Introduction to Microsoft R Server

Contents

Build your own Lab

Summary

This lab is intended to serve as an introduction to the Microsoft R Server functionality by running through a simple exploratory analytic example with some publicly available US flight delay data.

The lab will walk through using Visual Studio with R Tools for Visual Studio to create an R project and perform some exploratory data analysis with Microsoft R Server.

Alternative R IDE's can also be used with Microsoft R Server should you prefer or be more familiar with them. For example RStudio Desktop edition can be installed on your Data Science Virtual Machine.

Business Case

Since 1987 the US Federal Aviation authority has maintained and made public data related to all US internal flights with details of delays. Circa 30 years of data in total. The dataset size has got larger per annum with the increase in public air-travel. We will use the data from 2007 for our analysis.

Flights are often delayed for a number of reasons e.g. congestion, weather, mechanical faults. We will explore the data and ultimately determine whether airlines fly faster between destinations to make up for the departure delays.

There are a number of business applications that this data could be used for:

1. Insurance companies offering travel insurance may want to use the data to assist with pricing risk related to travel delays for travelers holidaying to a specific destination.

2. Travel Operators wishing to promote the best travel experience for their customers will want to understand the best airlines for certain flight legs, minimize delays and missed connections for their customers

3. Airlines will want to measure how they perform against their industry peers, across different routes. They may wish to incorporate additional operational data such as Aircraft servicing, weather, yield (seat utilization) etc.

Learning Objectives

Upon completing this lab, you will have hands-on experience with the following functions and concepts related to Microsoft R Server:

- Using Visual Studio & RTVS as an R IDE
- Working with Rx Data objects versus file-paths
- Working with CSV files
- Importing CSV files into an MRS xdf file
- Initial Data exploration

    - data-quality
    - summary statistics
    - distribution
- Creating additional derived features from a dataset
- Performing simple regression analysis

Lab Requirements / Pre-Requisites

We assume you have followed the course pre-requisites to create an Azure Data Science Virtual Machine. This should contain all the components required to run this lab, other than the data.

A Microsoft account is required to access an Azure Data Science Virtual Machine. If you don’t already have a Microsoft account, you can obtain one for free by following the link below:

https://www.microsoft.com/en-us/account/default.aspx

We will download a ~600MB sample of the dataset from Azure during the lab (later sections). Just to note, there are a number of places where you can obtain the full historical datasets for further analysis but Microsoft conveniently provides the data in a number of different formats here:

http://packages.revolutionanalytics.com/datasets/

For example the 2012 CSV files by month are here :
http://packages.revolutionanalytics.com/datasets/AirOnTimeCSV2012/

You can also download a large dataset containing multiple years to experiment with at the end of this lab, working with larger datasets.

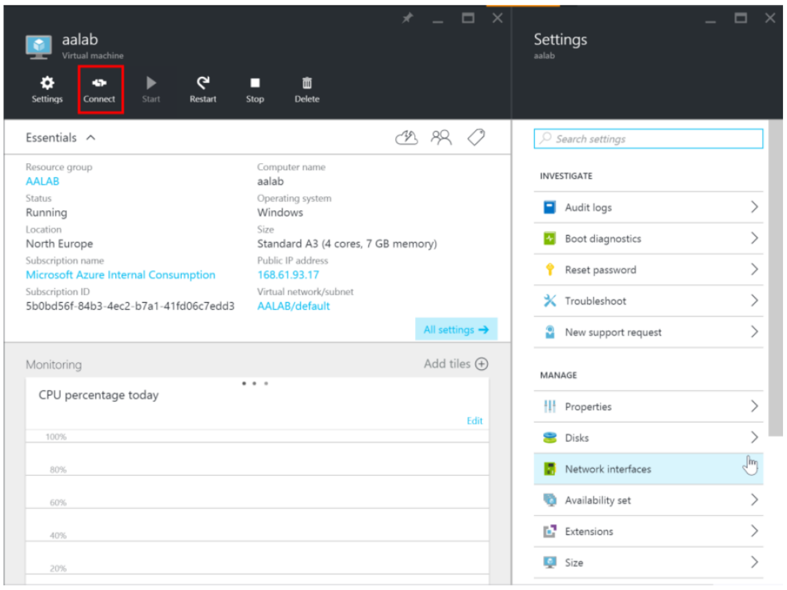
Getting Started

Open Visual Studio and Create an R Project

1. To get started, you will need to start your Azure Data Science Virtual Machine if you have not done so already.

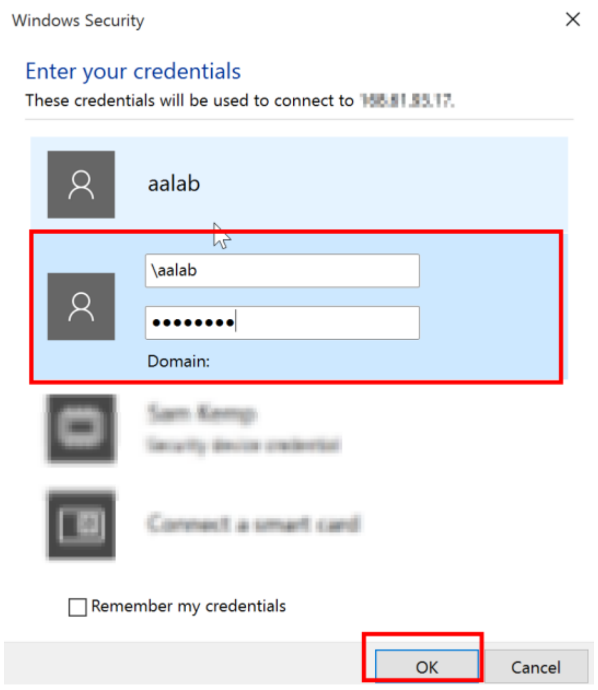
2. You connect to the VM using Remote Desktop on Windows. Click on the **Connect** button as highlighted above. Save the RDP file.

3. This will download the RDP connection profile and initiate a Remote Desktop Connection



Resource group: AALAB
Status: Running
Location: North Europe
Subscription name: Microsoft Azure Internal Consumption
Subscription ID: 5b0bd56f-84b3-4ec2-b7a1-41fd06c7edd3
Computer name: aalab
Operating system: Windows
Size: Standard A3 (4 cores, 7 GB memory)
Public IP address: 168.61.93.17
Virtual network/subnet: AALAB/default

4. Double click on the downloaded RDP file to connect to the VM and enter you credentials (note the \ before the username):



These credentials will be used to connect to [illegible].

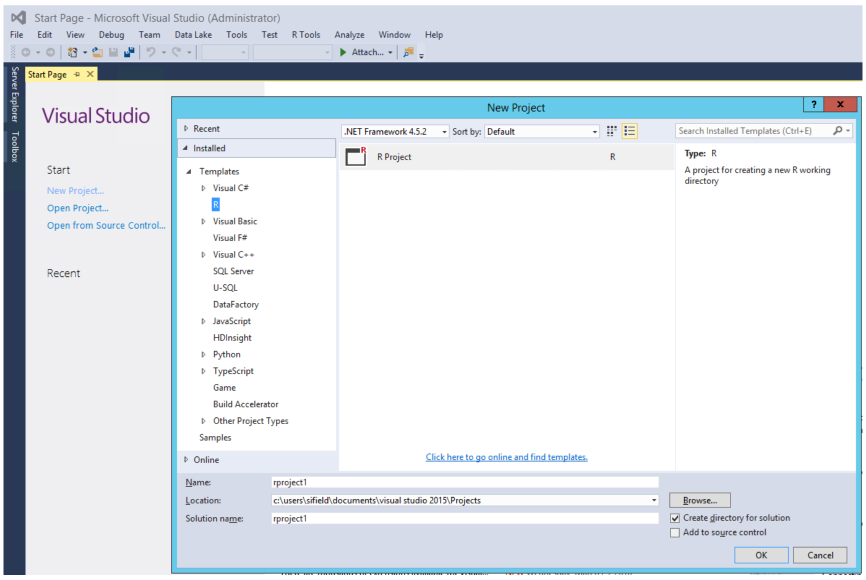
\aalab
••••••••

[illegible]

☐ Remember my credentials

5. On the desktop you will find a short cut to Visual Studio Community Edition, or alternatively go to Start menu and locate and start Visual Studio.

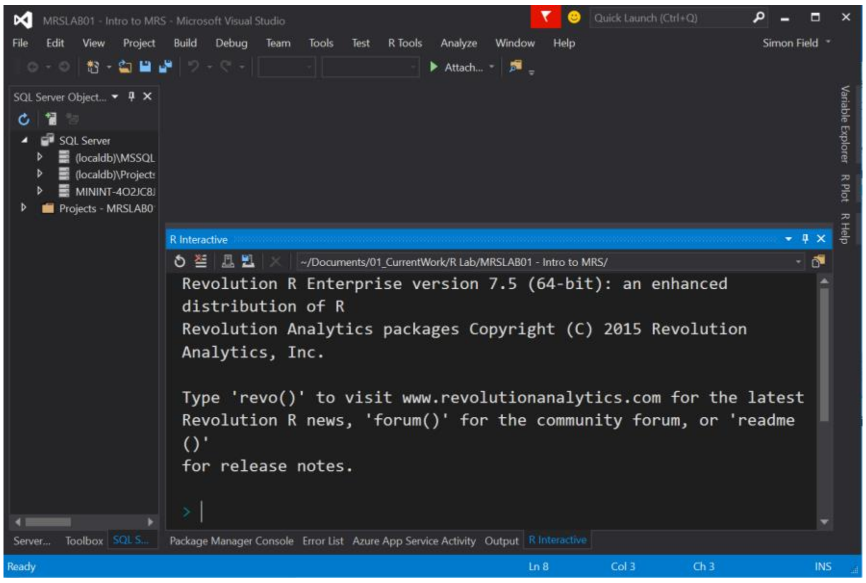
6. From the File menu → New → Project. You should see a dialog similar to that below. Select RProject and create a new R project. Name your project “FlightDelay”. Select the “MRSLAB01 - Intro to MRS” directory where you copied the lab materials. This will use this directory as the project home directory.



.NET Framework 4.5.2 · Sort by: Default

Name: rproject1
Location: c:\users\sifield\documents\visual studio 2015\Projects
Solution name: rproject1
[✓] Create directory for solution
[ ] Add to source control

7. When you create the project Visual Studio will start the R console and you should be presented with a screen similar to the one below with the R console started and an R prompt, ready for some R code!

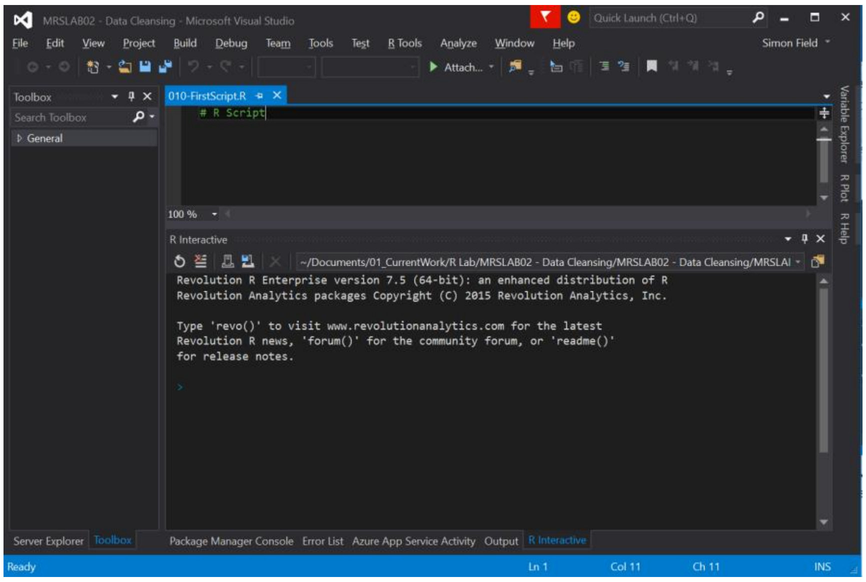

MRSLAB01 - Intro to MRS - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
File Edit View Project Build Debug Team Tools Test R Tools Analyze Window Help
Simon Field
Attach...
SQL Server Object...
SQL Server
(localdb)\MSSQL
(localdb)\Projects
MININT-4O2JC8J
Projects - MRSLAB0
R Interactive
~/Documents/01_CurrentWork/R Lab/MRSLAB01 - Intro to MRS/
Variable Explorer
R Plot
R Help
Server... Toolbox SQL S...
Package Manager Console Error List Azure App Service Activity Output R Interactive
Ready Ln 8 Col 3 Ch 3 INS

```
Revolution R Enterprise version 7.5 (64-bit): an enhanced
distribution of R
Revolution Analytics packages Copyright (C) 2015 Revolution
Analytics, Inc.

Type 'revo()' to visit www.revolutionanalytics.com for the latest
Revolution R news, 'forum()' for the community forum, or 'readme
()'
for release notes.

> |
```

8. Now go to the Project Menu and select Add New Item. Select R Script, and modify the file (script) name to something more meaningful.


MRSLAB02 - Data Cleansing - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test R Tools Analyze Window Help
Simon Field
Toolbox
General
010-FirstScript.R
R Interactive
~/Documents/01_CurrentWork/R Lab/MRSLAB02 - Data Cleansing/MRSLAB02 - Data Cleansing/MRSLAI
Server Explorer Toolbox Package Manager Console Error List Azure App Service Activity Output R Interactive
Ready Ln 1 Col 11 Ch 11 INS

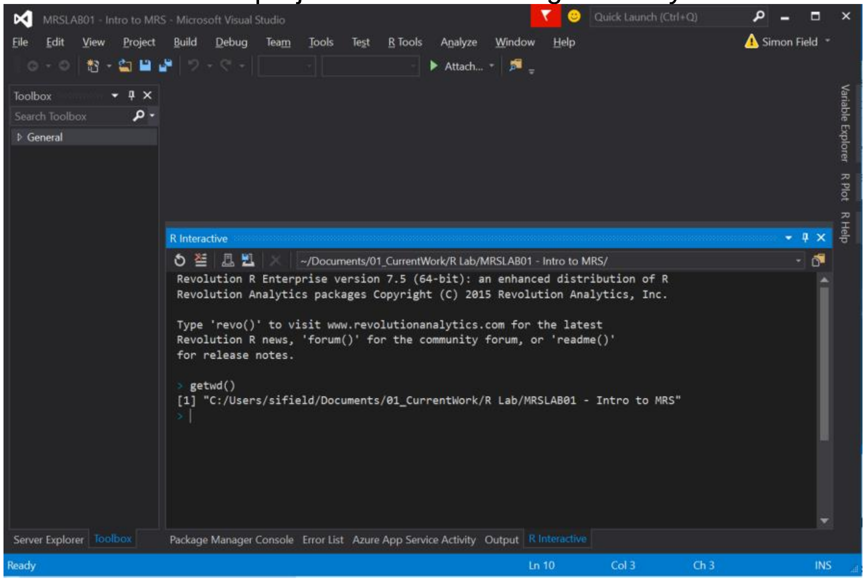
```
# R Script
```

```
Revolution R Enterprise version 7.5 (64-bit): an enhanced distribution of R
Revolution Analytics packages Copyright (C) 2015 Revolution Analytics, Inc.

Type 'revo()' to visit www.revolutionanalytics.com for the latest
Revolution R news, 'forum()' for the community forum, or 'readme()'
for release notes.

>
```

9. We can check the working directory for the project by entering `getwd()` in the R Interactive console and hitting < enter >. This is the projects default working directory.


MRSLAB01 - Intro to MRS - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test R Tools Analyze Window Help
Simon Field
Toolbox
General
R Interactive
~/Documents/01_CurrentWork/R Lab/MRSLAB01 - Intro to MRS/
Server Explorer Toolbox Package Manager Console Error List Azure App Service Activity Output R Interactive
Ready Ln 10 Col 3 Ch 3 INS

```
Revolution R Enterprise version 7.5 (64-bit): an enhanced distribution of R
Revolution Analytics packages Copyright (C) 2015 Revolution Analytics, Inc.

Type 'revo()' to visit www.revolutionanalytics.com for the latest
Revolution R news, 'forum()' for the community forum, or 'readme()'
for release notes.

> getwd()
[1] "C:/Users/sifield/Documents/01_CurrentWork/R Lab/MRSLAB01 - Intro to MRS"
> |
```

10. If you are unfamiliar with VisualStudio RTVS, you can find more information here : https://www.visualstudio.com/en-us/features/rtvs-vs.aspx

11. Microsoft R Server will work with any R IDE or tool, so if you prefer to use an alternative R IDE that you are already familiar with you can download and install that on your Data Science VM instead.

    Note: One advantage to using Visual Studio is that it supports other languages (e.g. SQL, Hadoop and C#). So when developing with MRS to interoperate with those environments, you can use a single editor to manage your project and code

Explore Microsoft R Server options and Help

Explore Microsoft R Server options and Help

1. We will start with a quick exploration of the MRS ScaleR package. Just as options() provides a set of parameters/defaults used by R the rxOptions() function provides a set of additional parameters/defaults used by MRS. Lets explore these now by running the following commands:

```
options()
rxOptions()
```

2. NumCoresToUse defines the number of cores that MRS functions will use by default to process parallelisable functions. This Defaults to total number of cores in the system. Run the following commands:

```
rxOptions()[["numCoresToUse"]]
```

3. Microsoft R Server ships with some sampleData for exploration using code from help files, product documentation. Note: There are a number of file-types available that are supported by MRS for processing.

   There are a mix of demo and installation test (IOQ) tests provided that use this sample data. Run the following commands:

```
sampleDataDir <- rxOptions()[["sampleDataDir"]]
list.files(sampleDataDir)

list.files(rxOptions()[["demoScriptsDir"]])
```

4. The package containing the main MRS user functionality is RevoScaleR. Just like any other R package, we can get help related to the package.

   Run the following command:

```
?RevoScaleR
```

Scroll down the help file to get an overview of the kind of functions available by area/use.

Additional packages are also included in MRS :

```
RevoIOQ, revolpe, RevoMods, RevoPemaR,
RevoRpeConnector, RevoRsrConnector, RevoScaleR,
RevoTreeView, RevoUtils, RevoUtilsMath, doRSR.
```

Exploring the US Airline Flight Dataset

Get the Data!

1. In **your R project** create a directory called **data**
2. In a web browser download the data - https://emealabs.blob.core.windows.net/2007data/2007.csv - place this in the data directory you created in step 1.

Using open-source R with the US Airline Flight Dataset

1. Let's use some open-source R functions to explore a subset of US Flight Departure data. We will explore the data and ascertain if delayed flights fly faster !

2. We will create a data directory under our project if we have not done so already and file-path objects referencing the directory. Copy the following code into your script and step through it.

```
# Configure Data and Working Directories
projectDir <- getwd()
if(dir.exists(file.path(projectDir, "data"))==FALSE)
  {dir.create(file.path(projectDir, "data"))}

dataRead <- file.path(projectDir,"data")
dataWrite <- dataRead
```

3. What would happen if we didn't use RevoScaleR "rx" functions and attempted to run all of this in memory? We can try using open source R functions run a regression of air speed on departure delays on a subset with airSpeed between 50 and 800. If you don't have enough RAM you might not be able to work with the entire data set. We may need to limit our dataset size to fit in memory. For example limit ourselves to a handful of variables to explore. Which columns should we read in?. How would we know that we are not leaving out useful information. Copy the following code into your script and step through it.

```
## define data file to read
list.files(dataRead)
airlineCsv <- file.path(dataRead,"2007.csv")

## define a colInfo vector
myColInfo = rep("NULL", 29)
myColInfo[c(12, 14, 15, 16, 19)] <- "numeric"

# Build a data frame in memory for analysis
airlineDF <- read.table(file=airlineCsv, header = TRUE, sep = ",",
                        colClasses=myColInfo)

# How big is it ?
object.size(airlineDF)
# Note: The dataframe for 2007 year and a subset of columns is ~300MB.
# There is total of 25+ years available and 30+ columns in the total dataset.

# How many rows and what is the structure
dim(airlineDF)
str(airlineDF)
```

4. Now let's do some initial data exploration on the file. Copy the following code into your script and step through it.

```
#  Summarise the data
summary(airlineDF)

# Create a new feature for airspeed
airlineDF$airSpeed <- airlineDF$Distance / airlineDF$AirTime * 60
head(airlineDF)

# Plot
hist(airlineDF$airSpeed,breaks = 100)
```

5. We can see from the histogram that we have some erroneous data causing outliers in our airspeed that we should remove.

   Copy the following code into your script and step through it.

```
# Build a dataframe with the outliers removed
airlineDF2 <- airlineDF[(airlineDF$airSpeed > 50) & (airlineDF$airSpeed < 800),]
dim(airlineDF2)
# make some room.  With open-source R if we leave objects in our workspace we will take
# up valuable memory for us and anyone else sharing the machine (e.g. shared R server)
rm(airlineDF)
gc()
# The histogram looks better with the outliers removed
hist(airlineDF2$airSpeed,breaks = 100)
```

6. We can now run a Linear Regression using lm.

   Copy the following code into your script and step through it.

```
system.time(reg3 <- lm(formula=airSpeed~DepDelay, data=airlineDF2))

summary(reg3)
```

Use MRS to explore the US Airline Flight Dataset

1. Let's now explore some of the MRS functionality available to see how we could perform a similar analysis using ScaleR functions.

2. We first need to create an Rx data object pointing to our data file. Note the capital "R" in Rx denotes that this function defines an object. Copy the following code into your script and step through it

```
# Redefine airlineCsv as a RxTextData object
airlineCsv <- RxTextData(file.path(dataRead,"2007.csv"))
```

1. We first need to create an Rx data object pointing to our data file. Note the capital "R" in Rx denotes that this function defines an object. Copy the following code into your script and step through it.

```
# Redefine airlineCsv as a RxTextData object
airlineCsv <- RxTextData(file.path(dataRead,"2007.csv"))
```

2. We can now run some R commands on it through functionality included in RevoScaleR. Copy the following code into your script and step through it

```
# Some open source R functions
head(airlineCsv)

# Or MRS functions
rxGetInfo(data=airlineCsv)
rxGetVarInfo(data=airlineCsv)
rxGetInfo(data=airlineCsv, getVarInfo = TRUE, computeInfo = TRUE, numRows = 5 )
```

3. All Microsoft R Server scalable functions can operate over text and other external file formats. e.g. ODBC/SQL, SAS etc.

   We also have the MRS External Data Frame (XDF) file format which is a column-compressed, block-based R binary format created by Revolution to optimise performance working with disk based datasets. Copy the following code into your script and step through it.

```
# We will import the data into XDF format
airlineXdf <- RxXdfData(file.path(dataWrite,"2007.xdf"))

# Import the data into xdf format for further exploratory analysis.
# We will time this one off import process
system.time(
  rxImport(inData=airlineCsv, outFile = airlineXdf,
           overwrite = TRUE)
)

# Get some information about the file and data
rxGetInfo(airlineXdf)
# Get some information about the columns and data in the xdf
rxGetVarInfo(data=airlineXdf)

object.size(airlineXdf)
# Note: the data is not stored in memory.  The airlineXdf is an object
# (RxXdfData).
file.size(file.path(dataWrite,"2007.xdf"))
file.size(file.path(dataWrite,"2007.csv"))
# The XDF file is giving us approx 5x saving in space and I/O
```

4. Simple Best Practices for Import

Import performance will vary depending on the data types

Character data is more "expensive" to import.

Importing categorical data can be accelerated by providing colInfo describing the categorical levels in the data

The file blocks are compressed, and the data is stored in 15 blocks based on the default block size of 100000 rows per block. Aim for each block to be circa 10M cells in size. cells = rows x columns. There are arguments that enable multiple blocks per read, but you cannot read less than a block in one I/O!

Copy the following code into your script and step through it.

```
  # Let’s define a colinfo clause to define factors and some data types

  airlineColInfo <- list(
    Year = list(newName = “Year”, type = “integer”),
    Month = list(newName = “Month”, type = “factor”,
                levels = as.character(1:12),
                newLevels = c(“Jan”, “Feb”, “Mar”, “Apr”, “May”, “Jun”,
                              “Jul”, “Aug”, “Sep”, “Oct”, “Nov”, “Dec”)),
    DayOfWeek = list(newName = “DayOfWeek”, type = “factor”,
                     levels = as.character(1:7),
                     newLevels = c(“Mon”, “Tues”, “Wed”, “Thur”, “Fri”, “Sat”,
                                   “Sun”)),
    UniqueCarrier = list(newName = “UniqueCarrier”, type =
                         “factor”),
    TailNum = list(newName = “TailNum”, type = “factor”),
    Origin = list(newName = “Origin”, type = “factor”),
    Dest = list(newName = “Dest”, type = “factor”)
  )
```

```
airlineCsv <- RxTextData(file.path(dataRead,”2007.csv”),
                         colInfo = airlineColInfo)

blockSize = 250000


# We can read a subset (first 100K) to see what the data looks like.
System.time(
  rxImport(inData=airlineCsv, outFile = airlineXdf,
           overwrite = TRUE, colInfo = airlineColInfo,
           rowsPerRead = blockSize,
           numRows = 100000)
)
# Get some information about the file and data
rxGetInfo(airlineXdf)
# Get some information about the columns and data in the xdf
rxGetVarInfo(data=airlineXdf)


# We can also perform on the fly data transformations as we import and
# calculate new features. For this we will create a helper functions
ConvertToDecimalTime <- function( tm ){(tm %/% 100) + (tm %% 100)/60}

system.time(
  rxImport(inData=airlineCsv, outFile = airlineXdf,
           overwrite = TRUE, colInfo = airlineColInfo,
           transforms = list(
             FlightDate = as.Date(as.character((as.numeric(Year)*10000)
                                  +(as.numeric(Month)*100)
                                  +as.integer((
                                    14ormat(as.numeric(DayOfWeek),
                                            width = 2, flag = “0”)))),
                                  format = “%Y%m%d”),
             CRSDepTime = ConvertToDecimalTimeFn(CRSDepTime),
             DepTime = ConvertToDecimalTimeFn(DepTime),
             CRSArrTime = ConvertToDecimalTimeFn(CRSArrTime),
             ArrTime = ConvertToDecimalTimeFn(ArrTime),
             MonthsSince198710 = as.integer((as.numeric(Year)-1987)*12 +
                                       as.numeric(Month) – 10),
             DaysSince19871001 = as.integer(FlightDate – as.Date(“1987-10-01”,
                                                     format = “%Y-%m-%d”))),
           transformObjects = list(ConvertToDecimalTimeFn = ConvertToDecimalTime),
           rowsPerRead = blockSize
  )
)
# Get some information about the file and data
rxGetInfo(airlineXdf)

# Get some information about the columns and data in the xdf
rxGetVarInfo(data=airlineXdf)
# Note: MRS pre-computes and stored the low/high values for numeric data which
# can be useful in optimising other processing.  Categorical data is stored as
# integer "levels" with metadata referencing the factor values.

# MRS provides methods for the following functions for supported data-sources
head(airlineXdf)
tail(airlineXdf)
names(airlineXdf)
dim(airlineXdf)
dimnames(airlineXdf)
nrow(airlineXdf)
ncol(airlineXdf)
```

```
# Read and display some rows from the file
rxReadXdf(airlineXdf, numRows=3)

# We can read from different points in the file
startPoint <- nrow(airlineXdf) - 5
rxReadXdf(airlineXdf, startRow=startPoint)

rxReadXdf(airlineXdf, startRow = 3983467, numRows=3)
```

5. Now we have the data in an optimal format we can perform some Exploratory Data Analysis on the data.

   Copy the following code into your script and step through it.

```
# Compute descriptive statistics using rxSummary()
rxSummary(~ActualElapsedTime + AirTime + DepDelay + Distance,
          data=airlineXdf)
# Note: MRS uses R's formula interface to enable a subset of columns
# to be analysed.
# For all numeric and factor columns you can use
rxSummary(~., airlineXdf,reportProgress = 1)
# or MRS provides a method for summary that does the same thing
summary(airlineXdf)

# Plot some histograms of the data
rxHistogram(~DepDelay, data=airlineXdf, reportProgress = 1 )

# We can remove outliers by restricting the x Axis min and max
rxHistogram(~DepDelay, data=airlineXdf, reportProgress = 1,
            xAxisMinMax=c(-100, 400), numBreaks=500,
            xNumTicks=10)

# We can see if the distribution is different by day of week
rxHistogram(~DepDelay | DayOfWeek, data=airlineXdf, reportProgress = 1,
            xAxisMinMax=c(-100, 400), numBreaks=500,
            xNumTicks=10)
```

6. To determine if delayed flights fly faster we will need the airspeed.

   Copy the following code into your script and step through it.

```
# Add a new feature (derived column) to the data
# rxDataStep processes data in blocks and applies the transforms to each block
rxDataStep(inData=airlineXdf,
           outFile=airlineXdf,
           varsToKeep=c("AirTime", "Distance"),
           transforms = list(AirSpeed = Distance / AirTime * 60),
           append="cols",
           overwrite=TRUE,
           reportProgress = 1)

# Check the new column is there. Couple of ways...
rxGetInfo(data=airlineXdf, getVarInfo=TRUE)
rxGetInfo(data=airlineXdf, getVarInfo=TRUE, varsToKeep="AirSpeed")

# Get summary information and histogram for AirSpeed
rxSummary(~AirSpeed, data=airlineXdf)
rxHistogram(~AirSpeed, data=airlineXdf)
# Clearly we have some outliers and caused by data quality issues

# Lets use the rowSelection argument to remove the obvious outliers!
rxHistogram(~AirSpeed, data=airlineXdf,
            rowSelection=(AirSpeed>50) & (AirSpeed<800),
            numBreaks=5000,
            xNumTicks=20,
            reportProgress = 1)
```

7. Is there an obvious correlation between arrival and departure delay and airspeed.

   Copy the following code into your script and step through it.

```
# Is there an obvious correlation between arrival and departure delay and airspeed
rxCor(formula=~DepDelay+ArrDelay+AirSpeed, data=airlineXdf)

# What if we remove the outliers ?
blah <- rxCor(formula=~DepDelay+ArrDelay+AirSpeed, data=airlineXdf,
      rowSelection=(AirSpeed>50) & (AirSpeed <800),
      reportProgress = 1)
str(blah)

# We can use open-source corrplot function to plot our correlation matrix.
# You may need to install this package if you don't already have it
# install.packages(corrplot)

library(corrplot)
corrplot(blah, method="circle", type = c("upper"))
```

8. As might be expected Arrival Delay is highly correlated to Departure Delay but much less correlated to Airspeed.

   Nevertheless, lets create a linear regression model for airSpeed by Departure Delay.

   Copy the following code into your script and step through it.

```
system.time({
  airline.model <- rxLinMod(formula=AirSpeed~DepDelay, data=airlineXdf,
                            rowSelection=(AirSpeed>50) & (AirSpeed <800),
                            reportProgress = 1)
})

airline.model

summary(airline.model)
```

Conclusion

Indeed! When you are late to leave, the plane flies faster.

Statistically significant faster. But the coefficient is very low - 0.0439 - for each minute of delay.

A staggering 2.6336 mph per each hour of delay!

Further analysis

Maybe the relationship is not monotonic?

- When delay is small, it can be absorbed by the airlines safety margins

- When delays are big, flying faster won't help

- Maybe there is a range in between where it matters?

Lets see do the following :-

- Create a factor variable for each 10 minutes of delay from -10 to 100 minutes
- Create a linear regression factor model.
- Examine the structure of the model object.
- Extract the coefficients of the model to plot them.

Answer

```
# Create a factor variable for each 10 minutes of delay from -10 to 100 minutes
rxDataStep(inData = airlineXdf,
           outFile = airlineXdf,
           transforms = list(
             F_DepDelay = cut(DepDelay, breaks = seq(from = -10, to = 100, by = 10))
           ),
           append = "cols",
           overwrite = TRUE,
           reportProgress = 1)

# Create a linear regression factor model
airFactorMod <- rxLinMod(formula = AirSpeed ~ F_DepDelay + -1,
                                 data = airlineXdf,
                                 rowSelection = (AirSpeed > 50) & (AirSpeed < 800),
                         cube = TRUE,
                         cubePredictions = TRUE,
                         reportProgress = 1)

# model summary
summary(airFactorMod)
# Let's examine the structure of the model object
str(airFactorMod)

# We can extract the coefficients of the model to plot them
plotDF <- data.frame(seq(from = -10, to = 90, by = 10),airFactorMod$coefficients)
rownames(plotDF) <- NULL
names(plotDF)[1] <- "DepDelayRange"
   rxLinePlot(AirSpeed ~ DepDelayRange,data = plotDF)
```

Terms of Use