REVOLUTION ANALYTICS WHITE PAPER

The RevoScaleR Data Step White Paper

By Joseph B. Rickert

INTRODUCTION

This paper provides an introduction to working with large data sets with Revolution Analytics' proprietary R package, RevoScaleR. Although the main focus is on the use and capabilities of the rxDataStep function, we take a broad view and describe the capabilities of the functions in the RevoScaleR package that may be useful for reading and manipulating large data sets, cleaning them, and preparing them for statistical analysis with R.

BIG DATA AND ITS CHALLENGES

Many What is a large data set? These days, one company's impossibly huge data set may be approximately the same size as another company's ordinary, typical size, morning log file. For the purpose of this paper, a large data set satisfies two criteria:

1. It is a file that contains data structured as rows of observations and columns of variables
2. It is too large for R to get all of the data into memory and analyze it on the computer that is available

The intention of the first criterion is to distinguish between all the data you may have, in a database or data warehouse and the data that need to be analyzed as a coherent whole for a particular project. No matter what its origin, we assume that the data to be analyzed can be shaped into a flat file with perhaps billions or rows and thousands of columns. The second criterion acknowledges that size of a data file must be measured relative to the amount of computing resources at hand. The R language is memory constrained. Normally, RevoScaleR and a few other big-data packages being exceptions, all of the data must fit into the available RAM, in addition to any copies or data created during analysis. So, though objectively your data file might not be *that* big, it doesn't much matter if your PC can't handle it.

The first challenge in analyzing a large data set is just to get hold of it: to see what variables it contains, to figure out their data types, compute basic summary statistics and get an idea of missing values and possible errors. The next task is to prepare the data for analysis, which in addition to cleaning the data may involve supplementing the data set with additional information, removing unnecessary variables and, perhaps, transforming some variables in a way that makes sense for the contemplated analysis.

Moreover, it is likely that the data set will eventually be reduced in size as the analysis proceeds. Perhaps only a subset of all the available variables are useful for the final model, or some natural stratification exists in the data set that would naturally lead to the data being distributed among many smaller files, or the some sort of aggregation step such as summing counts of time stamped data to form daily or monthly time series will boil down the data. It is for these purposes: cleaning and preparing large data sets for analysis that the data step functions of RevoScaleR were designed.

THE REVOSCALER APPROACH TO BIG DATA

Revolution Analytics' RevoScaleR package overcomes the memory limitation of R by providing a binary file (extension .xdf) format that is optimized for processing blocks of data at a time. The xdf file format stores data in a way that facilitates the operation of external memory algorithms. RevolScaleR provides functions for importing data from different kinds of sources into xdf files, manipulating data in xdf files and performing statistical analyses directly on data in these files.

In this paper we showcase the data handling capabilities of RevoScaleR for reading, writing, sorting, merging and transforming data stored in xdf files. Data handling in RevoScaleR is built around the function rxDataStep. We introduce this function in the next section, and then consider an example illustrating how the family of RevoScaleR data manipulation functions play together.

The rxDataStepFunction

The RevoScaleR function rxDataStep has been designed to be the work horse for manipulating xdf files and selecting, and transforming the data in these files. It can also be used with data frames in memory. The function can perform all of its work on a single xdf file, or it can be used to create a new xdf file; transforming, selecting and filtering data in the process. The key arguments for rxDataStep are[1]:

- `inData`: a data frame, .xdf file name, or a "data source" representation of an .xdf file
- `outFile`: an optional name of an .xdf file to store the output data. If omitted, a data frame is returned
- `varsToKeep` or `varsToDrop`: a vector of variable names to either keep in the data set or drop from the data set
- `rowSelection`: a logical expression indicating which rows or observations to keep in the data set
- `transforms`: a list of expressions for creating or transforming variables

We will exercise these and some of the additional available options while working through an extended example. Additionally, we will gain some experience with rxImport which is used to import data files, rxSort used to sort data sets, rxMerge used to merge two data sets, and rxCube, a cross tabulation function that is useful for reducing data stored in xdf files.

[1] Please see Appendix 1 for a description of all of rxDataStep parameters and the RevoScaleR User's Guide for a complete description for all RevoScaleR commands.

Importing Large Data Files

To illustrate some of the capabilities of rxImport, we will work with a .csv file containing "edge" data for US domestic flights from 1990 to 2009 that is available for free from infochimps.com[2]. At a little over 77MB and with 3.6 million records the file is of modest size but too big for Excel to open and big enough to get a feel for working with large files. Table 1 shows the first 10 records of the file which is named "flights_with_colnames.csv".

Table 1: First 10 records of airlines edge data file

origin_airport	destin_airport	passengers	flights	month
MHK	AMW	21	1	200810
EUG	RDM	41	22	199011
EUG	RDM	88	19	199012
EUG	RDM	11	4	199010
MFR	RDM	0	1	199002
MFR	RDM	11	1	199003
MFR	RDM	2	4	199001
MFR	RDM	7	1	199009
MFR	RDM	7	2	199011
SEA	RDM	8	1	199002

The first two columns of the airlines edge data file contain the three letter code for the origin and destination airports. The last column identifies the month for which the data were collected. The third and fourth columns contain the total number of passengers and total flights between the origin and destination airports for the month indicated.

RevoScaleR's main function for importing data files is rxImport. It imports delimited and fixed format text, SAS and SPSS files as well as data stored in a relational database via odbc. Smaller data sets can be imported into a data frame in memory; larger data sets are best stored in an xdf file.

The code in BLOCK 1 of the R script in Appendix 2 illustrates using the rxImport file to read the data in the .csv file into a .xdf file called "FlightsXdf". The colClasses option is used to make sure origin_airport and destin_airport get read in as factors and month gets read in as a character string. This is in preparation for breaking apart the month and year later on. The output from the rxGetInfo function given in Table 2 shows that we have imported 3,606,803 rows of data.

Table 2. First 10 rows of imported airline edge data

```
#File name: C:\Users\Joseph\Documents\Revolution\DATA STEP WHITE
PAPER\Flights.xdf
#Number of observations: 3606803
#Number of variables: 5
#Number of blocks: 8
#Data (10 rows starting with row 1):
   #origin_airport destin_airport passengers flights  month
#1             MHK            AMW         21       1 200810
#2             EUG            RDM         41      22 199011
#3             EUG            RDM         88      19 199012
#4             EUG            RDM         11       4 199010
#5             MFR            RDM          0       1 199002
#6             MFR            RDM         11       1 199003
```

[2] http://www.infochimps.com/datasets/us-domestic-flights-from-1990-to-2009

```
#7              MFR             RDM             2       4 199001
#8              MFR             RDM             7       1 199009
#9              MFR             RDM             7       2 199011
#10             SEA             RDM             8       1 199002
```

A variation of the rxGetInfo function (Table 3) interrogates the meta data stored in the xdf file and shows some information about the data types of the stored variables

Table 3. Meta data stored in xdf file for airlines edge data

```
#> rxGetInfo("Flights", getVarInfo=TRUE)
#File name: C:\Users\Joseph\Documents\Revolution\DATA STEP WHITE
PAPER\Flights.xdf
#Number of observations: 3606803
#Number of variables: 5
#Number of blocks: 8
#Variable information:
#Var 1: origin_airport
       #683 factor levels: MHK EUG MFR SEA PDX ... CRE BOK BIH MQJ LCI
#Var 2: destin_airport
       #708 factor levels: AMW RDM EKO WDG END ... COS HII PHD TBN OH1
#Var 3: passengers, Type: integer, Low/High: (0, 89597)
#Var 4: flights, Type: integer, Low/High: (0, 1128)
#Var 5: month, Type: character
```

Notice that there are 683 factor levels for the variable origin_airport and 708 destination airports. The maximum number of passengers carried between any origin/destination pair in one month was 89,597 and the maximum number of flights between any origin/destination pair was 1,128.

Merging Xdf Files

Before exploring the data, it would be nice to be able to interpret the airport codes. Infochimps.com has another file which gives the airport name associated with each code as well as additional information on airport locations.[3] Table 4 shows the first ten rows of this airport locations file. It is clear the variables Code and Airport_Name contain exactly the information required. Also note that the locations file is rather small (600KB). Even on a modest PC this entire file could be read into a data frame and concise R code written to merge it into an Xdf file. However, the approach we will take to merge the flights and locations file is treat them as if they are too big to be read into memory and perform the task entirely with xdf files as the only data structure.

Table 4. Airport location data

Code	Lattitude	Longitude	Airport_Name	City	Country	Country Code	GMT offset	Runway Length	Runway Elevation
MSW	15.67	39.37	Massawa International	Massawa	Eritrea	ER	3	11471	194
TES	15.1166	36.6833	Tessenei	Tessenei	Eritrea	ER	3	6234	2018
ASM	15.2919	38.9105	Yohannes IV	Asmara	Eritrea	ER	3	9843	7661
ASA	13.0716	42.6449	Assab International	Assab	Eritrea	ER	3	7546	46
NLK	-29.0416	167.9386	Norfolk Island	Norfolk Island	Norfolk Island	NF	11.5	6400	371
URT	9.1	99.3333	Surat Thani	Surat Thani	Thailand	TH	-7	8202	19

[3] http://www.infochimps.com/datasets/airports-and-their-locations

PHZ	8.1666	98.2833	Phi Phi Island	Phi Phi Island	Thailand	TH	-7		
PHS	16.7833	100.2666	Phitsanulok	Phitsanulok	Thailand	TH	-7	9843	145
UTP	12.6666	100.9833	Utapao	Utapao	Thailand	TH	7	11500	59
UTH	17.3863	102.7883	Udon Thani	Udon Thani	Thailand	TH	7	10000	579

Dealing with Factor Data

Conceptually, what we would like to do is read in this new file containing the airport names and merge it with our original data set using the airport codes which appear in both data sets as the key. Simple? Yes, conceptually simple, but we will be dealing with factor data and we must take care to ensure that the levels of the code variable in the new data set match the levels of the code variables origin_airport and destin_airport in the original data set. It is also the case that merging involves sorting, and sorting factor variables can be a cause for surprise to the uninitiated. In RevoScaleR, unless otherwise specified, factor levels are ordered by the order in which the values of the variable are read into the xdf file. This is because the data is read in in chunks. So if the first row with code TES happens to appear the first row with code ASA in a file, then, when sorted, the TES rows will appear before rows beginning with ASA. The following block of code uses a new function in RevoScaleR, rxFactors, to reset the levels of both the origin_airport factor variable and destin_airport factor variable to the union of the original levels for these variables. Note that a new flights file, FlightsRfXdf, is created in the process. The code to accomplish this is given in BLOCK 2 of the R script. Table 5 presents the meta-data for this new file and shows that origin_airport and destin_airport have the same levels which have been sorted into alphabetical order.

Table 5. Meta-information for new flights file FlightsRfXdf showing reset factor levels

```
#> rxGetInfo("FlightsRf",getVarInfo=TRUE)
#File name: C:\Users\Joseph\Documents\Revolution\DATA STEP WHITE
PAPER\FlightsRf.xdf
#Number of observations: 3606803
#Number of variables: 5
#Number of blocks: 8
#Variable information:
#Var 1: origin_airport
       #727 factor levels: 1B1 ABE ABI ABQ ABR ... YKN YNG YUM ZXX ZZV
#Var 2: destin_airport
       #727 factor levels: 1B1 ABE ABI ABQ ABR ... YKN YNG YUM ZXX ZZV
#Var 3: passengers, Type: integer, Low/High: (0, 89597)
#Var 4: flights, Type: integer, Low/High: (0, 1128)
#Var 5: month, Type: character
```

Next (BLOCK 3), we read in the airport location data file, using the colInfo parameter in rxImport to set the levels of the Code variable to the same as those in the flights data file. Explicitly setting the levels of a factor variable with the ColInfo command is the preferred method of importing factor levels. We also restrict the data import to the two variables of interest: Code and Airport_Name.

Merging Two Xdf Files

Before merging the two files, we first rename the variables in the AirportNames data file (BLOCK 4 of the R script) using the rxSetVarInfo function. Note that, when this process is completed, the variable origin_code will exist in both the flights data file FlightsRfXdf and AirportNames so that it can be used as the key in the merge step. We then use the rxMerge function to perform a left

outer join, with FlightsRfXdf as the "left file", the file for which all records will be retained, and AirportNames as the "right" file, the file whose records will be merged in. As mentioned above, the variable origin_airport which exists in both files is the matching variable. Table 6 shows five rows from the merged files beginning with row 300,000.

Table 6. 5 rows from the merged FlightsRf and AirportNames File

```
#> rxGetInfo(data="MergedFlights",numRows=5,startRow=300000)
#File name: C:\Users\Joseph\Documents\Revolution\DATA STEP WHITE
PAPER\MergedFlights.xdf
#Number of observations: 3606803
#Number of variables: 6
#Number of blocks: 16
#Data (5 rows starting with row 3e+05):
  #origin_airport destin_airport passengers flights  month        origin_name
#1            BGM            CVG       1830      83 200311 Greater Binghamton
#2            BGM            CVG        125       4 200311 Greater Binghamton
#3            BGM            CVG       1704      82 200304 Greater Binghamton
#4            BGM            CVG         93       5 200306 Greater Binghamton
#5            BGM            CVG         99       4 200304 Greater Binghamton
```

To get the full names of the destination airports into our newly created airlines data file, we repeat the entire process, again first renaming the variables in AirportNames (BLOCK 5). Table 7 show five lines from the final airlines flights file, FinalFlights.xdf starting at line 300,000. Notice that the data displayed are different from those displayed in Table 6. The reason for this is that during the second merge, the merge of the destination airport names, rxMerge internally sorted the resulting file by the destin_airport variable.

Table 7. 5 rows from the final airlines flights file, FinalFlights.xdf

```
> rxGetInfo(data="FinalFlights",numRows=5,startRow=300000)
#File name: C:\Users\Joseph\Documents\Revolution\DATA STEP WHITE
PAPER\FinalFlights.xdf
#Number of observations: 3606803
#Number of variables: 7
#Number of blocks: 32
#Data (5 rows starting with row 3e+05):
  #origin_airport destin_airport passengers flights  month     origin_name
#1            OXR            BFL         16       1 200602         Ventura
#2            PHX            BFL         53       1 199101 Sky Harbor Intl
#3            PHX            BFL         30       1 199210 Sky Harbor Intl
#4            PHX            BFL        102       1 199301 Sky Harbor Intl
#5            PHX            BFL         93       1 199407 Sky Harbor Intl
    #destin_name
#1 Meadows Field
#2 Meadows Field
#3 Meadows Field
#4 Meadows Field
#5 Meadows Field
```

Exploring the Data

Let's continue to exercise the data manipulation features of RevoScaleR by exploring the airlines flights data file (BLOCK 6). Just to pick a place to start consider the question: which origin airports had the most flights in any given month? A natural way to proceed might be to sort the data and have a look. First, we use rxSort to sort the data by the flights variable in decreasing order to produce a list showing the airport origin / destination pairs with the most flights in any month.

The rxGetInfo function which we have seen before extracts the first five lines of this file and writes it into a list (Table 8).

Table 8. List showing airport pairs with most flights in a month

```
#> mostflights5
#File name: C:\Users\Joseph\Documents\Revolution\DATA STEP WHITE
PAPER\sortFlights.xdf
#Number of observations: 3606803
#Number of variables: 7
#Number of blocks: 32
#Data (5 rows starting with row 1):
  #origin_airport destin_airport passengers flights  month
#1             SFO            LAX      83153    1128 199412
#2             LAX            SFO      80450    1126 199412
#3             HNL            OGG      73014    1058 199408
#4             OGG            HNL      77011    1056 199408
#5             OGG            HNL      63020    1044 199412
                  #origin_name                destin_name
#1 San Francisco International   Los Angeles International
#2   Los Angeles International San Francisco International
#3      Honolulu International                    Kahului
#4                     Kahului      Honolulu International
#5                     Kahului      Honolulu International
```

SF0 and LAX are right up there at the top and it appears that quite a bit of flying went on during the Christmas season of 1994. Also note that five airports each make the 10 ten list twice. It might be nice to see what activity was like for these airports over the time period of the whole data set. In preparation for generating a histogram showing the distribution of flights for these airports we will employ rxDataStep to create a file containing only the flights having SFO and LAX as the origin airports. The code to do this and some important housekeeping regarding the factor levels is in BLOCK 6. Notice that the variable top2 contains the character vector "SFO" "LAX". The mechanism for making this global variable from the R environment to the rxDataStep function which will select out the SFO and LAX flights into a new xdf file, "mostFlights" is through the argument transformObjects. This argument must be a named list, so we have transformObjects = list(top2to = top2), denoting a different name for the variable used in the transform environment.

The housekeeping is handled through the transforms argument in rxDataStep. This transform creates a new varaible origin having the two factor levels in top2 from the variable orgin_airport which has 683 factor levels.

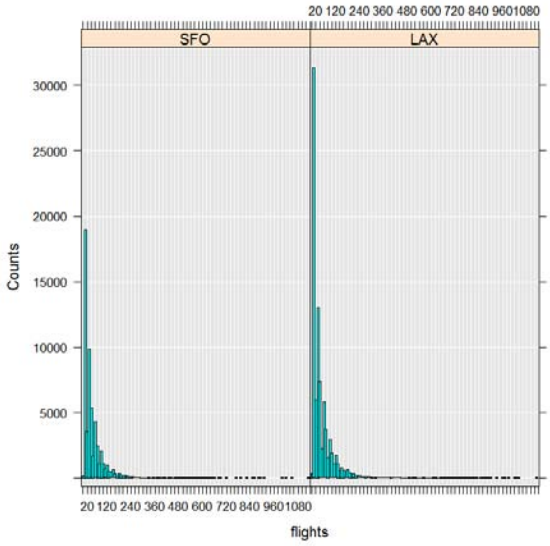
To summarize: the input file to rxDataStep is the flights data file, FinalFlightsXdf, the output file is the smaller, new file, mostFlightsXdf. The rowSelection parameter uses the transform object vector top2to to pick out only the 2 origin airports in which we are interested. The argument transforms is set to the expression we looked at above. The argument transformObjects is the bridge between the R environment and the transform environment. It makes the vector top2to available to the data step. Overwrite = TRUE merely lets one overwrite a file that already exists. Table 9 displays the variables in mostFlights.xdf. Notice that the new variable origin is a factor with only 2 levels, not the 683 levels of origin_airport. The reduced number of levels makes it possible to use origin as the selection variable in the histogram function.

With the new file in place a, call to rxHistogram plots the monthly flight data for the the top 2 origin / destination airport pairs by the origin airports using the new variable created for this purpose. Figure 1 shows the results.

Table 9: Contents of file mostFlights

```
#> rxGetInfo("mostFlights",getVarInfo=TRUE)
#File name: C:\Users\Joseph\Documents\Revolution\DATA STEP WHITE
PAPER\mostFlights.xdf
#Number of observations: 144505
#Number of variables: 8
#Number of blocks: 32
#Variable information:
#Var 1: origin_airport
       #727 factor levels: 1B1 ABE ABI ABQ ABR ... YKN YNG YUM ZXX ZZV
#Var 2: destin_airport
       #727 factor levels: 1B1 ABE ABI ABQ ABR ... YKN YNG YUM ZXX ZZV
#Var 3: passengers, Type: integer, Low/High: (0, 83153)
#Var 4: flights, Type: integer, Low/High: (0, 1128)
#Var 5: month, Type: character
#Var 6: origin_name, Type: character
#Var 7: destin_name, Type: character
#Var 8: origin
       #2 factor levels: SFO LAX
```

Figure 1: Monthly flight distributions of 2 busiest airports



Aggregating Data

The edge airlines file that we have been working contains time stamped data from which several time series may be easily extracted. Each record contains the number of flights and the number of passengers that flew from one airport to another during a given month. So, for each origin airport, by aggregating the counts in all of the destination airports, we can produce a monthly time series of outgoing flights. This is an example of a kind of data aggregation task at which R excels and for which the RevoScaleR package has a function (rxCube) that is particularly useful for boiling down large data sets. It fits nicely with our notion of an extended data step and provides another opportunity to work with rxDataStep.

Our first task will be to split apart the month field in Table 1 into separate month and year variables that we will eventually turn into a "Date" data type. We begin by writing new transforms to parse the character strings that comprise the month variable and use rxDataStep to create the

new variables, Month and Year, in the file (BLOCK 7). We also drop the variable origin_airport from the data set, and specify a new block size for the .xdf file of 50000 rows.

The next step is to use the function rxCube to aggregate the data into a time series. rxCube is a cross tabulation function that returns its output in long form (one row for each combination) rather than in a table. When the formula that describes the tabulation to be done contains a dependent numeric variable, rxCube computes the average of the instances that were tabulated along with a new variable, Counts, that contains the number of that went into computing the average. In our case, the Table 10 indicates that for the 1st month of 1990 the flights from SFO from 284 destination airports were summed and used to compute an average or 39.04 flights per destination. Hence, we can multiply Counts by flights to get the total number of flights out of SFO for each time period (11,088 flights for January 1990).

Table 10. Result of rxCube

```
#head(t1)
#    Year   Month origin  flights Counts
#1   1990       1    SFO 39.04225    284
#2   1991       1    SFO 38.42034    295
#3   1992       1    SFO 46.23954    263
#4   1993       1    SFO 44.39464    261
#5   1994       1    SFO 36.15417    240
#6   1995       1    SFO 45.76768    198
```

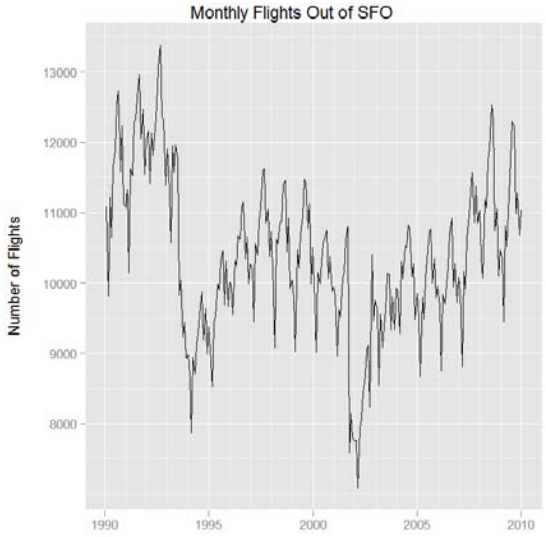
Next, we rename the variables and combine Month and Year into a variable of type Date. (Note that we arbitrarily selected the 28th of each month to fill out the date format.) Table 12 displays the first six lines of the new data frame.

Table 11. Data frame with date

```
# head(t1)
#  Year Month origin avg.flights.per.destin total.destin flights.out
Date
#1 1990     1    SFO               39.04225          284       11088 1990-
01-28
#2 1991     1    SFO               38.42034          295       11334 1991-
01-28
#3 1992     1    SFO               46.23954          263       12161 1992-
01-28
#4 1993     1    SFO               44.39464          261       11587 1993-
01-28
#5 1994     1    SFO               36.15417          240        8677 1994-
01-28
#6 1995     1    SFO               45.76768          198        9062 1995-
01-28
```

Finally, we select SFO, sort by date to form a time series of total monthly flights originating from SFO and plot. Figure 2 shows the monthly flights out of SFO.

Figure 2. Time series of flights out of SFO



Next, although it is really not in the scope of this paper to do any time series analysis, just for fun we perform a seasonal decomposition of the SFO time series and plot it in Figure 3. The first panel of Figure 3 reproduces the time series. The second panel shows the periodic, seasonal component. The third panel displays the trend and the fourth panel displays the residuals. Figure 4 and Figure 5 are the corresponding plots for the LAX time series.

Figure 3. Seasonal decomposition of the SFO series

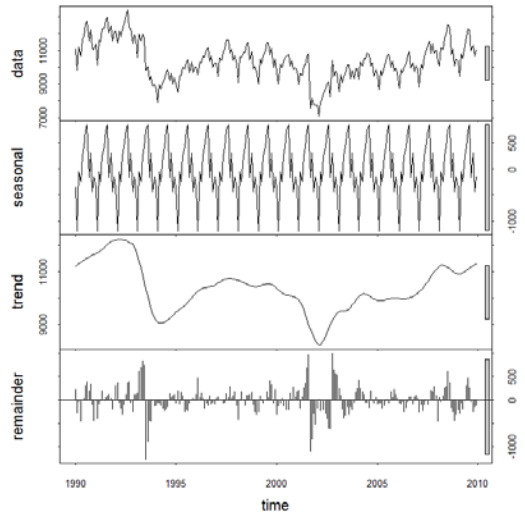


Figure 4. Time series of LAX flights

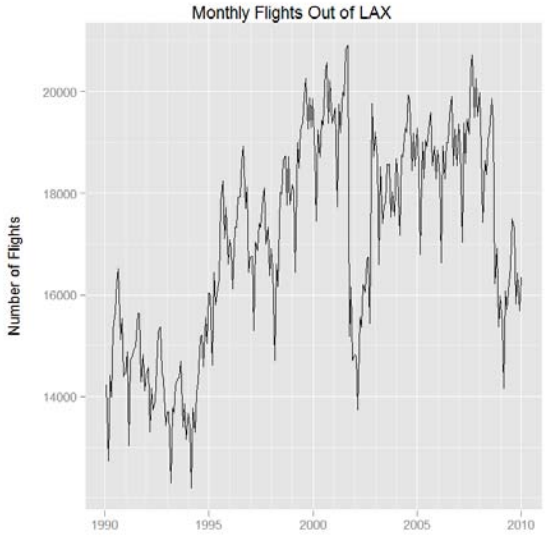
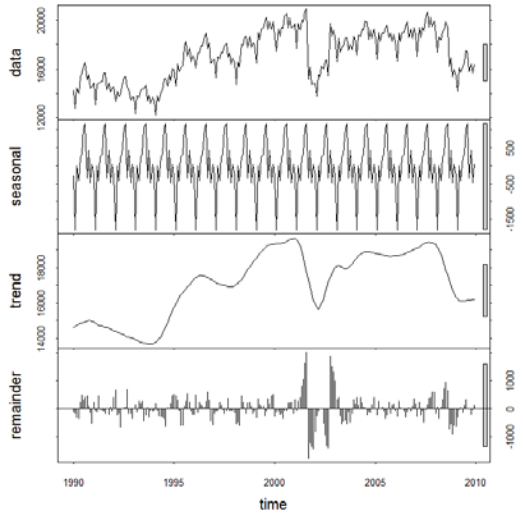


Figure 5. Seasonal decomposition of LAX flights



SUMMARY

In this paper we have explored some of the data handling capabilities of Revolution Analytics RevoScaleR package by working through a practical example that included reading, writing, sorting and merging xdf files; selecting and transforming variables; and aggregating time-stamped data to produce multiple time series. Although the data sets employed in the example were not very large, the section on merging files was done entirely with external memory algorithms and xdf files to show how a similar process could be accomplished with arbitrarily large files.

The extended example also illustrated the natural way in which the data structures and functions of RevoScaleR integrate into an interactive R workflow. RevoScaleR provides all of the data handling and data preparation capabilities required to R ready for big data.

About Revolution Analytics

Revolution Analytics is the leading commercial provider of software and services based on the open source R project for statistical computing. Led by predictive analytics pioneer Norman Nie, the company brings high performance, productivity and enterprise readiness to R, the most powerful statistics language in the world. The company's flagship Revolution R Enterprise product is designed to meet the production needs of large organizations in industries such as finance, life sciences, retail, manufacturing and media.

Please visit us at www.revolutionanalytics.com

APPENDIX 1: rxDataStep {RevoScaleR}

Data Step for .xdf Files and Data Frames

Description

Transform data from an input ‘.xdf’ file or data frame to an output ‘.xdf’ file or data frame

Usage

```
rxDataStep(inData = NULL, outFile = NULL, varsToKeep = NULL, varsToDrop = NULL,
              rowSelection = NULL, transforms = NULL, transformObjects = NULL,
              transformFunc = NULL, transformVars = NULL,
              transformPackages = NULL, transformEnvir = NULL,
              append = "none", overwrite = FALSE, removeMissings = FALSE,
              computeLowHigh = TRUE, maxRowsByCols = 3000000,
              rowsPerRead = -1, startRow = 1, numRows = -1,
             returnTransformObjects = FALSE,
              blocksPerRead = rxGetOption("blocksPerRead"),
              reportProgress = rxGetOption("reportProgress"), ...)
```

Arguments

inData	a data frame, a character string specifying the input ‘.xdf’ file, or an RxXdfData object. If NULL, a data set will be created automatically with a single variable, .rxRowNums, containing row numbers. It will have a total of numRows rows with rowsPerRead rows in each block.
inFile	either an RxXdfData object or a character string specifying the input ‘.xdf’ file.
outFile	a character string specifying the output ‘.xdf’ file or an RxXdfData object. If NULL, a data frame will be returned from rxDataStep unless returnTransformObjects is set to TRUE. Setting outFile to NULL and returnTransformObjects=TRUE allows chunkwise computations on the data without modifying the existing data or creating a new data set.
varsToKeep	character vector of variable names to include when reading from the input data file. If NULL, argument is ignored. Cannot be used with varsToDrop.
varsToDrop	character vector of variable names to exclude when reading from the input data file. If NULL, argument is ignored. Cannot be used with varsToKeep.
rowSelection	name of a logical variable or a logical expression (using variables in the data set) for row selection. For example, rowSelection = (age >= 20) & (age <= 65) & (value > 20) will select the rows corresponding to ages on the interval [20, 65] and with values greater than 20. As with all expressions, rowSelection (or transforms) can be defined outside of the function call using the expression function.
transforms	an expression of the form list(name = expression, ...) representing the first round of variable transformations. As with all expressions, transforms (or rowSelection) can be defined outside of the function call using the expression function.
transformObjects	a named list containing objects that can be referenced by transforms, transformsFunc, and rowSelection.
transformFunc	variable transformation function. See rxTransform for details.

`transformVars`	character vector of input data set variables needed for the transformation function. See rxTransform for details.
`transformPackages`	character vector defining additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions, e.g., those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` will be preloaded.
`transformEnvir`	user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.
`append`	either `"none"` to create a new files, `"rows"` to append rows to an existing file, or `"cols"` to append columns to an existing file. If `outFile` exists and `append` is `"none"`, the `overwrite` argument must be set to `TRUE`. Ignored for data frames.
`overwrite`	logical value. If `TRUE`, an existing `outFile` will be overwritten, or if appending columns existing columns with the same name will be overwritten. `overwrite` is ignored if appending rows. Ignored for data frames.
`removeMissings`	logical value. If `TRUE`, rows with missing values will not be included in the output data.
`computeLowHigh`	logical value. If `FALSE`, low and high values will not automatically be computed. This should only be set to `FALSE` in special circumstances, such as when `append` is being used repeatedly. Ignored for data frames.
`maxRowsByCols`	the maximum size of a data frame that will be returned if `outFile` is set to `NULL` and `inData` is an '`.xdf`' file , measured by the number of rows times the number of columns. If the number of rows times the number of columns being created from the '`.xdf`' file exceeds this, a warning will be reported and the number of rows in the returned data frame will be truncated. If `maxRowsByCols` is set to be too large, you may experience problems from loading a huge data frame into memory.
`rowsPerRead`	number of rows to read for each chunk of data read from the input data source. Use this argument for finer control of the number of rows per block in the output data source. If greater than 0, `blocksPerRead` is ignored. Cannot be used if `inFile` is the same as `outFile`. The default value of `-1` specifies that data should be read by blocks according to the `blocksPerRead` argument.
`startRow`	the starting row to read from the input data source. Cannot be used if `inFile` is the same as `outFile`.
`numRows`	number of rows to read from the input data source. If `rowSelection` or `removeMissings` are used, the output data set my have fewer rows than specified by `numRows`. Cannot be used if `inFile` is the same as `outFile`.
`returnTransformObjects`	logical value. If `TRUE`, the list of `transformObjects` will be returned instead of a data frame or data source object. If the input `transformObjects` have been modified, by using `.rxSet` or `.rxModify` in the `transformFunc`, the updated values will be returned. Any data returned from the `transformFunc` is ignored. If no `transformObjects` are used, `NULL` is returned. This argument allows for user-defined computations within a `transformFunc` without creating new data.
`blocksPerRead`	number of blocks to read for each chunk of data read from the data source. Ignored for data frames or if `rowsPerRead` is positive.
`reportProgress`	integer value with options: • `0`: no progress is reported. • `1`: the number of processed rows is printed and updated. • `2`: rows processed and timings are reported. • `3`: rows processed and all timings are reported.

APPENDIX 2: R SCRIPT

```
################ DATA STEP WHITE PAPER CODE ######################################
##################################################################################
# Joseph B. Rickert
# November 2011
##################################################################################
#
# READ MAIN DATA FILE
#
##################################################################################
# BLOCK 1
# READ IN PRIMARY DATA FILE
FlightsText <- "C:/Users/Joseph/Documents/DATA/US Flights/flights_with_colnames.csv"
#
# Note we make month a character field so we can parse it into month and year later
system.time(
    rxImport(inData = FlightsText, outFile = "Flights", overwrite = TRUE,
    colClasses=c(origin_airport = "factor", destin_airport = "factor",
month="character"))
)
#
# Look at meta-data stored with the file
rxGetInfo("Flights",  numRows = 10)
#
rxGetInfo(data = flightsDS, getVarInfo = TRUE)
##################################################################################
#
# MERGE FILES
#
##################################################################################
# BLOCK 2
# Rationalize the factor levels for the origin and
# destination airports in the flights file
flightVarInfo <- rxGetVarInfo("Flights")
originLevels <- flightVarInfo$origin_airport$levels
destinLevels <- flightVarInfo$destin_airport$levels
# Create a combined list of sorted, unique factors
airportLevels <- sort(unique(c(originLevels, destinLevels)))
# Use rxFactors to re-order the factor levels and make them the same
# FlightsRfXdf contains the Redone factor levels
rxFactors(inData = "Flights", outFile = "FlightsRf", overwrite = TRUE,
       factorInfo = list(
        destin_airport = list(newLevels = airportLevels),
        origin_airport = list(newLevels = airportLevels))
)
# Now origin_airport and destin_airport both have the same number of levels
rxGetInfo("FlightsRf", getVarInfo=TRUE)
#
##################################################################################
# BLOCK 3
# Read in the Airport Locations File,
# Set the Code factor levels to the same as those used in the flight data,
# and remove any names that are not used that aren't used
# Only read in the variables of interest: Code and Airport_Name
```

```
LocationsText <- "C:/Users/Joseph/Documents/DATA/Airport
Locations/Airport_locations2.csv"




rxImport(inData = LocationsText, outFile = "AirportNames",
            varsToKeep = c("Code", "Airport_Name"),
               colInfo = list(
                 Code = list(type = "factor", levels = airportLevels )),
            rowSelection = !is.na(Code),
                   overwrite=TRUE)

# Check to see that the levels are the same
rxGetInfo("AirportNames", getVarInfo=TRUE)
################################################################################
# BLOCK 4
#
# Rename the variables in AirportNames to match the desired names
# for the merged data set

nameVarInfo <- rxGetVarInfo("AirportNames")
nameVarInfo$Code$newName <- "origin_airport"
nameVarInfo$Airport_Name$newName <- "origin_name"
rxSetVarInfo(nameVarInfo, data = "AirportNames")

# Perform left merge, matching on origin airport
# FlightsRfXdf is the "left" file.  NamesOrigin is the "right" file.
rxMerge(inData1 = "FlightsRf", inData2 = "AirportNames",
        outFile = "MergedFlights.xdf",
       type = "left", matchVars = c("origin_airport"),
       overwrite = TRUE)
#
rxGetInfo(data = "MergedFlights", numRows=5, startRow=300000)
#
################################################################################
# BLOCK 5
# Rename the variables again to prepare for a merge
# for the destination airport
nameVarInfo <- rxGetVarInfo("AirportNames")
nameVarInfo$origin_airport$newName <- "destin_airport"
nameVarInfo$origin_name$newName <- "destin_name"
rxSetVarInfo(nameVarInfo, data = "AirportNames")
#
# Perform a second left merge, matching on destination airport
# MergedFlights is the "left" file.  NamesDestin is the "right" file.
rxMerge(inData1 = "MergedFlights", inData2 = "AirportNames",
        outFile = "FinalFlights.xdf",
       type = "left", matchVars = c("destin_airport"),
       overwrite = TRUE)
#
rxGetInfo(data = "FinalFlights", numRows=5, startRow=300000)
#
################################################################################
#
# DATA EXPLORATION
#
################################################################################
# BLOCK 6
```

```
# Find airports connections with most flights in a given month
rxSort(inData="FinalFlights", outFile = "sortFlights.xdf", sortByVars="flights",
               decreasing = TRUE,overwrite=TRUE)

rxGetInfo(data="sortFlights")
mostflights5 <- rxGetInfo(data = "sortFlights", numRows=5, startRow=1)
mostflights5
top5f <- as.data.frame(mostflights5[[5]])
topOA <- unique(as.vector(top5f$origin_airport))
topOA
# Select the top 2
top2 <- topOA[1:2]
top2
#
# Build a file with only flights originating at SFO and LAX
# Use a transform object to pass in the top2 levels

rxDataStep(inData = "FinalFlights", outFile = "mostFlights",
             rowSelection = origin_airport %in% top2to,
             transforms = list(
               origin = factor(origin_airport, levels = top2to, labels = top2to)),
             transformObjects = list(top2to = top2),
             overwrite = TRUE)

rxGetInfo("mostFlights", numRows=10, startRow=1)
rxGetInfo("mostFlights", getVarInfo=TRUE)
rxHistogram(~flights|origin, data="mostFlights")
#
###############################################################################
#
# AGGREGATING TIME SERIES DATA
#
###############################################################################
# BLOCK 7
# CREATE A TIME SERIES FILE
# Write transforms to separate out the month and year data

rxDataStep(inData = "mostFlights", outFile = "SFO.LAX",
        transforms = list(
            Month = as.integer(substring(month, 5, 6)),
            Year = as.integer(substring(month, 1, 4))),
            varsToDrop = c("origin_airport"),
            rowsPerRead = 50000,
            overwrite = TRUE )

rxGetInfo(data = "SFO.LAX", numRows=10)
#
#Aggregate data into monthly flights time series
t1 <- rxCube(flights ~ F(Year):F(Month):origin, removeZeroCounts=TRUE, data =
"SFO.LAX")
t1 <- rxResultsDF(t1)
head(t1)
# Compute total flights out and combine month and dat into a date
t1$flights_out <- t1$flights*t1$Counts
names(t1) <-
c("Year","Month","origin","avg.flights.per.destin","total.destin","flights.out")
t1$Date <- as.Date(as.character(paste(t1$Month,"- 28 -",t1$Year)),"%m - %d - %Y")
head(t1)
###############################################################################
# BLOCK 7
# Select SFO entries and plot
```

```
SFO.t1 <- t1[t1$origin=="SFO",]
head(SFO.t1)
SFO.t1 <- SFO.t1[order(SFO.t1$Date),]
# Make x axis a date and plot
x <-SFO.t1$Date
y <-SFO.t1$flights.out
library(ggplot2) # make sure ggplot is package is loaded
qplot(x,y, geom="line",xlab="", ylab="Number of Flights\n",main="Monthly Flights Out
of SFO")
summary(SFO.t1$flights.out)


#
# Try a seasonal decomposition
SFO.ts <- ts(y,start=c(1990,1),freq=12)
sd.SFO <- stl(SFO.ts,s.window="periodic")
plot(sd.SFO)
#
################################################################################
LAX.t1 <- t1[t1$origin=="LAX",]
LAX.t1 <- LAX.t1[order(LAX.t1$Date),]
#
# Make x axis a date and plot
x <-LAX.t1$Date
y <-LAX.t1$flights.out
qplot(x,y, geom="line",xlab="", ylab="Number of Flights\n",main="Monthly Flights Out
of LAX")
summary(LAX.t1$flights.out)
#
# Perform a seasonal decomposition
LAX.ts <- ts(y,start=c(1990,1),freq=12)
sd.LAX <- stl(LAX.ts,s.window="periodic")
plot(sd.LAX)
```