Microsoft

Apache Spark on Microsoft Azure HDInsight: Using Spark MLLib in HDInsight

Spark™

Change management

Version	Date Effective	Incorporated Changes	Requested By
1.0	06/01/2016	Initial Version	Nishant Thacker
1.1	07/12/2016	Formatting changes	Nishant Thacker
1.2	01/29/2017	TechReady 24	Nishant Thacker
1.3	7/17/2017	Ready 2017	Nishant Thacker

Contents

Introduction

This class specifically focuses on Spark ML component of Spark and highlights its value proposition in the Apache Spark Big Data processing framework.

Spark ML is Apache Spark's scalable machine learning library.

The main highlights of Spark ML are as follows (source: https://spark.apache.org/mllib/):

 # 1 Ease of Use.

Spark ML fits into Spark's APIs and one can use any Hadoop data source (e.g. HDFS, HBase, or local files), making it easy to plug into Hadoop workflows. Usable in Java, Scala, Python, and SparkR.

#2 Performance.

Spark ML runs high-quality algorithms about 100x faster than MapReduce.

#3 Easy to Deploy.

Spark ML, along with Spark runs on existing Hadoop clusters and data.

Business Use Case

Product Recommendations and Personalized Marketing for a retail website. For ex: ‘people like you also like this’.

Takeaways

The students will learn how online retailers provide product recommendations and market personalized items to them, while they are shopping online.

They will learn the machine learning algorithms, understand the data sets required to perform these computations and the overall process flow that brings it all together from a site visitor's perspective.

The ‘People Like You Also Like This’ use case: The algorithm to enable this functionality is implemented in SparkML and explained here:

Collaborative Filtering

http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html

This algorithm looks deep into site visitor's profile, previous browsing and shopping history. Additionally, it looks into the sales history of current items in shopping basket, general sales history of other items bought together with the current item, current item's product characteristics and properties and develop correlation's with other products in their catalog.

Ready special instructions

HDInsight cluster usually take 15-20 minutes to create. As a work around to the cluster create time, we've pre-provisioned HDI clusters for this lab and are sharing the credentials below. Note that these clusters are only active for TechReady and will be deleted after the event, so if you're trying the labs after TR, you should create your own clusters and use the cluster properties to proceed with the lab.

These credentials are shared in good faith and the understanding is that attendees will not misuse these for any purposes, including but not limited to this lab. If you have any concerns, please close this lab now and do not proceed any further.

Ready Cluster Credentials:

Note: The steps in the following section 'Provision HDInsight Linux Hadoop cluster with Azure Management Portal ' *should be ignored if you are provided a shared cluster. For Ready you're provided a cluster with the following credentials:*

Spark Cluster:

Cluster Name for Spark Cluster: nthdilabsspark##

Cluster URL (Ambari) for Hive Cluster: https://nthdilabsspark##.azurehdinsight.net/

Username: admin

Password: HDItut@123

Jupyter Notebook URL: https://nthdilabsspark##.azurehdinsight.net/jupyter/tree#notebooks

Username: admin

Password: HDItut@123

Class

Section 1: Provision an HDInsight Spark cluster

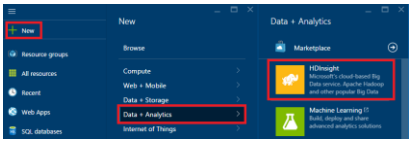
(Ignore for Ready as a cluster is already provisioned and provided)

Access Azure Preview Portal

1. Sign in to the **Azure preview portal**.

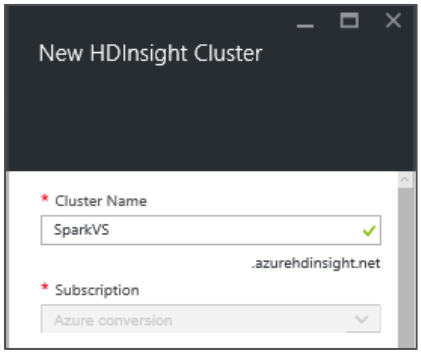
Create HDInsight Spark cluster

1. Click **NEW**, click **Data + Analytics**, and then click **HDInsight**.



Provide Cluster Details

1. In the New HDInsight Cluster blade , enter **Cluster Name.**



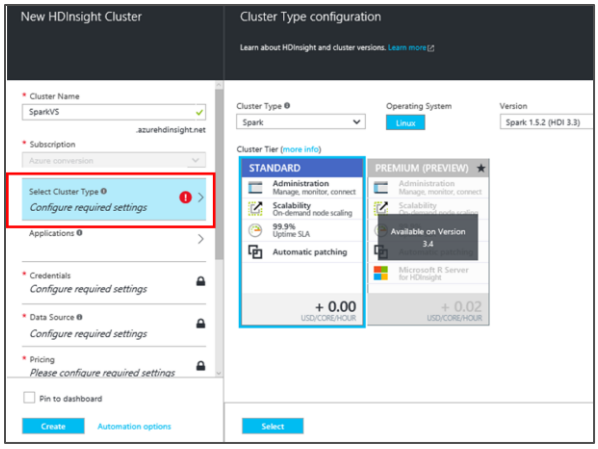
A green check mark appears beside the cluster name if it is available.

2. For **Subscription**, select **Azure Conversion.** If you have more than one subscription, click the Subscription entry to select the Azure subscription to use for the cluster.

1.

Configure Cluster Type

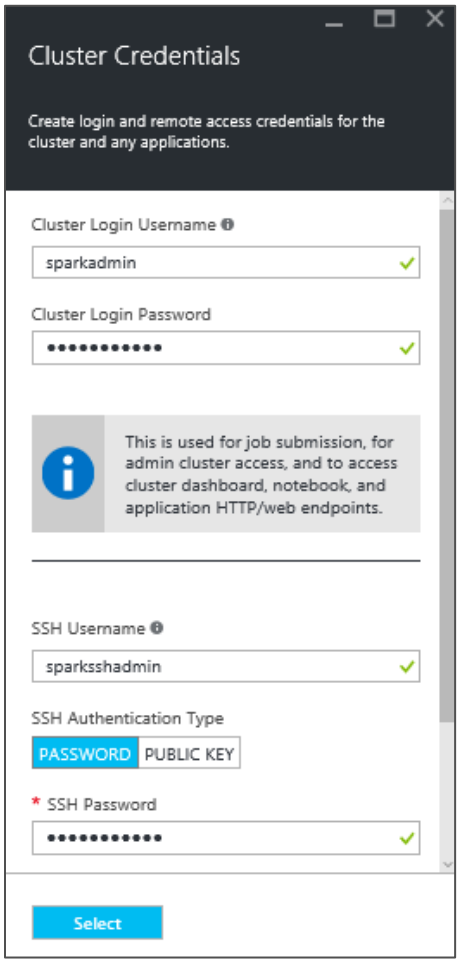
1. Click on **Select Cluster type.** This will open the Cluster Type configuration blade.



2. Select **Spark** as **Cluster Type.**
3. **Operating System** will be **Linux** by default.
4. Select **Version** as **Spark 1.6.2 (HDI 3.4)**
5. For **Cluster Tier**, select **STANDARD**
6. Click **SELECT** to complete the configuration settings

Provide credentials to access cluster

1. Click **Credentials** tab to open **Cluster Credentials blade.**



2. Enter **Cluster Login Username.**

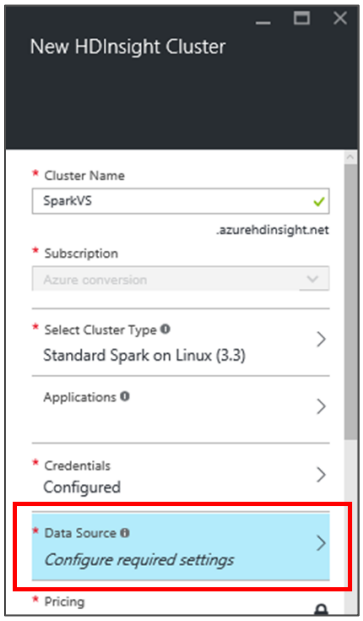
3. Enter **Cluster Login Password**.
4. Enter **SSH Username**.
5. Select **PASSWORD** as **SSH Authentication Type**.
6. Enter **SSH Password** and confirm it.
7. Click **Select** button to save the credentials.

**SSH Username and Password is required to remote session of Spark Cluster*

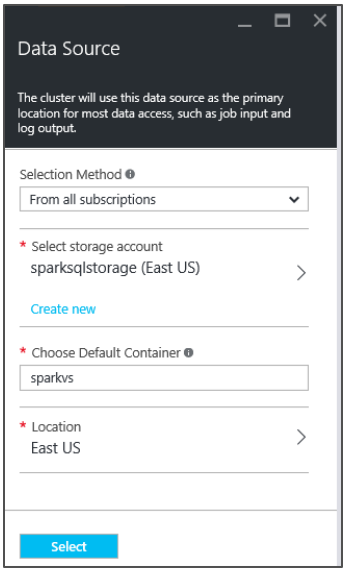
Provide Data Source

Data source will be used as primary location for most data access, such as job input and log output.

1. Click **Data Source tab** present on New HDInsight Cluster blade



2. In the Data source blade**, Selection Method** provides two options:



Option 1 - Set this to **Access Key** if you want to use existing storage account and you have **Storage Name** and **Access Key** of same, else

Option 2 - select **From all subscriptions** as Selection method.

Select **From all subscriptions** for purpose of this Lab exercise

3. To create new storage account enter a name for new storage account in **Create New Storage Account** input box

   Or
   Click on link **Select Existing** to select from existing accounts.
   For purspose of this exercise, we will create a new storage account.

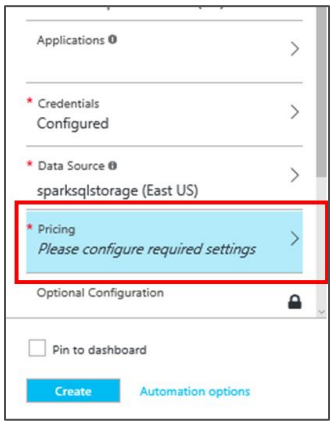
4. Enter name for default container to be designated for cluster in **Choose Default Container** field**.**
5. If existing storage account is selected then no need to provide **Location** else select appropriate **Location**.

   By default, the HDInsight cluster is provisioned in the same data center as the storage account you specify

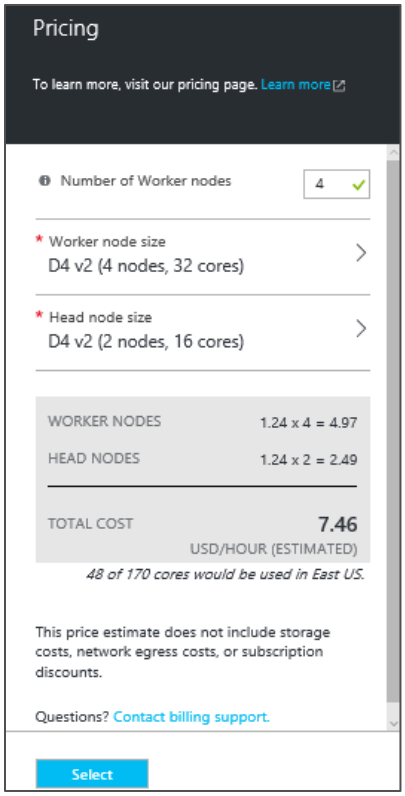
6. Click Select button at the bottom to save the data source configuration.

Set Pricing

1. Click **Pricing** tab to open the Pricing blade.



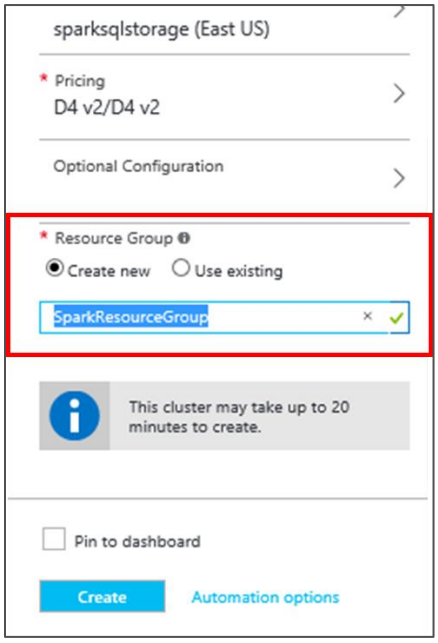
2. The Pricing blade provides options to the configure number of nodes in cluster, which will be the base pricing criteria. Enter number of worker node in **Number of Worker nodes** field, set it to **4** for this demo.



3. Leave all other values as default.
2. Note that based on the number of worker notes and size, the estimated cost of the cluster is calculated and displayed in USD/HOUR.
4. Click **Select** button to save node pricing configuration.

Provide Resource Group

1. In the **Resource Group** section, you have options:



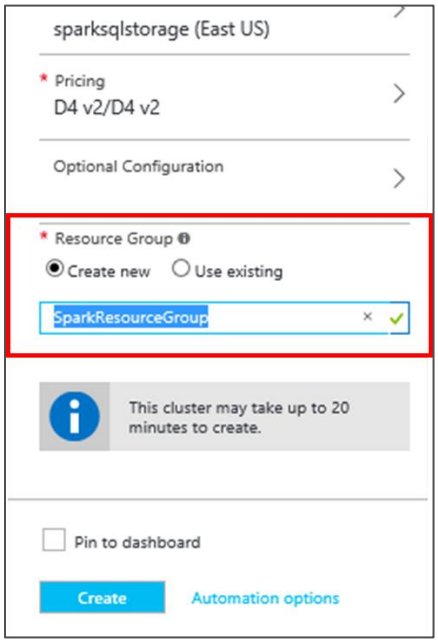
a) Click link Create New to provide new Resource Group.

Or

b) Use Existing by selecting appropriate Resource Group from list.

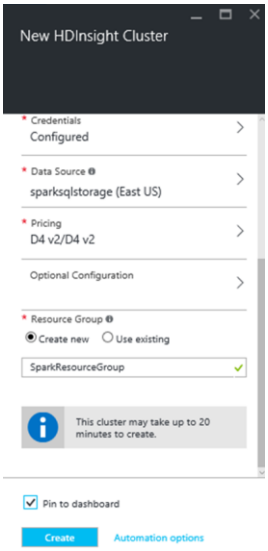
3. For the purpose of this lab, we will create a new resource group.
2. Enter name for Resource Group

   A green check appears beside the cluster name if this name is available.



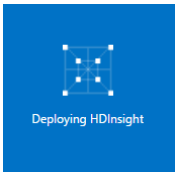
Provision cluster

1. After completing all the configraution, in the New HDInsight Cluster blade, make sure to tick on the 'Pin to dashboard' option.
2. Click **Create** button to finalize cluster creation.



This creates the cluster and adds a tile for it to the **Startboard** of your Azure portal.

The icon will indicate that the cluster is provisioning, and will change to display the HDInsight icon once provisioning has completed.



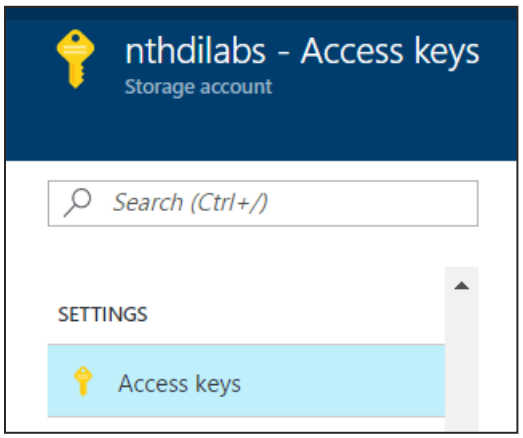
Section 2: Load datasets files to storage account.

In this section, you'll copy the files required for the lab to your storage account. You'll copy the files between two storage acount with the help of AzCopy utility. You can download the utility from here http://aka.ms/downloadazcopy

Note: Ignite Step 1 and 2 if you are provided a shared cluster (for TechReady)

To copy the files, follow the below steps.

1. Copy your Azure Storage account access keys. This is required to copy data from the source Azure Storage account to your Azure Storage account. To get your storage account access key, navigate to your storage account on the Azure Management Portal and select **Access keys** under **Settings**.



2. Click on the copy icon to copy **Key1** from the **Access Keys** pane.

3. Press Window + R to open the run window. Type cmd and press enter to open a new command console window.

4. Change the directory to C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy.

5. Copy and paste the following command on the console window to transfer **all spark lab assets needed** from the source storage account to your storage account.

```
AzCopy /Source:https://nthdilab.blob.core.windows.net/sparklabs/
/Dest:https://nthdilabs.blob.core.windows.net/nthdilabsspark##container/<yournam
e>/SparkLabDatasets/
/SourceKey:G1t6T13jn3K6w/4ZS2NG7RsTHe5YuusRLd9tKnRlJku7cjCwcRk5JxWAHmtrH1Dt03+nw
ttYbB2HuvHeI/UiNw==
/DestKey:QfPYhbGxokkcqjjEZIkbzzyDdMb7QHSrUjA6UnpfuLjC0Np4PSSjkfrsnVhGyOxJsKWEK9C
XNvPZo/L1w7T0ow== /S
```

Note: Replace <yourname> with your name or a unique ID, so that your files do not get processed by other jobs. Also replace ## with your cluster number, so that you only process your data.

Section 3: Creating a new Jupyter Notebook

For Ready, use the following credentials.

Jupyter Notebook URL: https://nthdilabsspark##.azurehdinsight.net/jupyter/tree#notebooks

Username: admin

Password: HDItut@123

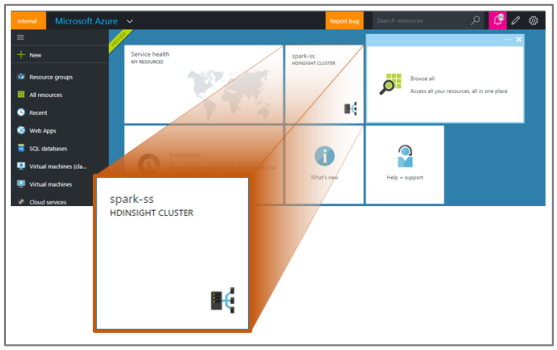
If creating your own cluster, use the instructions below, else skip to 'Create a new notebook'.

Access Azure Portal

1. Sign in to the Azure Portal.

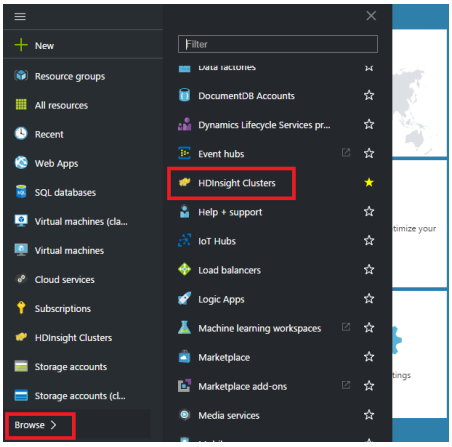
If Spark Cluster is pinned to the "StartBoard":

2. Click the tile for your Spark Cluster.

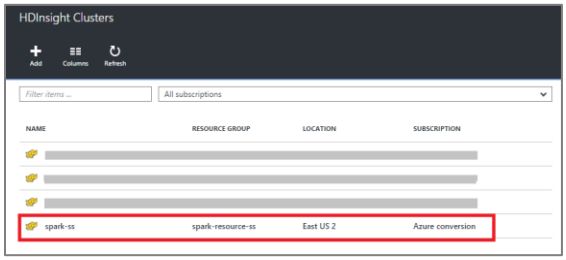


If Spark Cluster is not pinned to the "StartBoard":

1. Click Browse, select HDInsight Clusters.

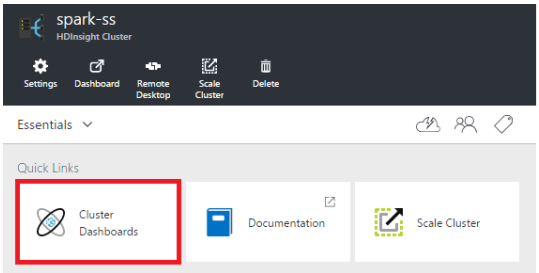


2. Select your Spark Cluster.

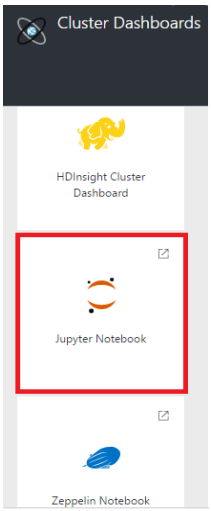


Launch Jupyter Notebook

1. Click on Cluster Dashboards tile displayed under the Quick Links of Cluster Blade.

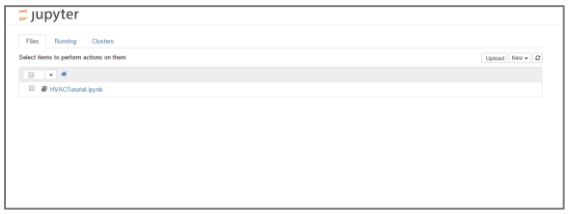


2. Locate **Jupyter Notebook** tile on Cluster Dashboards tile and click on it.



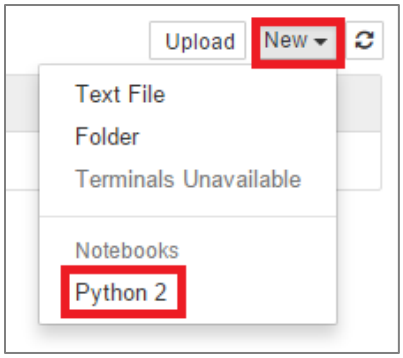
3. When prompted, enter the admin credentials for the Spark cluster.

This will open the Jupyter dashboard.



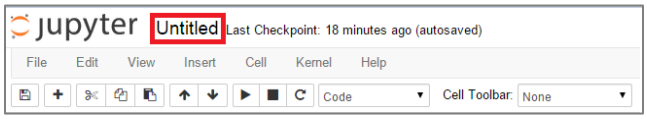
Create a new notebook

1. Click **New** dropdown button present at top right side of Jupyter Notebook screen.
2. Click Python 2 under Notebooks.

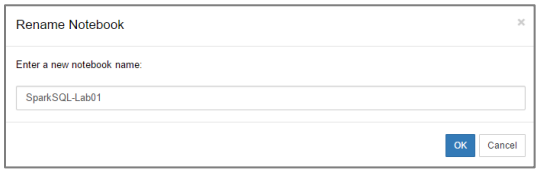


Assign a friendly name to the newly created notebook

1. A new notebook is created and has the default name: **Untitled.pynb.**
2. Click the notebook's name at the top to rename it.



3. Enter new notebook name as SparkSQL-Lab02 and press OK



Create the Spark context

The atexit module defines functions to register and unregister cleanup functions. Functions thus registered are automatically executed upon normal interpreter termination.

Import the required modules and create the Spark contexts.

1. Paste the following snippet in an empty cell.

```
import pyspark
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import SQLContext
import atexit
```

```
sqlContext = SQLContext(sc)
atexit.register(lambda: sc.stop())
```

```
In [ ]: import pyspark
        from pyspark import SparkConf
        from pyspark import SparkContext
        from pyspark.sql import SQLContext
        import atexit
        sc = SparkContext('yarn-client')
        sqlc = SQLContext(sc)
        atexit.register(lambda: sc.stop())
```

Here you are creating a SparkContext object, which is the main entry point for Spark functionality. A SparkContext represents the connection to a Spark cluster, and can be used to create RDDs, accumulators and broadcast variables on that cluster.

Press **SHIFT + ENTER**. Or press the **Play** button from tool bar to execute the code inside the cell. Wait till kernel becomes idle. 

2. On execution, you will see a confirmation of SparkContext creation.

```
Out[1]: <function __main__.<lambda>>
```

Section 4: Working with data

Import MLib namespace for recommendation

1. Paste the following snippet in an empty cell.

```
from pyspark.sql import *
from pyspark.mllib.recommendation import *
fdir = "/<yourname>/SparkLabDatasets/Lab03/"
salesf = fdir + "SaleTransactions.csv"
```

2. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.

Write a function to generate RDD from data lines

1. Paste the following snippet in an empty cell.

```
def toSalesRec(inline):
    l = inline.split(",")
    return
Row(SaleId=l[0],SessionDateTime=l[1],CustomerAction=l[2],CustomerId=int(l[3]),BookId=int
(l[4]),
               Name=l[5],BookPrice=l[6],Quantity=l[7],
TotalCharges=l[8],Transaction=l[9],OrderId=l[10])
```

2. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.

This function accepts the comma delimited string, splits that string based on comma and converts it into RDD.

Load data from BLOB storage and convert it into RDD

1 - **sc.textFile (string FilePath):** This function accepts file path in string format and loads the data at specified location

2 - **ActionPoints** defines key-value pair where key defines action item and value defines rating

3. Paste the following snippet in an empty cell.

```
print "Loading the Sales Data ..\n"
salesfrdd  = sc.textFile(salesf)
ActionPoints = {"Browsed":3, "Added to Cart":7, "Purchased":10}
salesrdd = salesfrdd.filter(lambda l: l[0] != ',' and l[0].isdigit()).map(lambda l:
toSalesRec(l)).filter(lambda s: s.CustomerAction in ActionPoints.keys())
print "Sales Data loaded..."
```

4. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
5. This code snippet loads data from BLOB storage and converts the data lines to stream of RDDs using function defined in previous step.
6. Wait till kernel becomes idle.

Convert the RDD stream to Ratings RDD format

1. Paste the following snippet in an empty cell.

```
print "Creating sales records from the Sales Data ..\n"
ratingsRdd = salesrdd.map(lambda s: [s.CustomerId, s.BookId,
ActionPoints[s.CustomerAction]])
print (repr(ratingsRdd.take(10)))
```

2. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
3. This code snippet converts salesrdd to ratingsRdd having format (UserId,ProductId,rating) required for ALS algorithm.
4. Refer output

```
Creating sales records from the Sales Data ..

[[469, 17935, 3], [263, 6050, 10], [131, 15182, 7], [140, 16324, 10], [474, 240
0, 3], [254, 5890, 7], [467, 493, 7], [142, 3904, 7], [418, 5208, 10], [324, 16
033, 3]]
```

Create trained ALS model

1 - ALS.trainImplicit (ratings, rank, iterations=5, lambda_=0.01, blocks=-1, alpha=0.01, nonnegative=False, seed=None)

ALS.trainImplicit train a matrix factorization model given an RDD of ‘implicit preferences’ given by users to some products, in the form of (userID, productID, preference) pairs. We approximate the ratings matrix as the product of two lower-rank matrices of a given rank (number of features). To solve for these features, we run a given number of iterations of ALS. This is done using a level of parallelism given by blocks.

2 - rank = 10 - means it has a collective rank of higher or equal to 5

3 - numIterations = 5 - means it has occurred atleast 5 times

1. Paste the following snippet in an empty cell.

```
rank = 10
numIterations = 5

print ("Creating an implicit / indirect prediction model (ALS.trainImplicit()) from the
Customer browsing history .. ")
model = ALS.trainImplicit(ratingsRdd, rank, numIterations, alpha=0.02)
```

```
savef = '%s/userRecommendationModel' %fdir
model.save(sc, savef)
```

2. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
3. Wait to see the Kernel status turn idle.
4. This code snippet creates trained ALS model using **ALS.trainImplicit** function and saves that model to BLOB storage at location **savef**

Section 5: Recommending products based on Collaborative Filtering

Import Namespaces

1. Paste the following snippet in an empty cell.

```
from ipywidgets import widgets
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating
from IPython.display import display
from pyspark.sql import SQLContext
```

2. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
3. This code snippet import namespaces for getting recommendation, display input box, import python widgets

Write Recommendation Function

1 - **MatrixFactorizationModel.load (sparkContext,path)** loads trained model, by accepting the object of sparkContext and path of location where trained data model is saved in earlier steps.

2 - **recommendedProductIds = sameModel.recommendProducts(customerId, 8)**: This statement gets recommended products from trained model based on provided customerId

3 - **recommendedProducts = recommendedProductsByIds.join(prodnamesRdd)**: This statement joins the **recommendedProductsByIds** RDD with **prodnamesRdd** based on **Product** to get details of product and stores final RDD to **recommendedProducts**

4 - **rpdf = recommendationDetails.toDF()**: This statement converts **recommendationDetails** to dataframe.

5 - **rpdf.write.save (path=savef,source='parquet',mode='overwrite')**: This statement saves the data in dataframe to path **savef** in BLOB storage in **paraquet** format with **overwrite** mode

6 **rpdf.registerTempTable("recommendedProducts")**: This statement creates temporary SQL table **recommendedProducts** based on **rpdf** dataframe and stores data in this table.

1. Paste the following snippet in an empty cell.

```
customerId=int("469")
print(customerId)
loadf = '%s/userRecommendationModel' %fdir
sameModel = MatrixFactorizationModel.load(sc, loadf)
recommendedProductIds = sameModel.recommendProducts(customerId, 8)
print('Recommended product Ids for user ' + str(customerId) +
'\n'.join(map(repr,recommendedProductIds)) + '\n')
prodnames = set(salesrdd.filter(lambda s: s.BookId in [rp[1] for rp in
recommendedProductIds]).map(lambda s: (s.BookId, s.Name)).collect())
recommendedProductsByIds = sc.parallelize(recommendedProductIds).keyBy(lambda b:
b.product)
prodnamesRddRaw = sc.parallelize(prodnames)
prodnamesRdd = prodnamesRddRaw.keyBy(lambda s: s[0])
recommendedProducts = recommendedProductsByIds.join(prodnamesRdd)
recommendationDet=recommendedProducts.map(lambda b:b[1])
recommendationDetails=recommendationDet.map(lambda
b:Row(userid=b[0].user,productid=b[0].product,rating=b[0].rating,productdetail=b[1]))
print(recommendationDetails.collect())
```

```
print('Find recommended products for user ' + str(customerId) + ':\n')
print('\n'.join(map(repr,recommendationDetails.collect())) + '\n')
savef = '%s/recommendedProducts' %fdir
print('Saving recommended products to %s ..\n' %savef)
rpdf = recommendationDetails.toDF()
rpdf.write.save(path=savef,source='parquet',mode='overwrite')

rpdf.registerTempTable("recommendedProducts")
sqlContext.sql("select * from recommendedProducts").show()
```

2. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
3. This function gets recommendations from saved trained ALS model.
4. Joins recommendation set with original dataset based on ProductId to get product details of recommended products
5. Final dataset is saved in BLOB storage as paraquet file and temporary SQL table.
6. Try out some Customer IDs for recommendations:

469
263
131
140
474
254
467
142

Section 6: Work with data for FP-Growth

Load data from BLOB storage and convert it into RDD

1 - **sc.textFile (string FilePath)**: This function accepts file path in string format and loads the data at specified location

7. Paste the following snippet in an empty cell.

```
import pyspark
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import *

fdir = "/<yourname>/SparkLabDatasets/Lab03/"
salesf = fdir + "SaleTransactions.csv"

sqlContext = SQLContext(sc)

print("Loading sales data from the cloud and parsing into records ..")
salesfrdd  = sc.textFile(salesf)
```

8. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
9. This code snippet creates SparkContext object and loads data from BLOB storage

```
import pyspark
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import *

fdir = "/LabData/"
salesf = fdir + "SaleTransactions.csv"

sc = SparkContext('yarn-client')
sqlContext = SQLContext(sc)

print("Loading sales data from the cloud and parsing into records ..")
salesfrdd  = sc.textFile(salesf)
```

```
Loading sales data from the cloud and parsing into records ..
```

Write a function to generate RDD from data lines

10. Paste the following snippet in an empty cell.

```
def toSalesRec(inline):
    l = inline.split(",")
    return
Row(SaleId=l[0],SessionDateTime=l[1],CustomerAction=l[2],CustomerId=l[3],BookId=l[4],
                Name=l[5],BookPrice=l[6],Quantity=l[7],
TotalCharges=l[8],Transaction=l[9],OrderId=l[10])
```

11. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
12. This function accepts the comma delimited string, splits that string based on comma and converts it into RDD.

Write a filter function which filters RDD based on CustomerAction and Transaction

13. Paste the following snippet in an empty cell.

```
def sfilter(salesrec):
    return salesrec.CustomerAction in ["Added to cart","Purchased"] and not
salesrec.Transaction in ["Cancelled", "Failed"]
```

14. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
15. This function filters RDD based on CustomerAction and Transaction, allows only successful purchase and ‘Added To Cart’ transactions.

Convert data lines to filtered RDD stream

16. Paste the following snippet in an empty cell.

```
all_salesrdd = salesfrdd.filter(lambda l: l[0] != ',').map(lambda l: toSalesRec(l))
salesrdd = all_salesrdd.filter(sfilter)
```

17. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
18. This code snippet removes blank data lines and converts data lines to RDD using function **toSalesRec.**
19. Filters out RDD which denotes failed or Cancelled Purchase transactions using filter function sfilter.

Aggregate RDDs based on OrderId

20. Paste the following snippet in an empty cell.

```
# Group by the orderIds to get the item sets
ordergroups = salesrdd.groupBy(lambda r: r.OrderId)
ordertxns = ordergroups.mapValues(lambda ll: set([l.BookId for l in ll]))
#print out a couple of samples
print("Printing some of the BookId sets contained within the Orders:\n ")
otake = ordertxns.take(3)
print('\n'.join(map(repr,otake)))
```

21. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
22. This code snippet brings the data in (transaction(t),(items (i1),(i2),(i3)..)) format required for FP Growth Algorithm.
23. Output similar to image marks successful execution of code.

```
Printing some of the BookId sets contained within the Orders:

(u'B6B10105-CD8A-4BF9-8929-F086BCD5D684', set([u'16266']))
(u'FF3DEDD6-0FA4-452D-83AD-282BF15A9DC3', set([u'20857', u'12417']))
(u'66200A77-4103-4198-85B9-D2560119048C', set([u'13368', u'11912', u'2940', u'8
674']))
```

Section 7: Train FP-Growth algorithm and find frequent itemsets

Train FP-Growth algorithm

1 - **FPGrowth.train(data, minSupport, numPartitions)**: Computes an FP-Growth model that contains frequent itemsets.

Parameters:

data - The input data set, each element contains a transaction.

minSupport - The minimal support level.

numPartitions - The number of partitions used by parallel FP-growth.

24. Paste the following snippet in an empty cell.

```
from pyspark.mllib.fpm import FPGrowth
minEntries = 3
maxEntries = 10
minSupport = 3
ngroups = ordertxns.count()
minSupportFrac = minSupport /ngroups
model = FPGrowth.train(ordertxns.values(), minSupport=minSupportFrac, numPartitions=10)
```

25. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
26. This code snippet import namespaces for FP-Growth algorithm and prepares the trained model using dataset created in previous section (Section-2)

Create filter function for item entries

27. Paste the following snippet in an empty cell.

```
def ilen(iset):
    return len(iset.items) >= minEntries and len(iset.items) <= maxEntries
```

28. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
29. This code snippet defines a filter which filters the frequent items set to contain entries having item count between **minEntries** and **maxEntries**.

Collect frequent item sets

freqItemsets(): Returns the frequent itemsets of this model.

30. Paste the following snippet in an empty cell.

```
fsets = model.freqItemsets().collect()
fsets = filter(ilen, fsets)
print("Num fpgrowth results: " + str(len(fsets)) + '\n')
print("Some item sets: \n")
print('\n'.join(map(lambda f: repr(f), fsets[0:5])))
```

31. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
32. This code snippet collects the frequent itemsets ad filters them using filter **ilen** defined in earlier step,

33. Output of above code resembles with image

```
Num fpgrowth results: 135

Some item sets:

FreqItemset(items=[u'5855', u'15071', u'3178'], freq=1)
FreqItemset(items=[u'1054', u'3493', u'1369'], freq=1)
FreqItemset(items=[u'65', u'6530', u'7409'], freq=1)
FreqItemset(items=[u'12643', u'2469', u'12927'], freq=1)
FreqItemset(items=[u'12643', u'5043', u'2469'], freq=1)
```

Define sql table schema

34. Paste the following snippet in an empty cell.

```
from pyspark.sql.types import *
dataSchema = StructType([StructField("item1", StringType(), True),
                         StructField("item2", StringType(), True),
                         StructField("item3", StringType(), True),
                        StructField("freq", StringType(), True)])
```

35. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
36. This code snippet define table schema for sql table.

Store frequent item sets to Paraquet file and temporary SQL table

1 - **df = rowRdd.toDF()**: This statement converts **rowRdd** to dataframe.

2 - **df.write.save(path=fresults, source='parquet', mode='overwrite')**: This statement saves the data in dataframe to path **fresults** in BLOB storage in **paraquet** format with **overwrite** mode

3 - **df.registerTempTable ("fpgResults")**: This statement creates temporary SQL table **fpgResults** based on **df** dataframe and stores data in this table.

37. Paste the following snippet in an empty cell.

```
rowRdd = sc.parallelize(fsets)
rowRdd1 = rowRdd.map(lambda f: Row(f.items[0],f.items[1],f.items[2],f.freq))
df = sqlContext.createDataFrame(rowRdd1,dataSchema)
fresults = fdir + '/fpgResults'
df.write.save(path=fresults,source='parquet',mode='overwrite')

df.registerTempTable("fpgResults")
```

38. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
39. This code snippet converts RDD to DataFrame and stores data in BLOB storage as Paraquet file and temporary SQL table.

Write event handler

1 - **sqlContext.sql(string query)**: Queries temporary SQL table based on sql query.

40. Paste the following snippet in an empty cell.

```
productId="5855"
print(productId)
query="select count(*) as counter,item1,item2,item3 from fpgResults where
(item1='"+productId+"' OR item2='"+productId+"' OR item3='"+productId+"') AND freq>0
group by item1,item2,item3 order by counter desc"
```

```
sqlContext.sql(query).show()
```

41. Try with other product IDs from the FreqItemSets above
42. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.
43. This event handler gets triggered on submit event of inputbox, used to perform query on temporary SQL table to get frequently bought products with product provided by user.

Bonus Section: Integrating R with Spark

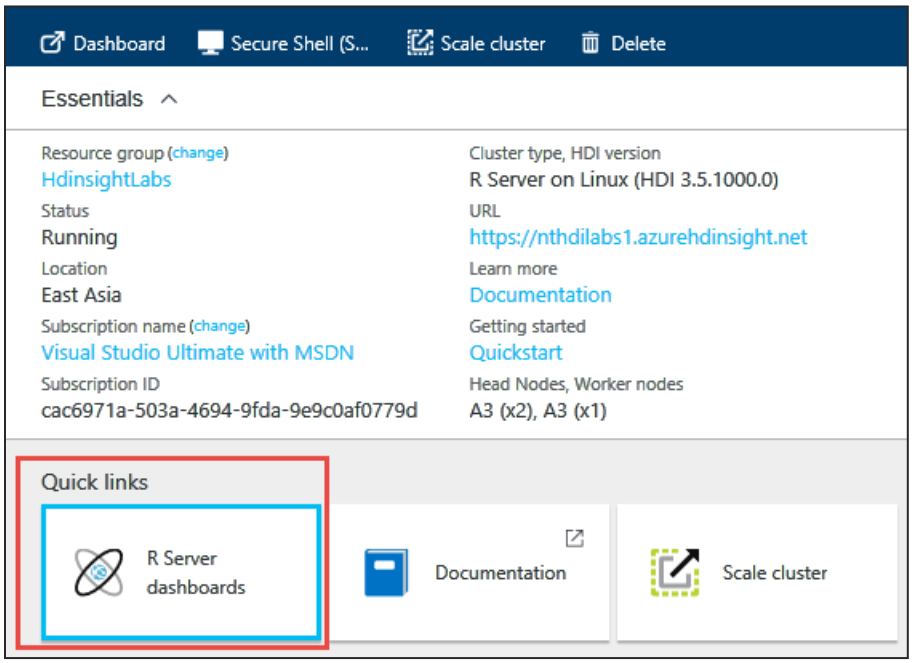
Executing R Script in RStudio

In this section, you'll learn to execute R Script with R Studio Community Edition.

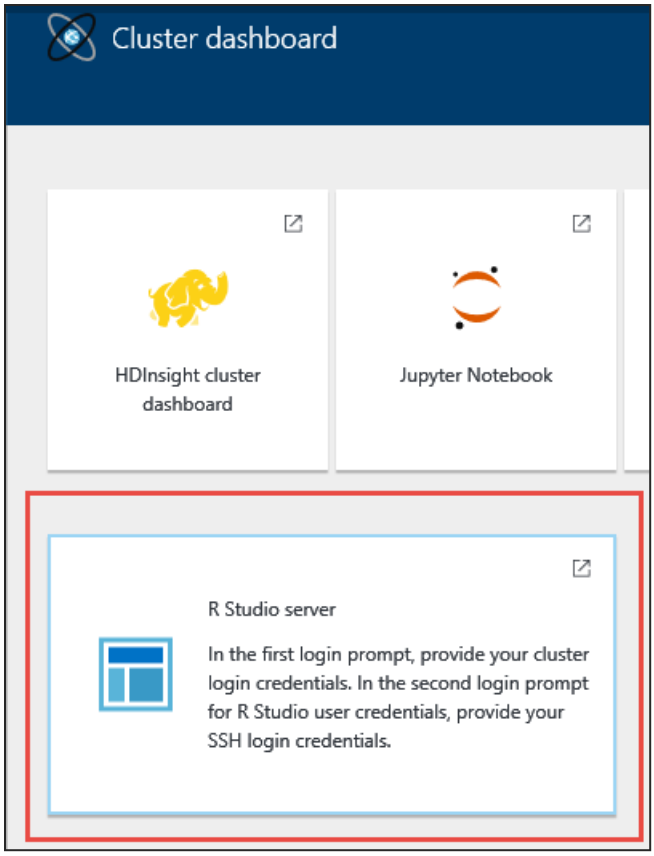
R Studio Community Edition

The R Studio Community edition is installed when provisioning R Server on HDInsight. To connect to the R Studio, follow the below steps

1. Navigate to the R Server on the Azure Management Portal, and select R Server Dashboards.

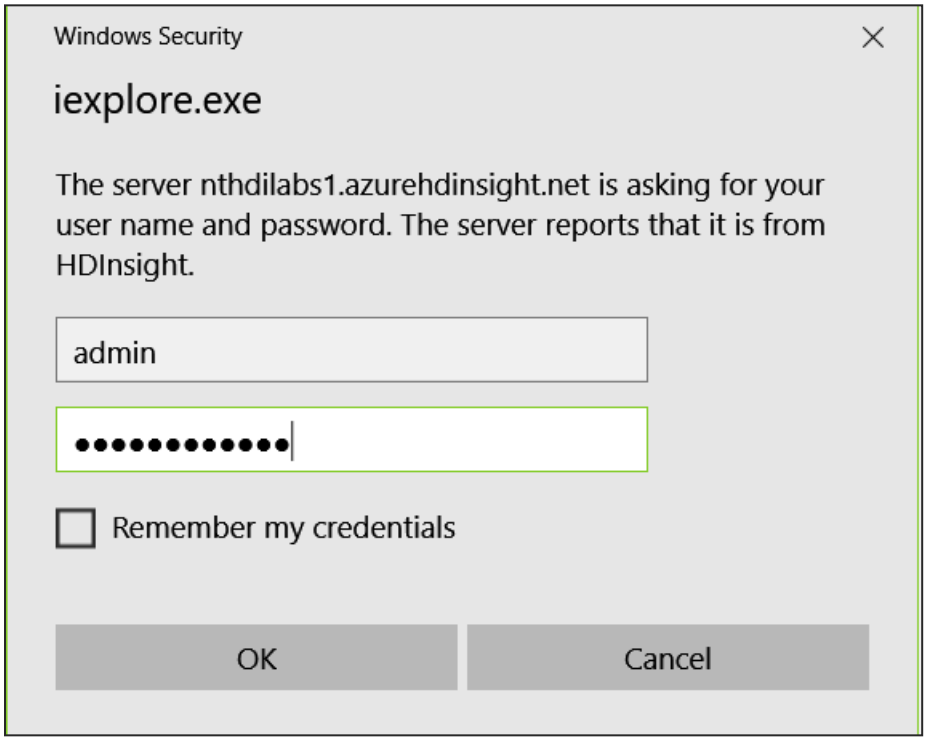


2. In the Cluster Dashboard pane, select R Studio Server and enter the passwords as mentioned in the tile.



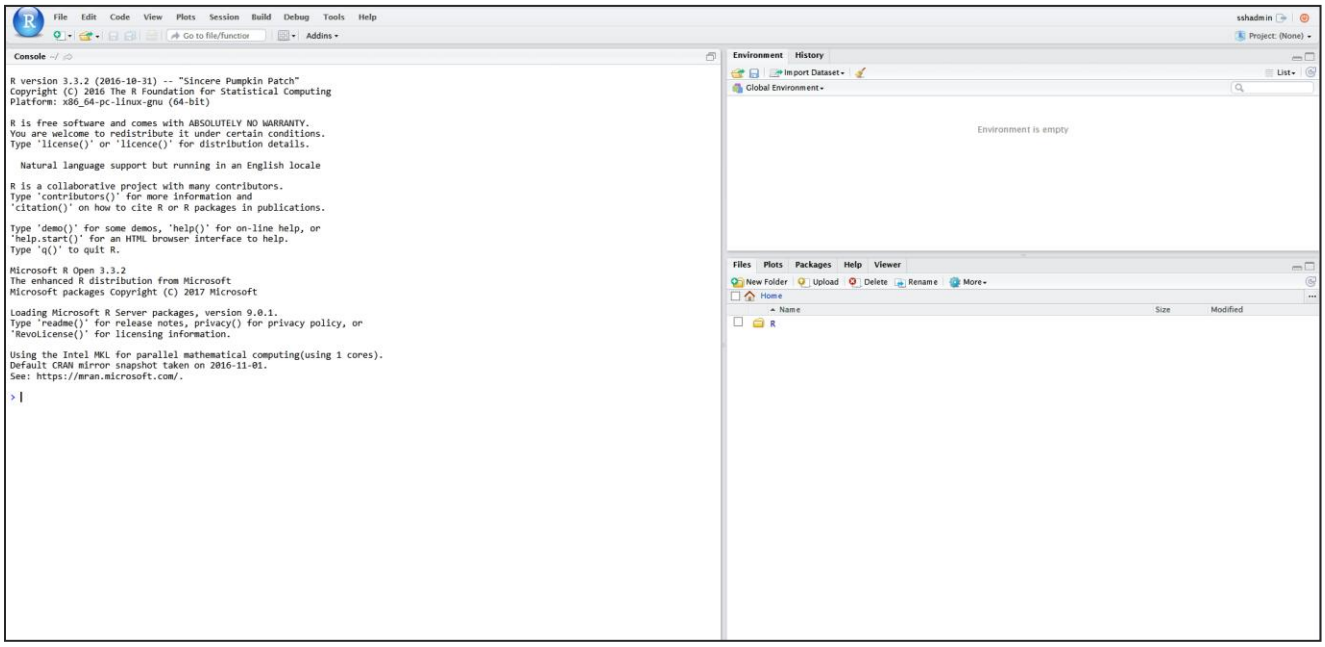
Note: You can also login R Studio server by opening https://nthdilabs.azurehdinsight.net/rstudio and following the below instructions. In case you are not using a shared cluster, replace nthdilabs with the name of the server you created under section "Provision HDInsight Linux R Server with Azure Management Portal".

3. In the first login prompt enter Cluster Admin Credentials



4. In the second login prompt, enter the SSH credentials.

5. You are now connected to R Studio.



6. Copy and paste the R Script in the R Studio console.

```
hdfsFS <- RxHdfsFileSystem()

# import data
fruits <- RxTextData("/example/data/fruits.txt",fileSystem=hdfsFS)

# get top 6 rows
head(fruits)
```

Note: The R script, modifies the storage context to HDFS file system. It then reads fruits.txt file located in R Server HDInsight cluster default storage account, ***example/data/fruits.txt*** *file.*

You should get the following output

```
>
> hdfsFS <- RxHdfsFileSystem()
>
> # import data
> fruits <- RxTextData("/example/data/fruits.txt",fileSystem=hdfsFS
>
> # get top 6 rows
> head(fruits)
         apple
1       banana
2 canary melon
3        grape
4        lemon
5       orange
6    pineapple
```

Importing Data from Azure Storage

In this section, you'll learn to import data into R Server from Azure Storage. You'll also learn about the xdf file format. We will download the data from a shared BLOB storage location to local disk, and then copy it to the default storage account.

The weblogs data has click stream data for the imaginary book store. The data description is given below.

Column	Description
TransactionDate	The date of the transaction
CustomerId	Unique Id assigned to the customer
BookId	Unique id assigned to a book in the book store
PurchaseType	1. **Purchased:** Customer bought the book 2. **Browsed:** Customer browsed but not purchased the book. 3. **Added to Cart:** Customer added the book to the shopping cart
TransactionId	Unique Id assigned to a transaction
OrderId	Unique order id
BookName	The name of the book accessed by the customer
CategoryName	The category of the book accessed by the customer
Quantity	Quantity of the book purchased. Valid only for PurchaseType = Purchased
ShippingAmount	Shipping cost
InvoiceNumber	Invoice number if a customer purchased the book
InvoiceStatus	The status of the invoice
PaymentAmount	Total amount paid by the customer. Valid only for PurchaseType = Purchased

To read the weblogs data into a data frame, follow the below steps.

Copy and paste following scripts one by one in Studio console.

1. To read the data from weblogs.csv, execute the following script in R Studio console window.

```
# download files to local
```

```
download.file("https://techready2017.blob.core.windows.net/techready2017/weblogs.csv",
"weblogs.csv")

# copy file to wasb storage
library(RevoScaleR)
rxHadoopCopyFromLocal("weblogs.csv", "wasb:///weblogs.csv")


# change the storage context
myNameNode <- "wasbs://<yourcontainername>@partnerairlift2017.blob.core.windows.net"
myPort <- 0

# change fileSystem to HDFS
hdfsFS <- RxHdfsFileSystem(hostName=myNameNode, port=myPort)

# import data into dataframe
weblogsDF <- RxTextData("/weblogs.csv", fileSystem=hdfsFS)

# get top 6 rows
head(weblogsDF)
```

Note: Replace <yourcontainername> with the name of the container you created in "Create a new Storage Account Section". Replace nthdilabs with the name of your storage account, if you aren't using the provided storage account

You should get the following output.

```
> # change the storage context
> myNameNode <- "wasbs://yourcontainername@nthdilabs.blob.core.windows.net"
> myPort <- 0
>
> # change fileSystem to HDFS
> hdfsFS <- RxHdfsFileSystem(hostName=myNameNode, port=myPort)
>
> # import data into dataframe
> weblogsDF <- RxTextData("/weblogs/weblogs.csv", fileSystem=hdfsFS)
>
> # get top 6 rows
> head(weblogsDF)
  TransactionDate CustomerId BookId  PurchaseType TransactionId OrderId                        BookName     CategoryName Quantity
1   3/8/2015 0:00          4      6     Purchased    KRFSTI561J     107   Advances in school psychology    World_History        5
2   3/8/2015 0:00          4      5     Purchased    MKJUDF993M      15   Advances in school psychology Automobile_books        1
3   3/8/2015 0:00          4      5 Added to Cart    ABERKF334I      95            New Christian poetry       Management       61
4   2/8/2015 0:00          3      7     Purchased     DS54EX316      91     The voyages of Captain Cook         Religion       61
5  12/8/2015 0:00         10      9     Purchased    JLTRBT354D      91     The voyages of Captain Cook         Religion       61
6  12/8/2015 0:00         10      8 Added to Cart    BRTDFS241G     154 Understanding American politics       Philosophy       74
  ShippingAmount InvoiceNumber InvoiceStatus PaymentAmount
1         225.00         97342        Issued        149.99
2         140.00        967445        Issued        625.00
3          25.00        967445     Cancelled          9.99
4         225.00         99568        Failed        120.00
5           5.00         88734     Completed        120.00
6          45.25         99554        Issued        299.99
```

The **RxHdfsFileSystem** function sets the file system to HDFS/Azure Storage from local and the default container to <yourcontainername>. The yourcontainername container has the weblogs.csv which is to be read.

The **RxTextData** function reads the **weblogs.csv** file and imports the data in memory in a data frame, weblogsDF.

2. To get information on the weblogsDF data frame created in previous step, execute the following script in R Studio console window.

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

You should get the following output.

```
> rxGetInfo(weblogsDF, getVarInfo = TRUE)
File name: /weblogs/weblogs.csv
Data Source: Text
Number of variables: 13
Variable information:
Var 1: TransactionDate, Type: character
Var 2: CustomerId, Type: integer
Var 3: BookId, Type: integer
Var 4: PurchaseType, Type: character
Var 5: TransactionId, Type: character
Var 6: OrderId, Type: integer
Var 7: BookName, Type: character
Var 8: CategoryName, Type: character
Var 9: Quantity, Type: integer
Var 10: ShippingAmount, Type: numeric, Storage: float32
Var 11: InvoiceNumber, Type: integer
Var 12: InvoiceStatus, Type: character
Var 13: PaymentAmount, Type: numeric, Storage: float32
```

The **rxGetInfo** function displays a summary of what's contained in the data frame. The getVarInfo includes variable names and the data types if set to True.

3. To import the data frame into an XDF format, execute the following script in R Studio console window.

```
# xdf file format: Save data frame as an xdf file
weblogsXDF <- rxImport(inData = weblogsDF, outFile = "weblogsXDF.xdf")
rxGetInfo(weblogsXDF, getVarInfo=TRUE)
```

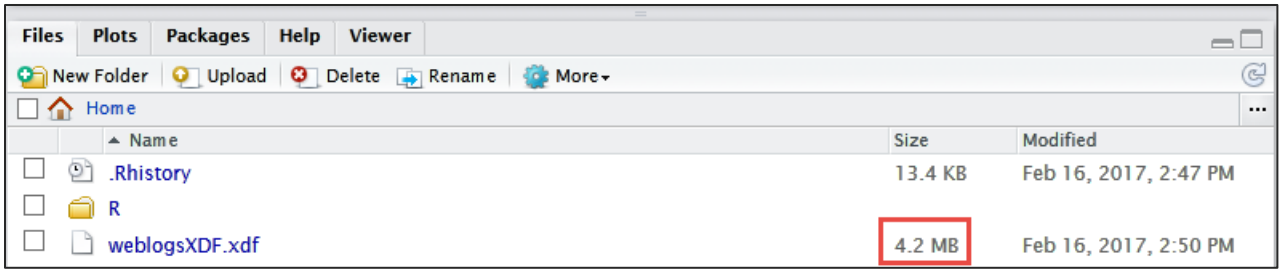
You should get the following output.

```
> weblogsXDF <- rxImport(inData = weblogsDF, outFile = "weblogsXDF.xdf", overwrite=TRUE)
Rows Read: 278972, Total Rows Processed: 278972, Total Chunk Time: 7.316 seconds
> rxGetInfo(weblogsXDF, getVarInfo=TRUE)
File name: /home/sshuser/weblogsXDF.xdf
Number of observations: 278972
Number of variables: 13
Number of blocks: 1
Compression type: zlib
Variable information:
Var 1: TransactionDate, Type: character
Var 2: CustomerId, Type: integer, Low/High: (1, 10)
Var 3: BookId, Type: integer, Low/High: (1, 11)
Var 4: PurchaseType, Type: character
Var 5: TransactionId, Type: character
Var 6: OrderId, Type: integer, Low/High: (10, 9010)
Var 7: BookName, Type: character
Var 8: CategoryName, Type: character
Var 9: Quantity, Type: integer, Low/High: (1, 74)
Var 10: ShippingAmount, Type: numeric, Storage: float32, Low/High: (2.9900, 305.0000)
Var 11: InvoiceNumber, Type: integer, Low/High: (87556, 967445)
Var 12: InvoiceStatus, Type: character
Var 13: PaymentAmount, Type: numeric, Storage: float32, Low/High: (9.9900, 625.0000)
>
```

The R script imports the data from the weblogsDF data frame, created in step 2 into a **weblogsXDF.xdf** file. The file is created locally on R Server at **/home/sshuser/weblogsXDF.xdf**.

XDF is a RevoScaleR package data file format, efficient in reading arbitrary rows and columns. It also offers compression levels 0 – 9, where 0 is no compression, 1 is the default compression level and 9 is the highest compression level.

The right-hand side of the R Studio, displays list of server objects created. Observer, that the weblogsXDF.xdf file has a size of 4.1 MB. The text file weblogs.csv is of 28 MB.



Note: You can use the Object explorer window to upload or delete R script, data files or any other required files. The weblogsXDF.xdf size may differ in your case.

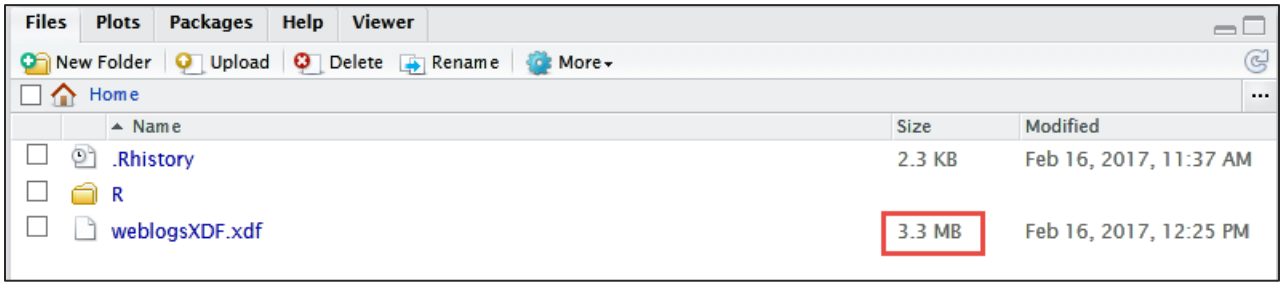
4. To specify the XDF compression level when importing data from a data frame, execute the following script in R Studio console window.

```
# specify xdf compression level
weblogsXDF <- rxImport(inData = weblogsDF, outFile = "weblogsXDF.xdf",
xdfCompressionLevel = 3,overwrite=TRUE)
rxGetInfo(weblogsXDF, getVarInfo = TRUE)
```

You should get the following output.

```
> weblogsXDF <- rxImport(inData = weblogsDF, outFile = "weblogsXDF.xdf", xdfCompressionLevel = 3,overwrite=TRUE)
Rows Read: 278972, Total Rows Processed: 278972, Total Chunk Time: 7.236 seconds
> rxGetInfo(weblogsXDF, getVarInfo = TRUE)
File name: /home/sshuser/weblogsXDF.xdf
Number of observations: 278972
Number of variables: 13
Number of blocks: 1
Compression type: zlib
Variable information:
Var 1: TransactionDate, Type: character
Var 2: CustomerId, Type: integer, Low/High: (1, 10)
Var 3: BookId, Type: integer, Low/High: (1, 11)
Var 4: PurchaseType, Type: character
Var 5: TransactionId, Type: character
Var 6: OrderId, Type: integer, Low/High: (10, 9010)
Var 7: BookName, Type: character
Var 8: CategoryName, Type: character
Var 9: Quantity, Type: integer, Low/High: (1, 74)
Var 10: ShippingAmount, Type: numeric, Storage: float32, Low/High: (2.9900, 305.0000)
Var 11: InvoiceNumber, Type: integer, Low/High: (87556, 967445)
Var 12: InvoiceStatus, Type: character
Var 13: PaymentAmount, Type: numeric, Storage: float32, Low/High: (9.9900, 625.0000)
```

Observe that with compression level 3, the **weblogsXDF.xdf** file size is further reduced to 3.3 MB.



Note: The weblogsXDF.xdf size may differ in your case.

Higher compression levels provide better compression at the cost of higher processing time.

5. To add a new column when importing data into XDF file, execute the following script in R Studio console window.

```
# transform data when reading
weblogsXDF <-rxImport(inData=weblogsDF, outFile = "weblogsXDF.xdf",
transforms=list(totalcost=ShippingAmount+PaymentAmount),overwrite=TRUE)
# verify data
rxGetInfo(weblogsXDF, getVarInfo = TRUE)
head(weblogsXDF)
```

You should get the following output.

```
> weblogsXDF <-rxImport(inData=weblogsDF, outFile = "weblogsXDF.xdf", transforms=list(totalcost=ShippingAmount+PaymentAmount),overwrite=TRUE)
Rows Read: 278972, Total Rows Processed: 278972, Total Chunk Time: 7.710 seconds
> rxGetInfo(weblogsXDF, getVarInfo=TRUE)
File name: /home/sshuser/weblogsXDF.xdf
Number of observations: 278972
Number of variables: 14
Number of blocks: 1
Compression type: zlib
Variable information:
Var 1: TransactionDate, Type: character
Var 2: CustomerId, Type: integer, Low/High: (1, 10)
Var 3: BookId, Type: integer, Low/High: (1, 11)
Var 4: PurchaseType, Type: character
Var 5: TransactionId, Type: character
Var 6: OrderId, Type: integer, Low/High: (10, 9010)
Var 7: BookName, Type: character
Var 8: CategoryName, Type: character
Var 9: Quantity, Type: integer, Low/High: (1, 74)
Var 10: ShippingAmount, Type: numeric, Storage: float32, Low/High: (2.9900, 305.0000)
Var 11: InvoiceNumber, Type: integer, Low/High: (87556, 967445)
Var 12: InvoiceStatus, Type: character
Var 13: PaymentAmount, Type: numeric, Storage: float32, Low/High: (9.9900, 625.0000)
Var 14: totalcost, Type: numeric, Low/High: (12.9800, 930.0000)
> head(weblogsXDF)
  TransactionDate CustomerId BookId  PurchaseType TransactionId OrderId                        BookName     CategoryName Quantity
1   3/8/2015 0:00          4      6     Purchased    KRFSTI561J     107   Advances in school psychology    World_History        5
2   3/8/2015 0:00          4      5     Purchased    MKJUDF993M      15   Advances in school psychology Automobile_books        1
3   3/8/2015 0:00          4      5 Added to Cart    ABERKF334I      95            New Christian poetry       Management       61
4   2/8/2015 0:00          3      7     Purchased     DS54EX316      91     The voyages of Captain Cook         Religion       61
5  12/8/2015 0:00         10      9     Purchased    JLTRBT354D      91     The voyages of Captain Cook         Religion       61
6  12/8/2015 0:00         10      8 Added to Cart    BRTDFS241G     154 Understanding American politics       Philosophy       74
  ShippingAmount InvoiceNumber InvoiceStatus PaymentAmount totalcost
1         225.00         97342        Issued        149.99    374.99
2         140.00        967445        Issued        625.00    765.00
3          25.00        967445     Cancelled          9.99     34.99
4         225.00         99568        Failed        120.00    345.00
5           5.00         88734     Completed        120.00    125.00
6          45.25         99554        Issued        299.99    345.24
```

The scripts adds a new column totalcost (ShippingAmount + PaymentAmount), when importing data from a data frame into XDF file.

6. To select only the required columns from a XDF file, execute the following script in R Studio console window.

```
# select only the required columns
webdataxdf <- rxDataStep(inData = "weblogsXDF.xdf",outFile="webdata.xdf",varsToKeep =
c("CustomerId", "BookId","PurchaseType","BookName","totalcost"), transforms
=list(PurchaseType=as.factor(PurchaseType),BookName=as.factor(BookName)),overwrite=TRUE)
# get info
rxGetInfo(webdataxdf, getVarInfo=TRUE)
# verify data
head(webdataxdf)
```

you should get the following output.

```
> webdataxdf <- rxDataStep(inData = "weblogsXDF.xdf",outFile="webdata.xdf",varsToKeep = c("CustomerId", "BookId","PurchaseType","BookName","t
talcost"), transforms =list(PurchaseType=as.factor(PurchaseType),BookName=as.factor(BookName)),overwrite=TRUE)
Rows Read: 278972, Total Rows Processed: 278972, Total Chunk Time: 1.267 seconds
>
> rxGetInfo(webdataxdf, getVarInfo=TRUE)
File name: /home/sshuser/webdata.xdf
Number of observations: 278972
Number of variables: 5
Number of blocks: 1
Compression type: zlib
Variable information:
Var 1: CustomerId, Type: integer, Low/High: (1, 10)
Var 2: BookId, Type: integer, Low/High: (1, 4)
Var 3: PurchaseType
       3 factor levels: AddedToCart Browsed Purchased
Var 4: BookName
       9 factor levels: Advances in school psychology History of political economy Prince Of Persia Science in Dispute Space  fact and fictio
The adventures of Arthur Conan Doyle The Book Of Witnesses The voyages of Captain Cook Understanding American politics
Var 5: totalcost, Type: numeric, Low/High: (12.9800, 930.0000)
> # verify data
> head(webdataxdf)
  CustomerId BookId PurchaseType                       BookName totalcost
1          4      1      Browsed   Advances in school psychology    374.99
2          4      1      Browsed   Advances in school psychology    765.00
3          4      3      Browsed                 Prince Of Persia     34.99
4          3      4      Browsed     The voyages of Captain Cook    345.00
5         10      4      Browsed     The voyages of Captain Cook    125.00
6         10      4      Browsed Understanding American politics    345.24
```

The R script imports only the selected columns specified by **varsToKeep** argument into a new XDF file, **webdata.xdf.** The script also converts the PurchaseType column data type from character to factor using the transforms function.

Statistical Analysis

In this section, you'll learn to view data summary using **rxSummary** and **rxCrossTab** functions.
The **rxSummary** function lists detailed statistics for the data using the formula argument, similar to the formula used by R modelling functions.

Execute the R Scripts one by one as mentioned in below steps:

1. To get summary statistics on a CustomerId column, execute the below script in R Studio console window.

```
rxSummary(formula = ~CustomerId, data = webdataxdf)
```

you should get the following output.

```
> rxSummary(~ CustomerId, data=webdataxdf)
Rows Read: 278972, Total Rows Processed: 278972, Total Chunk Time: 0.014 seconds
Computation time: 0.015 seconds.
Call:
rxSummary(formula = ~CustomerId, data = webdataxdf)

Summary Statistics Results for: ~CustomerId
Data: webdataxdf (RxXdfData Data Source)
File name: /home/sshuser/webdata.xdf
Number of valid observations: 278972

 Name       Mean     StdDev   Min Max ValidObs MissingObs
 CustomerId 5.496892 2.875599 1   10  278972   0
>
```

The R script lists the count, mean, median, standard deviation, valid and missing observations with for the CustomerId column.

2. To get summary statistics on CustomerId, BookId and PurchaseType column, execute the below script in R Studio console window.

```
rxSummary(formula = ~CustomerId + BookId + PurchaseType, data = webdataxdf)
```

You should get the following output.

```
> rxSummary(formula = ~CustomerId + BookId + PurchaseType, data = webdataxdf)
Rows Read: 278972, Total Rows Processed: 278972, Total Chunk Time: 0.049 seconds
Computation time: 0.051 seconds.
Call:
rxSummary(formula = ~CustomerId + BookId + PurchaseType, data = webdataxdf)

Summary Statistics Results for: ~CustomerId + BookId + PurchaseType
Data: webdataxdf (RxXdfData Data Source)
File name: /home/sshuser/webdata.xdf
Number of valid observations: 278972

 Name       Mean     StdDev   Min Max ValidObs MissingObs
 CustomerId 5.496892 2.875599 1   10  278972   0
 BookId     3.325370 1.058368 1    4  278972   0

Category Counts for PurchaseType
Number of categories: 3
Number of valid observations: 278972
Number of missing observations: 0

 PurchaseType Counts
 AddedToCart   97157
 Browsed      113717
 Purchased     68098
```

The R Scripts lists the count, mean, median, standard deviation, valid and missing observations for all three specified columns. Observer that for the PurchaseType column, you get count for all the three factors.

3. To get summary statistics for each BookId with respect to each PurchaseType, execute the below R script in R Studio console window.

```
rxSummary(~ BookId:PurchaseType, data=webdataxdf)
```

You should get the following output.

```
> rxSummary(~ BookId:PurchaseType, data=webdataxdf)
Rows Read: 278972, Total Rows Processed: 278972, Total Chunk Time: 0.030 seconds
Computation time: 0.039 seconds.
Call:
rxSummary(formula = ~BookId:PurchaseType, data = webdataxdf)

Summary Statistics Results for: ~BookId:PurchaseType
Data: webdataxdf (RxXdfData Data Source)
File name: /home/sshuser/webdata.xdf
Number of valid observations: 278972

 Name                Mean    StdDev   Min Max ValidObs MissingObs
 BookId:PurchaseType 3.32537 1.058368 1   4   278972   0

Statistics by category (3 categories):

 Category                            PurchaseType Means    StdDev   Min Max ValidObs
 BookId for PurchaseType=AddedToCart AddedToCart  3.329343 1.057498 1   4    97157
 BookId for PurchaseType=Browsed     Browsed      3.322089 1.059430 1   4   113717
 BookId for PurchaseType=Purchased   Purchased    3.325178 1.057833 1   4    68098
```

Observe, that you get summary statistics for each of the three categories.

4. To get summary statistics of each BookName against the PurchaseTypes, execute the following script in R Studio console window.

```
rxSummary(~ BookName:PurchaseType, data=webdataxdf)
```

You should get the following output.

```
> rxSummary(~ BookName:PurchaseType, data=webdataxdf)
Rows Read: 278972, Total Rows Processed: 278972, Total Chunk Time: 0.043 seconds
Computation time: 0.045 seconds.
Call:
rxSummary(formula = ~BookName:PurchaseType, data = webdataxdf)

Summary Statistics Results for: ~BookName:PurchaseType
Data: webdataxdf (RxXdfData Data Source)
File name: /home/sshuser/webdata.xdf
Number of valid observations: 278972


Category Counts for BookName
Number of categories: 27
Number of valid observations:
Number of missing observations:

 BookName                              PurchaseType Counts
 Advances in school psychology         AddedToCart  10927
 History of political economy          AddedToCart  10814
 Prince Of Persia                      AddedToCart  10750
 Science in Dispute                    AddedToCart  10573
 Space  fact and fiction               AddedToCart  10943
 The adventures of Arthur Conan Doyle AddedToCart  10781
 The Book Of Witnesses                 AddedToCart  10725
 The voyages of Captain Cook           AddedToCart  10936
 Understanding American politics       AddedToCart  10708
 Advances in school psychology         Browsed      12888
 History of political economy          Browsed      12738
 Prince Of Persia                      Browsed      12950
 Science in Dispute                    Browsed      12880
 Space  fact and fiction               Browsed      12698
 The adventures of Arthur Conan Doyle Browsed      12499
 The Book Of Witnesses                 Browsed      11572
 The voyages of Captain Cook           Browsed      12719
 Understanding American politics       Browsed      12773
 Advances in school psychology         Purchased     7625
 History of political economy          Purchased     7754
 Prince Of Persia                      Purchased     7571
 Science in Dispute                    Purchased     7552
 Space  fact and fiction               Purchased     7492
 The adventures of Arthur Conan Doyle Purchased     7550
 The Book Of Witnesses                 Purchased     7476
 The voyages of Captain Cook           Purchased     7540
 Understanding American politics       Purchased     7538
```

Observe that you get the count for each book against the different Purchase Type values.

5. To get some similar statistics, for CustomerId column against different PurchaseTypes, execute the following script in R Studio console window.

```
rxSummary(formula = ~F(CustomerId):PurchaseType, data = webdataxdf)
```

You should get the following output.

```
> rxSummary(~ F(CustomerId):PurchaseType, data=webdataxdf)
Rows Read: 278972, Total Rows Processed: 278972, Total Chunk Time: 0.036 seconds
Computation time: 0.044 seconds.
Call:
rxSummary(formula = ~F(CustomerId):PurchaseType, data = webdataxdf)

Summary Statistics Results for: ~F(CustomerId):PurchaseType
Data: webdataxdf (RxXdfData Data Source)
File name: /home/sshuser/webdata.xdf
Number of valid observations: 278972


Category Counts for F_CustomerId
Number of categories: 30
Number of valid observations:
Number of missing observations:

 F_CustomerId PurchaseType Counts
 1            AddedToCart   9796
 2            AddedToCart   9711
 3            AddedToCart   9577
 4            AddedToCart   9559
 5            AddedToCart   9766
 6            AddedToCart   9737
 7            AddedToCart   9738
 8            AddedToCart   9789
 9            AddedToCart   9830
 10           AddedToCart   9654
 1            Browsed      11606
 2            Browsed      11482
 3            Browsed      11589
 4            Browsed      10641
 5            Browsed      11472
 6            Browsed      11539
 7            Browsed      11310
 8            Browsed      11453
 9            Browsed      11217
 10           Browsed      11408
 1            Purchased     6819
 2            Purchased     6791
 3            Purchased     6774
 4            Purchased     6903
 5            Purchased     6845
 6            Purchased     6838
 7            Purchased     6776
 8            Purchased     6768
 9            Purchased     6772
 10           Purchased     6812
```

The F function converts the CustomerId column to factor variable on the fly. The above summary tells total number of books Browsed, Added To Cart or Purchased by each customer.

6. To get the summary statistics for books browsed, purchased or Added To Cart by each customers, execute the following query in R Studio console window.

```
rxSummary(formula = ~F(CustomerId,low = 1,high = 10):BookName:PurchaseType, data =
webdataxdf)
```

You should get the following output.

```
> rxSummary(formula = ~F(CustomerId,low = 1,high = 10):BookName:PurchaseType, data = webdataxdf)
Rows Read: 278972, Total Rows Processed: 278972, Total Chunk Time: 0.062 seconds
Computation time: 0.066 seconds.
Call:
rxSummary(formula = ~F(CustomerId, low = 1, high = 10):BookName:PurchaseType,
    data = webdataxdf)

Summary Statistics Results for: ~F(CustomerId, low = 1, high = 10):BookName:PurchaseType
Data: webdataxdf (RxXdfData Data Source)
File name: /home/sshuser/webdata.xdf
Number of valid observations: 278972


Category Counts for F_CustomerId_1_10_T
Number of categories: 270
Number of valid observations:
Number of missing observations:

 F_CustomerId_1_10_T BookName                          PurchaseType Counts
 1                   Advances in school psychology     AddedToCart  1111
 2                   Advances in school psychology     AddedToCart  1094
 3                   Advances in school psychology     AddedToCart  1080
 4                   Advances in school psychology     AddedToCart  1117
 5                   Advances in school psychology     AddedToCart  1080
 6                   Advances in school psychology     AddedToCart  1112
 7                   Advances in school psychology     AddedToCart  1080
 8                   Advances in school psychology     AddedToCart  1053
 9                   Advances in school psychology     AddedToCart  1120
 10                  Advances in school psychology     AddedToCart  1080
 1                   History of political economy      AddedToCart  1065
 2                   History of political economy      AddedToCart  1079
 3                   History of political economy      AddedToCart  1046
 4                   History of political economy      AddedToCart  1085
 5                   History of political economy      AddedToCart  1083
 6                   History of political economy      AddedToCart  1113
 7                   History of political economy      AddedToCart  1125
 8                   History of political economy      AddedToCart  1078
 9                   History of political economy      AddedToCart  1042
 10                  History of political economy      AddedToCart  1098
 1                   Prince Of Persia                  AddedToCart  1107
 2                   Prince Of Persia                  AddedToCart  1078
 3                   Prince Of Persia                  AddedToCart  1076
 4                   Prince Of Persia                  AddedToCart  1047
```

Note: The above image is trimmed for brevity.

Observe, that you get count for each book browsed, purchased and added to cart by all customers.

7. To save the summary statistics to an XDF file, execute the following script in R Studio console window.

```
rxSummary(~F(CustomerId):PurchaseType, data = webdataxdf,byGroupOutFile =
"byPurchaseType.xdf",overwrite = TRUE)
```

You should get the following output.

```
> rxSummary(~F(CustomerId):PurchaseType, data = webdataxdf,byGroupOutFile = "byPurchaseType",overwrite = TRUE)
Rows Read: 278972, Total Rows Processed: 278972, Total Chunk Time: 0.025 seconds
Computation time: 0.032 seconds.
Call:
rxSummary(formula = ~F(CustomerId):PurchaseType, data = webdataxdf,
    byGroupOutFile = "byPurchaseType", overwrite = TRUE)

Summary Statistics Results for: ~F(CustomerId):PurchaseType
Data: webdataxdf (RxXdfData Data Source)
File name: /home/sshuser/webdata.xdf
Number of valid observations: 278972


By-group statistics for F(CustomerId):PurchaseType contained in /home/sshuser/byPurchaseType.xdf
```

The script saves the summary data in **byPurchaseType.xdf** file, specified by the **byGroupOutFile** option in **rxSummary** function.

8. To get the information on the XDF file saved in previous step, execute the following query in R Studio console window.

```
rxGetInfo("byPurchaseType.xdf",numRows = 5)
```

You should get the following output.

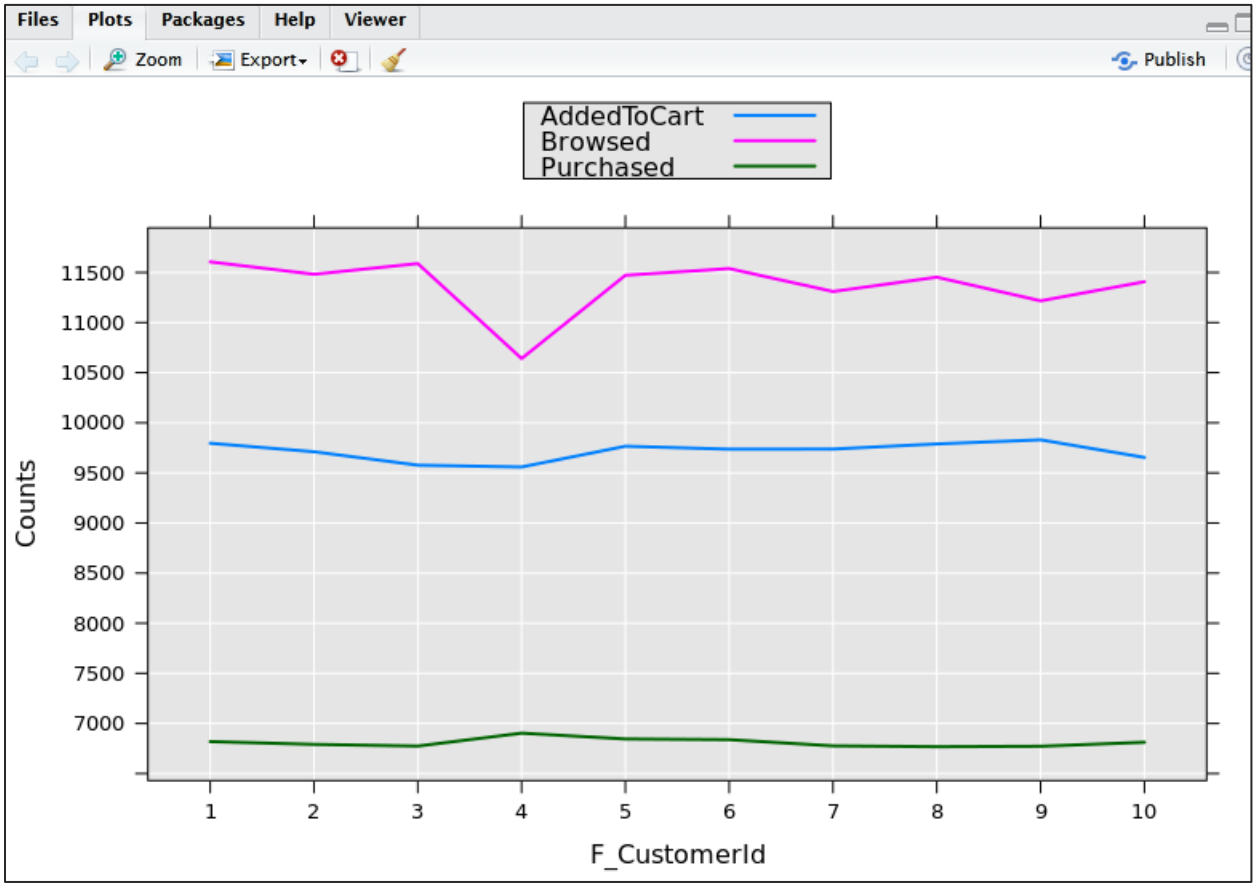
```
> rxGetInfo("byPurchaseType.xdf",numRows = 5)
File name: /home/sshuser/byPurchaseType.xdf
Number of observations: 30
Number of variables: 3
Number of blocks: 1
Compression type: none
Data (5 rows starting with row 1):
  F_CustomerId PurchaseType Counts
1            1  AddedToCart   9796
2            2  AddedToCart   9711
3            3  AddedToCart   9577
4            4  AddedToCart   9559
5            5  AddedToCart   9766
> rxLinePlot(Counts~F_CustomerId,groups=PurchaseType,data="byPurchaseType.xdf")
Rows Read: 30, Total Rows Processed: 30, Total Chunk Time: 0.001 seconds
```

The rxGetInfo function returns the top 5 rows in byPurchaseType.xdf file as specified by **numRows** option.

9. To plot the line chart for the summary statistics stored in byPurchaseType.xdf file, execute the following R script in R Studio console window.

```
rxLinePlot(Counts~F_CustomerId,groups=PurchaseType,data="byPurchaseType.xdf")
```

You will get the following line chart.

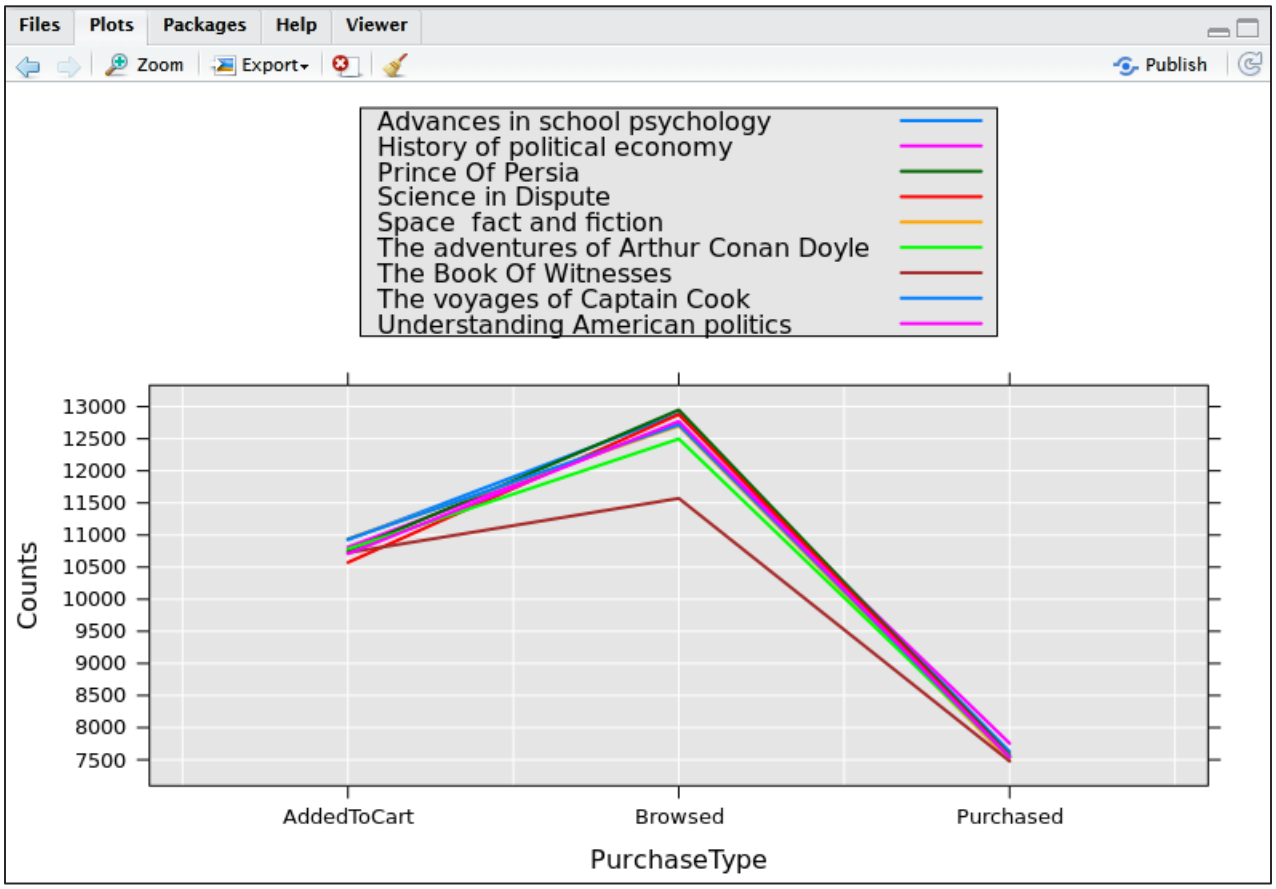


10. To plot the line chart for BookName and PurchaseType, execute the following query in R Studio console window.

```
# save summary stats for bookname per purchase type in byBookName.xdf
```

```
rxSummary(~BookName:PurchaseType, data = webdataxdf,byGroupOutFile =
"byBookName.xdf",overwrite = TRUE)
# get info on byBookName.xdf file
rxGetInfo("byBookName.xdf",numRows = 5)
# plot the line chart
rxLinePlot(Counts~PurchaseType,groups=BookName,data="byBookName.xdf")
```

You should get the following line chart.



11. To get summary statistics for totalcost against each book, execute the following script in R Studio console window.

    # save summary stats for bookname per purchase type in byBookName.xdf

```
rxSummary(~BookName:totalcost, data = webdataxdf,byGroupOutFile = "bytotalcost.xdf",
summaryStats = c("Means","StdDev"),overwrite = TRUE)
# get info on byBookName.xdf file
rxGetInfo("bytotalcost.xdf",numRows = 5)
```

    You should get the following output.

```
> rxSummary(~BookName:totalcost, data = webdataxdf,byGroupOutFile = "bytotalcost.xdf",summaryStats = c("Mean
Rows Read: 278972, Total Rows Processed: 278972, Total Chunk Time: 0.041 seconds
Computation time: 0.043 seconds.
Call:
rxSummary(formula = ~BookName:totalcost, data = webdataxdf, byGroupOutFile = "bytotalcost.xdf",
    summaryStats = c("Means", "StdDev"), overwrite = TRUE)

Summary Statistics Results for: ~BookName:totalcost
Data: webdataxdf (RxXdfData Data Source)
File name: /home/sshuser/webdata.xdf
Number of valid observations: 278972

 Name                Mean     StdDev
 BookName:totalcost 309.9707 207.9147

By-group statistics for BookName:totalcost contained in /home/sshuser/bytotalcost.xdf
> rxGetInfo("bytotalcost.xdf",numRows = 5)
File name: /home/sshuser/bytotalcost.xdf
Number of observations: 9
Number of variables: 3
Number of blocks: 1
Compression type: none
Data (5 rows starting with row 1):
                       BookName totalcost_Mean totalcost_StdDev
1 Advances in school psychology       308.7570         206.4804
2  History of political economy       310.3435         207.3664
3              Prince Of Persia       310.0510         207.6383
4            Science in Dispute       311.2443         207.5493
5       Space  fact and fiction       310.3253         207.2087
> rxLinePlot(totalcost_Mean~BookName,groups=BookName,data="bytotalcost.xdf")
```

Observe that you can specify which summary statistics to output using the **summaryStats** option in **rxSummary** function.

12. To find the aggregated total cost for each PurchaseType, execute the following R script in R Studio console window one by one.

```
ct <- rxCrossTabs(formula = totalcost ~ PurchaseType, data = webdataxdf)
ct
```

you should get the following output.

```
> ct<-rxCrossTabs(totalcost ~ PurchaseType,data=webdataxdf)
Rows Read: 278972, Total Rows Processed: 278972, Total Chunk Time: 0.043 seconds
Computation time: 0.045 seconds.
> ct
Call:
rxCrossTabs(formula = totalcost ~ PurchaseType, data = webdataxdf)

Cross Tabulation Results for: totalcost ~ PurchaseType
Data: webdataxdf (RxXdfData Data Source)
File name: /home/sshuser/webdata.xdf
Dependent variable(s): totalcost
Number of valid observations: 278972
Number of missing observations: 0
Statistic: sums

totalcost (sums):

AddedToCart 30168139
Browsed     35182567
Purchased   21122427
```

The **rxCrossTabs** function outputs the aggregated total cost across each Purchase Type.

```
summary(ct)
```

You should get the following output.

```
> summary(ct)
Call:
rxCrossTabs(formula = totalcost ~ PurchaseType, data = webdataxdf)

Cross Tabulation Results for: totalcost ~ PurchaseType
File name: /home/sshuser/webdata.xdf
Dependent variable(s): totalcost
Number of valid observations: 278972
Number of missing observations: 0
Statistic: sums

totalcost (sums):
                sums    sums %
AddedToCart 30168139  34.88730
Browsed     35182567  40.68613
Purchased   21122427  24.42658
Total       86473133 100.00000
```

The **summary** function over **rxCrossTab** gives the percentage of each PurchaseType contribute to the total cost. Observe, that 24% of the total cost is from Purchased.

13. To aggregate total quantity for all books across all PurchaseTypes, execute the following script in R Studio console window.

```
rxCrossTabs(N(Counts) ~ BookName:PurchaseType,data="byBookName.xdf")
```

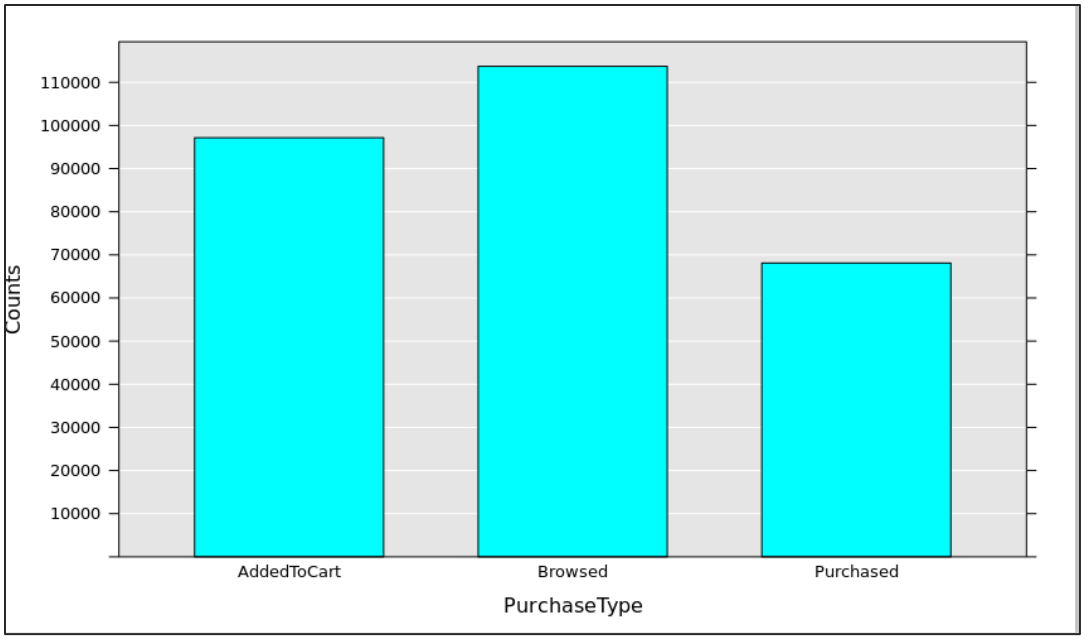
You should get the following output.

```
> rxCrossTabs(N(Counts) ~ BookName:PurchaseType,data="byBookName.xdf")
Rows Read: 27, Total Rows Processed: 27, Total Chunk Time: 0.001 seconds
Computation time: 0.003 seconds.
Call:
rxCrossTabs(formula = N(Counts) ~ BookName:PurchaseType, data = "byBookName.xdf")

Cross Tabulation Results for: N(Counts) ~ BookName:PurchaseType
Data: "byBookName.xdf" (RxXdfData Data Source)
File name: byBookName.xdf
Dependent variable(s): N(Counts)
Number of valid observations: 27
Number of missing observations: 0
Statistic: sums

Counts (sums):
                                    PurchaseType
BookName                             AddedToCart Browsed Purchased
  Advances in school psychology            10927   12888      7625
  History of political economy             10814   12738      7754
  Prince Of Persia                         10750   12950      7571
  Science in Dispute                       10573   12880      7552
  Space  fact and fiction                  10943   12698      7492
  The adventures of Arthur Conan Doyle     10781   12499      7550
  The Book Of Witnesses                    10725   11572      7476
  The voyages of Captain Cook              10936   12719      7540
  Understanding American politics          10708   12773      7538
```

14. To get a histogram with total count for all PurchaseTypes, execute the following script in R Studio console window.

```
rxHistogram(~PurchaseType,data=webdataxdf)
```

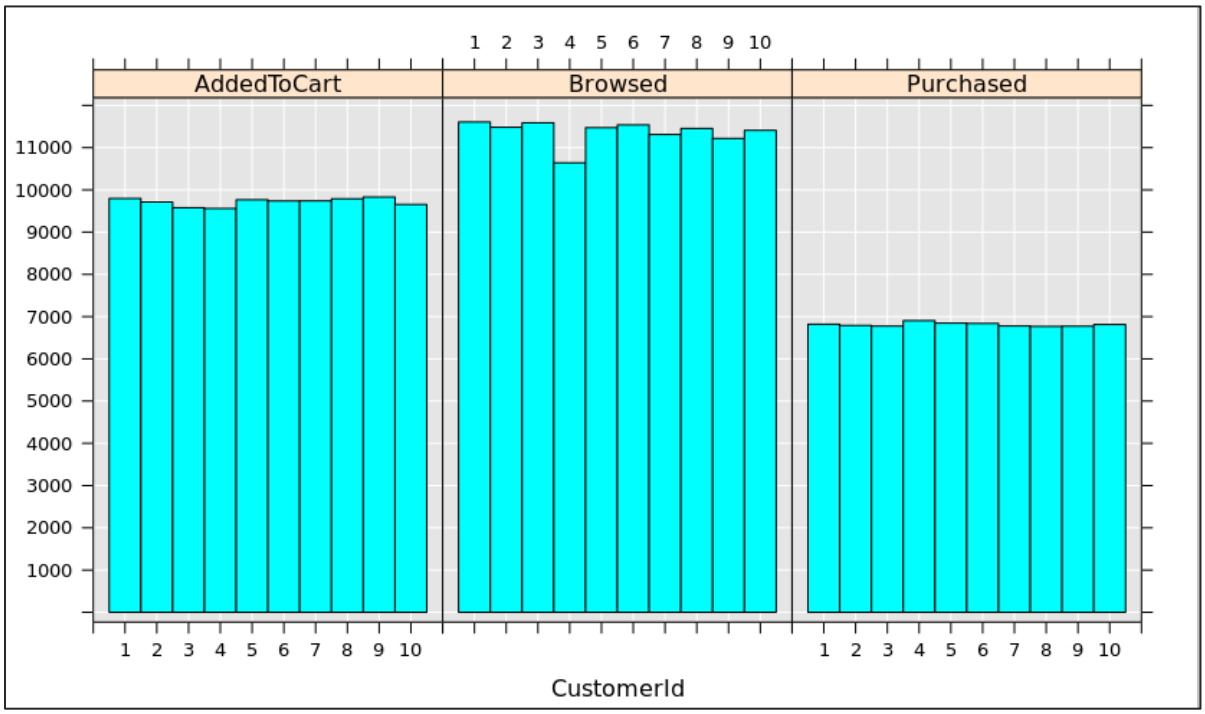
You should get the following output.



15. To get a histogram for CustomerId and PurchaseTypes, execute the following script in R Studio console window.

```
rxHistogram(~CustomerId|PurchaseType,data=webdataxdf)
```

You should get the following output.



Compute Context

R Server has four different compute contexts that determine the compute environment in which ScaleR calls are executed. The compute contexts are local sequential, local parallel, Map Reduce and Spark.

The local sequential does parallelized execution across cores of edge node, except the rxExec calls. Use this when amount of data to analyze is small and doesn't requires repeated analysis.

The local parallel does parallelized execution across cores of edge node.

The Spark compute context offers parallelized execution via Spark across the nodes of the HDInsight cluster. Use this when the amount of data is large and requires repeated analysis.

The MapReduce compute context offers parallelized execution via MapReduce across the nodes of the HDInsight cluster. Use this if Spark compute context has lot of problems. This is generally slower than Spark compute context.

To execute a function in Spark compute context, follow the below steps.

Execute the scripts one by one as shown in below steps in R Studio console window.

1. To read the weblogs.csv file in a data frame with Spark compute context, execute the following script in R Studio console window.

```
# set the HDFS file system
hdfsFS <- RxHdfsFileSystem()

# change compute context to Spark
rxSetComputeContext("spark")

# specify the input file
inputFile <-file.path("/weblogs","weblogs.csv")
# read the data into a data frame
weblogsDF <- RxTextData(inputFile, fileSystem=hdfsFS)
head(weblogsDF)
```

You should get the following output.

```
> myNameNode <- "wasbs://yourcontainername@nthdilabs.blob.core.windows.net"
> myPort <- 0
> hdfsFS <- RxHdfsFileSystem(hostName=myNameNode, port=myPort)
> mySparkCluster <- RxSpark(consoleOutput=TRUE, nameNode=myNameNode, port=myPort)
> rxSetComputeContext(mySparkCluster)
> inputFile <-file.path("weblogs","weblogs.csv")
> inputFile <-file.path("/weblogs","weblogs.csv")
> weblogsDF <- RxTextData(inputFile, fileSystem=hdfsFS)
> head(weblogsDF)
======  ed0-nthdil (Master HPA Process) has started run at Thu Feb 16 15:24:50 2017  ===
17/02/16 15:26:01 WARN azure.AzureFileSystemThreadPoolExecutor: Disabling threads for De
17/02/16 15:26:01 INFO azure.AzureFileSystemThreadPoolExecutor: Time taken for Delete op
======  ed0-nthdil (Master HPA Process) has completed run at Thu Feb 16 15:26:07 2017  =
  TransactionDate CustomerId BookId PurchaseType TransactionId OrderId
1   3/8/2015 0:00          4      1      Browsed    KRFSTI561J     107   Advances in sch
2   3/8/2015 0:00          4      1      Browsed    MKJUDF993M      15   Advances in sch
3   3/8/2015 0:00          4      3      Browsed    ABERKF334I      95                Pr
4   2/8/2015 0:00          3      4      Browsed     DS54EX316      91     The voyages o
5  12/8/2015 0:00         10      4      Browsed    JLTRBT354D      91     The voyages o
6  12/8/2015 0:00         10      4      Browsed    BRTDFS241G     154 Understanding Ame
  ShippingAmount InvoiceNumber InvoiceStatus PaymentAmount
1         225.00         97342        Issued        149.99
2         140.00        967445        Issued        625.00
3          25.00        967445     Cancelled          9.99
4         225.00         99568        Failed        120.00
5           5.00         88734     Completed        120.00
6          45.25         99554        Issued        299.99
```

Note: Replace <yourcontainername> with the name of your container specified in section "Create a New Azure Storage Container". Replace nthdilabs storage account with the storage account specified in section "Provision HDInsight Linux R Server with Azure Management Portal"", if you are not using the provided shared cluster.

The script defines to Spark compute context using RxSpark function. The default container is changed to yourcontainername as specified by the MyNameNode variable. The **rxSetComputeContext** changes the compute context to Spark, globally for a particular session.

The inputFile variable sets the path to weblogs.csv in yourcontainername container. The **RxTextData** functions read the weblogs.csv file into weblogsDF data frame.

2. Execute the following script to import the data frame into a XDF file. The XDF file is created in Azure Storage and not locally. This is because the compute context is Spark and not Local.

   # specify the output folder

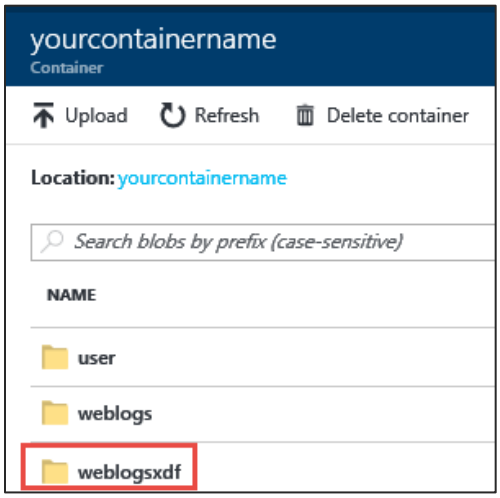
```
outFile <- "/weblogsxdf"
# specify the XDF target path
weblogsxdf <- RxXdfData(outFile, fileSystem = hdfsFS)

# import data from data frame into XDF
rxImport(inData=weblogsDF,outFile = weblogsxdf, overwrite=TRUE)
```

You should get the following output.

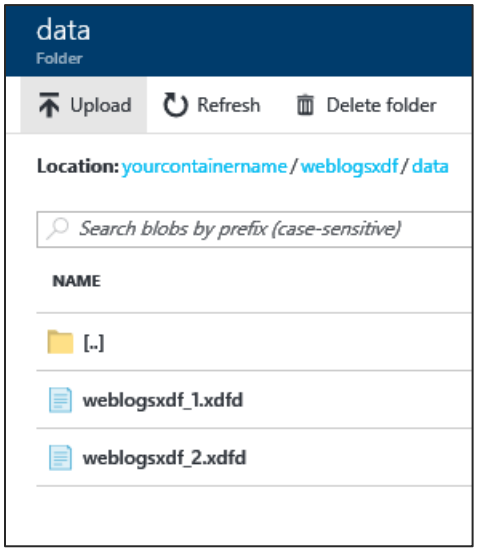
```
> outFile <- "/weblogsxdf"
>
> weblogsxdf <- RxXdfData(outFile, fileSystem = hdfsFS)
> rxImport(inData=weblogsDF,outFile = weblogsxdf)
======  ed0-nthdil (Master HPA Process) has started run at Thu Feb 16 15:27:41
17/02/16 15:28:40 WARN azure.AzureFileSystemThreadPoolExecutor: Disabling thre
17/02/16 15:28:40 INFO azure.AzureFileSystemThreadPoolExecutor: Time taken for
======  ed0-nthdil (Master HPA Process) has completed run at Thu Feb 16 15:29:
RxXdfData Source
"/weblogsxdf"
fileSystem:
    fileSystemType: hdfs
    hostName: wasbs://yourcontainername@nthdilabs1.blob.core.windows.net
    port: 0
    useWebHdfs: FALSE
    verbose: FALSE
```

The outFile variable specifies the location in Azure Storage. The **RxXdfData** specifies the XDF target on the Azure Storage at the path specified by outFile variable. The **rxImport** function imports the data frame into XDF file specified by RxXdfData.

Navigate to the Azure portal and open nthdilabs storage account. Browse to the container yourcontainername. Observer, that a folder weblogsxdf is created.



Navigate to **weblogsxdf/data** folder. Observer, that xdf files are created here.



3. Execute the following script perform transformation on the xdf file created in previous step.

```
outDir <- "/webdataxdf"
webdataxdf <- RxXdfData(outDir,fileSystem=hdfsFS)
rxDataStep(inData = weblogsxdf,outFile=webdataxdf,varsToKeep = c("CustomerId",
"BookId","PurchaseType","BookName"), transforms
=list(PurchaseType=as.factor(PurchaseType),BookName=as.factor(BookName),totalcost=Shippi
ngAmount+PaymentAmount),overwrite=TRUE)
head(webdataxdf)
```

You should get the following output.

```
> outDir <- "/webdataxdf"
> webdataxdf <- RxXdfData(outDir,fileSystem=hdfsFS)
> rxDataStep(inData = weblogsxdf,outFile=webdataxdf,varsToKeep = c("CustomerId", "BookId","PurchaseType","Boo
eType=as.factor(PurchaseType),BookName=as.factor(BookName),totalcost=ShippingAmount+PaymentAmount),overwrite=
======  ed0-nthdil (Master HPA Process) has started run at Thu Feb 16 16:11:39 2017  ======
17/02/16 16:12:41 WARN azure.AzureFileSystemThreadPoolExecutor: Disabling threads for Delete operation as thr
17/02/16 16:12:41 INFO azure.AzureFileSystemThreadPoolExecutor: Time taken for Delete operation is: 133 ms wi
======  ed0-nthdil (Master HPA Process) has completed run at Thu Feb 16 16:13:10 2017  ======
> head(webdataxdf)
======  ed0-nthdil (Master HPA Process) has started run at Thu Feb 16 16:14:17 2017  ======
======  ed0-nthdil (Master HPA Process) has completed run at Thu Feb 16 16:15:09 2017  ======
17/02/16 16:15:09 WARN impl.MetricsSinkAdapter: azurefs2 has a full queue and can't consume the given metrics
  CustomerId BookId PurchaseType                       BookName ShippingAmount PaymentAmount totalcost
1          4      1      Browsed   Advances in school psychology         225.00        149.99    374.99
2          4      1      Browsed   Advances in school psychology         140.00        625.00    765.00
3          4      3      Browsed                Prince Of Persia          25.00          9.99     34.99
4          3      4      Browsed     The voyages of Captain Cook         225.00        120.00    345.00
5         10      4      Browsed     The voyages of Captain Cook           5.00        120.00    125.00
6         10      4      Browsed Understanding American politics          45.25        299.99    345.24
```

The script performs the following transformation on the XDF file created in previous step.

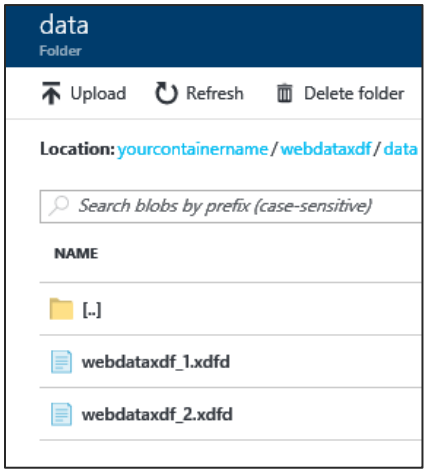
a. Adds the total cost column.

b. Modifies the data type of BookName and PurchaseType column from character to factor

c. Selects CustomerId, BookId, PurchaseType, BookName, ShippingCost and PaymentAmount columns to be included in the new XDF file.

A new file XDF is created at yourcontainername/webdataxdf location.

Navigate to **yourcontainername/webdataxdf/data** folder on Azure Management Portal in nthdilabs Azure Storage account.



Fitting Linear Model

In this section, you'll split the data into training and test data. Fit linear model to the training data and then test the model on the test data.

Follow the instructions given in section "Start new R Session" to start a new R session in R Studio.

Execute the scripts one by one as shown in the below steps.

1. To split the data into training and test data, execute the following script in R Studio console window.

```
# Step 1
# set the HDFS file system
hdfsFS <- RxHdfsFileSystem()
# read webdataxdf saved in last section
#set the source directory
outDir <- "/webdataxdf"
webdataxdf <- RxXdfData(outDir,fileSystem=hdfsFS)
# read data into a data frame
webdataXDF=rxDataStep(inData = webdataxdf)

# split data frame into train and test sets
index <- 1:nrow(webdataXDF)
tindex <- sample(index, trunc(length(index)/4))
testset <- webdataXDF[tindex, ]
trainset <- webdataXDF[-tindex, ]

# get info on training data
rxGetInfo(trainset,getVarInfo=TRUE)
# get info on test data
rxGetInfo(testset,getVarInfo=TRUE)
```

You should get the following output.

```
> webdataXDF=rxDataStep(inData = webdataxdf)
Rows Read: 139798, Total Rows Processed: 139798, Total Chunk Time: 0.897 seconds
Rows Read: 139174, Total Rows Processed: 278972, Total Chunk Time: 0.724 seconds
>
> # split data frame into train and test sets
> index <- 1:nrow(webdataXDF)
> tindex <- sample(index, trunc(length(index)/4))
> testset <- webdataXDF[tindex, ]
> trainset <- webdataXDF[-tindex, ]
>
> # get info on training data
> rxGetInfo(trainset,getVarInfo=TRUE)
Data frame: trainset
Number of observations: 209229
Number of variables: 7
Variable information:
Var 1: CustomerId, Type: integer, Low/High: (1, 10)
Var 2: BookId, Type: integer, Low/High: (1, 4)
Var 3: PurchaseType
       3 factor levels: AddedToCart Browsed Purchased
Var 4: BookName
       9 factor levels: Advances in school psychology History of political econo
d fiction The adventures of Arthur Conan Doyle The Book Of Witnesses The voyages
Var 5: ShippingAmount, Type: numeric, Low/High: (2.9900, 305.0000)
Var 6: PaymentAmount, Type: numeric, Low/High: (9.9900, 625.0000)
Var 7: totalcost, Type: numeric, Low/High: (12.9800, 930.0000)
> # get info on test data
> rxGetInfo(testset,getVarInfo=TRUE)
Data frame: testset
Number of observations: 69743
Number of variables: 7
Variable information:
Var 1: CustomerId, Type: integer, Low/High: (1, 10)
Var 2: BookId, Type: integer, Low/High: (1, 4)
Var 3: PurchaseType
       3 factor levels: AddedToCart Browsed Purchased
Var 4: BookName
       9 factor levels: Advances in school psychology History of political econo
d fiction The adventures of Arthur Conan Doyle The Book Of Witnesses The voyages
Var 5: ShippingAmount, Type: numeric, Low/High: (2.9900, 305.0000)
Var 6: PaymentAmount, Type: numeric, Low/High: (9.9900, 625.0000)
Var 7: totalcost, Type: numeric, Low/High: (12.9800, 930.0000)
>
```

The **webdataXDF** has **278972** observations. This is spilt into two separate data frames, **trainset** and **testset** in 75/25 ratio. The trainset has **209229** observations and testset has **69743** observations.

2. To fit the linear model on trainset, execute the following script in R Studio console window.

```
# fitting linear model on training set
lm <- rxLinMod(BookId~F(CustomerId):PurchaseType,data=trainset)
```

Execute the following script to get summary on the linear model.

```
# get model summary
summary(lm)
```

You should get the following output.

```
> # fitting linear model on training set
> lm <- rxLinMod(BookId~F(CustomerId):PurchaseType,data=trainset)
Rows Read: 209229, Total Rows Processed: 209229, Total Chunk Time: 0.008 seconds
Computation time: 0.014 seconds.
>
> # get model summary
> summary(lm)
Call:
rxLinMod(formula = BookId ~ F(CustomerId):PurchaseType, data = trainset)

Linear Regression Results for: BookId ~ F(CustomerId):PurchaseType
Data: trainset
Dependent variable(s): BookId
Total independent variables: 31 (Including number dropped: 1)
Number of valid observations: 209229
Number of missing observations: 0

Coefficients: (1 not defined because of singularities)
                                        Estimate Std. Error t value Pr(>|t|)
(Intercept)                            3.3276825  0.0146783 226.707 2.22e-16 ***
F_CustomerId=1, PurchaseType=AddedToCart   0.0183251  0.0191658   0.956    0.339
F_CustomerId=2, PurchaseType=AddedToCart  -0.0003419  0.0192386  -0.018    0.986
F_CustomerId=3, PurchaseType=AddedToCart   0.0119872  0.0192520   0.623    0.534
F_CustomerId=4, PurchaseType=AddedToCart  -0.0117983  0.0192442  -0.613    0.540
F_CustomerId=5, PurchaseType=AddedToCart   0.0150249  0.0192258   0.781    0.435
F_CustomerId=6, PurchaseType=AddedToCart  -0.0189540  0.0192143  -0.986    0.324
F_CustomerId=7, PurchaseType=AddedToCart   0.0082988  0.0192000   0.432    0.666
F_CustomerId=8, PurchaseType=AddedToCart   0.0119349  0.0192187   0.621    0.535
F_CustomerId=9, PurchaseType=AddedToCart   0.0130037  0.0191766   0.678    0.498
F_CustomerId=10, PurchaseType=AddedToCart -0.0017028  0.0192016  -0.089    0.929
F_CustomerId=1, PurchaseType=Browsed       0.0260948  0.0185383   1.408    0.159
F_CustomerId=2, PurchaseType=Browsed       0.0209888  0.0185502   1.131    0.258
F_CustomerId=3, PurchaseType=Browsed      -0.0062540  0.0185331  -0.337    0.736
```

Observer, that statistics are calculated for each customer against all of the PurchaseTypes. The BookId is a dependent variable and CustomerId and PurchaseType are independent variables.

3. To compute predictions over the test data set, execute the following script in R Studio console window.

```
# get predictions on test data
lmpred <- rxPredict(lm,data=testset)
```

Execute the below script to get the summary on lmpred object.

```
# get summary
summary(lmpred)
```

You should get the following output.

```
> # Step 3
> # get predictions on test data
> lmpred <- rxPredict(lm,data=testset)
Rows Read: 69743, Total Rows Processed: 69743, Total Chunk Time: 0.012 seconds
>
> # get summary
> summary(lmpred)
  BookId_Pred
 Min.   :3.253
 1st Qu.:3.320
 Median :3.328
 Mean   :3.327
 3rd Qu.:3.340
 Max.   :3.354
```

4. To get the predicted values, execute the following scripts in R studio console window.

```
# Create a function to get predicted values
predval <- function(cid){
```

```
      head(lmpred$BookId_Pred[which(testset$CustomerId==cid)])
}
# call predval function
predval(10)
```

The above script creates a functions **predval** which will return top 5 predicted BookId values for a particular customer as specified by the **cid** parameter.

You should get the following output.

```
>
> predval <- function(cid){
+     head(lmpred$BookId_Pred[which(testset$CustomerId==cid)])
+ }
> # get predicted value for CustomerId = 10
> predval(10)
[1] 3.327683 3.327683 3.325980 3.327683 3.328041 3.325980
```

5. To return prediction standard errors, confidence intervals and prediction intervals, execute the following script in R Studio console window.

```
lm <- rxLinMod(BookId~F(CustomerId):PurchaseType,data=trainset,covCoef=TRUE)
```

The covCoef=TRUE ensures that variance-covariance matrix is included in our linear model object.

Now, use rxPredict to get fitted values, prediction standard errors, and confidence intervals.

```
lmpred <- rxPredict(lm, data=testset, computeStdErrors=TRUE,interval="confidence",
writeModelVars = TRUE)
summary(lmpred)
```

You should get the following output.

```
> lm <- rxLinMod(BookId~F(CustomerId):PurchaseType,data=trainset,covCoef=TRUE)
Rows Read: 209229, Total Rows Processed: 209229, Total Chunk Time: 0.011 seconds
Computation time: 0.021 seconds.
> lmpred <- rxPredict(lm, data=testset, computeStdErrors=TRUE,interval="confidence", writeModelVars = TRUE)
Rows Read: 69743, Total Rows Processed: 69743, Total Chunk Time: 0.041 seconds
> summary(lmpred)
  BookId_Pred     BookId_StdErr       BookId_Lower     BookId_Upper        BookId        CustomerId          PurchaseType
 Min.   :3.253   Min.   :0.01131   Min.   :3.230   Min.   :3.276   Min.   :1.00   Min.   : 1.000   AddedToCart:24395
 1st Qu.:3.320   1st Qu.:0.01143   1st Qu.:3.295   1st Qu.:3.344   1st Qu.:3.00   1st Qu.: 3.000   Browsed    :28385
 Median :3.328   Median :0.01238   Median :3.303   Median :3.353   Median :4.00   Median : 6.000   Purchased  :16963
 Mean   :3.327   Mean   :0.01259   Mean   :3.302   Mean   :3.352   Mean   :3.32   Mean   : 5.496
 3rd Qu.:3.340   3rd Qu.:0.01246   3rd Qu.:3.315   3rd Qu.:3.365   3rd Qu.:4.00   3rd Qu.: 8.000
 Max.   :3.354   Max.   :0.01492   Max.   :3.332   Max.   :3.376   Max.   :4.00   Max.   :10.000
```

The standard errors are saved into variable **BookId_StdErr**. To get standard error, execute the following script in R Studio console window.

```
head(lmpred$BookId_StdErr)
```

In this section, you learnt to split data into training and testing set, fit linear model on training set and get prediction on the test set.

Interoperability for Microsoft R Server and Open Source R Libraries (sparklyr)

In this section, you'll learn how to use sparklyr package and MRS packages in one Spark session.

sparklyr, a package developed by RStudio, is an R interface to Apache Spark. It allows users to utilize Spark as the backend for dplyr, one of the most popular data manipulation packages. sparklyr also provides interfaces to Spark packages, allows users to query data in Spark using SQL, and develop extension in R by creating an interface to the full Spark API. Another key feature is it allows users the ability to use Spark integrated Machine Learning algorithms directly from within R. For H2O users, the Microsoft R Server sparklyr Interop can be used to covert sparklyr data frames to H2O data frames. This allows data imported from Microsoft R Server to be used with H2O modelling and data partitioning algorithms, via the rsparkling package. (to learn more about dplyr, please visit their CRAN site here.)

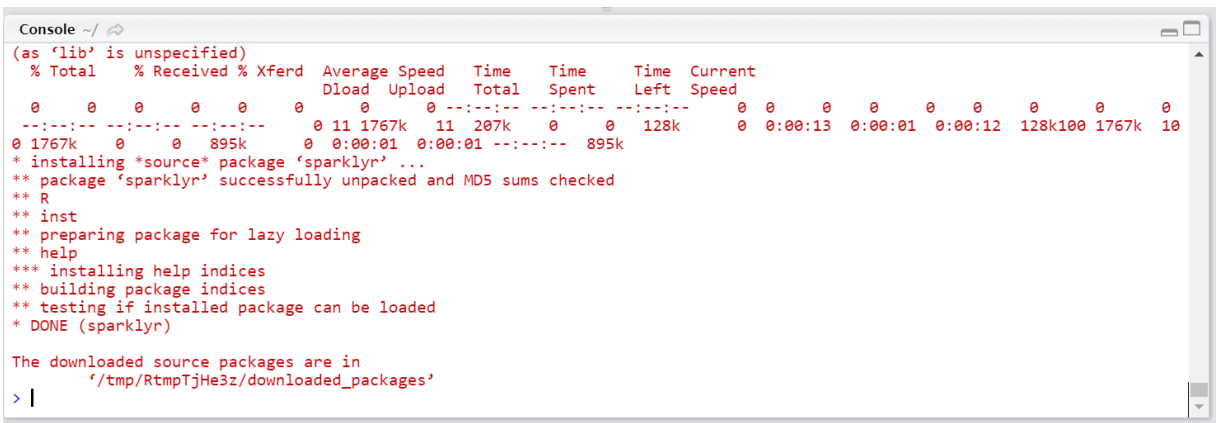
In the example below, we will re-do the example above to train a linear regression model. However, instead of using MRS to split the data and transform the data, we will use the popular sparklyr and dplyr package to transform the data, then use MRS algorithms to train a linear regression model.

Execute the scripts one by one as shown in the below steps.

1. To install sparklyr package, execute the following script in R Studio console window.

```
# set the repo to 2017-05-01 to download the latest sparklyr version
options(repos = "https://mran.microsoft.com/snapshot/2017-05-01")
install.packages("sparklyr")
```

You should get the following output and it will take around 2 minutes to complete.



2. To fit the linear model on trainset, execute the following script in R Studio console window.

```
# Load required libraries
library(RevoScaleR)
library(sparklyr)
library(dplyr)
cc <- rxSparkConnect(reset = TRUE, interop = "sparklyr")
# The returned Spark connection (sc) provides a remote dplyr data source
# to the Spark cluster using SparlyR within rxSparkConnect.
sc <- rxGetSparklyrConnection(cc)
spark_read_csv(sc, "weblogs", "wasb:///weblogs.csv")

partitions <- weblogs %>%  sdf_partition(training = 0.75, test = 0.25, seed = 1099)
```

```
Console ~/
> # to the Spark cluster using SparlyR within rxSparkConnect.
> sc <- rxGetSparklyrConnection(cc)
> spark_read_csv(sc, "weblogs", "wasb:///weblogs.csv")
Source:   query [2.79e+05 x 13]
Database: spark connection master=yarn-client app=sparklyr-scaleR-spark-won2r0FHOA-sshuser-20337-4680AA92F57248A1BB74BCA7721
0AE57 local=FALSE

   TransactionDate CustomerId BookId PurchaseType TransactionId OrderId                              BookName
             <chr>      <int>  <int>        <chr>         <chr>   <int>                                 <chr>
1    3/8/2015 0:00          4      1      Browsed    KRFSTI561J     107        Advances in school psychology
2    3/8/2015 0:00          4      1      Browsed    MKJUDF993M      15        Advances in school psychology
3    3/8/2015 0:00          4      3      Browsed    ABERKF334I      95                     Prince Of Persia
4    2/8/2015 0:00          3      4      Browsed     DS54EX316      91          The voyages of Captain Cook
5   12/8/2015 0:00         10      4      Browsed    JLTRBT354D      91          The voyages of Captain Cook
6   12/8/2015 0:00         10      4      Browsed    BRTDFS241G     154      Understanding American politics
7    7/8/2015 0:00          7      4      Browsed    MKJUDF993M     322          The voyages of Captain Cook
8   1/10/2015 0:00          1      4    Purchased    MKJUDF993M      61 The adventures of Arthur Conan Doyle
9    9/8/2015 0:00          8      4      Browsed    NMQR38SJDI     667 The adventures of Arthur Conan Doyle
10   2/8/2015 0:00          3      4      Browsed    JLTRBT354D      61                The Book Of Witnesses
# ... with 2.79e+05 more rows, and 6 more variables: CategoryName <chr>, Quantity <int>, ShippingAmount <dbl>,
#   InvoiceNumber <int>, InvoiceStatus <chr>, PaymentAmount <dbl>
>
> partitions <- weblogs %>%  sdf_partition(training = 0.75, test = 0.25, seed = 1099)
>
> |
```

Execute the following script to register the dataframe to Hive tables for further reuse in MRS functions.

```
# register as hive tables
sdf_register(partitions$training, "weblogs_training")
sdf_register(partitions$test, "weblogs_test")
```

You should get the following output.

```
> sdf_register(partitions$test, "weblogs_test")
Source:   query [6.934e+04 x 13]
Database: spark connection master=yarn-client app=sparklyr-scaleR-spark-won2r0FHOA-sshuser-20337-7A2C1D9B49A54E3EB07FC4B2913
57FF0 local=FALSE

   TransactionDate CustomerId BookId PurchaseType TransactionId OrderId                      BookName     CategoryName
             <chr>      <int>  <int>        <chr>         <chr>   <int>                         <chr>            <chr>
1   1/10/2015 0:00          1      1  AddedToCart    ABERKF334I     322 Advances in school psychology        Adventure
2   1/10/2015 0:00          1      1  AddedToCart    ABERKF334I     541 Advances in school psychology       Psychology
3   1/10/2015 0:00          1      1  AddedToCart    ARBUER437Y      61 Advances in school psychology          Fiction
4   1/10/2015 0:00          1      1  AddedToCart    ARBUER437Y     107 Advances in school psychology       Management
5   1/10/2015 0:00          1      1  AddedToCart    BRTDFS241G      61 Advances in school psychology      Non_Fiction
6   1/10/2015 0:00          1      1  AddedToCart    BRTDFS241G     541 Advances in school psychology    World_History
7   1/10/2015 0:00          1      1  AddedToCart    BTDRGL712H      10 Advances in school psychology    World_History
8   1/10/2015 0:00          1      1  AddedToCart    BTDRGL712H      15 Advances in school psychology Automobile_books
9   1/10/2015 0:00          1      1  AddedToCart     DS54EX316      10 Advances in school psychology             Cook
10  1/10/2015 0:00          1      1  AddedToCart     DS54EX316     107 Advances in school psychology      Non_Fiction
# ... with 6.933e+04 more rows, and 5 more variables: Quantity <int>, ShippingAmount <dbl>, InvoiceNumber <int>,
#   InvoiceStatus <chr>, PaymentAmount <dbl>
```

3. The next step is to read the Hive temporary table in MRS and use MRS functions to perform a simple linear regression method on the data. This can be achieved by executing the following code.

```
# reading data from temporary hive table which we just registered. Please be noted that
you need to read the PurchaseType column using "factor" type to make sure this column
can work in rxLinMod.
weblogs_training_hive <- RxHiveData(table = "weblogs_training", colInfo =
list(PurchaseType=list(type = "factor")))
weblogs_test_hive <- RxHiveData(table = "weblogs_test", colInfo =
list(PurchaseType=list(type = "factor")))
lm <- rxLinMod(BookId~F(CustomerId):PurchaseType,data=weblogs_training_hive)
```

Similarly with what we have executed in previous parts, you can use the summary function as well as rxPredict to see the summary of the model, or do predictions on the test dataset.

```
# get model summary
summary(lm)

# get predictions on test data
lmpred <- rxPredict(lm,data=weblogs_test_hive)
```

```
# get summary
summary(lmpred)
```

You should get the following output.

```
> # Step 3
> # get predictions on test data
> lmpred <- rxPredict(lm,data=testset)
Rows Read: 69743, Total Rows Processed: 69743, Total Chunk Time: 0.012 seconds
>
> # get summary
> summary(lmpred)
  BookId_Pred
 Min.   :3.253
 1st Qu.:3.320
 Median :3.328
 Mean   :3.327
 3rd Qu.:3.340
 Max.   :3.354
```

4. You can also do it in an opposite way – transform data in MRS, and train machine learning models using sparklyr, which ultimately uses Spark Mllib.

   Execute the code below to load data and transform data using MRS functions, and then use sparklyr to call Spark Mllib libraries to perform a linear regression.

```
# load necessary libraries
library(RevoScaleR)
library(sparklyr)
library(dplyr)
cc <- rxSparkConnect(reset = TRUE, interop = "sparklyr")
sc <- rxGetSparklyrConnection(cc)

hdfsFS <- RxHdfsFileSystem()
# import data into dataframe
weblogsDF <- RxTextData("wasb:///weblogs.csv", fileSystem=hdfsFS)

head(weblogsDF)

# put data into temporary hive tables
WebLogsHive <- RxHiveData(table="weblogs")
rxDataStep(inData = weblogsDF, outFile = WebLogsHive,
overwrite = TRUE)

# list all tables in this context
src_tbls(sc)

# Next, define a dplyr data source referencing the Hive table
# This caches the data in Spark
tbl_cache(sc, "weblogs")
weblogs_tbl <- tbl(sc, "weblogs")

# show tables
weblogs_tbl

# split data to test dataset and train dataset
weblogs_partition <- weblogs_tbl %>%
sdf_partition(train = 0.8, test = 0.2, seed = 6666)

sparkLinearRegression <- weblogs_partition$train %>%
```

```
ml_linear_regression(BookId ~ CustomerId + PaymentAmount)

summary(sparkLinearRegression)
```

you should see something similar with below:

```
> summary(sparkLinearRegression)
Call: ml_linear_regression(., BookId ~ CustomerId + PaymentAmount)

Deviance Residuals: (approximate):
    Min      1Q  Median      3Q     Max
-2.3359 -0.3282  0.6723  0.6757  0.6810

Coefficients:
                 Estimate  Std. Error  t value Pr(>|t|)
(Intercept)    3.3258e+00  5.5517e-03 599.0524   <2e-16 ***
CustomerId    -6.9632e-04  7.7939e-04  -0.8934   0.3716
PaymentAmount  1.7300e-05  1.3029e-05   1.3278   0.1842
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

R-Squared: 1.146e-05
Root Mean Squared Error: 1.058
```

In this section, you learnt to split data into training and testing set, fit linear model on training set and get prediction on the test set.

Disclaimer: Once you have completed the lab, to reduce costs associated with your Azure subscription, please delete your clusters.

Terms of use

By using this hands-on lab, you agree to the following terms:

The technology/functionality described in this hands-on lab is provided by Microsoft Corporation in a “sandbox” testing environment for purposes of obtaining your feedback and to provide you with a learning experience. You may only use the hands-on lab to evaluate such technology features and functionality and provide feedback to Microsoft. You may not use it for any other purpose. You may not modify, copy, distribute, transmit, display, perform, reproduce, publish, license, create derivative works from, transfer, or sell this hands-on lab or any portion thereof.

COPYING OR REPRODUCTION OF THE HANDS-ON LAB (OR ANY PORTION OF IT) TO ANY OTHER SERVER OR LOCATION FOR FURTHER REPRODUCTION OR REDISTRIBUTION IS EXPRESSLY PROHIBITED.

THIS HANDS-ON LAB PROVIDES CERTAIN SOFTWARE TECHNOLOGY/PRODUCT FEATURES AND FUNCTIONALITY, INCLUDING POTENTIAL NEW FEATURES AND CONCEPTS, IN A SIMULATED ENVIRONMENT WITHOUT COMPLEX SET-UP OR INSTALLATION FOR THE PURPOSE DESCRIBED ABOVE. THE TECHNOLOGY/CONCEPTS REPRESENTED IN THIS HANDS-ON LAB MAY NOT REPRESENT FULL FEATURE FUNCTIONALITY AND MAY NOT WORK THE WAY A FINAL VERSION MAY WORK. WE ALSO MAY NOT RELEASE A FINAL VERSION OF SUCH FEATURES OR CONCEPTS. YOUR EXPERIENCE WITH USING SUCH FEATURES AND FUNCTIONALITY IN A PHYSICAL ENVIRONMENT MAY ALSO BE DIFFERENT.

FEEDBACK. If you give feedback about the technology features, functionality, and/or concepts described in this hands-on lab to Microsoft, you give to Microsoft, without charge, the right to use, share, and commercialize your feedback in any way and for any purpose. You also give to third parties, without charge, any patent rights needed for their products, technologies, and services to use or interface with any specific parts of a Microsoft software or service that includes the feedback. You will not give feedback that is subject to a license that requires Microsoft to license its software or documentation to third parties because we include your feedback in them. These rights survive this agreement.

MICROSOFT CORPORATION HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS WITH REGARD TO THE HANDS-ON LAB, INCLUDING ALL WARRANTIES AND CONDITIONS OF MERCHANTABILITY, WHETHER EXPRESS, IMPLIED, OR STATUTORY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. MICROSOFT DOES NOT MAKE ANY ASSURANCES OR REPRESENTATIONS WITH REGARD TO THE ACCURACY OF THE RESULTS, OUTPUT THAT DERIVES FROM USE OF THE VIRTUAL LAB, OR SUITABILITY OF THE INFORMATION CONTAINED IN THE VIRTUAL LAB FOR ANY PURPOSE.