REVOLUTION ANALYTICS

Introduction to Revolution R Enterprise Module 4: Data Summaries and Cross Tabs

Data Statistics

Finally we are moving away from computer architecture and towards statistical computing. In this module we will start to execute statistical functions focusing particularly on data summaries and cross tabulation.

We will also cover some information on creating plots, including histograms and line plots. After completing this section, you should be able to start gaining important information from your data using ScaleR!

In the following slides, we will first consider informational and variable summaries of data sets, before moving on to numerical summaries and quantiles.

Helper Functions

First we will discuss the helper functions. These functions give you basic information about your data source, such as its size on the Hadoop cluster and the names of variables in the set:

- rxGetInfo
- rxGetVarInfo
- rxGetVarNames

Let's run each of these helper functions on our Bank data set located on the Hadoop cluster.

Example: RxGetInfo

The rxGetInfo function offers information on the data location, size, number of variables, and other related information.

```
infile <- file.path("data", "BankXDF.xdf")
BankDS <- RxXdfData(file = infile)

rxGetInfo(BankDS)

## File name: /AcademyR/Revolution_Course_Materials/modules/IntroToR/Descriptive_Statistics_with_R
## Number of observations: 45211
## Number of variables: 21
## Number of blocks: 5
## Compression type: zlib
```

From the above, we can see that our BankDS has 45,211 observations, 17 variables, and is contained in one block on the HDFS. Further, we know its location, as we defined it in an earlier section, and its compression type.

Example: RxGetVarInfo

To obtain information on the variables contained within a stored data set, use the rxGetVarInfo function:

```
rxGetVarInfo(BankDS)
```

Example: RxGetVarInfo

We can see the names of the 17 variables contained within the BankDS, including variable type and minimum/maximum values for each. For instance, day is an integer, and ranges between 1 and 31, as expected.

Example: RxGetVarNames

Finally, for a more concise output, we can use rxGetVarNames to obtain only the variable names in a data set source stored on the HDFS:

```
rxGetVarNames(BankXDF)
class(rxGetVarNames(BankDS))
dput(rxGetVarNames(BankDS))
```

Exercise: Helper Functions

Obtain information on the Churn data set using at least one of the above functions:

- rxGetInfo
- rxGetVarInfo
- rxGetVarNames

Exercise: Solution

We can obtain all of the qualitative information we want from ChurnDS using the following three commands:

```
rxGetInfo(ChurnDS)
rxGetVarInfo(ChurnDS)
rxGetVarNames(ChurnDS)
```

Numerical Data Summaries

Let's move to quantitative summaries of data stored on Hadoop. The rxSummary command provides output on the number of observations contained in a data set and, for a particular variable, the function provides output on its:

- Name
- Mean value
- Standard Deviation
- Minimum and Maximum value
- Number of valid observations
- Number of missing observations

rxSummary

The rxSummary() function takes a formula as its first argument, and the name of the data set as the second:

```
DSSummary <- rxSummary(~var1 + var2 + ..., data = DS)
```

In addition, variable subsets and transformations may also be computed as a sub-call to the function using the transforms (and so forth) commands.

Example: Summarizing your Data

Let's summarize the balance and age values for customers for our Bank data.

Using the rxSummary command, we specify the values balance and age as those which we want to summarize, and we also specify the data as the previously constructed BankDS:

```
BankSummary <- rxSummary(~balance + age, data = BankDS)
```

```
## Rows Read: 10000, Total Rows Processed: 10000, Total Chunk Time: 0.001 seconds
## Rows Read: 10000, Total Rows Processed: 20000, Total Chunk Time: 0.001 seconds
## Rows Read: 10000, Total Rows Processed: 30000, Total Chunk Time: 0.001 seconds
## Rows Read: 10000, Total Rows Processed: 40000, Total Chunk Time: 0.001 seconds
## Rows Read: 5211, Total Rows Processed: 45211, Total Chunk Time: Less than .001 seconds
## Computation time: 0.011 seconds.
```

```
BankSummary
```

```
## Call:
## rxSummary(formula = ~balance + age, data = BankDS)
##
```

Example: Summarizing your Data

Let's summarize the balance by categories of age for customers for our Bank data.

This time, balance is our dependent variable. We use F() function to convert age to a factor variable, on the fly.

```
BankSummary2 <- rxSummary(balance ~ F(age), data = BankDS)
```

```
## Rows Read: 10000, Total Rows Processed: 10000, Total Chunk Time: 0.001 seconds
## Rows Read: 10000, Total Rows Processed: 20000, Total Chunk Time: 0.005 seconds
## Rows Read: 10000, Total Rows Processed: 30000, Total Chunk Time: 0.002 seconds
## Rows Read: 10000, Total Rows Processed: 40000, Total Chunk Time: 0.001 seconds
## Rows Read: 5211, Total Rows Processed: 45211, Total Chunk Time: 0.001 seconds
## Computation time: 0.015 seconds.
```

```
BankSummary2
```

```
## Call:
## rxSummary(formula = balance ~ F(age), data = BankDS)
##
## Summary Statistics Results for: balance ~ F(age)
```

Quantiles

In addition to the rxSummary function, you can also compute approximate quantiles of your data (for XDF files and data frames) using the rxQuantile function. Basically, the function requires you to specify the name of the variable in you data set, as well as the data set itself. This function is Hadoop compatible!

```
rxQuantile(varName, data, ...)
```

However, this function does not sort the data, so use the rxSort function covered earlier before breaking your data into quantiles.

Exercise: Quantiles

Compute the quantiles of the BankSubDS data source for age, after first sorting the age variable in ascending order using rxSort.

Exercise: Solution

Remember to execute the rxSort function as in the previous section:

```
infile <- file.path("data", "BankSubXDF.xdf")
BankSubDS <- RxXdfData(file = infile)

rxSort(inData = BankSubDS, outFile = BankSubDS, sortByVars = "age", decreasing = FALSE,
    overwrite = TRUE)

## Number of rows written to file: 9043, Variable(s): balance, age, newBalance, Total number of rows i
## Number of rows written to file: 9043, Variable(s): balance, age, newBalance, Total number of rows i
## Number of rows written to file: 9043, Variable(s): balance, age, newBalance, Total number of rows i
## Number of rows written to file: 9043, Variable(s): balance, age, newBalance, Total number of rows i
## Number of rows written to file: 9039, Variable(s): balance, age, newBalance, Total number of rows i
## Time to sort data file: 0.023 seconds

rxGetInfo(BankSubDS, numRows = 3)

## File name: /AcademyR/Revolution_Course_Materials/modules/IntroToR/Descriptive_Statistics_with_R
## Number of observations: 45211
## Number of variables: 3
## Number of blocks: 5
## Compression type: zlib
## Data (3 rows starting with row 1):
```

Cross Tabulation

Cross Tabulation has the same meaning as Contingency Tables or Cross-tabulations: it is a convenient way to summarize cross-classified categorical data. More specifically, the data has multiple levels of two or more factors, and therefore is broken up into separate categories based on factors.

If only two factors are involved, the table is sometimes referred to as a two-way table; similarly, for three factors the table can be referred to as a three-way table.

Cross-Tabulate: rxCrossTabs

Cross-tabulation can be obtained by using either the rxCube function or the rxCrossTabs function.

The rxCrossTabs function performs a formula-based cross-tabulation of data. The basic idea behind executing the function is providing a formula of key variables you want to analyze and specifying the data set that those variables are from:

```
rxCrossTabs(formula, data, ...)
```

Cross-Tabulate: rxCube

Similarly, the rxCube function provides and alternative formula-based cross-tabulation returning ‘cube’ results. Like the rxCrossTabs function, you need to provide a formula of key variables you want to analyze and also specify the data set that those variables are from:

```
rxCube(formula, data, ...)
```

Example: Cross-Tabulate

Let's numerically look at the interaction between balance against marital status and the current employment of a customer in our Bank data set.

For this example, we convert age and campaign to factor variables on the fly with F(), which allows us to also limit the cross tabulation to a min and max value of each variable (see ?rxFormula). To compute the interaction between the two, use the colon.

```
rxCrossTabs(balance ~ F(age, low = 20, high = 30):F(campaign, low = 1, high = 5),
    data = BankDS)
```

Exercise: Cross-Tabulate

In this exercise, consider the interaction between the number of family members and the number of devices of customers from the Churn data set, or more specifically the variables n.family.members and n.devices. In order for this computation to work, remember to wrap continuously defined variables with F(). Also, use the Churn data source, ChurnDS, we created in an earlier exercise.

Exercise: Solution

Remember to wrap the variables using the F() command:

```
rxCrossTabs(~F(n.family.members):F(n.devices), data = ChurnDS)
```

Producing Basic Plots: Overview

Let's focus on producing basic plots. You can produce plots based on data on the Hadoop cluster just like in ScaleR!

Here are the ScaleR commands for basic plots (don't worry, we'll go over each of these):

- rxLinePlot
- rxHistogram

Producing Basic Plots: rxLinePlot

We can produce basic line plots using the rxLinePlot function

The rxLinePlot can analyze one variable against one or more other variables within a data set. Further, there are a variety of other options to plot with this command, such as row selection and grouping, that one may access by searching ?rxLinePlot.

```
rxLinePlot(y ~ x1 + x2 + ..., data = DS, .)
```

REVOLUTION ANALYTICS

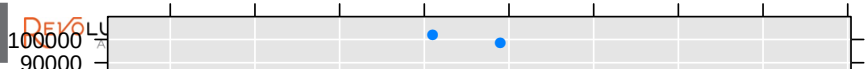
Example: rxLinePlot

Let's produce a line plot using the Bank subset data, which selects only variables age and balance. This should improve our computational time.

Let's plot customer balance against the age of the customer, plotting points at each data rather than connecting observations with lines:

```
rxLinePlot(balance ~ age, type = "p", data = BankSubDS)
```

```
## Rows Read: 9043, Total Rows Processed: 9043, Total Chunk Time: 0.002 seconds
## Rows Read: 9043, Total Rows Processed: 18086, Total Chunk Time: 0.002 seconds
## Rows Read: 9043, Total Rows Processed: 27129, Total Chunk Time: 0.002 seconds
## Rows Read: 9043, Total Rows Processed: 36172, Total Chunk Time: 0.002 seconds
## Rows Read: 9039, Total Rows Processed: 45211, Total Chunk Time: 0.002 seconds
```



Example: rxLinePlot

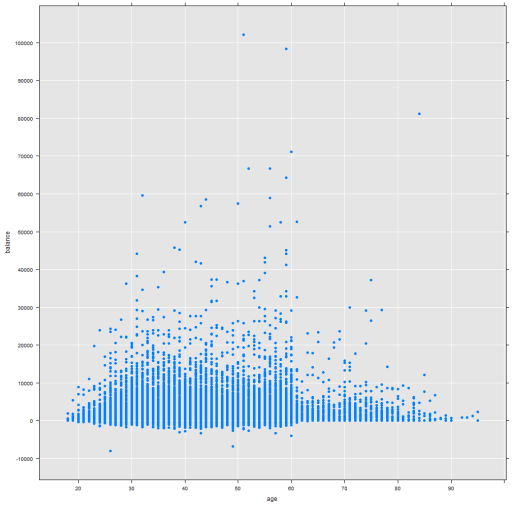


Figure:

Producing Basic Plots: rxHistogram

Similarly, we can produce basic histogram plots using the rxHistogram function.

The histogram function can analyze one variable by logging its frequency . Further, there are a variety of other options to plot with this command, such as row selection and grouping, that one may access by searching ?rxHistogram.

```
rxHistogram(y, data = DS, ...)
```

Example: RxHistogram

Let's verify the previous conclusion from our last example, this time treating age as a factor variable:

```
rxHistogram(~balance | F(age), data = BankSubDS)
```

Example: RxHistogram

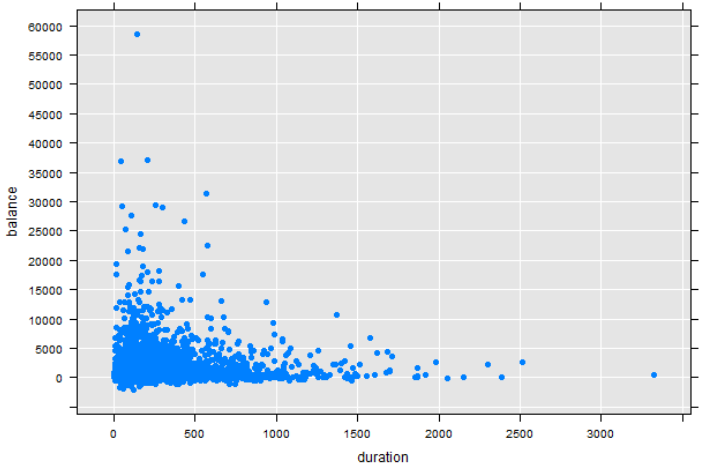


Figure:

Exercise: Refactoring Age

Since we treated age as a factor variable in our last example, the distribution of balances were computed for each separate age value. While this could be useful in some instances, the previous plot was a little over-complicated to interpret.

In this exercise, divide age into three categories by creating three new logical variables: - Young = (age <= 43), - Middle = (age > 43 & age <= 61), - Old = (age > 61 & age <= 95)

Then repeat the graphing exercise for each of the three categories.

Exercise: Solution

To divide the ages into separate categories, we need to specify the transforms parameter in the function call.

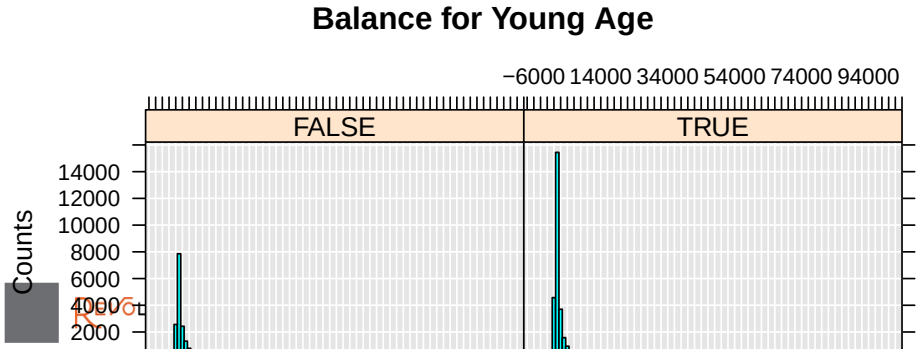
```
rxDataStep(inData = BankSubDS, outFile = BankSubDS, transforms = list(Young = (age <=
    43), Middle = (age > 43 & age <= 61), Old = (age > 61 & age <= 95)), overwrite = TRUE)
```

```
## Rows Read: 9043, Total Rows Processed: 9043, Total Chunk Time: 0.007 seconds
## Rows Read: 9043, Total Rows Processed: 18086, Total Chunk Time: 0.008 seconds
## Rows Read: 9043, Total Rows Processed: 27129, Total Chunk Time: 0.005 seconds
## Rows Read: 9043, Total Rows Processed: 36172, Total Chunk Time: 0.005 seconds
## Rows Read: 9039, Total Rows Processed: 45211, Total Chunk Time: 0.005 seconds
```

Exercise: Solution

```
rxHistogram(~balance | Young, main = "Balance for Young Age", data = BankSubDS)
```

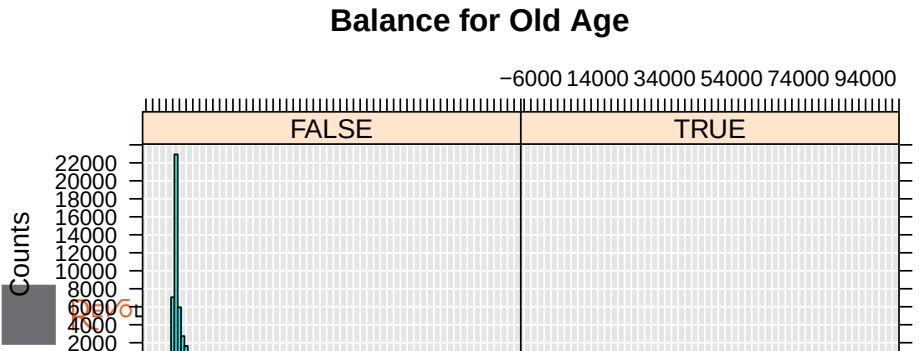
```
## Rows Read: 9043, Total Rows Processed: 9043, Total Chunk Time: 0.004 seconds
## Rows Read: 9043, Total Rows Processed: 18086, Total Chunk Time: 0.005 seconds
## Rows Read: 9043, Total Rows Processed: 27129, Total Chunk Time: 0.005 seconds
## Rows Read: 9043, Total Rows Processed: 36172, Total Chunk Time: 0.005 seconds
## Rows Read: 9039, Total Rows Processed: 45211, Total Chunk Time: 0.006 seconds
## Computation time: 0.031 seconds.
```



Exercise: Solution

```
rxHistogram(~balance | Old, main = "Balance for Old Age", data = BankSubDS)
```

```
## Rows Read: 9043, Total Rows Processed: 9043, Total Chunk Time: 0.004 seconds
## Rows Read: 9043, Total Rows Processed: 18086, Total Chunk Time: 0.005 seconds
## Rows Read: 9043, Total Rows Processed: 27129, Total Chunk Time: 0.005 seconds
## Rows Read: 9043, Total Rows Processed: 36172, Total Chunk Time: 0.005 seconds
## Rows Read: 9039, Total Rows Processed: 45211, Total Chunk Time: 0.005 seconds
## Computation time: 0.030 seconds.
```



Exercise: Brainstorm

What's another way that we could have addressed this instead of creating three new logical variables for these three categories?

How can we create one plot for each of the three age categories, plotted side by side in the same graphics device?

Exercise: Solution

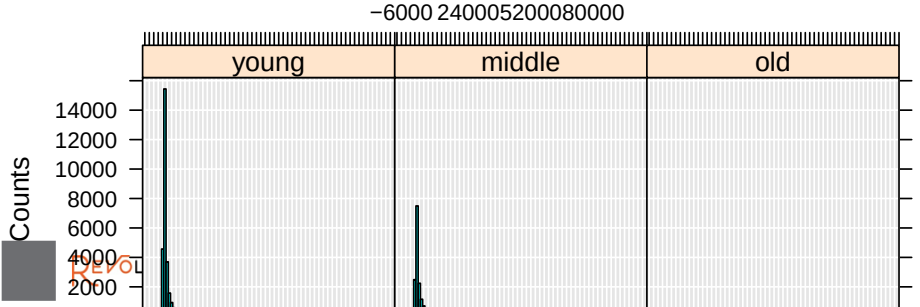
We could have defined one variable, a factor variable with three levels, one for each age category. In this case, we define this variable by executing a transformation which uses the base-R function cut().

```
outfile <- file.path("data", "BankFactors.xdf")

BankFactors <- rxDataStep(inData = BankSubDS, outFile = outfile, transforms = list(F_age = cut(ag
    breaks = c(0, 43, 61, 100), labels = c("young", "middle", "old"), right = TRUE)),
    overwrite = TRUE)


## Rows Read: 9043, Total Rows Processed: 9043, Total Chunk Time: 0.007 seconds
## Rows Read: 9043, Total Rows Processed: 18086, Total Chunk Time: 0.007 seconds
## Rows Read: 9043, Total Rows Processed: 27129, Total Chunk Time: 0.008 seconds
## Rows Read: 9043, Total Rows Processed: 36172, Total Chunk Time: 0.007 seconds
## Rows Read: 9039, Total Rows Processed: 45211, Total Chunk Time: 0.007 seconds

rxGetInfo(BankFactors, getVarInfo = TRUE, numRows = 5)

## File name: /AcademyR/Revolution_Course_Materials/modules/IntroToR/Descriptive_Statistics_with_R
## Number of observations: 45211
```

Exercise: Solution

```
rxHistogram(~balance | F_age, data = BankFactors)
```

```
## Rows Read: 9043, Total Rows Processed: 9043, Total Chunk Time: 0.004 seconds
## Rows Read: 9043, Total Rows Processed: 18086, Total Chunk Time: 0.005 seconds
## Rows Read: 9043, Total Rows Processed: 27129, Total Chunk Time: 0.111 seconds
## Rows Read: 9043, Total Rows Processed: 36172, Total Chunk Time: 0.005 seconds
## Rows Read: 9039, Total Rows Processed: 45211, Total Chunk Time: 0.005 seconds
## Computation time: 0.137 seconds.
```



Recap

Let's review some of the concepts covered in this module:

- What is the difference between qualitative and quantitative summaries?
- How do you obtain a quantile summary of your data?
- What does cross tabulation mean?
- How do you create a point plot rather than a line plot?

REVOLUTION ANALYTICS

Thank you

Revolution Analytics is the leading commercial provider of software and support for the popular open source R statistics language.

www.revolutionanalytics.com, 1.855.GET.REVO, Twitter: @Revo-lutionR