specify a column's data type when only decimal values (possibly mixed with missing (`NA`) values) are encountered upon first read of the data and the column's type information is not specified via `colInfo` or `colClasses`. Supported types are "float32" and "numeric", for 32-bit floating point and 64-bit floating point values, respectively.

`defaultMissingColType`

Used to specify a given column's data type when only missings (`NA`s) or blanks are encountered upon first read of the data and the column's type information is not specified via `colInfo` or `colClasses`. Supported types are

"float32", "numeric", and "character" for 32-bit floating point, 64-bit floating point and string values, respectively. Only supported for `useFastRead = TRUE`.

`writePrecision`

integer specifying the precision to use when writing numeric data to a file.

`stripZeros`

logical scalar. If `TRUE`, if there are only zeros after the decimal point for a numeric, it will be written as an integer to a file.

`quotedDelimiters`

logical scalar. If `TRUE`, delimiters within quoted strings will be ignored. This requires a slower parsing process. Only applicable to `useFastRead` is set to `TRUE`; delimiters within quotes are always supported when `useFastRead` is set to `FALSE`.

`isFixedFormat`

logical scalar. If `TRUE`, the input data file is treated as a fixed format file. Fixed format files must have a .sts file or a `colInfo` argument specifying the `start` and `width` of each variable. If `FALSE`, the input data file is treated as a delimited text file. If `NULL`, the text file type (fixed or delimited text) is auto-determined.

`useFastRead`

`NULL` or logical scalar. If `TRUE`, the data file is accessed directly by the Microsoft R Services Compute Engine. If `FALSE`, `StatTransfer` is used to access the data file. If `NULL`, the type of text import is auto-determined. `useFastRead` should be set to `FALSE` if importing `Date` or `POSIXct` data types, if the data set contains the delimiter character inside a quoted string, if the `decimalPoint` is not `". "`, if the `thousandsSeparator` is not ",", if the `quoteMark` is not `"\""`, or if `combineDelimiters` is set to `TRUE`. `useFastRead` should be set to `TRUE` if `rowsToSkip` or `defaultMissingColType` are set. If `useFastRead` is `TRUE`, by default variables containing the values `TRUE` and `FALSE` or `T` and `F` will be created as logical variables. `useFastRead` cannot be set to `FALSE` if an HDFS file system is being used. If an incompatible setting of `useFastRead` is detected, a warning will be issued and the value will be changed.

`createFileSet`

logical value or `NULL`. Used only when writing. If `TRUE`, a file folder of text files will be created instead of a single text file. A directory will be created whose name is the same as the text file that would otherwise be created, but with no extension. In the directory, the data will be split across a set of text files (see `rowsPerOutFile` below for determining how many rows of data will be in each file). If `FALSE`, a single text file will be created. If `NULL`, a folder of files will be created if the input data set is a composite XDF file or a folder of text files. This argument is ignored if the text file is fixed format.

`rowsPerOutFile`

numeric value or `NULL`. If a directory of text files is being created, and if the compute context is not `RxHadoopMR`, this will be the number of rows of data put into each file in the directory. When importing is being done on Hadoop using MapReduce, the number of rows per file is determined by the rows assigned to each MapReduce task. If `NULL`, the rows of data will match the input data.

`verbose`

integer value. If `0`, no additional output is printed. If `1`, information on the text type (`text` or `textFast`) is printed.

`checkVarsToKeep`

logical value. If `TRUE` variable names specified in `varsToKeep` will be checked against variables in the data set to make sure they exist. An error will be reported if not found. If there are more than 500 variables in the data set,

this flag is ignored and no checking is performed.

fileSystem

character string or [RxFileSystem](#) object indicating type of file system; "native" or [RxNativeFileSystem](#) object can be used for the local operating system, or an [RxHdfsFileSystem](#) object for the Hadoop file system.

inputEncoding

character string indicating the encoding used by input text. May be set to either "utf-8" (the default), or "gb18030", a standard Chinese encoding. Not supported when `useFastRead` is set to `TRUE`.

writeFactorsAsIndexes

logical. If `TRUE`, when writing to an [RxTextData](#) data source, underlying factor indexes will be written instead of the string representations. Not supported when `useFastRead` is set to `FALSE`.

Details

Delimited Data Type Details

Imported `POSIXct` will have the `tzone` attribute set to "GMT".

Decimal data in text files can be imported into .xdf files and stored as either 32-bit floats or 64-bit doubles. The default for this is 32-bit floats, which can be changed using [rxOptions](#). If the data type cannot be determined (i.e., only missing values are encountered), the column is imported as 64-bit doubles unless otherwise specified.

If stored in 32-bit floats, they are converted into 64-bit doubles whenever they are brought into R. Because there may be no exact binary representation of a particular decimal number, the resulting double may be (slightly) different from a double created by directly converting a decimal value to a double. Thus exact comparisons of values may result in unexpected behavior. For example, if `x` is imported from a text file with a decimal value of "1.2" and stored as a float in an .xdf file, the closest decimal representation of the stored value displayed as a double is 1.2000000476837158. So, bringing it into R as a double and comparing with a decimal constant of 1.2, e.g. `x == 1.2`, will result in `FALSE` because the decimal constant 1.2 in the code is being converted directly to a double.

To store imported text decimal data as 64-bit doubles in an .xdf file, set the `defaultDecimalColType` to "numeric". Doing so will increase the size of the .xdf file, since the size of a double is twice that of a float.

Fixed Format Schema Details

The .sts schema for a fixed format data file consists of two required components (`FILE` and `VARIABLES`) and one optional component (`FIRST LINE`), representing the first line of the data file to read.

FILE file name and path specification

FIRST LINE n when required

VARIABLES

Variable name | variable list | variable range columns (A)

where (A) is used to tag character columns, else use no tag for numeric columns.

If the data file names and .sts file name are different, the full path must be specified in the `FILE` component.

See `file.show(file.path(rxGetOption("sampleDataDir"), "claims.sts"))` for an example fixed format schema file.

If a .sts schema file is used in addition to a `colInfo` argument, schema file is read first, then the `colInfo` is used to modify that information (for example, to specify that a variable should be read as a factor).

Encoding Details

Character data in the input file must be encoded as ASCII or UTF-8 in order to be imported correctly. If the data contains UTF-8 multibyte (i.e., non-ASCII) characters, make sure the `useFastRead` parameter is set to `FALSE` as the 'fast' version of this object may not handle extended UTF-8 characters correctly.

Value

object of class RxTextData.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxTextData-class](#), [rxImport](#), [rxNewDataSource](#).

Examples

```

### Read from a csv file ####
# Create a csv data source
claimsCSVFileName <- file.path(rxGetOption("sampleDataDir"), "claims.txt")
claimsCSVDataSource <- RxTextData(claimsCSVFileName)

# Specify the location for the new .xdf file
claimsXdffFileName <- file.path(tempdir(), "importedClaims.xdf")

# Import the data into the xdf file
claimsDS <- rxImport(inData = claimsCSVDataSource,
                      outFile = claimsXdffFileName, overwrite = TRUE)

# Look at file information
rxGetInfo( claimsDS, getVarInfo = TRUE )

# Clean-up: delete the new file
file.remove( claimsXdffFileName )



### Read from a fixed format file ####
# Create a fixed format data source
claimsFFFFileName <- file.path(rxGetOption("sampleDataDir"), "claims.dat")
claimsFFColInfo <-
  list(rownames = list(start = 1, width = 3),
       age = list(type = "factor", start = 4, width = 5),
       car.age = list(type = "factor", start = 9, width = 3),
       type = list(type = "factor", start = 12, width = 1),
       cost = list(type = "numeric", start = 13, width = 6),
       number = list(type = "integer", start = 19, width = 3))
claimsFFSource <- RxTextData(claimsFFFFileName, colInfo = claimsFFColInfo)

# Import the data into a data frame
claimsIn <- rxImport(inData = claimsFFSource)
rxGetInfo( claimsIn, getVarInfo = TRUE)

#####
# Import a space delimited text file without variable names
# Specify names for the data, and use "." for missing values
# Perform additional transformations when importing
#####

unitTestDataDir <- rxGetOption("unitTestDataDir")
gardenDAT <- file.path(unitTestDataDir, "Garden.dat")
gardenDS <- RxTextData(file=gardenDAT, firstRowIsColNames = FALSE,
                      missingValueString=".",
                      colInfo = list(
                        list(index = 1, newName = "Name"),
                        list(index = 2, newName = "Tomato"),
                        list(index = 3, newName = "Zucchini"),
                        list(index = 4, newName = "Peas"),
                        list(index = 5, newName = "Grapes")) )

rsrData <- rxImport(inData=gardenDS,
                     transforms = list(
                       Zone = rep(14, .rxNumRows),
                       Type = rep("home", .rxNumRows),
                       Zucchini = Zucchini * 10,
                       Total = Tomato + Zucchini + Peas + Grapes,
                       PerTom = 100*(Tomato/Total)))
)
rsrData

```

rxTextToXdf: Text File Data Import (to .xdf)

7/12/2022 • 15 minutes to read • [Edit Online](#)

Description

Import text data into to an .xdf file using `fastText` mode. Can also use rxImport.

Usage

```
rxTextToXdf(inFile, outFile, rowSelection = NULL, rowsToSkip = 0,
            transforms = NULL, transformObjects = NULL,
            transformFunc = NULL, transformVars = NULL,
            transformPackages = NULL, transformEnvir = NULL,
            append = "none", overwrite = FALSE, numRows = -1,
            stringsAsFactors = FALSE, colClasses = NULL, colInfo = NULL,
            missingValueString = "NA",
            rowsPerRead = 500000, columnDelimiters = NULL,
            autoDetectColNames = TRUE, firstRowIsColNames = NULL,
            rowsToSniff = 10000, defaultReadBufferSize = 10000,
            defaultDecimalColType = rxGetOption("defaultDecimalColType"),
            defaultMissingColType = rxGetOption("defaultMissingColType"),
            reportProgress = rxGetOption("reportProgress"),
            xdfCompressionLevel = rxGetOption("xdfCompressionLevel"),
            createCompositeSet = FALSE,
            blocksPerCompositeFile = 3)
```

Arguments

`inFile`

character string specifying the input comma or whitespace delimited text file.

`outFile`

either an RxXdfData object or a character string specifying the output .xdf file.

`rowSelection`

name of a logical variable in the data set (in quotes) or a logical expression using variables in the data set to specify row selection. For example, `rowSelection = "old"` will use only observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` will use only observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

`rowsToSkip`

integer indicating number of leading rows to ignore.

`transforms`

an expression of the form `list(name = expression, ...)` representing variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

`transformObjects`

a named list containing objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

`transformFunc`

variable transformation function. See [rxTransform](#) for details.

`transformVars`

character vector of input data set variables needed for the transformation function. See [rxTransform](#) for details.

`transformPackages`

character vector defining additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, e.g., those explicitly defined in RevoScaleR functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.

`transformEnvir`

user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

`append`

either `"none"` to create a new .xdf file or `"rows"` to append rows to an existing .xdf file. If `outFile` exists and `append` is `"none"`, then `overwrite` argument must be set to `TRUE`. Note that appending to a factor column normally extends its levels with whatever values are in the appended data, but if levels are explicitly specified using `colInfo` then it will be extended only with the specified levels - any other values will be added as missings.

`overwrite`

logical value. If `TRUE`, an existing `outFile` will be overwritten, or if appending columns existing columns with the same name will be overwritten. `overwrite` is ignored if appending rows.

`numRows`

integer value specifying the maximum number of rows to import. If set to `-1`, all rows will be imported.

`stringsAsFactors`

logical indicating whether or not to automatically convert strings to factors on import. This can be overridden by specifying `"character"` in `colClasses` and `colInfo`. If `TRUE`, the factor levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use `colInfo` with specified `"levels"`.

`colClasses`

character vector specifying the column types to use when converting the data. The element names for the vector are used to identify which column should be converted to which type.

- Allowable column types are:

- "logical" (stored as `uchar`),
- "integer" (stored as `int32`),
- "factor" (stored as `uint32`),
- "ordered" (an ordered factor stored as `uint32`. Ordered factors are treated the same as factors in RevoScaleR analysis functions.).

- "float32" (the default for floating point data for .xdf files),
- "numeric" (stored as `float64` as in R),
- "character" (stored as `string`),
- "int16" (alternative to integer for smaller storage space),
- "uint16" (alternative to unsigned integer for smaller storage space)
- Note for `"factor"` and `"ordered"` type, the levels will be coded in the order encountered. Since this factor level ordering is row dependent, the preferred method for handling factor columns is to use `colInfo` with specified `"levels"`.
- Note that equivalent types share the same bullet in the list above; for some types we allow both 'R-friendly' typenames, as well as our own, more specific type names for .xdf data.
- Note also that specifying the column as a "factor" type is currently equivalent to "string" - for the moment, if you wish to import a column as factor data you must use the `colInfo` argument, documented below.

`colInfo`

list of named variable information lists. Each variable information list contains one or more of the named elements given below. The information supplied for `colInfo` overrides that supplied for `colClasses`.

- Currently available properties for a column information list are:
- `type(character)` - character string specifying the data type for the variable. See `colClasses` argument description for the available types.
- `newName(character)` - character string specifying a new name for the variable.
- `description(character)` - character string specifying a description for the variable.
- `levels(character vector)` - levels specified for a column of type "factor". If the `levels` property is not provided, factor levels will be determined by the entries in the source column. Blanks or empty strings will be set to missing values. If levels are provided, any source column entry that does not match a provided level will be converted to a missing value. If an ".rxOther" level is included, all missings will be assigned to that level.
- `newLevels(character vector)` - new or replacement levels specified for a column of type "factor". Must be used in conjunction with the `levels` component of `colClasses`. After reading in the original data, the labels for each level will be replaced with the `newLevels`.
- `low` - the minimum data value in the variable (used in computations using the `F()` function).
- `high` - the maximum data value in the variable (used in computations using the `F()` function).

`missingValueString`

character string containing the missing value code.

`rowsPerRead`

number of rows to read for each chunk of data; if `NULL`, all rows are read. This is the number of rows written per block to the .xdf file unless the number of rows in the read chunk is modified through code in a `transformFunc`.

`columnDelimiters`

character string containing characters to be used as delimiters. If `NULL`, the file is assumed to be either comma or tab delimited.

`autoDetectColNames`

DEPRECATED. Use `firstRowIsColNames`.

`firstRowIsColNames`

logical indicating if the first row represents column names. If `firstRowIsColNames` is `NULL`, then column names are auto-detected. The logic for auto-detection is: if the first row contains all values that are interpreted as character and the second row contains at least one value that is interpreted as numeric, then the first row is considered to be column names; else the first row is considered to be the first data row.

`rowsToSniff`

number of rows to use in determining column type.

`defaultReadBufferSize`

number of rows to read into a temporary buffer. This value could affect the speed of import.

`defaultDecimalColType`

Used to specify a column's data type when only decimal values (possibly mixed with missing (`NA`) values) are encountered upon first read of the data and the column's type information is not specified via `colInfo` or `colClasses`. Supported types are "float32" and "numeric", for 32-bit floating point and 64-bit floating point values, respectively.

`defaultMissingColType`

Used to specify a given column's data type when only missings (`NA`s) or blanks are encountered upon first read of the data and the column's type information is not specified via `colInfo` or `colClasses`. Supported types are "float32", "numeric", and "character" for 32-bit floating point, 64-bit floating point and string values, respectively.

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`xdfCompressionLevel`

integer in the range of -1 to 9. The higher the value, the greater the amount of compression - resulting in smaller files but a longer time to create them. If `xdfCompressionLevel` is set to 0, there will be no compression and files will be compatible with the 6.0 release of Revolution R Enterprise. If set to -1, a default level of compression will be used.

`createCompositeSet`

logical value. EXPERIMENTAL. If `TRUE`, a composite set of files will be created instead of a single .xdf file. A directory will be created whose name is the same as the .xdf file that would otherwise be created, but with no extension. Subdirectories data and metadata will be created. In the data subdirectory, the data will be split across a set of .xdff files (see `blocksPerCompositeFile` below for determining how many blocks of data will be in each file). In the metadata subdirectory there is a single .xdfm file, which contains the meta data for all of the .xdff files in the data subdirectory. When the `fileSystem` is "hdfs" a composite set of files is always created.

`blocksPerCompositeFile`

integer value. EXPERIMENTAL. If `createCompositeSet=TRUE`, and if the compute context is not `RxHadoopMR`, this will be the number of blocks put into each .xdff file in the composite set. When importing is being done on Hadoop using MapReduce, the number of rows per .xdff file is determined by the rows assigned to each MapReduce task, and the number of blocks per .xdff file is therefore determined by `rowsPerRead`.

Details

Decimal data in text files can be imported into .xdf files and stored as either 32-bit floats or 64-bit doubles. The default for this is 32-bit floats, which can be changed using `rxOptions`.

If stored in 32-bit floats, they are converted into 64-bit doubles whenever they are brought into R. Because there may be no exact binary representation of a particular decimal number, the resulting double may be (slightly) different from a double created by directly converting a decimal value to a double. Thus exact comparisons of values may result in unexpected behavior. For example, if `x` is imported from a text file with a decimal value of `"1.2"` and stored as a float in an .xdf file, the closest decimal representation of the stored value displayed as a double is `1.2000000476837158`. So, bringing it into R as a double and comparing with a decimal constant of `1.2`, e.g. `x == 1.2`, will result in `FALSE` because the decimal constant `1.2` in the code is being converted directly to a double.

To store imported text decimal data as 64-bit doubles in an .xdf file, set the `"defaultDecimalColType"` to `"numeric"`. Doing so will increase the size of the .xdf file, since the size of a double is twice that of a float.

An error message will be issued if the data being appended is of a different type than the existing data and the conversion might result in a loss of data.

For example, appending a float column to an integer column will result in an error unless the column type is explicitly set to `"integer"` using the `colClasses` argument. If explicitly set, the float data will then be converted to integer data when imported.

`Date` is not currently supported in `colClasses` or `collInfo`, but `Date` variables can be created by importing string data and converting to a `Date` variable using `as.Date` in a `transforms` argument.

Encoding Details

If the input data source or file contains non-ASCII character information, please use the function `rxImport` with the encoding settings recommended in the RxImport documentation under 'Encoding Details'

Note

For reasons of performance, `rxTextToXdf` does not properly handle text files that contain the delimiter character inside a quoted string (for example, the entry `"Wade, John"` inside a comma delimited file. See `rxImport` with `type = "text"` for importing this type of data).

In addition `rxTextToXdf` currently requires that all rows of data in the text file contain the same number of entries. Date, time, and currency data types are not currently supported and are imported as character data.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxImport](#), [rxDataStep](#), [rxFactors](#), [rxTransform](#).

Examples

```
###  
# Example 1: Exploration of data types  
###  
  
# 1a) Target column types automatically determined  
# New .xdf file has one integer column, three character columns,  
# and two 32-bit single precision floating point columns  
sampleDataDir <- rxGetOption("sampleDataDir")
```

```

inputFile <- file.path(sampleDataDir, "claims.txt")
outputFile <- file.path(tempdir(), "basicClaims.xdf")
rxTextToXdf(inFile = inputFile, outFile = outputFile, overwrite = TRUE)
rxGetInfo(data = outputFile, getVarInfo = TRUE, numRows = 5)
file.remove(outputFile)

# 1b) Convert strings columns to factors
# Result is similar to 1a) with the character columns now being stored
# as factors
sampleDataDir <- rxGetOption("sampleDataDir")
inputFile <- file.path(sampleDataDir, "claims.txt")
outputFile <- file.path(tempdir(), "claimsWithFactors.xdf")
rxTextToXdf(inFile = inputFile, outFile = outputFile, stringsAsFactors = TRUE, overwrite = TRUE)
rxGetVarInfo(data = outputFile, getValueLabels = TRUE)
file.remove(outputFile)

# 1c) Interpret numeric columns as 64-bit doubles
# Result is similar to 1a) with the floating point columns now being
# stored as 64-bit double precision floating point columns
sampleDataDir <- rxGetOption("sampleDataDir")
inputFile <- file.path(sampleDataDir, "claims.txt")
outputFile <- file.path(tempdir(), "claimsWithDoubles.xdf")
rxTextToXdf(inFile = inputFile, outFile = outputFile,
            colClasses = c(cost = "numeric", number = "numeric"),
            overwrite = TRUE)
rxGetInfo(data = outputFile, getVarInfo = TRUE, numRows = 5)
file.remove(outputFile)

# 1d) Combination of 1b) and 1c)
# Result is similar to what is produced by utils::read.csv
sampleDataDir <- rxGetOption("sampleDataDir")
inputFile <- file.path(sampleDataDir, "claims.txt")
outputFile <- file.path(tempdir(), "claimsWithFactorsAndDoubles.xdf")

# Set options so that the default decimal type is numeric
myDefaultColType <- rxGetOption( "defaultDecimalColType" )
rxOptions(defaultDecimalColType = "numeric")
rxTextToXdf(inFile = inputFile, outFile = outputFile, stringsAsFactors = TRUE,
            overwrite = TRUE)
# Reset options to original settings
rxOptions( defaultDecimalColType = myDefaultColType )
rxGetInfo(data = outputFile, getVarInfo = TRUE, numRows = 5)
claimsWithFactorsAndDoubles <- rxDataStep(inData = outputFile, reportProgress = 0)

claimsFromReadCsv <- read.csv(inputFile)
levels(claimsFromReadCsv[["car.age"]])
claimsFromReadCsv[["car.age"]] <-
  factor(claimsFromReadCsv[["car.age"]],
         levels = c("0-3", "4-7", "8-9", "10+"))
all.equal(claimsWithFactorsAndDoubles, claimsFromReadCsv)
file.remove(outputFile)

# 1e) Build off of 1d), convert "number" column to a 16-bit integer
# After reading into R, results still similar to utils::read.csv output
sampleDataDir <- rxGetOption("sampleDataDir")
inputFile <- file.path(sampleDataDir, "claims.txt")
outputFile <- file.path(tempdir(), "claimsWithFactorsDoublesAnd16BitInt.xdf")
rxTextToXdf(inFile = inputFile, outFile = outputFile, stringsAsFactors = TRUE,
            colClasses =
              c(number = "int16", cost = "numeric", number = "numeric"),
            overwrite = TRUE)
rxGetInfo(data = outputFile, getVarInfo = TRUE, numRows = 5)
claimsWithFactorsDoublesAnd16BitInt <- rxDataStep(inData = outputFile, reportProgress = 0)
all.equal(claimsWithFactorsDoublesAnd16BitInt, claimsFromReadCsv)
file.remove(outputFile)

###  

# Example 2: Exploration of factor levels

```

```

###

# 2a) For column "type" convert all non "A" and "C" values to missing values
sampleDataDir <- rxGetOption("sampleDataDir")
inputFile <- file.path(sampleDataDir, "claims.txt")
outputFile <- file.path(tempdir(), "claimsFactorExploration.xdf")
rxTextToXdf(inFile = inputFile, outFile = outputFile,
            colInfo = list(type = list(type = "factor", levels = c("A", "C"))),
            overwrite = TRUE)
rxGetInfo(data = outputFile, getVarInfo = TRUE, numRows = 5)

# 2b) Append new data to .xdf file, where new data will extend the factor levels
appendFile <- file.path(sampleDataDir, "claimsExtra.txt")
rxTextToXdf(inFile = appendFile, outFile = outputFile,
            colInfo =
            list(type = list(type = "factor", levels = c("E", "F", "G"))),
            append = "rows")
rxGetInfo(data = outputFile, getVarInfo = TRUE, numRows = 5)
file.remove(outputFile)

# 2c) Reimport claims data, making car.age an ordered factor
# Then extract a data frame with cars more than 3 years old
sampleDataDir <- rxGetOption("sampleDataDir")
inputFile <- file.path(sampleDataDir, "claims.txt")
outputFile <- file.path(tempdir(), "claimsFactorExploration.xdf")
rxTextToXdf(inFile = inputFile, outFile = outputFile,
            colInfo = list(car.age = list(type = "ordered",
                                           levels = c("0-3", "4-7", "8-9", "10+"))),
            overwrite = TRUE)
oldCars <- rxDataStep(inData = outputFile, rowSelection = car.age > "0-3")

####
# Example 3: Exploring dynamic variable transformation and row selection
###

# Create local data source and write as a .csv file
tinyData <- data.frame(a = 1:4, b = c(4.1, 5.2, 6.3, 1.2),
                       c = c("one", "two", "three", "four"),
                       stringsAsFactors = FALSE)
tinyText <- file.path(tempdir(), "rxTiny.txt")
if (file.exists(tinyText)) file.remove(tinyText)
write.csv(tinyData, file = tinyText, row.names = FALSE)

# Create paths for .xdf file
tinyXdf <- file.path(tempdir(), "rxTiny.xdf")
if (file.exists(tinyXdf)) file.remove(tinyXdf)

# Create a transformation list.
# New variables : bSquared, bHalved, bIncremented
# Alter existing variables : a, c
# Delete existing variables : b
transforms <- expression(list(
  bSquared = b^2,
  bHalved = b / 2,
  bIncremented = b + 1,
  c = toupper(c),
  a = rev(a),
  b = NULL))

# Convert the existing text file to tinyXdf and transform variables along the way.
# Read the tinyXdf data into a data frame and compare to the original.
# Column "b" is explicitly typed as a 64-bit floating point number to show
# equality with the original data
rxTextToXdf(inFile = tinyText, outFile = tinyXdf, colClasses = c(b = "numeric"),
            firstRowIsColNames = TRUE, transforms = transforms)
tinyDataFromXdf <- rxDataStep(inData = tinyXdf, reportProgress = 0)

# Do a similar conversion but use a row selection expression to filter the rows.

```

```

" do a similar conversion but use a row selection expression to filter the rows.

# * The row selection expression is based on a variable that is derived from
#   column "b". In general, any column created using the transforms and
#   transformFunc arguments can be used in the row selection process.
# * Column "b" is explicitly typed as a 64-bit floating point number so one
#   of its derived variables can be compared with an R numeric value in the
#   row selection criterion.

rowSelection <- expression(bHalved > 2.05)
if (file.exists(tinyXdf)) file.remove(tinyXdf, firstRowIsColNames = TRUE,
  transforms = transforms, rowSelection = rowSelection)
tinyDataSubset <- rxDataStep(inData = tinyXdf, reportProgress = 0)
tinyDataSubset
file.remove(tinyText)
file.remove(tinyXdf)

####

# Example 4: Importing character data into a Date variable
####

# The following can be used in as.Date format strings:
# %b abbreviated month name
# %B full month name
# %m month as number (1-12)
# %d day of the month
# %y year, last two digits
# %Y year,

```

txtFile <- file.path(rxGetOption("unitTestDataDir"), "AirlineSampleDate.csv")
 xdfFile <- file.path(tempdir(), ".rxTemp.xdf")

```

# Import as character data
rxTextToXdf( inFile = txtFile, outFile = xdfFile, overwrite=TRUE)
rxDataStep(inData = xdfFile, numRows = 5)
# Example of Date variable is 1996-08-18

# Reimport as Date variable
rxTextToXdf( inFile = txtFile, outFile = xdfFile,
  transforms = list(Date = as.Date(Date, "%Y-%m-%d")),
  overwrite = TRUE)

# Look at variable information
rxGetVarInfo( data = xdfFile )

# Read first 10 rows of the file into a data frame
myData <- rxDataStep(inData = xdfFile, numRows = 10)
myData

# Clean up
file.remove( xdfFile )

```

rxTlcBridge: TLC Bridge

7/12/2022 • 3 minutes to read • [Edit Online](#)

Description

Bridge code for additional packages

Usage

```
rxTlcBridge(formula = NULL, data = NULL, outData = NULL,
            outDataFrame = FALSE, overwrite = FALSE,
            weightVar = NULL, groupVar = NULL, nameVar = NULL,
            customVar = NULL, analysisType = NULL,
            mamlCode = NULL, mamlTransformVars = NULL,
            modelInfo = NULL,
            testModel = FALSE, saveTextData = FALSE, saveBinaryData = FALSE,
            rowSelection = NULL, varsToKeep = NULL,
            transforms = NULL, transformObjects = NULL,
            transformFunc = NULL, transformVars = NULL,
            transformPackages = NULL, transformEnvir = NULL,
            rowsPerWrite = 500000,
            blocksPerRead = rxGetOption("blocksPerRead"),
            reportProgress = rxGetOption("reportProgress"), verbose = 1,
            computeContext = rxGetOption("computeContext"), ...)
```

Arguments

`formula`

formula as described in [rxFormula](#).

`data`

either a data source object, a character string specifying a .xdf file, a data frame object, or `NULL`.

`outData`

optional output data source.

`outDataFrame`

logical. If `TRUE`, a data frame is returned.

`overwrite`

logical. If `TRUE`, the output data source will be overwritten.

`weightVar`

character string or `NULL`. Optional name of weight variable.

`groupVar`

character string or `NULL`. Optional name of group variable.

`nameVar`

character string or `NULL`. Optional name of name variable.

`customVar`

character string or `NULL`. Optional name of custom variable.

`analysisType`

character string or `NULL`. Type of analysis being processed.

`mamlCode`

character string or `NULL`. Desired MAML code.

`mamlTransformVars`

character vector of variable names used in MAML transforms.

`modelInfo`

list or `NULL`. List containing additional model information.

`testModel`

If `TRUE`, the trained model is tested.

`saveTextData`

If `TRUE`, the data used in the model is saved to a text file.

`saveBinaryData`

If `TRUE`, the data used in the model is saved to a binary file.

`rowSelection`

name of a logical variable in the data set (in quotes) or a logical expression using variables in the data set to specify row selection. For example, `rowSelection = "old"` will use only observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` will use only observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

`varsToKeep`

`NULL` or character vector specifying the variables to keep when writing out the data set. Ignored if a model is specified.

`transforms`

an expression of the form `list(name = expression, ...)` representing the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

`transformObjects`

a named list containing objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

`transformFunc`

variable transformation function. See [rxTransform](#) for details.

`transformVars`

character vector of input data set variables needed for the transformation function. See [rxTransform](#) for details.

`transformPackages`

character vector defining additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, e.g., those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.

`transformEnvir`

user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

`rowsPerWrite`

Not implemented.

`blocksPerRead`

number of blocks to read for each chunk of data read from the data source.

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`verbose`

integer value. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information are provided.

`computeContext`

a valid **RxComputeContext**. The and `RxHadoopMR` compute context distributes the computation among the nodes specified by the compute context; for other compute contexts, the computation is distributed if possible on the local computer.

`...`

additional arguments to be processed or passed to the base computational function.

Details

This function does not provide direct user functionality.

Value

an `rxTlcBridge` object, or an output data source or data frame

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

rxTransform: Dynamic Variable Transformation Functions

7/12/2022 • 7 minutes to read • [Edit Online](#)

Description

Description of the recommended method for creating *on the fly* variables.

Details

Analytical functions within the **RevoScaleR** package use a formal transformation function framework for generating *on the fly* variables as opposed to the traditional R method of including transformations directly in a `formula` statement. The **RevoScaleR** approach is to use the following analytical function arguments to create new named variables that can be referenced by name within a `formula` statement:

- `transformFunc` : R function whose first argument and return value are named R lists with equal length vector elements. The output list can contain modifications to the input elements as well as newly named elements. R lists, instead of R data frames, are used to minimize function execution timings, implying that row name information will not be available during calculations.
- `transformVars` : character vector selecting the names of the variables to be represented as list elements for the input to `transformFunc`.

In application, the `transformFunc` argument is analogous to the `FUN` argument in R's `apply`, `lapply`, etc. functions. Just as with the `*apply` functions, the `transformFunc` function does not receive results from previous chunks as input and so an external mechanism is required to carry over results from one chunk to the next. Additional information variables are available for use within a transformation function when working in a local compute context:

- `.rxStartRow` : the row number from the original data set of the first row in the chunk of data that is being processed.
- `.rxChunkNum` : the chunk number of the data being processed, set to 0 if a test sample of data is being processed. For example, if there are 10 blocks of data in an .xdf file, and `blocksPerRead` is set to 2, there will be 5 chunks of data.
- `.rxNumRows` : the number of rows in the current chunk.
- `.rxReadFileName` : character string containing the name of the .xdf file being processed.
- `.rxIsTestChunk` : logical set to `TRUE` if the chunk of data being processed is a test sample of data.
- `.rxIsPrediction` : logical set to `TRUE` if the chunk of data being processed is being used in a prediction rather than a model estimation.
- `.rxTransformEnvir` : environment containing the data specified in `transformObjects`, which is also the parent environment to the transformation functions.

Three functions are also available for use within transformation functions that facilitate the interaction with `transformObjects` objects packed into transformation function environment (`.rxTransformEnvir`):

- `.rxGet(objName)` : Get the value of an object in the transform environment, e.g., `.rxGet("myObject")`.

- `.rxSet(objName, objValue)` : Set the value of an object in the transform environment, e.g.,
`.rxSet("myObject", pi)`.
- `.rxModify(objName, ..., FUN = "+")` : Modify the value of an object in the transform environment by applying to the current value a specified function, `FUN`. The first argument passed (behind the scenes) to `FUN` is always the current value of `"objName"`. The ellipsis argument (...) denotes additional arguments passed to `FUN`. By default, `FUN = "+"`, the addition operator. e.g.,
 - Increment the current value of "myObject" by 2: `.rxModify("myObject", 2)`
 - Decrement the current value of "myObject" by 4: `.rxModify("myObject", 4, FUN = "-")`
 - Convert "myObject", assumed to be a vector, to a factor with specified levels:
`.rxModify("myObject", levels = 1:5, exclude = 6:10, FUN = "factor")`

See Also

[rxFormula](#), [rxTransform](#), [rxCrossTabs](#), [rxCube](#), [rxLinMod](#), [rxLogit](#), [rxSummary](#), [rxDataStep](#), [rxImport](#).

Examples

```
# Develop a data.frame to serve as a local data source
set.seed(100)
N <- 100
feet <- round(runif(N, 4, 6))
inches <- floor(runif(N, 1, 12))
sportLevels <- c("tennis", "golf", "cycling", "running")
myDF <- data.frame(sex = sample(c("Male", "Female"), N, replace = TRUE),
                    age = round(runif(N, 15, 40)),
                    height = paste(feet, "ft", inches, "in", sep = ""),
                    pounds = rnorm(N, 140, 20),
                    sport = factor(sample(sportLevels, N, replace = TRUE),
                                  levels = sportLevels))

# Take a peek at the data source
head(myDF)

# Develop a transformation function
xform <- function(dataList)
{
  # Create a new continuous variable from an existing continuous variable:
  # convert weight in units of pounds to kilograms.
  dataList$kilos <- dataList$pounds * 0.45359237

  # Remove an existing variable: 'pounds'
  dataList$pounds <- NULL

  # Create a new continuous variable from an existing factor variable:
  # convert 'height' categories (character strings) to a continuous
  # variable named 'heightInches' containing the height in inches.
  heightStrings <- as.character(dataList$height)
  feet <- as.numeric(sub("ft.*", "", heightStrings))
  inches <- as.numeric(sub("in", "", sub(".*ft", "", heightStrings)))
  dataList$heightInches <- feet * 12 + inches

  # Alter an existing factor variable: for plotting purposes,
  # coerce the 'sport' factor levels to be in alphabetical order.
  dataList$sport <- factor(dataList$sport,
                            levels = sort(levels(dataList$sport)))

  # Return the adapted variable list
  dataList
}
```

```

# Perform a cube. Note the use of the 'kilos' variable in the formula, which
# is created in the variable transformation function.
cubeWeight <- rxCube(kilos ~ sport : sex, data = myDF, transformFunc = xform,
                      transformVars = c("pounds", "height", "sport"))

# Analyze the cube results
cubeWeight
if ("lattice" %in% .packages())
{
  barchart(kilos ~ sport | sex, data = cubeWeight, xlab = "Sport",
            ylab = "Mean Weight (kg)")
}

# Perform a cube with multiple dependent variables using a formula string
cubeHW <- rxCube(cbind(kilos, heightInches) ~ sport : sex, data = myDF,
                  transformFunc = xform,
                  transformVars = c("pounds", "height", "sport"))

cubeHW
if ("lattice" %in% .packages())
{
  barchart(heightInches + kilos ~ sex | sport, data = cubeHW, xlab = "Sex",
            ylab = "Mean Height (in) | Mean Weight (kg)")
}

# Use a transformation function to match external group data to
# individual observations

censusWorkers <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers.xdf")

# Create a function that creates a transformation function
makeTransformFunc <- function()
{
  # Put average per capita educational expenditures into a named vector
  educExp <- c(Connecticut=1795.57, Washington=1170.46, Indiana = 1289.66)

  function( dataList )
  {
    dataList$stateEducExpPC <- educExp[match(dataList$state, names(educExp))]
    return( dataList )
  }
}

linModObj <- rxLinMod(incwage~sex + age + stateEducExpPC, data=censusWorkers,
                      transformFun=makeTransformFunc(), transformVars="state")
summary(linModObj)

###  

# Illustrate the use of .rxGet, .rxSet and .rxModify  

###  

# This file has five blocks and so our data step will loop through the
# transformation function (defined below) six times: once as a test
# check performed by rxDataStep and once per block in the file.
# The test step is skipped if returnTransformObjects is TRUE

inFile <- file.path(rxGetOption("unitTestDataDir"), "test100r5b.xdf")

# Basic example
"myTestFun1" <- function(dataList)
{
  # If returnTransformObjects is TRUE, we won't get a test chunk (0)
  print(paste("Processing chunk", .rxChunkNum))

  numRows <- length(dataList[[1]])
  # Increment numRows by the number of rows in this chunk
  .rxSet("numRows", .rxGet("numRows") + numRows)

  # Increment numChunks by 1
}

```

```

# INCREMENT numChunks by 1
.rxModify("numChunks", 1L, FUN = "+")

# Multiply current count3 by twice its value
sumxnum1 <- sum(dataList$xnum1)
print(paste("Sum of chunk xnum1:", sumxnum1))
.rxModify("sumxnum1", sumxnum1, FUN = "+")

# Don't return any data, since we're just want transformObjects
return( NULL )
}

newValues <- rxDataStep( inData = inFile, transformFunc = myTestFun1,
    transformObjects = list(numRows = 0L, numChunks = 0L, sumxnum1 = 0L),
    returnTransformObjects = TRUE)
newValues

# More complicated example using transformEnv
"myFunction" <- function()
{
  count1 <- 100L

  "myTestFun" <- function(dataList)
  {
    if (!.rxIsTestChunk)
    {
      # Ladder update: this only updates local count1: better to use .rxSet or .rxModify
      # You should get a warning about two variables with the name 'count1'
      # Chunk 0: 100
      # Chunk 1: 101
      # Chunk 2: 102
      # Chunk 3: 103
      # Chunk 4: 104
      # Chunk 5: 105
      count1 <-> count1 + 1L
      print(count1)

      # Targeted update: increment count2
      .rxSet("count2", .rxGet("count2") + 1L)

      # Targeted update with .rxModify: increment count3
      .rxModify("count3", 1L, FUN = "+")

      # Targeted update with .rxModify: multiply current count4 by twice its value
      .rxModify("count4", 2L, FUN = "*")

      # Targeted update with .rxModify: permute second term of series, add nothing
      # Chunk 0: 1 2 3 4 5
      # Chunk 1: 1 3 4 5 2
      # Chunk 2: 1 4 5 2 3
      # Chunk 3: 1 5 2 3 4
      # Chunk 4: 1 2 3 4 5
      # Chunk 5: 1 3 4 5 2
      .rxModify("series", n = 2L, bias = 0L, FUN = "permute")
    }

    return(dataList)
  }

  myEnv <- new.env(hash = TRUE, parent = baseenv())
  assign("permute", function(x, n = 1L, bias = 10) c(x[-n], x[n]) + bias, envir = myEnv)
  rxDataStep( inData = inFile, transformFunc = myTestFun,
    transformObjects = list(count1 = 0L, count2 = 0L, count3 = 0L, count4 = 1L, series = 1:5),
    transformEnv = myEnv)

  myEnv
}

# Obtain the transform environment containing the results

```

```

transformEnvir <- myFunction()

# Check results
all.equal(transformEnvir[["count1"]], 0L)
all.equal(transformEnvir[["count2"]], 5L)
all.equal(transformEnvir[["count3"]], 5L)
all.equal(transformEnvir[["count4"]], 32L)
all.equal(transformEnvir[["series"]], c(1L, 3L, 4L, 5L, 2L))

#####
# Exclude computations for dependent variable when using rxPredict

myTransform <- function(dataList)
{
  if (!.rxIsPrediction)
  {
    dataList$SepalLengthLog <- log(dataList$Sepal.Length)
  }
  dataList$SepalWidthLog <- log(dataList$Sepal.Width)
  return( dataList )
}
linModOut <- rxLinMod(SepalLengthLog~SepalWidthLog, data = iris,
  transformFunc = myTransform, transformVars = c("Sepal.Length", "Sepal.Width"))

# Copy the iris data and remove Sepal.Length - used for the dependent variable
iris1 <- iris
iris1$Sepal.Length <- NULL
# Run a prediction using the smaller data set
predOut <- rxPredict(linModOut, data = iris1)

```

rxTweedie: Tweedie Generalized Linear Models

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Produces a dummy generalized linear model family object that can be used with `rxGlm` to fit Tweedie generalized linear regression models. This does NOT produce a full family object, but gives `rxGlm` enough information to call a C++ implementation that fits a Tweedie model. The excellent R package `tweedie` by Gordon Smyth does provide a full Tweedie family object, and that can also be used with `rxGlm`.

Usage

```
rxTweedie(var.power = 0, link.power = 1 - var.power)
```

Arguments

`var.power`

index of power variance function.

`link.power`

index of power link function. Setting `link.power` to `0` produces a `log` link function. Setting it to `1` is the identity link. The default is a canonical link equal to `1 - var.power`

Details

This provides a way to specify the `var.power` and the `link.power` arguments to the Tweedie distribution for `rxGlm`.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

Peter K Dunn (2011). `tweedie`: Tweedie exponential family models. R package version 2.1.1.

See Also

`rxGlm`

Examples

```
# In the claims data, the cost is set to NA if no claim was made
# Convert NA to 0 for the cost, and read data into a data frame

claimsXdf <- file.path(rxGetOption("sampleDataDir"), "claims.xdf")
claims <- rxDataStep(inData = claimsXdf,
                     transforms = list(cost = ifelse(is.na(cost), 0, cost)))

# Estimate using a Tweedie family
claimsTweedie <- rxGlm(cost ~ age + car.age + type ,
                        data=claims, family = rxTweedie(var.power = 1.5))
summary(claimsTweedie)

# Re-estimate using a Tweedie family setting link.power to 0,
# resulting in a log link function
claimsTweedie <- rxGlm(cost ~ age + car.age + type ,
                        data=claims, family = rxTweedie(var.power = 1.5, link.power = 0))
summary(claimsTweedie)
```

rxSerializeModel: RevoScaleR Model Serialization and Unserialization

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Serialize a **RevoScaleR/MicrosoftML** model in raw format to enable saving the model. This allows model to be loaded into SQL Server for real-time scoring.

Usage

```
rxSerializeModel(model, metadata = NULL, realtimeScoringOnly = FALSE, ...)  
  
rxUnserializeModel(serializedModel, ...)
```

Arguments

`model`

`RevoScaleR / MicrosoftML` model to be serialized

`metadata`

Arbitrary metadata of `raw` type to be stored with the serialized model. Metadata will be returned when unserialized.

`realtimeScoringOnly`

Drops fields not required for real-time scoring. NOTE: Setting this flag could reduce the model size but `rxUnserializeModel` can no longer retrieve the RevoScaleR model

`serializedModel`

Serialized model to be unserialized

Details

`rxSerializeModel` converts models into `raw` bytes to allow them to be saved and used for real-time scoring.

The following is the list of models that are currently supported in real-time scoring:

* **RevoScaleR**

* `rxLogit`

* `rxLinMod`

* `rxBTrees`

* `rxDTree`

* `rxDForest`

- * MicrosoftML
- * rxFastTrees
- * rxFastForest
- * rxLogisticRegression
- * rxOneClassSvm
- * rxNeuralNet
- * rxFastLinear

RevoScaleR models containing R transformations or transform based formula (e.g "A ~ log(B)") are not supported in real-time scoring. Such transforms to input data may need to be handled before calling real-time scoring.

`rxUnserializeModel` method is used to retrieve the original R model object and metadata from the serialized raw model.

Value

`rxSerializeModel` returns a serialized model.

`rxUnserializeModel` returns original R model object. If metadata is also present returns a list containing the original model object and metadata.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

Examples

```
myIris <- iris
myIris[1:5,]
form <- Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width + Species
irisLinMod <- rxLinMod(form, data = myIris)

# Serialize model for scoring
serializedModel <- rxSerializeModel(irisLinMod)

# Save model to file or SQL Server (use rxWriteObject)
# serialized model can now be used for real-time scoring

unserializedModel <- rxUnserializeModel(serializedModel)
```

rxWaitForJob: Wait for Distributed Job to Complete

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Causes R to block on an existing distributed job until completion.

Usage

```
rxWaitForJob(jobInfo)
```

Arguments

`jobInfo`

A `jobInfo` object.

Details

This function essentially changes a non-blocking distributed computing job to a blocking job. You can change a blocking job to non-blocking on Windows by pressing the Esc key, then using `rxGetJobStatus` and `rxGetJobResults` on the object `rxgLastPendingJob`. This function does not, however, modify the compute context.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

Examples

```
## Not run:  
  
rxWaitForJob( rxgLastPendingJob )  
rxWaitForJob( myNonWaitingJob )  
## End(Not run)
```

rxWriteObject: Manage R objects in ODBC Data Sources

7/12/2022 • 4 minutes to read • [Edit Online](#)

Description

Store/Retrieve R objects to/from ODBC data sources. The APIs are modelled after a simple key value store. These are generic APIs and if the ODBC data source isn't specified in the argument, the function does serialization or deserialization of the R object with the specified compression if any.

Usage

```
## S3 method for class `RxOdbcData':  
rxWriteObject (odbcDestination, key, value, version = NULL, keyName = "id", valueName = "value",  
versionName = "version", serialize = TRUE, overwrite = FALSE, compress = "gzip")  
## S3 method for class `default':  
rxWriteObject (value, serialize = TRUE, compress = "gzip")  
## S3 method for class `RxOdbcData':  
rxReadObject (odbcSource, key, version = NULL, keyName = "id", versionName = "version", valueName =  
"value", deserialize = TRUE, decompress = "gzip")  
## S3 method for class `raw':  
rxReadObject (rawSource, deserialize = TRUE, decompress = "gzip")  
## S3 method for class `RxOdbcData':  
rxDeleteObject (odbcSource, key, version = NULL, keyName = "id", valueName = "value", versionName =  
"version", all = FALSE)  
## S3 method for class `RxOdbcData':  
rxListKeys (odbcSource, key = NULL, version = NULL, keyName = "id", versionName = "version")
```

Arguments

`odbcDestination`

an RxOdbcData object identifying the destination to which the data is to be written.

`odbcSource`

an RxOdbcData object identifying the source from which the data is to be read.

..

`rawSource`

a raw R object to be read from.

`key`

a character string identifying the R object to be written or read. The intended use is for the key+version to be unique.

`value`

the R object being stored into the data source.

`version`

`NULL` or a character string which carries the version of the object. Combined with key identifies the object.

`keyName`

character string specifying the column name for the key in the underlying table.

`valueName`

character string specifying the column name for the objects in the underlying table.

`versionName`

character string specifying the column name for the version in the underlying table.

`serialize`

logical value. Dictates whether the object is to be serialized. Only raw values are supported if serialization is off.

`compress`

character string defining the compression algorithm to use for memCompress.

`deserialize`

logical value. Defines whether the object is to be de-serialized.

`decompress`

character string defining the compression algorithm to use for memDecompress.

`overwrite`

logical value. If `TRUE`, `rxWriteObject` first removes the key (or the key+version combination) before writing the new value. Even when `overwrite` is `FALSE`, `rxWriteObject` may still succeed if there is no database constraint (or index) enforcing uniqueness.

`all`

logical value. `TRUE` to remove all objects from the data source. If `TRUE`, the 'key' parameter is ignored.

Details

`rxWriteObject` stores an R object into the specified ODBC data destination. The R object to be written is identified by a key, and optionally, by a version (key+version). By default, the object is serialized and compressed. Returns `TRUE` if successful. If the ODBC data destination is not specified, the default implementation is dispatched which takes only the value of the R object with `serialize` and `compress` arguments.

`rxReadObject` loads an R object from the ODBC data source (or from a raw) decompressing and unserializing it (by default) in the process. Returns the R object. If the data source parameter defines a query, the `key` and the `version` parameters are ignored.

`rxDeleteObject` deletes an R object from the ODBC data source. If there are multiple objects identified by the key/version combination, all are deleted.

`rxListKeys` enumerates all keys or versions for a given key, depending on the parameters. When `key` is `NULL`, the function enumerates all unique keys in the table. Otherwise, it enumerates all versions for the given key. Returns a single column data frame.

The `key` and the `version` column should be of some SQL character type (`CHAR`, `VARCHAR`, `NVARCHAR`, etc) supported by the data source. The `value` column should be a binary type (`VARBINARY` for instance). Some conversions to other types might work, however, they are dependant on the ODBC driver and on the underlying package functions.

Value

`rxReadObject` returns an R object. `rxWriteObject` and `rxDeleteObject` return logical, `TRUE` on success.

`rxListKeys` returns a single column data frame containing strings.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

Examples

```
## Not run:

# Setup the connection string
conStr <- 'Driver={SQL Server};Server=localhost;Database=storedb;Trusted_Connection=true'

# Create the data source
ds <- RxOdbcData(table="robjects", connectionString=conStr)

# Re-create the table
if(rxSqlServerTableExists(ds$table, ds@connectionString)) {
  rxSqlServerDropTable(ds$table, ds@connectionString)
}

# An example table definition for SQL Server (no version)
ddl <- paste(" create table [", ds$table, "] (",
            "     [id] varchar(200) not null, ",
            "     [value] varbinary(max), ",
            "     constraint unique_id unique (id))",
            sep = "")

rxOpen(ds, "w")
rxExecuteSQLDDL(ds, ddl)
rxClose(ds)

# Fit a logit model
infertLogit <- rxLogit(case ~ age + parity + education + spontaneous + induced,
                        data = infert)

# Store the model in the database
rxWriteObject(ds, "logit.model", infertLogit)

# Load the model
infertLogit2 <- rxReadObject(ds, "logit.model")

all.equal(infertLogit, infertLogit2)
# TRUE

## End(Not run)
```

RxXdfData-class: Class RxXdfData

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Xdf data source connection class.

Generators

The targeted generator [RxXdfData](#) as well as the general generator [rxNewDataSource](#).

Extends

Class RxFileData, directly. Class RxDataSource, by class RxFileData.

Methods

`colnames`

`signature(x = "RxXdfData") : ...`

`dim`

`signature(x = "RxXdfData") : ...`

`dimnames`

`signature(x = "RxXdfData") : ...`

`formula`

`signature(x = "RxXdfData") : ...`

`length`

`signature(x = "RxXdfData") : ...`

`names`

`signature(x = "RxXdfData") : ...`

`names<-`

`signature(x = "RxXdfData") : ...`

`row.names`

`signature(x = "RxXdfData") : ...`

`show`

`signature(object = "RxXdfData") : ...`

`str`

`signature(object = "RxXdfData") : ...`

Author(s)

See Also

[RxDataSource-class](#), [RxXdfData](#), [rxNewDataSource](#)

Examples

```
DS <- RxXdfData(file.path(rxGetOption("sampleDataDir"), "fourthgraders.xdf"))

head(DS)
tail(DS)
names(DS)
dim(DS)
dimnames(DS)
nrow(DS)
ncol(DS)
str(DS)

# formula examples
formula(DS)
formula(DS, varsToDrop = "male")
formula(DS, depVar = "height")
formula(DS, depVar = "height", inter = list(c("male", "eyecolor")))

# summarize variables in data source
summary(DS)

# renaming variables in .xdf file via replacement method
XDF <- file.path(tempdir(), "iris.xdf")
rxDataStep(iris, XDF, overwrite = TRUE)
irisDS <- RxXdfData(XDF)
names(irisDS)
names(irisDS) <- c("cow", "horse", "chicken", "goat", "squirrel")
names(irisDS)
if (file.exists(XDF)) file.remove(XDF)
```

RxXdfData: Generate Xdf Data Source Object

7/12/2022 • 3 minutes to read • [Edit Online](#)

Description

This is the main generator for S4 class RxXdfData, which extends RxDataSource.

Usage

```
RxXdfData(file, varsToKeep = NULL, varsToDelete = NULL, returnDataFrame = TRUE,
           stringsAsFactors = FALSE, blocksPerRead = rxGetOption("blocksPerRead"),
           fileSystem = NULL, createCompositeSet = NULL, createPartitionSet = NULL,
           blocksPerCompositeFile = 3)

## S3 method for class `RxXdfData':
head  (x, n = 6L, reportProgress = 0L, ...)

## S3 method for class `RxXdfData':
summary (object, ...)

## S3 method for class `RxXdfData':
tail   (x, n = 6L, addrownums = TRUE, reportProgress = 0L, ...)
```

Arguments

`file`

character string specifying the location of the data. For single Xdf, it is a .xdf file. For composite Xdf, it is a directory like /tmp/airline. When using distributed compute contexts like `RxSpark`, a directory should be used since those compute contexts always use composite Xdf.

`varsToKeep`

character vector of variable names to keep around during operations. If `NULL`, argument is ignored. Cannot be used with `varsToDelete`.

`varsToDelete`

character vector of variable names to drop from operations. If `NULL`, argument is ignored. Cannot be used with `varsToKeep`.

`returnDataFrame`

logical indicating whether or not to convert the result to a data frame when reading with `rxReadNext`. If `FALSE`, a list is returned when reading with `rxReadNext`.

`stringsAsFactors`

logical indicating whether or not to convert strings into factors in R (for reader mode only). It currently has no effect.

`blocksPerRead`

number of blocks to read for each chunk of data read from the data source.

`fileSystem`

character string or `RxFileSystem` object indicating type of file system; "native" or `RxNativeFileSystem` object can be used for the local operating system, or an `RxHdfsFileSystem` object for the Hadoop file system. If `NULL`, the file system will be set to that in the current compute context, if available, otherwise the `fileSystem` option.

`createCompositeSet`

logical value or `NULL`. Used only when writing. If `TRUE`, a composite set of files will be created instead of a single .xdf file. Subdirectories data and metadata will be created. In the data subdirectory, the data will be split across a set of .xdfd files (see `blocksPerCompositeFile` below for determining how many blocks of data will be in each file). In the metadata subdirectory there is a single .xdfm file, which contains the meta data for all of the .xdfd files in the data subdirectory. When the compute context is `RxHadoopMR` or `RxSpark`, a composite set of files are always created.

`createPartitionSet`

logical value or `NULL`. Used only when writing. If `TRUE`, a set of files for partitioned Xdf will be created when assigning this `RxXdfData` object for `outData` of `rxPartition`. Subdirectories data and metadata will be created. In the data subdirectory, the data will be split across a set of .xdf files (each file stores data of a single data partition, see `rxPartition` for details). In the metadata subdirectory there is a single .xdfp file, which contains the meta data for all of the .xdf files in the data subdirectory. The partitioned Xdf object is currently supported only in `rxPartition` and `rxGetPartitions`.

`blocksPerCompositeFile`

integer value. If `createCompositeSet=TRUE`, and if the compute context is not `RxHadoopMR`, this will be the number of blocks put into each .xdfd file in the composite set. When importing is being done on Hadoop using MapReduce, the number of rows per .xdfd file is determined by the rows assigned to each MapReduce task, and the number of blocks per .xdfd file is therefore determined by `rowsPerRead`.

`x`

an `RxXdfData` object

`object`

an `RxXdfData` object

`n`

positive integer. Number of rows of the data set to extract.

`addrownums`

logical. If `TRUE`, row numbers will be created to match the original data set.

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`...`

arguments to be passed to underlying functions

Value

object of class RxXdfData.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxXdfData-class](#), [rxNewDataSource](#), [rxOpen](#), [rxReadNext](#).

Examples

```
myDataSource <- RxXdfData(file.path(rxGetOption("sampleDataDir"), "claims"))
# both of these should return TRUE
is(myDataSource, "RxXdfData")
is(myDataSource, "RxDataSource")

names(myDataSource)

modelFormula <- formula(myDataSource, depVars = "cost", varsToDrop = "RowNum")
```

rxXdfFileName-methods: Retrieve .xdf file name

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

Get the .xdf file path from a character string or RxXdfData object.

Usage

```
rxXdfFileName( x )
```

Arguments

x

character string containing the file name or an RxXdfData object.

Value

a character string containing the path and name of the .xdf file

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[RxXdfData](#)

Examples

```
# Create an RxXdfData object
ds <- RxXdfData(file.path(rxgetOption("sampleDataDir"), "claims.xdf"))
# Retrieve the file name with path
fileName <- rxXdfFileName(ds)
fileName
```

rxXdfToText: Export .xdf File to Delimited Text File

7/12/2022 • 3 minutes to read • [Edit Online](#)

Description

Write .xdf file content to a delimited text file. `rxDataStep` recommended.

Usage

```
rxXdfToText(inFile, outFile, varsToKeep = NULL,  
            varsToDrop = NULL, rowSelection = NULL,  
            transforms = NULL, transformObjects = NULL,  
            transformFunc = NULL, transformVars = NULL,  
            transformPackages = NULL, transformEnvir = NULL,  
            overwrite = FALSE, sep = ",", quote = TRUE, na = "NA",  
            eol = "\n", col.names = TRUE,  
            blocksPerRead = rxGetOption("blocksPerRead"),  
            reportProgress = rxGetOption("reportProgress"), ...)
```

Arguments

`inFile`

either an RxXdfData object or a character string specifying the input .xdf file.

`outFile`

character string specifying the output delimited text file.

`varsToKeep`

character vector of variable names to include when reading from `inFile`. If `NULL`, argument is ignored. Cannot be used with `varsToDrop`.

`varsToDrop`

character vector of variable names to exclude when reading from `inFile`. If `NULL`, argument is ignored. Cannot be used with `varsToKeep`.

`rowSelection`

name of a logical variable in the data set (in quotes) or a logical expression using variables in the data set to specify row selection. For example, `rowSelection = "old"` will use only observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` will use only observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

`transforms`

an expression of the form `list(name = expression, ...)` representing the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

`transformObjects`

a named list containing objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

`transformFunc`

variable transformation function. See [rxTransform](#) for details.

`transformVars`

character vector of input data set variables needed for the transformation function. See [rxTransform](#) for details.

`transformPackages`

character vector specifying additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, both those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments and those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.

`transformEnvir`

user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

`overwrite`

logical value. If `TRUE`, the existing `outFile` will be overwritten.

`sep`

character(s) specifying the separator between columns.

`quote`

logical value or numeric vector. If `TRUE`, any character or factor columns will be surrounded by double quotes. If a numeric vector, its elements are taken as the indices of columns to quote. In both cases, row and column names are quoted if they are written. If `FALSE`, nothing is quoted.

`na`

character string to use for missing values in the data.

`eol`

character(s) to print at the end of each line (row). For example, `eol = "\r\n"` will produce Windows' line endings on a Unix-alike OS, and `eol = "\r"` will produce files as expected by Mac OS Excel 2004.

`col.names`

a logical value indicating whether the column names in the `inFile` should be written as the first row in the output text file.

`blocksPerRead`

number of blocks to read for each chunk of data read from the data source.

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.

- 2 : rows processed and timings are reported.
- 3 : rows processed and all timings are reported.

...

additional arguments passed to the write.table function.

Details

For most purposes, the more general [rxDataStep](#) is preferred for writing to text files.

Value

An [RxTextData](#) object representing the output text file.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

See Also

[rxDataStep](#), [rxImport](#), [write.table](#), [rxSplit](#).

Examples

```
sampleDataDir <- rxGetOption("sampleDataDir")
censusXdfFile <- file.path(rxGetOption("sampleDataDir"), "CensusWorkers.xdf")

# Write all data in a text file
censusCsvFile <- file.path(tempdir(), "CensusWorkers.csv")
rxXdfToText(inFile = censusXdfFile, outFile = censusCsvFile, overwrite = TRUE)

# Alternatively, use rxDataStep
rxDataStep(inData = censusXdfFile, outFile = censusCsvFile, overwrite = TRUE)

# Write subset of transformed data in a text file
censusSubsetCsvFile <- file.path(tempdir(), "CensusSubset.csv")
rxXdfToText(inFile = censusXdfFile, outFile = censusSubsetCsvFile,
            varsToKeep = c("age", "incwage"),
            rowSelection = age < 40,
            transforms = list(logIncwage = log1p(incwage), incwage = NULL),
            overwrite = TRUE)

# Again, this can be done using rxDataStep
rxDataStep(inData = censusXdfFile, outFile = censusSubsetCsvFile,
            varsToKeep = c("age", "incwage"),
            rowSelection = age < 40,
            transforms = list(logIncwage = log1p(incwage), incwage = NULL),
            overwrite = TRUE)

# Clean-up
file.remove(censusCsvFile)
file.remove(censusSubsetCsvFile)
```

Comparison of Base R and RevoScaleR Functions

7/12/2022 • 2 minutes to read • [Edit Online](#)

This article provides a list of the functions provided by the **RevoScaleR** package and lists comparable functions included in the base distribution of R.

Data Input and Output

RX FUNCTION	DESCRIPTION	NEAREST BASE R FUNCTION
rxGetInfo	Retrieves header information from an .XDF file or summary information from a data frame	<code>str()</code> <code>names()</code> <code>colNames()</code>
rxGetVarInfo	Retrieves variable information from an .XDF file or data frame	<code>names()</code> <code>str()</code> <code>nrow()</code> <code>min()</code> <code>max()</code>
RxSasData	Creates a SAS data source object	<code>foreign::read.ssd()</code>
RxSpssData	Creates an SPSS data source object	<code>foreign::read.ssps()</code>
rxOpen	Opens a data source for reading	<code>read.table()</code> etc.
rxReadNext	Reads data from a data source	<code>read.table()</code> , etc.

Data Manipulation and Chunking

RX FUNCTION	DESCRIPTION	NEAREST BASE R FUNCTION
rxDataStep	Transforms and subsets data in .XDF files or data frames	<code>transform()</code> <code>with()</code> <code>within()</code> <code>subset()</code>
rxFactors	Recodes a factor variable, or converts a non-factor variable into a factor	<code>factor()</code>

RX FUNCTION	DESCRIPTION	NEAREST BASE R FUNCTION
rxSort	Performs multi-key sorting of the variables in an .XDF file or data frame	<code>sort()</code> <code>order()</code>
rxMerge	Merges two .XDF files or two data frames using a variety of merge types	<code>merge()</code> <code>rbind()</code> <code>cbind()</code>
rxSplit	Splits an .XDF file or a data frame into multiple .XDF files or data frames	<code>split()</code>

Descriptive Statistics and Cross-Tabulation

RX FUNCTION	DESCRIPTION	NEAREST BASE R FUNCTION
rxSummary	Generates summary statistics for a data frame, including computations by group	<code>summary()</code> <code>lapply(x, ...)</code>
rxQuantile	Computes approximate quantiles for an .XDF file or data frame without sorting	<code>quantile()</code>
rxCrossTabs	Creates a cross-tabulation of data based on a formula provided as parameter	<code>xtabs()</code>
rxCube	Creates a cross-tabulation of data based on formula provided as parameter This function is an alternative to <code>rxCrossTabs</code> and is designed for efficient representation.	<code>xtabs()</code>
rxMarginals	Creates a marginal summary for an <code>xtab</code> object	<code>addmargins()</code> <code>colSums()</code> <code>rowSums()</code>
as.crosstabs	Converts cross tabulation results to an <code>xtab</code> object	<code>xtabs()</code>
rxChiSquaredTest	Performs a chi-squared test on an <code>xtab</code> object	<code>chisq.test()</code>
rxFisherTest	Performs Fisher's Exact Test on an <code>xtab</code> object	<code>fisher.test()</code>

RX FUNCTION	DESCRIPTION	NEAREST BASE R FUNCTION
rxKendallCor	Computes Kendall's Tau Rank Correlation Coefficient using an xtab object	<code>cor(..., method="kendall")</code>

Statistical Modeling

RX FUNCTION	DESCRIPTION	NEAREST BASE R FUNCTION
rxLinMod	Fits a linear model to data	<code>lm()</code>
rxCovCor	Calculates the covariance, correlation, or sum of squares (cross-product) matrix for a set of variables	<code>cor()</code> <code>cov()</code> <code>crossprod()</code>
rxCov	Calculates the covariance matrix for a set of variables	<code>cov()</code>
rxCor	Calculates the correlation matrix for a set of variables	<code>cov()</code>
rxLogit	Fits a logistic regression model to data	<code>glm(..., family="binomial")</code>
rxGlm	Fits a generalized linear model to data	<code>glm()</code>
rxDTree	Fits a classification or regression tree to data	<code>tree::tree()</code> <code>rpart::rpart()</code>
rxPredict	Calculates predictions for fitted models	<code>predict()</code>
rxKmeans	Performs K-means clustering	<code>cluster::kmeans()</code>

Basic Graphing

RX FUNCTION	DESCRIPTION	NEAREST BASE R FUNCTION
rxHistogram	Creates a histogram from data	<code>hist()</code>
rxLinePlot	Creates a line plot from data	<code>plot()</code> <code>lines()</code>

See Also

[SQL Server R Services Features and Tasks](#)

RevoScaleR-defunct: Defunct functions in RevoScaleR

7/12/2022 • 2 minutes to read • [Edit Online](#)

Description

The functions or variables listed here are no longer part of RevoScaleR as they are no longer needed.

Usage

```
rxGetVarInfoXdf(file, getValueLabels = TRUE, varsToKeep = NULL,  
                 varsToDrop = NULL, computeInfo = FALSE)  
  
rxGetInfoXdf(file, getVarInfo = FALSE, getBlockSizes = FALSE,  
              getValueLabels = NULL, varsToKeep = NULL, varsToDrop = NULL,  
              startRow = 1, numRows = 0, computeInfo = FALSE, verbose = 0)
```

Arguments

`file`

either an RxXdfData object or a character string specifying the .xdf file. If a local compute context is being used, this argument may also be a list of data sources, in which case the output will be returned in a named list. See the details section for more information.

`getValueLabels`

logical value. If `TRUE`, value labels (including factor levels) are included in the output if present.

`getVarInfo`

logical value. If `TRUE`, variable information is returned.

`getBlockSizes`

logical value. If `TRUE`, block sizes are returned in the output for an .xdf file, and when printed the first 10 block sizes are shown.

`varsToKeep`

character vector of variable names for which information is returned. If `NULL`, argument is ignored. Cannot be used with `varsToDrop`.

`varsToDrop`

character vector of variable names for which information is not returned. If `NULL`, argument is ignored. Cannot be used with `varsToKeep`.

`startRow`

starting row for retrieval of data if a data frame or .xdf file.

`numRows`

number of rows of data to retrieve if a data frame or .xdf file.

`computeInfo`

logical value. If `TRUE`, variable information (e.g., high/low values) for non-xdf data sources will be computed by reading through the data set.

`verbose`

integer value. If `0`, no additional output is printed. If `1`, additional summary information is printed for an .xdf file.

Details

`rxGetVarInfo` should be used instead of `rxgetvarinfo`. `rxGetInfo` should be used instead of `rxGetInfoXdf`.

See Also

[rxGetVarInfo](#), [rxGetInfo](#), [RevoScaleR-deprecated](#).

RevoScaleR-deprecated: Deprecated functions in RevoScaleR

7/12/2022 • 24 minutes to read • [Edit Online](#)

Description

These functions are provided for compatibility with older versions of RevoScaleR only, and may be defunct as soon as the next release.

Usage

```
rxGetNodes(headNode, includeHeadNode = FALSE, makeRNodeNames = FALSE, getWorkersOnly=TRUE)
RxHpcServer(object, headNode = "", revoPath = NULL, shareDir = "", workingDir = NULL,
            dataPath = NULL, outDataPath = NULL, wait = TRUE, consoleOutput = FALSE,
            configFile = NULL, nodes = NULL, computeOnHeadNode = FALSE, minElems = -1, maxElems = -1,
            priority = 2, exclusive = FALSE, autoCleanup = TRUE, dataDistType = "all",
            packagesToLoad = NULL, email = NULL, resultsTimeout = 15, groups = "ComputeNodes"
            )

rxImportToXdf(inSource, outSource, rowSelection = NULL,
              transforms = NULL, transformObjects = NULL,
              transformFunc = NULL, transformVars = NULL,
              transformPackages = NULL, transformEnvir = NULL,
              append = "none", overwrite = FALSE, numRows = -1,
              maxRowsByCols = NULL,
              reportProgress = rxGetOption("reportProgress"),
              verbose = 0,
              xdfCompressionLevel = rxGetOption("xdfCompressionLevel"),
              createCompositeSet = NULL,
              blocksPerCompositeFile = 3
              )

rxDataStepXdf(inFile, outFile, varsToKeep = NULL, varsToDelete = NULL,
               rowSelection = NULL, transforms = NULL, transformObjects = NULL,
               transformFunc = NULL, transformVars = NULL,
               transformPackages = NULL, transformEnvir = NULL,
               append = "none", overwrite = FALSE, removeMissingsOnRead = FALSE, removeMissings = FALSE,
               computeLowHigh = TRUE, maxRowsByCols = 3000000,
               rowsPerRead = -1, startRow = 1, numRows = -1,
               startBlock = 1, numBlocks = -1, returnTransformObjects = FALSE,
               inSourceArgs = NULL, blocksPerRead = rxGetOption("blocksPerRead"),
               reportProgress = rxGetOption("reportProgress"),
               xdfCompressionLevel = rxGetOption("xdfCompressionLevel"),
               checkVarsToKeep = TRUE, cppInterp = NULL)

rxDataFrameToXdf(data, outFile, varsToKeep = NULL, varsToDelete = NULL,
                  rowVarName = NULL, append = "none", overwrite = FALSE,
                  computeLowHigh = TRUE,
                  xdfCompressionLevel = rxGetOption("xdfCompressionLevel"))

rxXdfToDataFrame(file, varsToKeep = NULL, varsToDelete = NULL, rowVarName = NULL,
                  rowSelection = NULL, transforms = NULL, transformObjects = NULL,
                  transformFunc = NULL, transformVars = NULL,
                  transformPackages = NULL, transformEnvir = NULL,
                  removeMissings = FALSE, stringsAsFactors = FALSE,
                  blocksPerRead = rxGetOption("blocksPerRead"),
                  maxRowsByCols = 3000000,
                  reportProgress = rxGetOption("reportProgress"),
                  cppInterp = NULL)
```

```

rxSortXdf(inFile, outFile, sortByVars, decreasing = FALSE,
          type = "auto", missingsLow = TRUE, caseSensitive = FALSE,
          removeDupKeys = FALSE, varsToKeep = NULL, varsToDrop = NULL,
          dupFreqVar = NULL, overwrite = FALSE, bufferLimit = -1,
          reportProgress = rxGetOption("reportProgress"),
          verbose = 0, xdfCompressionLevel = rxGetOption("xdfCompressionLevel"),
          blocksPerRead = -1, ...)

RxHadoopMR(object,
            hdfsShareDir = paste( "/user/RevoShare", Sys.info()[["user"]], sep="/" ),
            shareDir = paste( "/var/RevoShare", Sys.info()[["user"]], sep="/" ),
            clientShareDir = rxGetDefaultTmpDirByOS(),
            hadoopRPath = rxGetOption("unixRPath"),
            hadoopSwitches = "",
            revoPath = rxGetOption("unixRPath"),
            sshUsername = Sys.info()[["user"]],
            sshHostname = NULL,
            sshSwitches = "",
            sshProfileScript = NULL,
            sshClientDir = "",
            usingRunAsUserMode = FALSE,
            nameNode = rxGetOption("hdfsHost"),
            jobTrackerURL = NULL,
            port = rxGetOption("hdfsPort"),
            onClusterNode = NULL,
            wait = TRUE,
            consoleOutput = FALSE,
            showOutputWhileWaiting = TRUE,
            autoCleanup = TRUE,
            workingDir = NULL,
            dataPath = NULL,
            outDataPath = NULL,
            fileSystem = NULL,
            packagesToLoad = NULL,
            resultsTimeout = 15,
            ... )

```

Arguments

`inSource`

a non-RxXdfData RxDataSource object representing the input data source or a non-empty character string representing a file path. If a character string is supplied, the type of file is inferred from its extension, with the default being a text file.

`file`

either an RxXdfData object or a character string specifying the .xdf file.

`outSource`

an RxXdfData object or non-empty character string representing the output .xdf file.

`outFile`

a character string specifying the output .xdf file, an [RxXdfData](#) object, a [RxOdbcData](#) data source, or a [RxTeradata](#) data source. If `NULL`, a data frame will be returned from `rxDataStep` unless `returnTransformObjects` is set to `TRUE`. Setting `outFile` to `NULL` and `returnTransformObjects=TRUE` allows chunkwise computations on the data without modifying the existing data or creating a new data set. `outFile` can also be a delimited [RxTextData](#) data source if using a native file system and not appending.

`rowSelection`

name of a logical variable in the data set (in quotes) or a logical expression using variables in the data set to specify row selection. For example, `rowSelection = "old"` will use only observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` will use only observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

`transforms`

an expression of the form `list(name = expression, ...)` representing the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

`transformObjects`

a named list containing objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

`transformFunc`

variable transformation function. See [rxTransform](#) for details.

`transformVars`

character vector of input data set variables needed for the transformation function. See [rxTransform](#) for details.

`transformPackages`

character vector defining additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, e.g., those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.

`transformEnvir`

user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

`append`

either `"none"` to create a new .xdf file or `"rows"` to append rows to an existing .xdf file. If `outSource` exists and `append` is `"none"`, the `overwrite` argument must be set to `TRUE`.

`overwrite`

logical value. If `TRUE`, the existing `outSource` will be overwritten.

`numRows`

integer value specifying the maximum number of rows to import. If set to -1, all rows will be imported.

`maxRowsByCols`

the maximum size of a data frame that will be read in if `outData` is set to `NULL`, measured by the number of rows times the number of columns. If the number of rows times the number of columns being imported exceeds this, a warning will be reported and a smaller number of rows will be read in than requested. If `maxRowsByCols` is set to be too large, you may experience problems from loading a huge data frame into memory.

`reportProgress`

integer value with options:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`verbose`

integer value. If `0`, no additional output is printed. If `1`, information on the import type is printed.

`xdfCompressionLevel`

integer in the range of -1 to 9. The higher the value, the greater the amount of compression - resulting in smaller files but a longer time to create them. If `xdfCompressionLevel` is set to 0, there will be no compression and files will be compatible with the 6.0 release of Revolution R Enterprise. If set to -1, a default level of compression will be used.

`createCompositeSet`

logical value or `NULL`. If `TRUE`, a composite set of files will be created instead of a single .xdf file. A directory will be created whose name is the same as the .xdf file that would otherwise be created, but with no extension. Subdirectories data and metadata will be created. In the data subdirectory, the data will be split across a set of .xdff files (see `blocksPerCompositeFile` below for determining how many blocks of data will be in each file). In the metadata subdirectory there is a single .xdfm file, which contains the meta data for all of the .xdff files in the data subdirectory. When the compute context is `RxHadoopMR` a composite set of files is always created.

`blocksPerCompositeFile`

integer value. If `createCompositeSet=TRUE`, and if the compute context is not `RxHadoopMR`, this will be the number of blocks put into each .xdff file in the composite set. When importing is being done on Hadoop using MapReduce, the number of rows per .xdff file is determined by the rows assigned to each MapReduce task, and the number of blocks per .xdff file is therefore determined by `rowsPerRead`. If the `outSource` is an `RxXdfData` object, set the value for `blocksPerCompositeFile` there instead.

`inFile`

either an `RxXdfData` object or a character string specifying the input .xdf file.

`varsToKeep`

character vector of variable names to include when reading from the input data file. If `NULL`, argument is ignored. Cannot be used with `varsToDelete` or when `outFile` is the same as the input data file. Variables used in transformations or row selection will be retained even if not specified in `varsToKeep`. If `newName` is used in `colInfo` in a non-xdf data source, the `newName` should be used in `varsToKeep`. Not supported for `RxTeradata`, `RxOdbcData`, or `RxSqlServerData` data sources.

`varsToDelete`

character vector of variable names to exclude when reading from the input data file. If `NULL`, argument is ignored. Cannot be used with `varsToKeep` or when `outFile` is the same as the input data file. Variables used in transformations or row selection will be retained even if specified in `varsToDelete`. If `newName` is used in `colInfo` in a non-xdf data source, the `newName` should be used in `varsToDelete`. Not supported for `RxTeradata`, `RxOdbcData`, or `RxSqlServerData` data sources.

`removeMissingsOnRead`

logical value. If `TRUE`, rows with missing values will be removed on read.

`removeMissings`

logical value. If `TRUE`, rows with missing values will not be included in the output data.

`computeLowHigh`

logical value. If `FALSE`, low and high values will not automatically be computed. This should only be set to `FALSE` in special circumstances, such as when `append` is being used repeatedly. Ignored for data frames.

`rowsPerRead`

number of rows to read for each chunk of data read from the input data source. Use this argument for finer control of the number of rows per block in the output data source. If greater than 0, `blocksPerRead` is ignored. Cannot be used if `inFile` is the same as `outFile`. The default value of `-1` specifies that data should be read by blocks according to the `blocksPerRead` argument.

`startRow`

the starting row to read from the input data source. Cannot be used if `inFile` is the same as `outFile`.

`returnTransformObjects`

logical value. If `TRUE`, the list of `transformObjects` will be returned instead of a data frame or data source object. If the input `transformObjects` have been modified, by using `.rxSet` or `.rxModify` in the `transformFunc`, the updated values will be returned. Any data returned from the `transformFunc` is ignored. If no `transformObjects` are used, `NULL` is returned. This argument allows for user-defined computations within a `transformFunc` without creating new data. `returnTransformObjects` is not supported in distributed compute contexts such as [RxHadoopMR](#).

`startBlock`

starting block to read. Ignored if `startRow` is set to greater than 1.

`numBlocks`

number of blocks to read; all are read if set to -1. Ignored if `numRows` is not set to -1.

`blocksPerRead`

number of blocks to read for each chunk of data read from the data source. Ignored for data frames or if `rowsPerRead` is positive.

`inSourceArgs`

an optional list of arguments to be applied to the input data source.

`checkVarsToKeep`

logical value. If `TRUE` variable names specified in `varsToKeep` will be checked against variables in the data set to make sure they exist. An error will be reported if not found. Ignored if more than 500 variables in the data set.

`headNode`

A compute context (preferred), a `jobInfo` object, or (deprecated) a character scalar containing the name of the head node of a Microsoft HPC cluster.

`includeHeadNode`

logical scalar. Indicates whether to include the name of the head node in the list of returned nodes.

`makeRNodeNames`

Determines if the names of the nodes should be normalized for use as R variables. See also [rxMakeRNodeNames](#) for details on name mangling.

`getWorkersOnly`

logical. If `TRUE`, returns only those nodes within the cluster that are configured to actually execute jobs (where applicable).

`object`

object of class `RxHpcServer`. This argument is optional. If supplied, the values of the other specified arguments are used to replace those of `object` and the modified object is returned.

`revoPath`

character string specifying the path to the directory on the cluster nodes containing the files `R.exe` and `Rterm.exe`. The invocation of R on each node must be identical (this is an HPC constraint). See the Details section for more information regarding the path format.

`shareDir`

character string specifying the directory on the head node that is shared among all the nodes of the cluster and any client host. You must have permissions to read and write in this directory. See the Details section for more information regarding the path format.

`workingDir`

character string specifying a working directory for the processes on the cluster. If `NULL`, will default to the standard Windows user directory (that is, the value of the environment variable `USERPROFILE`).

`dataPath`

character vector defining the search path(s) for the data source(s). See the Details section for more information regarding the path format.

`outDataPath`

`NULL` or character vector defining the search path(s) for new output data file(s). If not `NULL`, this overrides any specification for `dataPath` in `rxOptions`

`wait`

logical value. If `TRUE`, the job will be blocking and will not return until it has completed or has failed. If `FALSE`, the job will be non-blocking return immediately, allowing you to continue running other R code. The object `rxgLastPendingJob` is created with the job information. You can pass this object to the `rxGetJobStatus` function to check on the processing status of the job. `rxWaitForJob` will change a non-waiting job to a waiting job. Conversely, pressing ESC changes a waiting job to a non-waiting job, provided that the HPC scheduler has accepted the job. If you press ESC before the job has been accepted, the job is canceled.

`consoleOutput`

logical scalar. If `TRUE`, causes the standard output of the R processes to be printed to the user console.

`configFile`

character string specifying the path to the XML template (on your local computer) that will be consumed by the job scheduler. If `NULL` (the default), the standard template is used. See the Details section for more information regarding the path format. We recommend that the user rely upon the default setting for `configFile`.

`nodes`

character string containing a comma-separated list of requested nodes, e.g., `"compute1,compute2,compute3"`, or a character vector like `c("compute1", "compute2", "compute3")`. If you specify the nodes to be used, `minElems` and `maxElems` are ignored if greater than the number of specified nodes. See the Details section for more information. The `nodes` parameter forces use of the specified nodes, rather than simply requesting them. This feature is provided so that jobs requiring only 10 out of 100 nodes, for example, do not force you to deploy your data to all 100 nodes. Should you need finer control, you will need to generate a new XML template through the

MS HPC job scheduler. See the Microsoft HPC Server 2008 documentation for details.

computeOnHeadNode

If `FALSE` and nodes are automatically being selected, the head node of the cluster will not be among the nodes to be used. Furthermore, if `FALSE` and the head node is explicitly specified in the node list, a warning is issued, but the job proceeds with a warning and the head node excluded from the node list. If `TRUE`, then the head node is treated exactly as any other node for the purpose of job resource allocations. Note that setting this flag to `TRUE` does not guarantee inclusion of the head node in a job; it simply allows the head node to be listed in an explicit list, or to be included by automatic node usage generation.

minElems

minimum number of nodes required for a run. If `-1`, the value for `maxElems` is used. If both `minElems` and `maxElems` are `-1`, the number of nodes specified in the `nodes` argument is used. See the Details section for more information.

maxElems

maximum number of nodes required for a run. If `-1`, the value for `minElems` is used. If both `minElems` and `maxElems` are `-1`, the number of nodes specified in the `nodes` argument is used. See the Details section for more information.

priority

The priority of the job. Allowable values are 0 (lowest), 1 (below normal), 2 (normal, the default), 3 (above normal), and 4 (highest). See the Microsoft HPC Server 2008 documentation for using job scheduling policy options and job templates to manage job priorities. These policy options may also affect the behavior of the `exclusive` argument.

exclusive

logical value. If `TRUE`, no other jobs can run on a compute node at the same time as this job. This may fail if you do not have administrative privileges on the cluster, and may also fail depending on the settings of your job scheduling policy options.

autoCleanup

logical scalar. If `TRUE`, the default behavior is to clean up the temporary computational artifacts and delete the result objects upon retrieval. If `FALSE`, then the computational results are not deleted, and the results may be acquired using `rxGetJobResults`, and the output via `rxGetJobOutput` until the `rxCleanupJobs` is used to delete the results and other artifacts. Leaving this flag set to `FALSE` can result in accumulation of compute artifacts which you may eventually need to delete before they fill up your hard drive. If you set `autoCleanup=TRUE` and experience performance degradation on a Windows XP client, consider setting `autoCleanup=FALSE`.

dataDistType

a character string denoting how the data has been distributed. Type `"all"` means that the entire data set has been copied to each compute node. Type `"split"` means that the data set has been partitioned and that each compute node contains a different set of rows.

packagesToLoad

optional character vector specifying additional packages to be loaded on the nodes when jobs are run in this compute context.

email

optional character string specifying an email address to which a job complete email should be sent. Note that the cluster administrator will have to enable email notifications for such job completion mails to be received.

resultsTimeout

A numeric value indicating for how long attempts should be made to retrieve results from the cluster. Under normal conditions, results are available immediately upon completion of the job. However, under certain high load conditions, the processes on the nodes have reported as completed, but the results have not been fully committed to disk by the operating system. Increase this parameter if results retrieval is failing on high load clusters.

groups

Optional character vector specifying the groups from which nodes should be selected. If `groups` is specified and `nodes` is `NULL`, all the nodes in the groups specified will be candidates for computations. If both `nodes` and `groups` are specified, the candidate nodes will be the intersection of the set of nodes explicitly specified in `nodes` and the set of all the nodes in the `groups` specified. Note that the default value of `"ComputeNodes"` is the name of the Microsoft defined group that includes all physically present nodes. Another Microsoft default group name used is `"HeadNodes"` which includes all nodes set up to act as head nodes. While these default names can be changed, this is not recommended. Note also that `rxGetNodeInfo` will honor group filtering. See the help for `rxGetNodeInfo` for more information.

data

data frame to put into .xdf file. See details section for a listing of the supported column types in the data frame.

rowVarName

character string or `NULL`. If `NULL`, the data frame's row names will be dropped. If a character string, an additional variable of that name will be added to the data set containing the data frame's row names.

stringsAsFactors

logical indicating whether or not to convert strings into factors in R.

sortByVars

character vector containing the names of the variables to use for sorting. If multiple variables are used, the first `sortByVars` variable is sorted and common values are grouped. The second variable is then sorted within the first variable groups. The third variable is then sorted within groupings formed by the first two variables, and so on.

decreasing

a logical scalar or vector defining the whether or not the `sortByVars` variables are to be sorted in decreasing or increasing order. If a vector, the length `decreasing` must be that of `sortByVars`. If a logical scalar, the value of `decreasing` is replicated to the length `sortByVars`.

type

a character string defining the sorting method to use. Type `"auto"` automatically determines the sort method based on the amount of memory required for sorting. If possible, all of the data will be sorted in memory. Type `"mergeSort"` uses a merge sort method, where chunks of data are pre-sorted, then merged together. Type `"varByVar"` uses a variable-by-variable sort method, which assumes that the `sortByVars` variables and the calculated sorted index variable can be held in memory simultaneously. If `type="varByVar"`, the variables in the sorted data are re-ordered so that the variables named in `sortByVars` come first, followed by any remaining variables.

missingsLow

a logical scalar for controlling the treatment of missing values. If `TRUE`, missing values in the data are treated as the lowest value; if `FALSE`, they are treated as the highest value.

caseSensitive

a logical scalar. If `TRUE`, case sensitive sorting is performed for character data.

`removeDupKeys`

logical scalar. If `TRUE`, only the first observation will be kept when duplicate values of the key (`sortByVars`) are encountered. The sort `type` must be set to `"auto"` or `"mergeSort"`.

`cppInterp`

NOT SUPPORTED information to be sent to the C++ interpreter.

`...`

additional arguments to be passed to the input data source.

`object`

object of class RxHadoopMR. This argument is optional. If supplied, the values of e other specified arguments are used to replace those of `object` and the modified object is turned.

`hdfsShareDir`

character string specifying the file sharing location within HDFS. You must ave permissions to read and write to this location.

`shareDir`

character string specifying the directory on the master (perhaps edge) node that is ared among all the nodes of the cluster and any client host. You must have permissions to read and write this directory.

`clientShareDir`

character string specifying the absolute path of the temporary directory on the client. faults to /tmp for POSIX-compliant non-Windows clients. For Windows and non-compliant POSIX clients, faults to the value of the TEMP environment variable if defined, else to the TMP environment variable defined, else to `NULL`. If the default directory does not exist, defaults to NULL. C paths (" \\host\dir ") are not supported.

`hadoopRPath`

character string specifying the path to the directory on the cluster mpute nodes containing the files R.exe and Rterm.exe. The invocation of R on each de must be identical.

`revoPath`

character string specifying the path to the directory on the master (perhaps edge) node ntaining the files R.exe and Rterm.exe.

`hadoopSwitches`

character string specifying optional generic Hadoop command line switches, r example `-conf myconf.xml` . e <http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/CommandsManual.html> r details on the Hadoop command line generic options.

`sshUsername`

character string specifying the username for making an ssh connection to the doop cluster. This is not needed if you are running your R client directly on the cluster. faults to the username of the user running the R client (that is, the value of `sys.info()[["user"]]`).

`sshHostname`

character string specifying the hostname or IP address of the Hadoop cluster de or edge node that the client will log into for launching Hadoop jobs and for copying files tween the client machine and the Hadoop cluster. Defaults to the hostname of the machine running e R client (that is, the value of `sys.info()[["nodename"]]`) This

field is only used if `odeonClusterNode` is `NULL` or `FALSE`. If you are using PuTTY on a Windows system, this can be the name of a saved PuTTY session that includes the user name and authentication file to use.

`sshSwitches`

character string specifying any switches needed for making an ssh connection to the doop cluster. This is not needed if one is running one's R client directly on the cluster.

`sshProfileScript`

Optional character string specifying the absolute path to a profile script that will exist on the `hHostname` host. This is used when the target ssh host does not automatically read in a `.bash_profile`, `.profile` or other shell environment configuration file for the definition of requisite variables such as `HADOOP_STREAMING`.

`sshClientDir`

character string specifying the Windows directory where Cygwin's `ssh.exe`, `scp.exe` or PuTTY's `plink.exe` and `pscp.exe` executables can be found. Needed only for Windows. It is needed if these executables are on the Windows Path or if Cygwin's location can be found in the Windows Registry. Defaults to the empty string.

`usingRunAsUserMode`

logical scalar specifying whether run-as-user mode is being used on the Hadoop cluster. When using run-as-user mode, local R processes started by the map-reduce framework will run as the same user that started the job, and will have any allocated local permissions. When not using run-as-user mode (the default for many Hadoop systems), all R processes will run as user `mapred`. Note that when running as user `mapred`, permissions for files and directories will be more open in order to allow hand-offs between the user and `mapred`. Run-as-user mode is controlled for the Hadoop map-reduce framework by the xml setting, `odemapped.task.tracker.task-controller`, in the `mapred-site.xml` configuration file. If it is set to the value

`org.apache.hadoop.mapred.LinuxTaskController`, then run-as-user mode is in use. If it is set to the value

`org.apache.hadoop.mapred.DefaultTaskController`, then run-as-user mode is not in use.

`nameNode`

character string specifying the Hadoop name node hostname or IP address. Typically you can leave this at its default value, set to a value other than "default" or the empty string (see below), or it must be an address that can be resolved by the data nodes and used by them to contact the master node. Depending on your cluster, it may need to be set to a private network address such as `"master.local"`. If set to the empty string, "", then the master process will set it to the name of the node on which it is running, as returned by `sys.info()[[["nodename"]]]`. It is likely to work when the `sshHostname` points to the name node or the `sshHostname` is not specified and the R client is running on the name node. Defaults to `rxGetOption("hdfsHost")`.

`jobTrackerURL`

character scalar specifying the full URL for the jobtracker web interface. It is used only for the purpose of loading the job tracker web page from the `rxLaunchClusterJobManager` convenience function. It is never used for job control, and its specification in the compute context is completely optional. See the [rxLaunchClusterJobManager](#) page for more information.

`port`

numeric scalar specifying the port used by the name node for hadoop jobs. Needs to be able to be cast to an integer. Defaults to `rxGetOption("hdfsPort")`.

`onClusterNode`

logical scalar or `NULL` specifying whether the user is initiating the job from a client that will connect to the edge node or an actual cluster node, directly from either an edge node or node within the cluster. If set to `odeFALSE` or `NULL`, then `sshHostname` must be a valid host.

`wait`

logical value. If `TRUE`, the job will be blocking and will not return until it has completed or has failed. If `FALSE`, the job will be non-blocking and will return immediately, allowing you to continue running other R code. The object `rxgLastPendingJob` is created with the job information. You can pass this object to the `oderxGetJobStatus` function to check on the processing status of the job. `oderxWaitForJob` will change a non-waiting job a waiting job. Conversely, pressing ESC changes a waiting job to a non-waiting job, avoiding that the HPC scheduler has accepted the job. If you press ESC before the job has been accepted, the job is canceled.

`consoleOutput`

logical scalar. If `TRUE`, causes the standard output the R processes to be printed to the user console.

`showOutputWhileWaiting`

logical scalar. If `TRUE`, causes the standard output the remote primary R and hadoop job process to be printed to the user console while waiting for (blocking on) job.

`autoCleanup`

logical scalar. If `TRUE`, the default behavior is to clean up the temporary computational artifacts and delete the result objects upon retrieval. If `FALSE`, then the computational results are not deleted, and the results may be acquired using `oderxGetJobResults`, and the output via `rxGetJobOutput` until the `oderxCleanupJobs` is used to delete the results and other artifacts. Leaving this argument set to `FALSE` can result in accumulation of compute artifacts which you may eventually need to delete before they fill up your hard drive.

`workingDir`

character string specifying a working directory for the processes the master node.

`dataPath`

NOT YET IMPLEMENTED. character vector defining the search path(s) for the data source(s).

`outDataPath`

NOT YET IMPLEMENTED. `NULL` or character vector defining the search path(s) for new output data file(s). If not `NULL`, this overrides any specification for `outDataPath` in `rxOptions`.

`fileSystem`

`NULL` or an `RxHdfsFileSystem` to use as the default file system for data sources when created when this compute context is active.

`packagesToLoad`

optional character vector specifying additional packages to be loaded on the nodes when jobs are run in this compute context.

`resultsTimeout`

A numeric value indicating for how long attempts should be made to retrieve results from the cluster. Under normal conditions, results are available immediately. However, under certain high load conditions, the processes on the nodes have reported as completed, but the results have not been fully committed to disk by the operating system. Increase this parameter if results retrieval is failing on high load clusters.

`...`

additional arguments to be passed directly to the Microsoft R Services Compute Engine.

Details

Use `rxImport` instead of `rxImportToXdf`. Use `rxDataStep` instead of `rxDataStepXdf`. Use `rxDataStep` instead of `rxDataFrameToXdf`. Use `rxDataStep` instead of `rxXdfToDataFrame`. Use `rxSort` instead of `rxSortXdf`. Use

`rxGetAvailableNodes` instead of `rxGetNodes`. Use `rxSparkConnect` instead of `RxHadoopMR`.

Value

For `rxSortXdf` : If sorting is successful, `TRUE` is returned; otherwise `FALSE` is returned.

See Also

[rxImport](#), [rxDataStep](#), [rxSparkConnect](#), "RevoScaleR-defunct".

RevoScaleR Functions for Spark on Hadoop

7/12/2022 • 5 minutes to read • [Edit Online](#)

The [RevoScaleR package](#) provides a set of portable, scalable, distributable data analysis functions. This page presents a curated list of functions that might be particularly interesting to Hadoop users. These functions can be called directly from the command line.

The RevoScaleR package supports two Hadoop compute contexts:

- RxSpark (recommended), a distributed compute context in which computations are parallelized and distributed across the nodes of a Hadoop cluster via Apache Spark. This provides up to a 7x performance boost compared to [RxHadoopMR](#). For guidance, see [How to use RevoScaleR on Spark](#).
- RxHadoopMR (deprecated), a distributed compute context on a Hadoop cluster. This compute context can be used on a node (including an edge node) of a Cloudera or Hortonworks cluster with a RHEL operating system, or a client with an SSH connection to such a cluster. For guidance, see [How to use RevoScaleR on Hadoop MapReduce](#).

On Hadoop Distributed File System (HDFS), the XDF file format stores data in a composite set of files rather than a single file.

Data Analysis Functions

Import and Export Functions

FUNCTION NAME		DESCRIPTION	HELP
rxDataStep		Transform and subset data. Creates an .xdf file, a comma-delimited text file, or data frame in memory (assuming you have sufficient memory to hold the output data) from an .xdf file or a data frame.	View
RxXdfData		Creates an efficient XDF data source object.	View
RxTextData		Creates a comma delimited text data source object.	View
rxGetInfo		Retrieves summary information from a data source or data frame.	View
rxGetVarInfo		Retrieves variable information from a data source or data frame.	View

<code>rxGetVarNames</code>		Retrieves variable names from a data source or data frame.	View
<code>rxHdfsFileSystem</code>		Creates an HDFS file system object.	View

Manipulation, Cleansing, and Transformation Functions

FUNCTION NAME		DESCRIPTION	HELP
<code>rxDataStep</code>		Transform and subset data. Creates an .xdf file, a comma-delimited text file, or data frame in memory (assuming you have sufficient memory to hold the output) from an .xdf file or a data frame.	View
<code>rxFactors</code>		Create or recode factor variables in a composite XDF file in HDFS. A new file must be written out.	View

Analysis Functions for Descriptive Statistics and Cross-Tabulations

FUNCTION NAME		DESCRIPTION	HELP
<code>rxQuantile</code>		Computes approximate quantiles for .xdf files and data frames without sorting.	View
<code>rxSummary</code>		Basic summary statistics of data, including computations by group. Writing by group computations to .xdf file not supported.	View
<code>rxCrossTabs</code>		Formula-based cross-tabulation of data.	View
<code>rxCube</code>		Alternative formula-based cross-tabulation designed for efficient representation returning 'cube' results. Writing output to .xdf file not supported.	View

Analysis, Learning, and Prediction Functions for Statistical Modeling

FUNCTION NAME		DESCRIPTION	HELP

<code>rxLinMod</code>		Fits a linear model to data.	View
<code>rxLogit</code>		Fits a logistic regression model to data.	View
<code>rxGlm</code>		Fits a generalized linear model to data.	View
<code>rxCovCor</code>		Calculate the covariance, correlation, or sum of squares / cross-product matrix for a set of variables.	View
<code>rxDTree</code>		Fits a classification or regression tree to data.	View
<code>rxBTrees</code>		Fits a classification or regression decision forest to data using a stochastic gradient boosting algorithm.	View
<code>rxDForest</code>		Fits a classification or regression decision forest to data.	View
<code>rxPredict</code>		Calculates predictions for fitted models. Output must be an XDF data source.	View
<code>rxKmeans</code>		Performs k-means clustering.	View
<code>rxNaiveBayes</code>		Fit Naive Bayes Classifiers on an .xdf file or data frame for small or large data using parallel external memory algorithm.	View

Compute Context Functions

FUNCTION NAME		DESCRIPTION	HELP
<code>RxHadoopMR</code>		Creates an in-data, file-based Hadoop compute context.	View
<code>RxSpark</code>		Creates an in-data, file-based Spark compute context. Computations are parallelized and distributed across the nodes of a Hadoop cluster via Apache Spark.	View

<code>rxSparkConnect</code>		Creates a persistent Spark compute context.	View
<code>rxSparkDisconnect</code>		Disconnects a Spark session and return to a local compute context.	View
<code>rxInstalledPackages</code>		Returns the list of installed packages for a compute context.	View
<code>rxFindPackage</code>		Returns the path to one or more packages for a compute context.	View

Data Source Functions

Of course, not all data source types are available on all compute contexts. For the Hadoop compute contexts, two types of data sources can be used.

FUNCTION NAME		DESCRIPTION	HELP
<code>RxXdfData</code>		Creates an efficient XDF data source object.	View
<code>RxTextData</code>		Creates a comma delimited text data source object.	View
<code>RxHiveData</code>		Generates a Hive Data Source object.	View
<code>RxParquetData</code>		Generates a Parquet Data Source object.	View
<code>rxSparkDataOps</code>		Lists cached <code>RxParquetData</code> or <code>RxHiveData</code> data source objects.	View
<code>rxSparkRemoveData</code>		Removes cached <code>RxParquetData</code> or <code>RxHiveData</code> data source objects.	View

High Performance Computing and Distributed Computing Functions

The Hadoop compute context has a number of helpful functions used for high performance computing and distributed computing. Learn more about the entire set of functions in the [Distributed Computing guide](#).

FUNCTION NAME		DESCRIPTION	HELP
---------------	--	-------------	------

<code>rxExec</code>	Run an arbitrary R function on nodes or cores of a cluster.	View
<code>rxGetJobStatus</code>	Get the status of a non-waiting distributed computing job.	View
<code>rxGetJobResults</code>	Get the return object(s) of a non-waiting distributed computing job.	View
<code>rxGetJobOutput</code>	Get the console output from a non-waiting distributed computing job.	View
<code>rxGetJobs</code>	Get the available distributed computing job information objects.	View

Hadoop Convenience Functions

RevoScaleR also provides some wrapper functions for accessing Hadoop/HDFS functionality via R. These functions require access to Hadoop, either locally or remotely via the RxHadoopMR or RxSpark compute contexts.

FUNCTION NAME	DESCRIPTION	HELP
<code>rxHadoopCommand</code>	Execute an arbitrary Hadoop command. Allows you to run basic Hadoop commands.	View
<code>rxHadoopVersion</code>	Return the current Hadoop version.	View
<code>rxHadoopCopyFromClient</code>	Copy a file from a remote client to the Hadoop cluster's local file system, and then to HDFS.	View
<code>rxHadoopCopyFromLocal</code>	Copy a file from the native file system to HDFS. Wraps the Hadoop <code>fs -copyFromLocal</code> command.	View
<code>rxHadoopCopy</code>	Copy a file in the Hadoop Distributed File System (HDFS). Wraps the Hadoop <code>fs -cp</code> command.	View
<code>rxHadoopRemove</code>	Remove a file in HDFS. Wraps the Hadoop <code>fs -rm</code> command.	View

<code>rxHadoopListFiles</code>	List files in an HDFS directory. Wraps the Hadoop <code>fs -ls</code> or <code>fs -lsr</code> command.	View
<code>rxHadoopMakeDir</code>	Make a directory in HDFS. Wraps the Hadoop <code>fs -mkdir</code> command.	View
<code>rxHadoopMove</code>	Move a file in HDFS. Wraps the Hadoop <code>fs -mv</code> command.	View
<code>rxHadoopRemoveDir</code>	Remove a directory in HDFS. Wraps the Hadoop <code>fs -rmdir</code> command.	View

RevoScaleR Functions for a SQL Server compute context

7/12/2022 • 2 minutes to read • [Edit Online](#)

This article provides an overview of the main RevoScaleR functions for use with SQL Server, along with comments on their syntax.

Functions for working with SQL Server Data Sources

The following functions let you define a SQL Server data source. A data source object is a container that specifies a connection string together with the set of data that you want, defined either as a table, view, or query. Stored procedure calls are not supported.

In addition to defining a data source, you can execute DDL statements from R, if you have the necessary permissions on the instance and database.

- [RxSqlServerData](#) - Define a SQL Server data source object
- [rxSqlServerDropTable](#) - Drop a SQL Server table
- [rxSqlServerTableExists](#) - Check for the existence of a database table or object
- [rxExecuteSQLDDL](#) - Execute a command to define, manipulate, or control SQL data, but not return data

Functions for Defining or Managing a Compute Context

The following functions let you define a new compute context, switch compute contexts, or identify the current compute context.

- [RxComputeContext](#) - Create a compute context.
- [rxInSqlServer](#) - Generate a SQL Server compute context that lets RevoScaleR functions run in SQL Server R Services.
- [rxGetComputeContext](#) - Get the current compute context.
- [rxSetComputeContext](#) - Specify which compute context to use. The local compute context is available by default, or you can specify the keyword **local**.

Functions for Using a Data Source

After you have created a data source object, you can open it to get data, or write new data to it. Depending on the size of the data in the source, you can also define the batch size as part of the data source and move data in chunks.

- [rxIsOpen](#) - Check whether a data source is available
- [rxOpen](#) - Open a data source for reading
- [rxReadNext](#) - Read data from a source
- [rxWriteNext](#) - Write data to the target
- [rxClose](#) - Close a data source

Functions that work with XDF Files

The following functions can be used to create a local data cache in the XDF format. This file can be useful when working with more data than can be transferred from the database in one batch, or more data than can fit in

memory.

If you regularly move large amounts of data from a database to a local workstation, rather than query the database repeatedly for each R operation, you can use the XDF file to save the data locally and then work with it in your R workspace, using the XDF file as the cache.

- `rxImport` - Move data from an ODBC source to the XDF file
- `RxXdfData` - Create an XDF data object
- `RxDataStep` - Read data from XDF int a data frame

See Also

[Comparison of RevoScaleR and CRAN R Functions](#)

RevoScaleR Functions for Teradata

7/12/2022 • 3 minutes to read • [Edit Online](#)

Microsoft R Server 9.1 was the last release to include the RxInTeradata and RxInTeradata-class functions for creating a remote compute context on a Teradata appliance. This release is fully supported as described in the [service support policy](#), but the Teradata compute context is not available in Machine Learning Server 9.2.1 and later.

Data import using the RxTeradata data source object remains unchanged. If you have existing data in a Teradata appliance, you can create an [RxTeradata object](#) to import your data into Machine Learning Server.

For backwards compatibility purposes, this page presents a curated list of functions specific to the Teradata DB compute contexts, as well as those that may not be fully supported.

These functions can be called directly from the command line. For guidance on using these functions, see the [RevoScaleR Teradata DB Getting Started Guide](#).

Data Analysis Functions

Import and Export Functions

FUNCTION NAME		DESCRIPTION	HELP
<code>rxDataStep</code>		Transform and subset data. Creates an .xdf file, a comma-delimited text file, or data frame in memory (assuming you have sufficient memory to hold the output data) from an .xdf file or a data frame.	View
<code>RxXdfData</code>		Creates an efficient XDF data source object.	View
<code>RxTextData</code>		Creates a comma delimited text data source object.	View
<code>rxGetInfo</code>		Retrieves summary information from a data source or data frame.	View
<code>rxGetVarInfo</code>		Retrieves variable information from a data source or data frame.	View
<code>rxGetVarNames</code>		Retrieves variable names from a data source or data frame.	View

Manipulation, Cleansing, and Transformation Functions

FUNCTION NAME		DESCRIPTION	HELP
<code>rxDataStep</code>		Transform and subset data. Creates an .xdf file, a comma-delimited text file, or data frame in memory (assuming you have sufficient memory to hold the output) from an .xdf file or a data frame.	View
<code>rxFactors</code>		Create or recode factor variables in a composite XDF file in HDFS. A new file must be written out.	View

Analysis Functions for Descriptive Statistics and Cross-Tabulations

FUNCTION NAME		DESCRIPTION	HELP
<code>rxQuantile</code>		Computes approximate quantiles for .xdf files and data frames without sorting.	View
<code>rxSummary</code>		Basic summary statistics of data, including computations by group. Writing by group computations to .xdf file not supported.	View
<code>rxCrossTabs</code>		Formula-based cross-tabulation of data.	View
<code>rxCube</code>		Alternative formula-based cross-tabulation designed for efficient representation returning 'cube' results. Writing output to .xdf file not supported.	View

Analysis, Learning, and Prediction Functions for Statistical Modeling

FUNCTION NAME		DESCRIPTION	HELP
<code>rxLinMod</code>		Fits a linear model to data.	View
<code>rxLogit</code>		Fits a logistic regression model to data.	View
<code>rxGlm</code>		Fits a generalized linear model to data.	View

<code>rxCovCor</code>		Calculate the covariance, correlation, or sum of squares / cross-product matrix for a set of variables.	View
<code>rxDTree</code>		Fits a classification or regression tree to data.	View
<code>rxBTrees</code>		Fits a classification or regression decision forest to data using a stochastic gradient boosting algorithm.	View
<code>rxForest</code>		Fits a classification or regression decision forest to data.	View
<code>rxPredict</code>		Calculates predictions for fitted models. Output must be an XDF data source.	View
<code>rxKmeans</code>		Performs k-means clustering.	View

Compute Context Functions

FUNCTION NAME		DESCRIPTION	HELP
<code>RxTeradata</code>		Creates a Teradata data source object.	View
<code>RxInTeradata</code>		Creates an in-database compute context for Teradata.	View
<code>rxSetComputeContext</code>		Sets a compute context.	View
<code>rxGetComputeContext</code>		Gets the current compute context.	View
<code>rxInstalledPackages</code>		Returns the list of installed packages for a compute context.	View
<code>rxFindPackage</code>		Returns the path to one or more packages for a compute context.	View

Data Source Functions

Of course, not all data source types are available on all compute contexts.

FUNCTION NAME		DESCRIPTION	HELP
<code>rxTeradataTableExists</code>		Check for the existence of a database table or object.	View
<code>rxTeradataDropTable</code>		Drop a table.	View
<code>RxXdfData</code>		Creates an efficient XDF data source object.	View
<code>RxTextData</code>		Creates a comma delimited text data source object.	View

High Performance Computing and Distributed Computing Functions

The Teradata compute context has a number of helpful functions used for high performance computing and distributed computing. Learn more about the entire set of functions in the [Distributed Computing guide](#).

FUNCTION NAME		DESCRIPTION	HELP
<code>rxExec</code>		Run an arbitrary R function on nodes or cores of a cluster.	View
<code>rxGetJobStatus</code>		Get the status of a non-waiting distributed computing job.	View)
<code>rxGetJobResults</code>		Get the return object(s) of a non-waiting distributed computing job.	View
<code>rxGetJobOutput</code>		Get the console output from a non-waiting distributed computing job.	View
<code>rxGetJobs</code>		Get the available distributed computing job information objects.	View
<code>rxGetAvailableNodes</code>		Get all the available nodes on a distributed compute context.	View
<code>rxgetNodeInfo</code>		Get information on nodes specified for a distributed compute context.	View
<code>rxPingNodes</code>		Test round trip from user through computation node(s) in a cluster or cloud.	View

Support Timeline for Machine Learning Server & Microsoft R Server

7/12/2022 • 2 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Machine Learning Server 9.4.7 is supported until July 1, 2022. All other versions are no longer supported.

For more information, see [What's happening to Machine Learning Server?](#)

At the end of a version's life cycle, the software is removed from the Microsoft download sites and obsolete feature documentation is archived.

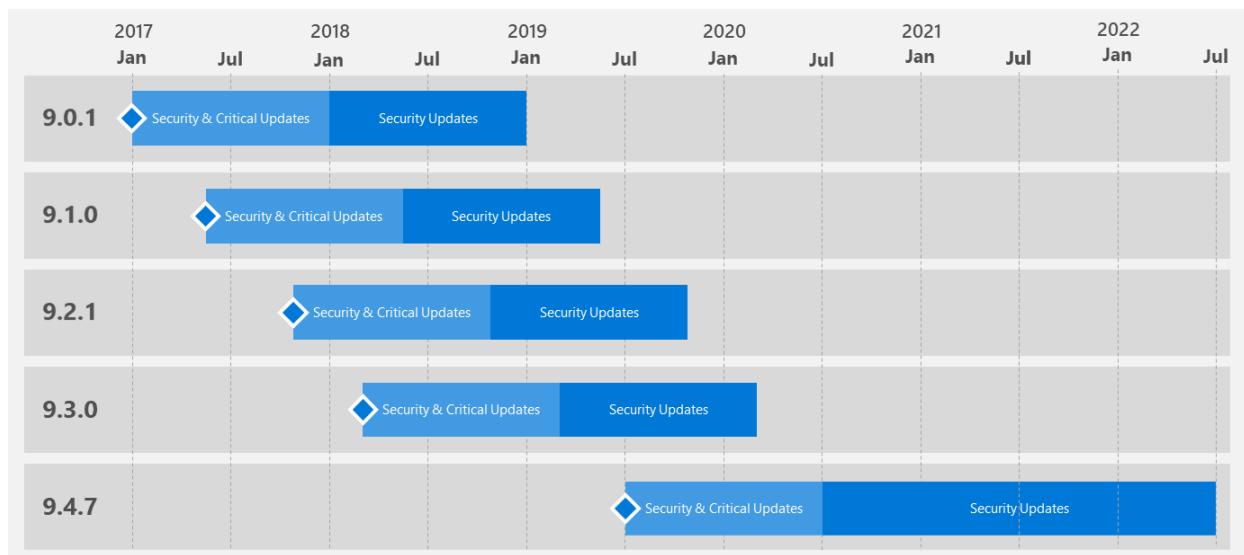


Figure 1. Example of servicing support

VERSION HISTORY	AVAILABILITY	SUPPORT END DATE
Machine Learning Server 9.4.7	7/30/2019	7/1/2022
Machine Learning Server 9.3.0	3/1/2018	3/1/2020
Machine Learning Server 9.2.1	10/1/2017	10/1/2019
Microsoft R Server 9.1.0	5/1/2017	5/1/2019
Microsoft R Server 9.0.1	1/1/2017	1/1/2019
Microsoft R Server 8.0.5	7/1/2016	7/1/2018
Microsoft R Server 8.0.0	1/1/2016	1/1/2018

Known issues in Machine Learning Server

7/12/2022 • 18 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

The following issues are known in the 9.4 release.

Known issues in 9.4

1. Missing `azure-ml-admin-cli` extension on DSVM environments

If for some reason your `azure-ml-admin-cli` extension is not available or has been removed, you will be met with the following error:

```
# With elevated privileges, run the following commands.  
$ az mlserver admin --help  
  
az: error: argument _command_package: invalid choice: mlserver  
usage: az [-h] [--verbose] [--debug] [--output {tsv,table,json,jsonc}]  
          [--query JMESPATH]  
          {aks,backup,redis,network,cosmosdb,batch,iot,dla,group,webapp,acr,dls,  
           storage,mysql,vm,reservations,account,keyvault,sql,vmss,eventgrid,  
           managedapp,ad,advisor,postgres,container,policy,lab,batchai,  
           functionapp,identity,role,cognitiveservices,monitor,sf,resource,cdn,  
           tag,feedback,snapshot,disk,extension,acs,provider,cloud,lock,image,  
           find,billing,appservice,login,consumption,feature,logout,configure,  
           interactive}
```

If you encounter this error, you can re-add the extension as such:

Windows:

```
$ az extension add --source "C:\Program Files\Microsoft\ML Server\Setup\azure_ml_admin_cli-0.0.1-py2.py3-none-any.whl" --yes
```

Linux:

```
az extension add --source /opt/microsoft/mlserver/9.4.7/o16n/azure_ml_admin_cli-0.0.1-py2.py3-none-any.whl -  
-yes
```

2. Compute nodes fail on a Python-only install on Ubuntu 14.04

This issue applies to both 9.3 and 9.2.1 installations. On an Ubuntu 14.04 installation of a Python-only Machine Learning Server configured for operationalization, the compute node eventually fails. For example, if you run [diagnostics](#), the script fails with "BackEndBusy Exception".

To work around this issue, comment out the stop service entry in the config file:

1. On the compute node, edit the `/etc/init/computenode.service` file.

2. Comment out the command: "stop on stopping rserve" by inserting # at beginning of the line.

3. Restart the compute node: `az ml admin node start --computenode`

For more information on service restarts, see [Monitor, stop, and start web & compute nodes](#).

3. ImportError for Matplotlib.pyplot

This is a [known Anaconda issue](#) not specific to Machine Learning Server, but Matplotlib.pyplot fails to load on some systems. Since using Matplotlib.pyplot with revoscalepy is a common scenario, we recommend the following workaround if you are blocked by an import error. The workaround is to assign a non-interactive backend to matplotlib prior to loading pyplot:

```
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt
```

For more information, search for "Agg backend" in the [Matplotlib FAQ](#).

4. Model deserialization on older remote servers

Applies to: [rxSerializeModel \(RevoScaleR\)](#), referencing "Error in memDecompress(data, type = decompress)"

If you customarily switch the compute context among multiple machines, you might have trouble deserializing a model if the RevoScaleR library is out of sync. Specifically, if you serialized the model on a newer client, and then attempt deserialization on a remote server having older copies of those libraries, you might encounter this error:

```
"Error in memDecompress(data, type = decompress) :
  internal error -3 in memDecompress(2)"
```

To deserialize the model, switch to a newer server or consider upgrading the older remote server. As a best practice, it helps when all servers and client apps are at the same functional level.

5. azureml-model-management-sdk only supports up to 3 arguments as the input of the web service

When consuming the web services using python, sending multiple variables (more than three) as inputs of `consume()` or the alias function is returning `KeyError` or `TypeError`. Alternative: use `DataFrames` as the input type.

```

# example:
def func(Age, Gender, Height, Weight):
    pred = mod.predict(Age, Gender, Height, Weight)
    return pred
#error 1:
service.consume(Age = 25.0, Gender = 1.0, Height = 180.0, Weight = 200.0)
#-----
#TypeError: consume() got multiple values for argument 'Weight'
#-----
#error 2:
service.consume(25.0, 1.0, 180.0, 200.0)
#-----
#KeyError: 'weight'
#-----

#workaround:
def func(inputDatf):
    features = ['Age', 'Gender', 'Height', 'Weight']
    inputDatf = inputDatf[features]
    pred = mod.predict(inputDatf)
    inputDatf['predicted']=pred
    outputDatf = inputDatf
    return outputDatf

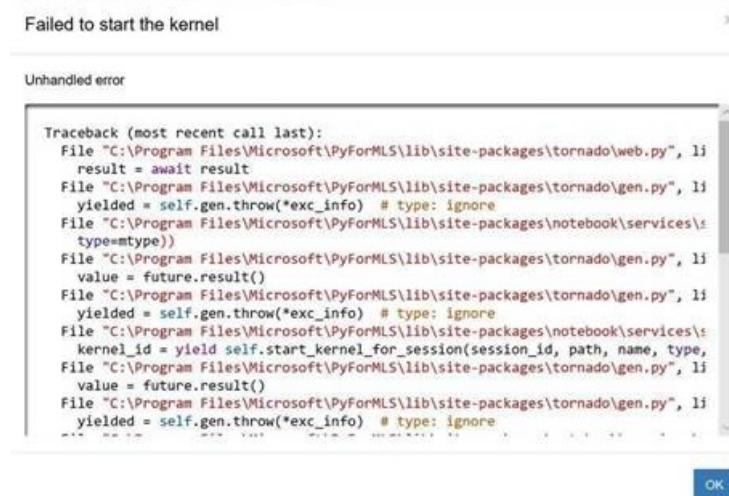
service = client.service(service_name)\n        .version('1.0')\n        .code_fn(func)\n        .inputs(inputDatf=pd.DataFrame)\n        .outputs(outputDatf=pd.DataFrame)\n        .models(mod=mod)\n        .description('Calories python model')\n        .deploy()

res=service.consume(pd.DataFrame({ 'Age':[1], 'Gender':[2], 'Height':[3], 'Weight':[4] }))
```

6. Python 3 Kernel error when using Jupyter Notebooks and Python Client 9.4.7

A Python 3 Kernel error may occur when using Jupyter Notebooks for Microsoft Machine Learning Server with ML Python Client 9.4.7.

For example:



The workaround is to edit the file

`C:\Program Files\Microsoft\PyForMLS\share\jupyter\kernels\python3\kernel.json` and replace all the contents with the following:

```
{
  "display_name": "Python 3",
  "language": "python",
  "argv": [
    "C:\\Program Files\\Microsoft\\PyForMLS\\python.exe",
    "-m",
    "ipykernel_launcher",
    "-f",
    "{connection_file}"
  ]
}
```

The file paths of `kernel.json` and `python.exe` may be different if the client was installed to a different folder.

Previous releases

This document also describes the known issues for the last several releases:

- [Known issues for 9.3](#)
- [Known issues for 9.2.1](#)
- [Known issues for 9.1.0](#)
- [Known issues for 9.0.1](#)
- [Known issues for 8.0.5](#)

Machine Learning Server 9.3

1. Missing `azure-ml-admin-cli` extension on DSVM environments

If for some reason your `azure-ml-admin-cli` extension is not available or has been removed, you will be met with the following error:

```
# With elevated privileges, run the following commands.
$ az ml admin --help

az: error: argument _command_package: invalid choice: ml
usage: az [-h] [--verbose] [--debug] [--output {tsv,table,json,jsonc}]
          [--query JMESPATH]
          {aks,backup,redis,network,cosmosdb,batch,iot,dla,group,webapp,acr,dls,
           storage,mysql,vm,reservations,account,keyvault,sql,vmss,eventgrid,
           managedapp,ad,advisor,postgres,container,policy,lab,batchai,
           functionapp,identity,role,cognitiveservices,monitor,sf,resource,cdn,
           tag,feedback,snapshot,disk,extension,acs,provider,cloud,lock,image,
           find,billing,appservice,login,consumption,feature,logout,configure,
           interactive}
```

If you encounter this error, you can re-add the extension as such:

Windows:

```
$ az extension add --source 'C:\\Program Files\\Microsoft\\ML Server\\Setup\\azure_ml_admin_cli-0.0.1-py2.py3-none-any.whl' --yes
```

Linux:

```
az extension add --source /opt/microsoft/mlserver/9.3.0/o16n/azure_ml_admin_cli-0.0.1-py2.py3-none-any.whl --yes
```

2. Compute nodes fail on a Python-only install on Ubuntu 14.04

This issue applies to both 9.3 and 9.2.1 installations. On an Ubuntu 14.04 installation of a Python-only Machine Learning Server configured for operationalization, the compute node eventually fails. For example, if you run [diagnostics](#), the script fails with "BackEndBusy Exception".

To work around this issue, comment out the stop service entry in the config file:

1. On the compute node, edit the /etc/init/computenode.service file.
2. Comment out the command: "stop on stopping rserve" by inserting # at beginning of the line.
3. Restart the compute node: `az ml admin node start --computenode`

For more information on service restarts, see [Monitor, stop, and start web & compute nodes](#).

3. ImportError for Matplotlib.pyplot

This is a [known Anaconda issue](#) not specific to Machine Learning Server, but Matplotlib.pyplot fails to load on some systems. Since using Matplotlib.pyplot with revoscalepy is a common scenario, we recommend the following workaround if you are blocked by an import error. The workaround is to assign a non-interactive backend to matplotlib prior to loading pyplot:

```
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt
```

For more information, search for "Agg backend" in the [Matplotlib FAQ](#).

4. Model deserialization on older remote servers

Applies to: [rxSerializeModel \(RevoScaleR\)](#), referencing "Error in memDecompress(data, type = decompress)"

If you customarily switch the compute context among multiple machines, you might have trouble deserializing a model if the RevoScaleR library is out of sync. Specifically, if you serialized the model on a newer client, and then attempt deserialization on a remote server having older copies of those libraries, you might encounter this error:

```
"Error in memDecompress(data, type = decompress) :
  internal error -3 in memDecompress(2)"
```

To deserialize the model, switch to a newer server or consider upgrading the older remote server. As a best practice, it helps when all servers and client apps are at the same functional level.

5. azureml-model-management-sdk only supports up to 3 arguments as the input of the web service

When consuming the web services using python, sending multiple variables (more than three) as inputs of `consume()` or the alias function is returning `KeyError` or `TypeError`. Alternative: use `DataFrames` as the input type.

```

# example:
def func(Age, Gender, Height, Weight):
    pred = mod.predict(Age, Gender, Height, Weight)
    return pred
#error 1:
service.consume(Age = 25.0, Gender = 1.0, Height = 180.0, Weight = 200.0)
#-----
#TypeError: consume() got multiple values for argument 'Weight'
#-----
#error 2:
service.consume(25.0, 1.0, 180.0, 200.0)
#-----
#KeyError: 'weight'
#-----

#workaround:
def func(inputDatf):
    features = ['Age', 'Gender', 'Height', 'Weight']
    inputDatf = inputDatf[features]
    pred = mod.predict(inputDatf)
    inputDatf['predicted']=pred
    outputDatf = inputDatf
    return outputDatf

service = client.service(service_name) \
    .version('1.0') \
    .code_fn(func) \
    .inputs(inputDatf=pd.DataFrame) \
    .outputs(outputDatf=pd.DataFrame) \
    .models(mod=mod) \
    .description('Calories python model') \
    .deploy()

res=service.consume(pd.DataFrame({ 'Age':[1], 'Gender':[2], 'Height':[3], 'Weight':[4] }))
```

Machine Learning Server 9.2.1

The following issues are known in this release:

- [Configure Machine Learning Server web node warning: "Web Node was not able to start because it is not configured."](#)
- [Client certificate is ignored when the Subject or Issue is blank.](#)
- [Web node connection to compute node times out during a batch execution.](#)

NOTE

Other release-specific pages include [What's New in 9.2.1](#) and [Deprecated and Discontinued Features](#). For known issues in the previous releases, see [Previous Releases](#).

1. Configure Machine Learning Server web node warning: "Web Node was not able to start because it is not configured."

When configuring your web node, you might see the following message: "Web Node was not able to start because it is not configured." Typically, this is not really an issue since the web node is automatically restarted within 5 minutes by an auto-recovery mechanism. After five minutes, run the [diagnostics](#).

2. Client certificate is ignored when the Subject or Issue is blank.

If you are using a client certificate, both the Subject AND Issuer need to be set to a value in order for the certificate to be used. If any of those is not set, the certificate settings are ignored without warning.

3. Web node connection to compute node times out during a batch execution.

If you are consuming a long-running web service via batch mode, you may encounter a connection timeout between the web and compute node. In batch executions, if a web service is still running after 10 minutes, the connection from the web node to the compute node times out. The web node then starts another session on another compute node or shell. The initial shell that was running when the connection timed out continues to run but never returns a result.

The workaround to bypass the timeout is to modify the web node appsetting.json file.

1. Change the field "ConnectionTimeout" under the "ComputeNodesConfiguration" section. The default value is "01:00:00", which is one hour.
2. Add a new field "BatchExecutionCheckoutTimeSpan" at the base level of the json file. For example:

```
"MaxNumberOfThreadsPerBatchExecution": 100,  
"BatchExecutionCheckoutTimeSpan": "03:00:00",
```

The value of "BatchExecutionCheckoutTimeSpan" and "ConnectionTimeout" should be set to same value. If both web and compute nodes are on the same machine (a one-box configuration) or on the same virtual network, then the "ConnectionTimeout" can be shorter because there is minimal latency.

To reduce the risk of timeouts, we recommend same-machine or same-network deployments. On Azure, you can set these up easily using a template. For more information, see [Configure Machine Learning Server using Resource Manager templates](#).

Microsoft R Server 9.1.0

1. [RevoScaleR: rxMerge\(\) behaviors in RxSpark compute context](#)
2. [RevoScaleR: rxExecBy\(\) terminates unexpectedly when NA values do not have a factor level](#)
3. [MicrosoftML error: "Transform pipeline 0 contains transforms that do not implement IRowToRowMapper"](#)
4. [Spark compute context: modelCount=1 does not work with rxTextData](#)
5. [Cloudera: "install_mrs_parcel.py" does not exist](#)
6. [Cloudera: Connection error due to libjvm and libhdfs package dependencies](#)
7. [Long delays when consuming web service on Spark](#)

Other release-specific pages include [What's New in 9.1](#) and [Deprecated and Discontinued Features](#). For known issues in the 9.0.1 or 8.0.5 releases, see [Previous Releases](#).

1. rxMerge() behaviors in RxSpark compute context

Applies to: RevoScaleR package > rxMerge function

In comparison with the local compute context, rxMerge() used in a RxSpark compute context has slightly different behaviors:

1. NULL return value.
2. Column order may be different.
3. Factor columns may be written as character type.
4. In a local compute context, duplicate column names are made unique by adding "", plus the extensions provided by the user via the duplicateVarExt parameter (for example "Visibility.Origin"). In an RxSpark compute context, the "" is omitted.

2. rxExecBy() terminates unexpectedly when NA values do not have a factor level

Applies to: RevoScaleR package > rxExecBy function

R script using rxExecBy suddenly aborts when the data set presents factor columns containing NA values, and NA is not a factor level. For example, consider a variable for Gender with three factor levels: Female, Male,

Unknown. If an existing value is not represented by one of the factors, the function fails.

There are two possible workarounds:

- Option 1: Add an 'NA' level using `addNA()` to catch the "not applicable" case.
- Option 2: Clean the input dataset (remove the NA values).

Pseudo code for the first option might be:

```
> dat$Gender = addNA(dat$Gender)
```

Output would now include a fourth factor level called NA that would catch all values not covered by the other factors:

```
> rxGetInfo(dat, getVarInfo = TRUE)

Data frame: dat
Number of observations: 97
Number of variables: 1
Variable information:
Var 1: Gender
  4 factor levels: Female Male Unknown NA
```

3. MicrosoftML error: "Transform pipeline 0 contains transforms that do not implement IRowToRowMapper"

Applies to: MicrosoftML package > Ensembling

Certain machine learning transforms that don't implement the `IRowToRowMapper` interface fail during Ensembling. Examples include `getSentiment()` and `featurizeImage()`.

To work around this error, you can pre-featurize data using `rxFeaturize()`. The only other alternative is to avoid mixing Ensembling with transforms that produce this error. Finally, you could also wait until the issue is fixed in the next release.

4. Spark compute context: modelCount=1 does not work with rxTextData

Applies to: MicrosoftML package > Ensembling

`modelCount = 1` does not work when used with `rxTextData()` on Hadoop/Spark. To work around this issue, set the property to greater than 1.

5. Cloudera: "install_mrs_parcel.py" does not exist

If you are performing a [parcel installation of R Server in Cloudera](#), you might notice a message directing you to use a python installation script for automated deployment. The exact message is "If you wish to automate the Parcel installation please run:", followed by "install_mrs_parcel.py". Currently, that script is not available. Ignore the message.

6. Cloudera: Connection error related to libjvm or libhdfs package dependencies

R Server has a package dependency that is triggered only under a very specific configuration:

- R Server was installed on CDH via parcel generator
- RStudio is the IDE
- Operation runs in local compute context on an edge node in a Hadoop cluster

Under this configuration, a failed operation could be the result of a package dependency, which is evident in the error message stack through warnings about a missing libjvm or libhdfs package.

The workaround is to recreate the symbolic link, update the site file, and restart R Studio.

1. Create this symlink:

```
sudo ln -s /usr/java/jdk1.7.0_67-cloudera/jre/lib/amd64/server/libjvm.so  
/opt/cloudera/parcels/MRS/hadoop/libjvm.so
```

2. Copy the site file to the parcel repo and rename it to RevoHadoopEnvVars.site:

```
sudo cp ~/.RevoHadoopEnvVars.site /opt/cloudera/parcels/MRS/hadoop  
sudo mv /opt/cloudera/parcels/MRS/hadoop/.RevoHadoopEnvVars.site  
/opt/cloudera/parcels/MRS/hadoop/RevoHadoopEnvVars.site
```

3. Restart RStudio after the changes:

```
sudo rstudio-server restart
```

7. Long delays when consuming web service on Spark

If you encounter long delays when trying to consume a web service created with mrsdeploy functions in a Spark compute context, you may need to add some missing folders. The Spark application belongs to a user called 'rserve2' whenever it is invoked from a web service using mrsdeploy functions.

To work around this issue, create these required folders for user 'rserve2' in local and hdfs:

```
hadoop fs -mkdir /user/RevoShare/rserve2  
hadoop fs -chmod 777 /user/RevoShare/rserve2  
  
mkdir /var/RevoShare/rserve2  
chmod 777 /var/RevoShare/rserve2
```

Next, create a new Spark compute context:

```
rxSparkConnect(reset = TRUE)
```

When 'reset = TRUE', all cached Spark Data Frames are freed and all existing Spark applications belonging to the current user are shut down. If you encounter long delays when trying to consume a web service created with mrsdeploy functions in a Spark compute context, you may need to add some missing folders. The Spark application belongs to a user called "rserve2" whenever it is invoked from a web service using mrsdeploy functions.

To work around this issue, create these required folders for user "rserve2" in local and hdfs:

```
hadoop fs -mkdir /user/RevoShare/rserve2  
hadoop fs -chmod 777 /user/RevoShare/rserve2  
  
mkdir /var/RevoShare/rserve2  
chmod 777 /var/RevoShare/rserve2
```

Now, to create a clean Spark compute context, then run:

```
rxSparkConnect(reset = TRUE)
```

The 'reset' parameter kills all pre-existing yarn applications, and create a new one.

8. Web node connection to compute node times out during a batch execution.

If you are consuming a long-running web service via batch mode, you may encounter a connection timeout between the web and compute node. In batch executions, if a web service is still running after 10 minutes, the connection from the web node to the compute node times out. The web node then starts another session on another compute node or shell. The initial shell that was running when the connection timed out continues to run but never returns a result.

Microsoft R Server 9.0.1

Package: RevoScaleR > Distributed Computing

- On SLES 11 systems, there have been reports of threading interference between the Boost and MKL libraries.
- The value of `consoleOutput` that is set in the `RxHadoopMR` compute context when `wait=FALSE` determines whether or not `consoleOutput` is displayed when `rxGetJobResults` is called; the value of `consoleOutput` in the latter function is ignored.
- When using `RxInTeradata`, if you encounter an intermittent failure, try resubmitting your R command.
- The `rxDataStep` function does not support the `varsToKeep` and `varsToDrop` arguments in `RxInTeradata`.
- The `dataPath` and `outDataPath` arguments for the `RxHadoopMR` compute context are not yet supported.
- The `rxSetVarInfo` function is not supported when accessing xdf files with the `RxHadoopMR` compute context.
- When specifying a non-default `RNGkind` as an argument to `rxExec`, identical random number streams can be generated unless the `RNGseed` is also specified.
- When using small test data sets on a Teradata appliance, some test failures may occur due to insufficient data on each AMP.
- Adding multiple new columns using `rxDataStep` with `RxTeradata` data sources fails in local compute context. As a workaround, use `RxOdbcData` data sources or the `RxInTeradata` compute context.

Package: RevoScaleR > Data Import and Manipulation

- Appending to an existing table is not supported when writing to a Teradata database.
- When reading `VARCHAR` columns from a database, white space is trimmed. To prevent this, enclose strings in non-white-space characters.
- When using functions such as `rxDataStep` to create database tables with `VARCHAR` columns, the column width is estimated based on a sample of the data. If the width can vary, it may be necessary to pad all strings to a common length.
- Using a transform to change a variable's data type is not supported when repeated calls to `rxImport` or `rxTextToXdf` are used to import and append rows, combining multiple input files into a single .xdf file.
- When importing data from the Hadoop Distributed File System, attempting to interrupt the computation may result in exiting the software.

Package: RevoScaleR > Analysis Functions

- Composite xdf data set columns are removed when running `rxPredict(.)` with `rxDForest(.)` in Hadoop and writing to the input file.
- The `rxDTTree` function does not currently support in-formula transformations; in particular, using the `F()` syntax for creating factors on the fly is not supported. However, numeric data is automatically binned.
- Ordered factors are treated the same as factors in all RevoScaleR analysis functions except `rxDTTree`.

Package: RevoIOQ

- If the `RevoIOQ` function is run concurrently in separate processes, some tests may fail.

Package: RevoMods

- The `RevoMods` `timestamp()` function, which masks the standard version from the `utils` package, is unable to find the `c_adchistory` object when running in an Rgui, Rscript, etc. session. If you are calling `timestamp()`, call the `utils` version directly as `utils::timestamp()`.

- In the `nls` function, use of the `port` algorithm occasionally causes the R front end to stop unexpectedly. The `nls` help file advises caution when using this algorithm. We recommend avoiding it altogether and using either the default Gauss-Newton or plinear algorithms.

Operationalize (Deploy & Consume Web Services) features formerly referred to as DeployR

- When Azure active directory authentication is the only form of authentication enabled, it is not possible to run diagnostics.

Microsoft R Server 8.0.5

Package: RevoScaleR > Distributed Computing

- On SLES 11 systems, there have been reports of threading interference between the Boost and MKL libraries.
- The value of `consoleOutput` defined in the `RxHadoopMR` compute context when `wait=FALSE` determines whether `consoleOutput` is displayed when `rxGetJobResults` is called. The value of `consoleOutput` in the latter function is ignored.
- When using `RxInTeradata`, if you encounter an intermittent failure, try resubmitting your R command.
- The `rxDataStep` function does not support the `varsToKeep` and `varsToDrop` arguments in `RxInTeradata`.
- The `dataPath` and `outDataPath` arguments for the `RxHadoopMR` compute context are not yet supported.
- The `rxSetVarInfo` function is not supported when accessing xdf files with the `RxHadoopMR` compute context.

Package: RevoScaleR > Data Import and Manipulation

- Appending to an existing table is not supported when writing to a Teradata database.
- When reading `VARCHAR` columns from a database, white space is trimmed. To prevent this, enclose strings in non-white-space characters.
- When using functions such as `rxDataStep` to create database tables with `VARCHAR` columns, the column width is estimated based on a sample of the data. If the width can vary, it may be necessary to pad all strings to a common length.
- Using a transform to change a variable's data type is not supported when repeated calls to `rxImport` or `rxTextToXdf` are used to import and append rows, combining multiple input files into a single .xdf file.
- When importing data from the Hadoop Distributed File System, attempting to interrupt the computation may result in exiting the software.

Package: RevoScaleR > Analysis Functions

- Composite xdf data set columns are removed when running `rxPredict()` with `rxDForest()` in Hadoop and writing to the input file.
- The `rxDTTree` function does not currently support in-formula transformations; in particular, using the `F()` syntax for creating factors on the fly is not supported. However, numeric data is automatically binned.
- Ordered factors are treated the same as factors in all RevoScaleR analysis functions except `rxDTTree`.

DeployR

- On Linux, if you attempt to change the DeployR RServe port using the `adminUtilities.sh`, the script incorrectly updates Tomcat's `server.xml` file, which prevents Tomcat from starting, and does not update the necessary the `Rserv.conf` file. You must revert back to an earlier version of `server.xml` to restore service.
- Using `deployrExternal()` on the DeployR Server to reference a file that in a specified folder produces a 'Connection Error' due to an improperly defined environment variable. For this reason, you must log into the Administration Console and go to The Grid tab. In that tab, edit **Storage Context value for each and every node** and specify the full path to the external data directory on that node's machine, such as `<DEPLOYR_INSTALLATION_DIRECTORY>/deployr/external/data`.

RevoIOQ Package

- If the `RevoIOQ` function is run concurrently in separate processes, some tests may fail.

RevoMods Package

- The `RevoMods timestamp()` function, which masks the standard version from the `utils` package, is unable to find the `C_addhistory` object when running in an Rgui, Rscript, etc. session. If you are calling `timestamp()`, call the `utils` version directly as `utils::timestamp()`.

R Base and Recommended Packages

- In the `nls` function, use of the `port` algorithm occasionally causes the R front end to stop unexpectedly. The `nls` help file advises caution when using this algorithm. We recommend avoiding it altogether and using either the default Gauss-Newton orplinear algorithms.

Opting out of usage data collection (Machine Learning Server)

7/12/2022 • 2 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

By default, telemetry data is collected during your usage of Machine Learning Server and R Client for the purpose of improving products and services. Anonymous usage data includes device information, operating system version, regional and language settings, and errors reports. You can review the [Microsoft privacy statement](#) for a detailed explanation.

To turn data collection on or off, use `rxPrivacyControl` from [RevoScaleR](#) on any platform, or `rx_privacy_control` from [revoscalepy](#).

Version Requirements

Privacy controls are in server version 9.x and later, and R Client 3.4.x and later. To get version information, open an R IDE, such as the R Console (RGui.exe). The console app reports server version information for Microsoft R Open, R Client, and R Server. From the console, you can use the `print` command to return verbose version information:

```
> print(Revo.version)
```

Permission Requirements

Opting out of telemetry requires administrator rights. The instructions below explain how to run console applications as an administrator.

As an administrator, running the R command `rxPrivacyControl(TRUE)` will permanently change the setting to TRUE, and `rxPrivacyControl(FALSE)` will permanently change the setting to FALSE. There is no user-facing way to change the setting for a single session.

Without a parameter, running `rxPrivacyControl()` returns the current setting. Similarly, if you are not an administrator, `rxPrivacyControl()` returns just the current setting.

How to opt out (Python)

The revoscalepy package providing `rx-privacy-control` is installed when you add Python support to Machine Learning Server:

1. Log on as root: `sudo su`
2. Start a Python session: `mlserver-python`
3. Load the library: `import revoscalepy`
4. Return the current value: `revoscalepy.rx_privacy_control()`
5. Change the current value: `revoscalepy.rx_privacy_control(TRUE)` to turn on telemetry data collection.

Otherwise, set it to FALSE.

How to opt out (R)

The RevoScaleR package provides `rxPrivacyControl` is installed and loaded in both R Client and R Server. To turn off telemetry data collection, you can use the built-in R console application, which loads RevoScaleR automatically.

On Linux

1. Log on as root: `sudo su`
2. Start an R session: `Revo64`
3. Return the current value: `rxPrivacyControl`
4. Change the current value: `rxPrivacyControl(TRUE)` to turn on telemetry data collection. Otherwise, set it to FALSE.

On Windows

1. Log in to the computer as an administrator.
2. Go to C:\Program Files\Microsoft\ML Server\R_SERVER\bin\x64.
3. Right-click Rgui.exe and select Run as administrator.
4. Type `rxPrivacyControl` to return the current value.
5. Type `rxPrivacyControl(FALSE)` to turn off telemetry data collection.

The `rxPrivacyControl` command sets the state to be opted-in or out for anonymous usage collection.

Command syntax

```
rxPrivacyControl(optIn)
```

This command takes one parameter, `optIn`, set to either 'TRUE' or 'FALSE' to opt out. Left unspecified, the current setting is returned.

How to Contact Us

Please contact Microsoft if you have any questions or concerns.

- For general privacy question or a question for the Chief Privacy Officer of Microsoft or want to request access to your personal information, please contact us by using our [Web form](#).
- For technical or general support question, please visit <https://support.microsoft.com/> to learn more about Microsoft Support offerings.
- If you have a Microsoft account password question, please visit [Microsoft account support](#).
- By mail: Microsoft Privacy, Microsoft Corporation, One Microsoft Way, Redmond, Washington 98052 USA
- By Phone: 425-882-8080

Additional Resources for Machine Learning Server and Microsoft R

7/12/2022 • 10 minutes to read • [Edit Online](#)

IMPORTANT

This content is being retired and may not be updated in the future. The support for Machine Learning Server will end on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Use the links in this article for recommended resources about R and Python.

Blogs and resources

- [Product web page](#)
- [Supported platforms](#)
- [About R](#)
- [R Language Resources](#)
- [R Blog](#)
- [Operationalization Examples](#)
- [Galaxy classification with neural networks: a data science workflow](#)
- [Running your R code on Azure with mrsdeploy](#)

Support

- [Machine Learning Server support forum](#)
- [Microsoft R Open support forum](#)

Sample datasets

- [Sample Dataset Directory](#)
- [More Data Sources](#)

New to Python

Python is installed through a distribution of Anaconda, adding tools and packages that are in common use across the development community. Jupyter Notebooks, included in Anaconda, is added to your system when you install Machine Learning Server with Python support. We recommend using Jupyter Notebooks for beginners. Its interactive environment, with visualization support, is easy to use and a mainstay of Python training content.

To start Jupyter Notebooks, go to C:\Program Files\Microsoft\ML Server\PYTHON_SERVER\Scripts and run Jupyter-Notebook.exe. A browser window will open to localhost, with options for creating or opening new Python files. For help on getting started, see [Jupyter Notebook Tips, Tricks, and Shortcuts](#).

New to R

If you are just getting started with R, we recommend the R Core Team manuals, which are part of every R

distribution, including *An Introduction to R*, *The R Language Definition*, *Writing R Extensions*. You can access that on the CRAN website or in your R installation directory.

Beyond the standard R manuals, there are many books available to help you learn R, and to help you use R to do particular tasks. The rest of this article helps point you in the right direction.

For beginners

- *R for Dummies* by Andrie de Vries and Joris Meys
Excellent starting place if you are new to R, filled with examples and tips
- *R FAQ* by Kurt Hornik
- *An Introduction to R* by the R Development Core Team, 2008
Based on "Notes on S-Plus" by Bill Venables and David Smith. Includes an extensive sample session in Appendix A

Intermediate R Users

- *O'Reilly's R Cookbook* by Paul Teator
A book filled with recipes to help you accomplish specific tasks.
- *The Essential R Reference* by Mark Gardener
A dictionary-like reference to more than 400-R commands, including cross-references and examples

For SAS or SPSS Users

- *R for SAS and SPSS Users* by Robert Muenchen
Good starting point for users of SAS or SPSS who are new to R
- *SAS and R: Data Management, Statistical Analysis, and Graphics* by Ken Kleinman and Nicholas J. Horton
- *Analysis of Correlated Data with SAS and R* by Mohamed M. Shoukri and Mohammad A. Chaudhary

Information on Data Analysis and Statistics

A good source of information on introductory data analysis and statistics is Peter Dalgaard's *Introductory Statistics with R*. After a chapter on basic R operations, Dalgaard discusses probability and distributions, descriptive statistics and graphics, one- and two-sample tests, regression and correlation, ANOVA and Kruskal-Wallis, tabular data, power and computation of sample size, multiple regression, linear models, logistic regression, and survival.

For more advanced techniques, the obvious starting point is *Modern Applied Statistics with S* by Bill Venables and Brian Ripley. This book starts with four introductory chapters on R, then gets into statistics from univariate statistics (chapter 5) to optimization (chapter 16). Along the way, the authors touch on many widely used techniques, including linear models, generalized linear models, clustering, tree-based methods, survival analysis, and many others.

Rapidly becoming the book for aspiring data scientists is *The Elements of Statistical Learning* by Trevor Hastie, Robert Tibshirani, and Jerome Friedman. This book covers a variety of statistical techniques important in big data analysis and machine learning, including various tree-based methods, support vector machines, graphical models, and more.

Linear models, generalized linear models, and other regression techniques are the subject of a number of texts, including Frank Harrell's *Regression Modeling Strategies*, John Fox's *Applied Regression Analysis and Generalized Linear Models* and his R-specific companion volume, *An R and S-PLUS Companion to Applied Regression*, and *Data Analysis Using Regression and Multilevel/Hierarchical Models* by Andrew Gelman and Jennifer Hill.

Other useful books that take you into more advanced statistics are *R in Action* by Robert I. Kabacoff, *A Handbook of Statistical Analyses Using R* by Brian Everitt and Torsten Hothorn, *Data Analysis and Graphics Using R* by John

Maindonald and John Braun, and *The R Book* by Michael Crawley.

If you are interested in an overview of the multiple uses of big data analytics, the book *Big Data, Big Analytics* by Michael Minelli, Michele Chambers, and Ambiga Dhiraj gives you an excellent start in understanding what big data is and how it is used in real-world business applications.

Information on Programming with R

The newest book from John M. Chambers, *Software for Data Analysis: Programming with R*, gives a thorough description of programming in R, including tips on debugging, writing packages, creating classes and methods, and interfacing to code in other languages. It also includes a useful chapter describing how R works.

R in a Nutshell by Joseph Adler is unlike most books on R in that it deals with R first and foremost as a programming language; it does touch on statistical topics, but that is not its main focus.

The book *S Programming* by Venables and Ripley is a concise, readable guide to programming in the S family of languages. Most of their advice remains valid, but the book was published when R was still at a pre-release version (0.90.1), some details have changed over time.

The Blue Book, White Book, and Green Book all have one or more chapters devoted to programming in S, with different points of emphasis. The Blue Book focuses on basic function writing. The White Book describes the S Version 3 class system and how to define classes, generic functions, and methods in that system. The Green Book describes the S Version 4 class system and how to define classes and methods in that system.

The manual *Writing R Extensions* by the R Core Team describes how to write complete R packages, including documentation.

Information on Getting Data Into and Out of R

The manual *R Data Import/Export* by the R Core Team describes how to read data into R from a variety of sources using both built-in R tools and additional packages. The book *Data Manipulation with R* by Phil Spector includes information on reading and writing data, and also further manipulation within R. Also, be sure to look at the *RevoScaleR User's Guide* for information on data import and export capabilities provided by **RevoScaleR**.

Information on Creating Graphics with R

All of the references mentioned up to now contain at least some material on graphics, because graphical exploration is a primary motivation for using R in the first place. The Blue Book, in particular, describes in detail the "traditional S graphics" framework.

A popular graphics package that is rapidly growing its own complete package ecosystem is Hadley Wickham's *ggplot2* package, documented in Wickham's *ggplot2: Elegant Graphics for Data Analysis*. The *ggplot2* package implements in R many of the ideas from Leland Wilkinson's *The Grammar of Graphics*.

The *ggplot2* package is a high-level graphics package. For lower-level graphics functionality, the definitive reference is Paul Murrell's *R Graphics*, which describes both the traditional S graphics framework (in particular, its implementation in R by Ross Ihaka) and the grid graphics framework developed by Murrell. It also describes the lattice system, developed by Deepayan Sarkar, that uses the grid framework to implement the Trellis graphics system developed by Rick Becker and Bill Cleveland. Serious users of the lattice system also consult Sarkar's book, *Lattice: Multivariate Data Visualization with R*.

Trellis graphics are discussed thoroughly in Cleveland's *Visualizing Data*. Cleveland's earlier book, now its second edition, *The Elements of Graphing Data* remains essential reading for anyone interested in data visualization.

Interactive and Dynamic Graphics for Data Analysis by Dianne Cook and Deborah Swayne describes using R together with the GGobi visualization program for dynamic graphics.

R Productivity Environment (RPE) The RPE is an older development tool. RPE documentation can be found at the following links:

- [RPE Getting Started](#)
- [RPE User's Guide](#)

Revolution R Enterprise Docs Prior to Machine Learning Server and Microsoft R Server, the product was called Revolution R Enterprise (RRE).

Here is a list of the available archived documentation sets for RRE:

- [RRE 7.5.0 Docs](#)
- [RRE 7.4.1 Docs](#)
- [RRE 7.4.0 Docs](#)
- [RRE 7.3.0 Docs](#)
- [RRE 7.2.0 Docs](#)

DeployR 8.x Docs

The documentation for these releases has been archived. [See here.](#)

More Books on R

- Adler, J. (2010). *R in a Nutshell*. Sebastopol, CA: O'Reilly.
- Becker, R. A., Chambers, J. M., & Wilks, A. R. (1988). *The New S Language: A Programming Environment for Data Analysis and Graphics*. New York: Chapman and Hall.
- Chambers, J. M. (1998). *Programming with Data: A Guide to the S Language*. New York: Springer.
- Chambers, J. M. (2008). *Software for Data Analysis: Programming with R*. New York: Springer.
- Chambers, J. M., & Hastie, T. J. (Eds.). (1992). *Statistical Models in S*. New York: Chapman and Hall.
- Cleveland, W. S. (1993). *Visualizing Data*. Summit, New Jersey: Hobart Press.
- Cleveland, W. S. (1994). *The Elements of Graphing Data* (second ed.). Summit, New Jersey: Hobart Press.
- Cook, D., & Swain, D. F. (2008). *Interactive and Dynamic Graphics for Data Analysis: With R and GGobi*. New York: Springer.
- Crawley, M. J. (2013). *The R Book* (Second ed.). Chichester: John Wiley & Sons Ltd.
- Dalgaard, P. (2002). *Introductory Statistics with R*. New York: Springer.
- de Vries, A., & Meys, J. (2012). *R for Dummies*. Chichester: John Wiley & Sons.
- Everitt, B. S., & Hothorn, T. (2006). *A Handbook of Statistical Analyses Using R*. Boca Raton, Florida: Chapman & Hall/CRC.
- Fox, J. (2002). *An R and S-PLUS Companion to Applied Regression*. Thousand Oaks, CA: Sage.
- Fox, J. (2008). *Applied Regression Analysis and Generalized Linear Models*. Thousand Oaks, CA: Sage.
- Gardener, M. (2013). *The Essential R Reference*. Indianapolis, IN: John Wiley & Sons.
- Gelman, A., & Hill, J. (2007). *Data Analysis Using Regression and Multilevel/Hierarchical Models*. New York: Cambridge University Press.
- Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1), 5-48.
- Harrell, F. E. (2001). *Regression Model Strategies: with applications to linear models, logistic regression, and*

survival analysis. New York: Springer.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd ed.). New York: Springer.

Ihaka, R., & Gentleman, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3), 299-314.

Kabacoff, R. I. (2011). *R in Action*. Shelter Island, NY: Manning.

Kleinman, K., & Horton, N. J. (2010). *SAS and R: Data Management, Statistical Analysis, and Graphics*. Boca Raton, FL: Chapman & Hall/CRC.

Maindonald, J., & Braun, J. (2007). *Data Analysis and Graphics Using R: An Example-based Approach* (second ed.). Cambridge: Cambridge University Press.

Matloff, N. (2011). *The Art of R Programming*. San Francisco: no starch press.

Minelli, M., Chambers, M., & Dhiraj, A. (2013). *Big Data, Big Analytics*. Hoboken, NJ: John Wiley & Sons.

Muenchen, R. A. (2009). *R for SAS and SPSS Users*. New York: Springer.

Murrell, P. (2006). *R Graphics*. Boca Raton, FL: Chapman & Hall/CRC.

R Development Core Team. (2008). *R: A Language and Environment for Statistical Computing*. Vienna: R Foundation for Statistical Computing.

R Development Core Team. (2008). *An Introduction to R*. Vienna: R Foundation for Statistical Computing.

Sarkar, D. (2008). *Lattice: Multivariate Data Visualization with R*. New York: Springer.

Shoukri, M. M., & Chaudhary, M. A. (2007). *Analysis of Correlated Data with SAS and R* (third ed.). Boca Raton, FL: Chapman & Hall/CRC.

Spector, P. (2008). *Data Manipulation with R*. New York: Springer.

Teator, P. (2011). *R Cookbook*. Sebastopol, CA: O'Reilly.

Venables, W. N., & Ripley, B. D. (1999). *S Programming*. New York: Springer.

Venables, W. N., & Ripley, B. D. (2002). *Modern Applied Statistics with S* (Fourth Edition ed.). New York: Springer.

Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis*. New York: Springer.

Wilkinson, L. (2005). *The Grammar of Graphics* (second ed.). New York: Springer.

Find [additional resources here](#).