What’s New in
Microsoft R Services
The correct bibliographic citation for this manual is as follows: Microsoft Corporation. 2016. *What’s New in Microsoft R Services*. Microsoft Corporation, Redmond, WA.

**What’s New in Microsoft R Services**  
Copyright © 2016 Microsoft Corporation. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Microsoft Corporation.

U.S. Government Restricted Rights Notice: Use, duplication, or disclosure of this software and related documentation by the Government is subject to restrictions as set forth in subdivision (c) (1) (ii) of The Rights in Technical Data and Computer Software clause at 52.227-7013.


Other product names mentioned herein are used for identification purposes only and may be trademarks of their respective owners.

Microsoft  
One Microsoft Way  
Redmond WA 98052  
U.S.A.

Revised on November 18, 2015

We want our documentation to be useful, and we want it to address your needs. If you have comments on this or any Revolution document, write to revodoc@microsoft.com.
## Contents

1  Overview .................................................................................................................. 1

   1.1  Features New in Revolution R Enterprise 7.5 ...................................................... 1

   1.2  Features New in Revolution R Enterprise 7.4 ...................................................... 1

   1.3  Features New in Revolution R Enterprise 7.3 ...................................................... 2

   1.4  Features New in Revolution R Enterprise 7.2 ...................................................... 3

   1.5  Features New in Revolution R Enterprise 7.1 ...................................................... 4

   1.6  Features New in Revolution R Enterprise 7.0 ...................................................... 5

2  Running the Examples in This Guide ...................................................................... 6

3  Fuzzy Matching with rxGetFuzzyKeys and rxGetFuzzyDist .................................. 6

4  Specifying Low and High Values ........................................................................... 11

5  Naïve Bayes Classifiers ......................................................................................... 15

   5.1  A Simple Naïve Bayes Classifier ...................................................................... 15

   5.2  Missing Data with Naïve Bayes ........................................................................ 16

6  Stepwise Variable Selection .................................................................................. 17

   6.1  Stepwise Variable Selection with rxLinMod...................................................... 17

   6.2  Stepwise Variable Selection with rxLogit ......................................................... 20

   6.3  Stepwise Variable Selection with rxGlm ......................................................... 22

   6.4  Plotting Model Coefficients ............................................................................ 23

   6.5  Variable Selection with Wide Data .................................................................. 26

7  Performance Improvements .................................................................................. 27

   7.1  Wide Dataset Import Improvements ................................................................ 27

   7.2  Modeling Performance for Tree Based Algorithms .......................................... 27

   7.3  Prediction Performance for rxDForest and rxBTrees ...................................... 27

8  Stochastic Gradient Boosting ............................................................................... 27

   8.1  A Simple Binary Classification Forest ............................................................ 27

   8.2  A Simple Regression Forest ............................................................................ 28

   8.3  A Simple Multinomial Forest Model ............................................................... 28

9  Decision Forests ..................................................................................................... 29
9.1 A Simple Classification Forest ................................................................. 29
9.2 A Simple Regression Forest .................................................................. 30
10 Removing Duplicates While Sorting .................................................. 30
11 Using rxDataStep and rxPredict to Create Delimited Text Files .......... 35
12 Converting RevoScaleR Objects for Use in PMML .............................. 38
13 The RevoTreeView Package ................................................................. 41
1 Overview

This guide is an introduction to using the new features of Microsoft R Services. The current release is Microsoft R Services 2016, which includes the following new features:

- New Features
  - It is now Microsoft R Services! With the acquisition of Revolution Analytics by Microsoft, we are quickly moving to integrate the product line. With this release you will see Microsoft R Server 8.0 on Linux and Revolution R Enterprise 8.0 on Windows, both bringing you the full feature set of the Revolution R Enterprise 7, and more!
  - Revolution R Open has become Microsoft R Open.
  - RevoScaleR now includes two new functions (as experimental) for cleaning and analyzing text data: rxGetFuzzyKeys and rxGetFuzzyDist.
- Bug Fixes
  - When using rxDataStep, new variables created in a transformation no longer inherit the rxLowHigh attribute of the variable used to create them.

1.1 Features New in Revolution R Enterprise 7.5

Revolution R Enterprise 7.5 included the following new features and bug fixes:

- New Features
  - Revolution R Open for Revolution R Enterprise, a single installer, replaces the previous RRO and Revolution R Connector installers.
  - RevoScaleR now includes an in-database compute context, RxInSqlServer, for use with SQL Server 2016 CTP 3. See the RevoScaleR SQL Server Getting Started Guide for a tutorial introduction.
  - RevoScaleR now includes a SQL Server data source, RxSqlServerData.
  - The RxOdbcData data source now has write capability.
- Bug Fixes
  - rxGetInfo was failing when an extended class of RxXdfData was used.

1.2 Features New in Revolution R Enterprise 7.4

Revolution R Enterprise 7.4 included the following new features and bug fixes:

- New Features
R is no longer built into Revolution R Enterprise. Instead, Revolution R Enterprise is installed into an existing RRO 8.0.3 installation, after you have also installed the Revolution R Connector.

- The Revobase and RevobaseEnt packages have been removed.
- The new RevoUtils package contains basic utility functions such as `readNews`.
- The new RevoUtilsMath package contains functions to get and set MKL threads.
- The RxHadoopMR compute context now supports additional distributions of Hadoop, including Cloudera CDH 5.2 and CDH 5.3 (with or without Cloudera Manager 5), Hortonworks HDP 2.2, and MapR 4.0.2.
- The new RevoIOQ package contains the RevoIOQ function for running an installation and operational qualification test suite on a single machine.
- RevoScaleR now features a two-process architecture—R itself runs in one process, while RevoScaleR’s underlying computational engine runs in a separate process. For RRE 7.4, this is primarily an infrastructure investment, but we expect to take advantage of the new architecture in future releases by providing exciting new interfaces to the compute engine.
- A new function, rxNaiveBayes, performs classification using Bayes Theorem to determine the probability that an observation belongs to a certain class.
- A new argument, keepStepCoefs, has been added to the rxStepControl function. If TRUE, a data frame, stepCoefs, is returned with the fitted model with rows corresponding to the coefficients and columns corresponding to the iterations. These stepwise components can be visualized by plotting the fitted model with the new rxStepPlot function.

### 1.3 Features New in Revolution R Enterprise 7.3

Revolution R Enterprise 7.3 included the following new features and bug fixes:

- **New Features**
  - The included version of R has been updated to R 3.1.1.
  - The RxHadoopMR compute context now supports additional distributions of Hadoop, including Cloudera CDH 5.1 (with or without Cloudera Manager 5), and MapR 4.0.1.
  - A new function, rxBTrees, provides decision forests fit using a stochastic gradient boosting algorithm. Regression models as well as binary and multi-class classification models are supported.
  - New argument `scheduleOnce` to rxDForest and rxBTrees allows models fit in a Hadoop compute context to be fit using a single MapReduce job.
  - New coercion methods, as.randomForest and as.gbm, are available to allow for greater compatibility of RevoScaleR model objects with third-party packages such as pmml.
  - New argument, `extraVarsToWrite`, to rxPredict allows additional variables to be written to the prediction output.
The RevoPemaR package now supports distributed compute contexts.
The Linux installer has been enhanced to simplify installation on Hadoop clusters.
The Cloudera Manager 5 parcel install has been enhanced.
In-Teradata computations have been tested with Teradata 15.
In-Teradata memory management has been improved.

- **Bug Fixes**
  - rxImport and rxDataStep now work in a local compute context with inData from HDFS when returning a data frame.
  - rxSplit no longer attempts to convert all character data to factors.
  - The Teradata “number” data type is now supported for reading.
  - rxImport from RxOdbcData data source no longer imports logical NULL values as FALSE.
  - The Hadoop convenience function rxHadoopMakeDir no longer throws warnings about NAs; the underlying rxHadoopVersion function is more robust and is now exported.
  - Calls to rxExec on Hadoop clusters now correctly load the methods package.
  - Calls to rxExec no longer fail when used with R primitive functions.

### 1.4 Features New in Revolution R Enterprise 7.2

Revolution R Enterprise 7.2 included the following new features and bug fixes:

- **New Features**
  - The included version of R has been updated to R 3.0.3.
  - The RxHadoopMR compute context now supports Hadoop systems that are Kerberos-enabled. Users must obtain Kerberos tickets via kinit or keytab authorization before running Hadoop jobs.
  - The RxHadoopMR compute context now supports additional distributions of Hadoop, including Cloudera CDH5 (with or without Cloudera Manager 5), Hortonworks 2.x, and MapR 3.0.2 and 3.1.1 (3.1.0 is supported with the Escalated Bug Fix patch for a file overwrite issue).
  - A new argument, useSparseCube, has been added to the rxCube, rxCrossTabs, and rxSummary functions. If TRUE, RevoScaleR attempts to perform cross-tabulations using a sparse cube. If your data has many factor levels and the full cube will not fit into memory, setting this argument to TRUE may allow your cross-tabulation to complete, because cells with no entries are omitted from the table in memory.
  - The RxTextData data source now supports additional arguments when useFastRead=TRUE: these are the combineDelimiters, quoteMark, decimalPoint, and thousandsSeparator arguments (though note that thousandsSeparator is only used if decimalPoint is specified).
  - A new, experimental R package, RevoPemaR, provides an R-level interface to RevoScaleR’s Parallel External Memory Algorithm (PEMA) infrastructure.
The rxTeradataDropTable and rxTeradataTableExists functions can now optionally take a data source with a table specification as its table argument.

The rxSetVarInfo function can now be used to change variable names, factor level names, and low/high values (used in the F() function) for composite XDF files in HDFS. Only the composite metafile is rewritten.

The rxFactors function can now be used to create or recode factor variables in a composite XDF file in HDFS. A new file must be written out.

### Bug Fixes

- The numCoresToUse argument of rxOptions is now being respected by the nodes of an LSF cluster when performing RevoScaleR high-performance analytics.
- Jobs running in-database on Teradata no longer build up empty Rtmp directories in the system tmp directory.
- The rxReadXdf function and rxGetInfo functions were ignoring startRow if set to a number larger than 2147483647.
- The rxDataStep function now reports a warning when maxRowsByCols is exceeded.
- The rxDTTree function has an improved error message when the fweights argument is set incorrectly.
- The rxGetVarInfo function now returns 1 as the low value for a factor variable, and the number of levels as the high value.
- When using rxDataStep to write to a .csv file, character data were ignoring the missing value string.
- When using an RxTextData data source with a delimited text data file without column names and with useFastRead set to FALSE, the column names now default to V1, V2, etc.
- rxImport was failing with an error when varsToDrop and transforms were both used with an input data frame.
- Error handling has been improved when importing data from a delimited text data file with a varying number of columns.

### 1.5 Features New in Revolution R Enterprise 7.1

Revolution R Enterprise 7.1 included the following new features and bug fixes:

#### New Features

- New compute context, RxInTeradata, allows RevoScaleR High Performance Analytics (HPA) functions such as rxSummary and rxLinMod to run in-database on Teradata data warehouses. Complete details for getting started with RevoScaleR in Teradata can be found in an additional guide, *RevoScaleR Teradata Getting Started Guide* (RevoScaleR_Teradata_Getting_Started.pdf).
- The RxHadoopMR compute context now supports Hadoop systems configured in “run-as-user” mode. This support required a change in the RxHadoopMR constructor, so jobs created with earlier versions of the RxHadoopMR constructor will not be compatible with Revolution R Enterprise 7.1—delete any leftover jobs in your shared directories.
- The RxHadoopMR compute context no longer uses ssh when running directly on a cluster or edge node.
- You can now specify low and high values for variables with a data source's `colInfo` argument.
- The RxTextData data source has a new argument `quotedDelimiters`; if TRUE, delimiters within quoted strings are ignored.
- The included version of the Intel Math Kernel Library has been updated to Version 11.1.1.

**Bug Fixes**
- The rxCov, rxCor, and rxSCCP functions can now be used with a transforms argument in a distributed compute context.
- A memory leak when writing character data to a compressed .xdf file has been fixed.
- The rxKmeans function now respects the `blocksPerCompositeFile` slot of a composite .xdf outFile.
- The outFile argument for the rxCube function is now supported in an RxHadoopMR compute context.
- When printing data sources, information about non-default settings is now included.
- The rxImport function now successfully handles comma-delimited text data files without column names containing only character data.
- Labels for rxHistogram are improved when using non-xdf data sources with `startVal` and `endVal` provided.
- The rxLogit and rxGlm functions now support specification of low and high values with F in a formula, such as y ~ F(x, low=1, high=10).
- The rxLogit and rxGlm functions no longer remove all missing values on read.
- The function rxGetVarInfo no longer computes factor levels for non-xdf data sources unless `computeInfo` is set to TRUE.
- In rxImport, `collInfo` specifications are no longer lost when `varsToKeep` includes all variables in order.
- In rxImport, handling of warning and error messages has been improved when importing a delimited text file containing a varying number of columns per row.

### 1.6 Features New in Revolution R Enterprise 7.0

The principal new features introduced in Revolution R Enterprise 7.0 are the following:

- A newer version of R, R 3.0.2, is included.
- For the included RevoScaleR package:
  - New compute context, RxHadoopMR, allows RevoScaleR High Performance Analytics (HPA) functions such as `rxSummary` and `rxLinMod` to run as MapReduce jobs on a Hadoop cluster. In addition, the `rxExec` function can be used to compute arbitrary R functions in parallel. Complete details for getting started with RevoScaleR in Hadoop can be found in an additional guide, *RevoScaleR Hadoop Getting Started Guide* (RevoScaleR_Hadoop_Getting_Started.pdf).
  - Stepwise functionality has been added to rxLogit and rxGlm.
Running the Examples in This Guide

The examples in this guide use the following data sets:

- **ccFraud.csv**: a comma-separated text data file consisting of 10 million observations of 9 variables related to credit card fraud. This file can be found and online.
- **TestStepwise.xdf**: a .xdf format data file created in the stepwise regression section of this guide.
- **AirlineDemoSmall.xdf**: a smaller sample of the airline data used in one of the examples of removing duplicates while sorting. This data set is built into RevoScaleR as part of its SampleData directory.
- **kyphosis**: a small data frame used in the stepwise variable selection section. This data set is available as part of the rpart package.
- **claims.xdf**: a classic example of automobile insurance claims data that is also built into RevoScaleR as part of its SampleData directory.

3 Fuzzy Matching with rxGetFuzzyKeys and rxGetFuzzyDist

RevoScaleR now provides two functions (as experimental) that are useful for cleaning and analysing text data:
- rxGetFuzzyKeys will provide a phonetic transformation of text data. It can process a character vector of data, a character column in a data frame, or a string variable in a RevoScaleR data source.
- rxGetFuzzyDist with provide a similarity distance measure.

Two basic algorithms are provided for phonetic transformations: Soundex and Metaphone3, with variations on both of them. The Soundex variants are as follows:

- The simplified Soundex algorithm creates a 4-letter code: the first letter of the word as is, followed by 3 numbers representing consonants in the word, or zeros if necessary to fill out the four characters. Non-alphabetical characters are ignored. The Soundex method is used widely, and is available, for example, on the RootsWeb web site http://searches.rootsweb.ancestry.com/cgi-bin/Genea/soundex.sh.
- American Soundex differs slightly from standard simplified Soundex in its treatment of ‘H’ and ‘W’, which are treated differently when present between two other consonants. Setting the keyType argument to soundexAm provides this variant. This is described by the U.S. National Archives, http://www.archives.gov/research/census/soundex.html.
- Another variant of Soundex is to code the first letter in the same way that all letters are coded, giving the first letter less importance. Setting the keyType argument to soundexNum provides this variant.
- Another popular variant of Soundex is to continue coding all letters rather than stopping at 4. Extra zeros are not added to fill out the code, so the result may be less than 4 characters. For this variant, use soundexAll as the keyType. Note that in this case, spaces are treated as vowels in separating consonants.

The Metaphone3 algorithm was developed by Lawrence Philips, who designed and developed the original Metaphone algorithm in 1990 and the Double Metaphone algorithm in 2000. Metaphone3 is reported to increase the accuracy of phonetic encoding from the 89% of Double Metaphone to 98%, as tested against a database of the most common English words, and names and non-English words familiar in North America. The Metaphone3 variants are as follows:

- In Metaphone3, all vowels are encoded to the same value - 'A'. If mphone3 is used as the keyType, only initial vowels will be encoded at all.
- If mphone3Vowels is used as the keyType, 'A' will be encoded at all places in the word that any vowels are normally pronounced. 'W' as well as 'Y' are treated as vowels.
- The mphone3Exact keyType is a variant of Metaphone3 that allows you to encode consonants as exactly as possible. This does not include 'S' vs. 'Z', since Americans will pronounce 'S' at the at the end of many words as 'Z', nor does it include 'CH' vs. 'SH'. It does cause a distinction to be made between 'B' and 'P', 'D' and 'T', 'G' and 'K', and 'V' and 'F'.

In phonetic matching, all non-alphabetic characters are ignored, so that any strings containing numbers may give misleading results. This is typically handled by removing
numbers from strings and handling them separately, but occasionally it is convenient to have the numbers converted to their phonetic equivalent. This is accomplished by setting the `ignoreNumbers` parameter to `FALSE`.

Phonetic matching also typically treats the entire string as a single word. Setting the `ignoreSpaces` argument to `TRUE` results in each word being processed separately, with the result concatenated into a single (longer) code.

In addition, some simple non-phonetic options are provided:

- If `all` is used as the `keyType`, all characters are retained (no transformation occurs).
- If `alphanum` is used as the `keyType`, only alphanumeric characters are retained; no other transformation is performed.
- If `alpha` is used as the `keyType`, only letters are retained; no other transformation is performed.

Text transformations are often combined with similarity distance measures to determine if the strings match. Four similarity distance measures are provided: Levenshtein, Jaro, Jaro-Winkler, and Bag of Words.

- The Levenshtein edit-distance algorithm computes the least number of edit operations (number of insertions, deletions, and substitutions) that are necessary to modify one string to obtain another string.
- The Jaro distance measure considers the number of matching characters and transpositions.
- Jaro-Winkler adds a prefix-weighting, giving higher match values to strings where prefixes match.
- The Bag of Words measure looks at the number of matching words in a phrase, independent of order.

In the implementation used here, all measures are normalized to the same scale, where 0 is no match and 1 is a perfect match.

Some simple examples follow:

1) Compare soundex and American soundex

```r
origStr <- c("Ashcroft", "FLASHCARD")
soundexCode <- rxGetFuzzyKeys(stringsIn = origStr, keyType = "soundex")
soundexAmCode <- rxGetFuzzyKeys(stringsIn = origStr, keyType = "soundexAm")
# Show the original string, Soundex, and American Soundex codes
data.frame(origStr, soundexCode, soundexAmCode)
```

```
data.frame(origStr, soundexCode, soundexAmCode)
   origStr      soundexCode soundexAmCode
1  Ashcroft        A226          A261
2 FLASHCARD        F422          F426
```
2) Compare Metaphone3 with Metaphone3 with internal vowels

```r

mphone3Code <- rxGetFuzzyKeys(stringsIn = origStr, keyType = "mphone3")
mphone3VowelsCode <- rxGetFuzzyKeys(stringsIn = origStr, keyType = "mphone3Vowels")
data.frame(origStr, mphone3Code, mphone3VowelsCode)
```

```
origStr mphone3Code mphone3VowelsCode
1 Phorensics FRNSKS FARANSAK
2 forensics FRNSKS FARANSAK
3 Nicholas NKLS NAKALAS
4 Nicolas NKLS NAKALAS
5 Nikolas NKLS NAKALAS
6 Knight NT NAT
7 Night NT NAT
8 Stephen STFN STAFAN
9 Steven STFN STAFAN
10 Matthew M0 MA0A
11 Matt MT MAT
12 Shuan XN XAN
13 Shawn XN XAN
14 McDonald MKTNLT MKTANALT
15 MacDonald MKTNLT MAKTANAL
16 Schwarzenegger XRTSNKR XARTSANA
17 Shwardseneger XRTSN3R XARTSANA
18 Ellen LN ALAN
19 Elena LN ALANA
20 Allen LN ALAN
```

3) Include numbers and spaces with Metaphone3

```r
origStr <- c("10 Main Apt 410", "20 Main Apt 300", "20 Main Apt 302")
mphone3Code <- rxGetFuzzyKeys(stringsIn = origStr, keyType = "mphone3")
mphone3NumCode <- rxGetFuzzyKeys(stringsIn = origStr, keyType = "mphone3", ignoreNumbers = FALSE)
mphone3NumSpCode <- rxGetFuzzyKeys(stringsIn = origStr, keyType = "mphone3", ignoreNumbers = FALSE, ignoreSpaces = FALSE)
data.frame(origStr, mphone3Code, mphone3NumCode, mphone3NumSpCode)
```

```
origStr mphone3Code mphone3NumCode mphone3NumSpCode
1 10 Main Apt 410 MNPT NSRMNPTF NSR MN APT FRNSR
2 20 Main Apt 300 MNPT TSRMNPT0 TSR MN APT 0RSRSR
3 20 Main Apt 302 MNPT TSRMNPT0 TSR MN APT 0RSRT
```
4) Perform data clean-up on an xdf file

```r
# Create temporary xdf file with data to be cleaned

# Put the data frame into an .xdf file with 2 blocks
tempInFile <- tempfile(pattern = "fuzzyDistIn", fileext = ".xdf")
rxDataStep(inData = inData, outFile = tempInFile, rowsPerRead = 10)

# Create a dictionary of institution names

tempOutFile <- tempfile(pattern = "fuzzyDistOut", fileext = ".xdf")

# Use rxGetFuzzyKeys to create a new variable of cleaned names
outDataSource <- rxGetFuzzyKeys(stringsIn = "institution", data = tempInFile, outFile = tempOutFile, dictionary = uDictionary, ignoreWords = c("University", "Univ", "of", "U"), ignoreCase = TRUE, matchMethod = "bag", ignoreSpaces = FALSE, minDistance = .4, keyType = "mphone3", overwrite = TRUE)

rxGetVarInfo(outDataSource)

Var 1: institution, Type: character
Var 2: institution.key, Type: character

# Read the new data set back into memory
newData <- rxDataStep(outDataSource)

# See the 'cleaned-up' institution names in the new institution.key variable
# Note that one input, Seattle Inst Tech, was not
```
# cleaned
defined

<table>
<thead>
<tr>
<th>institution</th>
<th>institution.key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle Pacific U</td>
<td>Seattle Pacific University</td>
</tr>
<tr>
<td>SEATTLE UNIV</td>
<td>Seattle University</td>
</tr>
<tr>
<td>Seattle Central</td>
<td>Seattle Central College</td>
</tr>
<tr>
<td>U Washington</td>
<td>University of Washington</td>
</tr>
<tr>
<td>UNIV WASH BOTHELL</td>
<td>University of Washington, Bothell</td>
</tr>
<tr>
<td>Puget Sound Univ</td>
<td>Puget Sound University</td>
</tr>
<tr>
<td>ANTIOC U SEATTLE</td>
<td>Antioch University, Seattle</td>
</tr>
<tr>
<td>North Seattle Univ</td>
<td>North Seattle College</td>
</tr>
<tr>
<td>North Seatle U</td>
<td>North Seattle College</td>
</tr>
<tr>
<td>Seattle Inst Tech</td>
<td>Seattle Inst Tech</td>
</tr>
<tr>
<td>Seattle College Nth</td>
<td>North Seattle College</td>
</tr>
<tr>
<td>UNIV waSHINGTON</td>
<td>University of Washington</td>
</tr>
<tr>
<td>University Seattle</td>
<td>Seattle University</td>
</tr>
<tr>
<td>Seattle Antioch U</td>
<td>Antioch University, Seattle</td>
</tr>
<tr>
<td>Technology Institute Seattle</td>
<td>Seattle Institute of Technology</td>
</tr>
<tr>
<td>Pgt Sound U</td>
<td>Puget Sound University</td>
</tr>
<tr>
<td>Seattle Central Univ</td>
<td>Seattle Central College</td>
</tr>
<tr>
<td>Univ North Seattle</td>
<td>North Seattle College</td>
</tr>
<tr>
<td>Bothell - Univ of Wash</td>
<td>University of Washington, Bothell</td>
</tr>
<tr>
<td>ANTIOCH SEATTLE</td>
<td>Antioch University, Seattle</td>
</tr>
</tbody>
</table>

def.clean(tempInFile)
def.clean(tempOutFile)

4 Specifying Low and High Values

RevoScaleR uses minimum and maximum values in computing histograms, and can very efficiently convert integer data to categorical factor data “on-the-fly” using the F function. You can include the specification of high and low values in a RxTextData (or any) data source.

We start by creating a simple RxTextData data source, which we will use to obtain some summary information about the ccFraud.csv data set:

```r
fraudTextDS <- RxTextData("C:/data/ccFraud.csv",
  rowsPerRead = 150000)
rxGetVarInfo(fraudTextDS)
```

You should see the following information:

Var 1: custID, Type: integer
Var 2: gender, Type: integer
Var 3: state, Type: integer
Var 4: cardholder, Type: integer
Var 5: balance, Type: integer
Var 6: numTrans, Type: integer
Specifying Low and High Values

Var 7: numIntlTrans, Type: integer
Var 8: creditLine, Type: integer
Var 9: fraudRisk, Type: integer

We can then run rxSummary on our data source to find high and low values for our variables:

```r
rxSummary(~ custID + gender + state + cardholder + balance +
           numTrans + numIntlTrans + creditLine + fraudRisk,
           data = fraudTextDS)
```

Call:
```
rxSummary(formula = ~ custID + gender + state + cardholder + balance +
           numTrans + numIntlTrans + creditLine + fraudRisk, data =
           fraudTextDS)
```

Summary Statistics Results for: ~ custID + gender + state + cardholder +
balance + numTrans + numIntlTrans + creditLine + fraudRisk
Data: fraudTextDS (RxTextData Data Source)
File name: C:/data/ccFraud.csv
Number of valid observations: 1e+07

<table>
<thead>
<tr>
<th>Name</th>
<th>Mean</th>
<th>StdDev</th>
<th>Min</th>
<th>Max</th>
<th>ValidObs</th>
<th>MissingObs</th>
</tr>
</thead>
<tbody>
<tr>
<td>custID</td>
<td>5.000001e+06</td>
<td>2.886751e+06</td>
<td>1</td>
<td>10000000</td>
<td>1e+07</td>
<td>0</td>
</tr>
<tr>
<td>gender</td>
<td>1.382177e+00</td>
<td>4.859195e-01</td>
<td>1</td>
<td>2</td>
<td>1e+07</td>
<td>0</td>
</tr>
<tr>
<td>state</td>
<td>2.466127e+01</td>
<td>1.497012e+01</td>
<td>1</td>
<td>51</td>
<td>1e+07</td>
<td>0</td>
</tr>
<tr>
<td>cardholder</td>
<td>1.030004e+00</td>
<td>1.705991e-01</td>
<td>1</td>
<td>2</td>
<td>1e+07</td>
<td>0</td>
</tr>
<tr>
<td>balance</td>
<td>4.109920e+03</td>
<td>3.996847e+03</td>
<td>0</td>
<td>41485</td>
<td>1e+07</td>
<td>0</td>
</tr>
<tr>
<td>numTrans</td>
<td>2.893519e+01</td>
<td>2.655378e+01</td>
<td>0</td>
<td>100</td>
<td>1e+07</td>
<td>0</td>
</tr>
<tr>
<td>numIntlTrans</td>
<td>4.047190e+00</td>
<td>8.602970e+00</td>
<td>0</td>
<td>60</td>
<td>1e+07</td>
<td>0</td>
</tr>
<tr>
<td>creditline</td>
<td>9.134469e+00</td>
<td>9.641974e+00</td>
<td>1</td>
<td>75</td>
<td>1e+07</td>
<td>0</td>
</tr>
<tr>
<td>fraudRisk</td>
<td>5.960140e-02</td>
<td>2.367469e-01</td>
<td>0</td>
<td>1</td>
<td>1e+07</td>
<td>0</td>
</tr>
</tbody>
</table>

We can refine our fraudTextDS data source by adding high and low information for
balance, numTrans, numIntlTrans, and creditLine to the colInfo used to create the
data source (we have also included information to make the gender, cardholder, and state
variables more informative factor variables):

```r
stateAbb <- c("AK", "AL", "AR", "AZ", "CA", "CO", "CT",
               "DC", "DE", "FL", "GA", "HI", "IA", "ID", "IL", "IN", "KS",
               "KY", "LA", "MA", "MD", "ME", "MI", "MN", "MO", "MS", "MT",
               "NB", "NC", "ND", "NH", "NJ", "NM", "NV", "NY", "OH", "OK",
               "OR", "PA", "RI", "SC", "SD", "TN", "TX", "UT", "VA", "VT",
               "WA", "WI", "WV", "WY")
ccColInfo <- list(
  gender = list(
    type = "factor",
    levels = c("1", "2"),
    newLevels = c("Male", "Female")),
  cardholder = list(
    type = "factor",
    levels = c("1", "2"),
    newLevels = c("Principal", "Secondary")),
  state = list(
    type = "factor",
    levels = c("1", "2"),
    newLevels = c("Principal", "Secondary")))
```
levels = as.character(1:51),
newLevels = stateAbb),
balance = list(low = 0, high = 41485),
numTrans = list(low = 0, high = 100),
numIntlTrans = list(low = 0, high = 60),
creditLine = list(low = 1, high = 75)
)

Again, recreate an RxTextData data source, adding the additional column information:

fraudTextDS <- RxTextData("C:/data/ccFraud.csv",
colInfo=ccColInfo, rowsPerRead=150000)

The \textit{rxHistogram} function will show us the distribution of any of the variables in our data set. For example, let’s look at \textit{creditLine} by \textit{gender}. Internally, \textit{rxHistogram} will call the \textit{RevoScaleR rxCube} analytics function, which will perform the required computations in-database in Teradata and return the results to your local workstation for plotting:

\texttt{rxHistogram(~creditLine|gender, data = fraudTextDS, histType = "Percent")}

We can also call the \textit{rxCube} function directly and use the results with one of many of R’s plotting functions. For example, \textit{rxCube} can compute group means, so we can compute the mean of \textit{fraudRisk} for every combination of \textit{numTrans} and \textit{numIntlTrans}. We’ll use the \textit{F()} notation to have integer variables treated as categorical variables (with a level for
Specifying Low and High Values

each integer value). The low and high levels specified in `colInfo` will automatically be used.

\[
\text{cubel} \leftarrow \text{rxCube} (\text{fraudRisk} \sim F(\text{numTrans}) : F(\text{numIntlTrans}), \text{data} = \text{fraudTextDS})
\]

The `rxResultsDF` function will convert the results of the `rxCube` function into a data frame that can easily be used in one of R’s standard plotting functions. Here we’ll create a heat map using the `levelplot` function from the `lattice` package:

\[
\text{cubePlot} \leftarrow \text{rxResultsDF(cubel)}
\text{levelplot}(\text{fraudRisk} \sim \text{numTrans} * \text{numIntlTrans}, \text{data} = \text{cubePlot})
\]

We can see that the risk of fraud increases with both the number of transactions and the number of international transactions:
5 Naïve Bayes Classifiers

Naïve Bayes is a simple and effective way to classify your data. The algorithm, which is based on Bayes Theorem, creates the decision criteria for assigning class from a training data set which has classes assigned manually to each observation. In RevoScaleR this is done using the `rxNaiveBayes` function. Our implementation assumes the distribution of any continuous independent variables to be Gaussian. The class membership for a test data set is then decided by choosing the class with the largest posterior probability for each observation. This is accomplished with the `rxPredict` function using the Naïve Bayes object from the call to `rxNaiveBayes`.

5.1 A Simple Naïve Bayes Classifier

In this example, we fit a Naïve Bayes model to rpart’s kyphosis data to see which patients are more likely to acquire Kyphosis based on age, number and start. We use the `rxNaiveBayes` function to construct a classifier for the kyphosis data:

```r
data("kyphosis", package="rpart")
kyphNaiveBayes <- rxNaiveBayes(Kyphosis ~ Age + Start + Number, 
data = kyphosis)
kyphNaiveBayes
```

Call:
`rxNaiveBayes(formula = Kyphosis ~ Age + Start + Number, data = kyphosis)`

A priori probabilities:
Kyphosis
  absent  present
0.7901235 0.2098765

Predictor types:
  Variable   Type
1    Age numeric
2    Start numeric
3   Number numeric

Conditional probabilities:
$Age
  Means   StdDev
absent 79.89062 61.86111
present 97.82353 39.27505

$Start
  Means   StdDev
absent 12.609375 4.427967
present  7.294118 4.283175

$Number
  Means   StdDev
absent 3.750000 1.414214
present 5.176471 1.878673
The returned object `kyphNaiveBayes` is an object of class `rxNaiveBayes`. Objects of this class provide the following useful components: `apriori` and `tables`. The `apriori` component contains the conditional probabilities for the response variable, in this case the Kyphosis variable. The `tables` component contains a list of tables, one for each predictor variable. For a categorical variable, the table contains the conditional probabilities of the variable given the target level of the response variable. For a numeric variable, the table contains the mean and standard deviation of the variable given the target level of the response variable. These components are printed in the output above.

We can use our Naïve Bayes object with `rxPredict` to re-classify the Kyphosis variable for each child in our original dataset:

```r
kyphPred <- rxPredict(kyphNaiveBayes, kyphosis)
```

When we table the results from the Naïve Bayes classifier with the original Kyphosis variable, it appears that 13 of 81 children are misclassified:

```r
table(kyphPred[['Kyphosis_Pred']], kyphosis[['Kyphosis']])
```

<table>
<thead>
<tr>
<th></th>
<th>absent</th>
<th>present</th>
</tr>
</thead>
<tbody>
<tr>
<td>absent</td>
<td>59</td>
<td>8</td>
</tr>
<tr>
<td>present</td>
<td>5</td>
<td>9</td>
</tr>
</tbody>
</table>

### 5.2 Missing Data with Naïve Bayes

You can control the handling of missing data using the `byTerm` argument in the `rxNaiveBayes` function. By default, `byTerm` is set to `TRUE`, which means that missings are removed by variable before computing the conditional probabilities. If you prefer to remove observations with missings in any variable before computations are done, set the `byTerm` argument to `FALSE`.

On the other hand, when your training data are missing certain levels of a predictor or predictors that you expect to be present in your test data you might want to add the `smoothingFactor` argument to your `rxNaiveBayes` call. In general, smoothing is used to avoid overfitting your model. It follows that to achieve the optimal classifier you may want to smooth the conditional probabilities even if every level of each variable is observed in your training data.
6 Stepwise Variable Selection

6.1 Stepwise Variable Selection with rxLinMod

Stepwise linear regression is an algorithm that helps you determine which variables are most important to a regression model. You provide a minimal, or lower, model formula and a maximal, or upper, model formula, and using forward selection, backward elimination, or bidirectional search, the algorithm determines the model formula that provides the best fit based on an AIC selection criterion (by default) or a significance level criterion.

In SAS, stepwise linear regression is implemented through PROC REG. In open source R, it is implemented through the function `step`. The problem with using the function `step` in R is that the size of the data set that can be analyzed is severely limited by the requirement that all computations must be done in memory.

RevoScaleR provides an implementation of stepwise linear regression that is not constrained by the use of "in-memory" algorithms. Stepwise linear regression in RevoScaleR is implemented by the functions `rxLinMod` and `rxStepControl`.

To illustrate the functionality, we'll first create a function to create a simple data set, stored on disk as an .xdf file:

```r
createTestData <- function(rowsPerBlock = 100000, numBlocks = 10, fileName) {
  set.seed(100)
  nextData <- function(nrows) {
    x1 <- rnorm(nrows)
    x2 <- rnorm(nrows)
    x3 <- rnorm(nrows)
    x4 <- rnorm(nrows)
    x5 <- rnorm(nrows)
    x6 <- rnorm(nrows)
    y <- 2*x1 + 3*x3 + 4*x5 + 300*rnorm(nrows)
    yLogical <- (x2 + x4 + x6 + rnorm(nrows)) > 0
    df <- data.frame(x1, x2, x3, x4, x5, x6, y, yLogical)
  }
  append <- "none"
  for (i in 1:numBlocks) {
    df <- nextData(nrows = rowsPerBlock)
    rxDataStep(inData = df, outFile = fileName, append = append)
    append <- "rows"
  }
}
```

Now, specify the location for your new file, and call the function to create a dataset with a million rows, stored in 10 blocks:

```r
outputDir <- getwd()  # Specify the location to write your file
```
Now let's fit a linear model with the variable y as the dependent variable, using stepwise variable selection to help us determine which independent variables to include in the model. To use stepwise selection in RevoScaleR, you add the `variableSelection` argument to your call to `rxLinMod`. The `variableSelection` argument is a list, most conveniently created by using the `rxStepControl` function. Using `rxStepControl`, you specify the method (the default, "stepwise", specifies a bidirectional search), the scope (lower and/or upper formulas for the search), and various control parameters. In this case, we'll start with $x_1$ as the only independent variable, and specify a scope of $x_1$ through $x_6$. By default, variable selection is determined using the Akaike Information Criterion, or AIC; this is the R standard. To see the steps of the estimation process instead of the rows processed, we will set `verbose` to 1 and `reportProgress` to 0.

```
linModOut1 <- rxLinMod(y ~ x1, data = testFile, 
                         variableSelection = rxStepControl( 
                             method = "stepwise", 
                             scope = ~ x1 + x2 + x3 + x4 + x5 + x6),
                         reportProgress = 0, verbose = 1)
```

```
> linModOut1 <- rxLinMod(y ~ x1, data = testFile, 
+ variableSelection = rxStepControl( method = "stepwise", 
+ scope = ~ x1 + x2 + x3 + x4 + x5 + x6), 
+ reportProgress = 0, verbose = 1)
```

```
Start: AIC=1.14099e+007
y~x1+x5
DataSet[7, 6]
Row        StepMethod Step  Df    Sum of Sq       RSS    AIC
AIC
[  1,.]      1 x5       1 13335329.2177 90193504065.9490
11409718.6865
[  2,.]      1 x3       1 11722384.1109 90195117011.0558
11409736.5695
[  3,.] <none>   NA     NA    NA 90206839395.1667
11409864.5280
[  4,.]      1 x4       1 146865.2151 90206692529.9515
11409865.8999
[  5,.]      1 x6       1  99464.1783 90206739930.9883
11409865.4253
[  6,.]      1 x2       1   9515.7550 90206829879.4117
11409866.4225
[  7,.]     -1 x1      -1 2392657.1983 90209232052.3650
11409889.0517

Step 1: AIC=1.14097e+007
y~x1+x5
DataSet[6, 6]
Row        StepMethod Step  Df    Sum of Sq       RSS    AIC
AIC
[  1,.]      1 x3       1 11702330.4673 90193358816.6116
11409865.4253
[  2,.] <none>   NA     NA    NA 90206739930.9883
11409865.4253
[  3,.]      1 x4       1  145249.3374 90206692529.9515
11409864.8999
[  4,.]      1 x6       1   99464.1783 90206739930.9883
11409865.4253
[  5,.]      1 x2       1   9515.7595 90206829879.4117
11409866.4225
[  6,.]     -1 x1      -1 2392657.1983 90209232052.3650
11409889.0517
```
You may have noticed when we created the data set that y was not dependent on x2, x4, or x6, so it is not surprising to see that the final model estimated excludes these variables.

If you want a stepwise selection that is more SAS-like, you can specify `stepCriterion="SigLevel"`. If this is set, rxLinMod uses either an F-test (default) or Chi-square test to determine whether to add or drop terms from the model. You can specify which test to perform using the `test` argument. For example, we can refit our model using the SigLevel step criterion with an F-test as follows:

```r
linModOut2 <- rxLinMod(y ~ x1, data = testFile, variableSelection = rxStepControl( method = "stepwise", scope = ~ x1 + x2 + x3 + x4 + x5 + x6, stepCriterion="SigLevel" ))
summary(linModOut2)

> summary(linModOut2)
```
20 Stepwise Variable Selection

Call:
rXLinMod(formula = y ~ x1, data = testFile, variableSelection =
rXStepControl(method = "stepwise",
    scope = ~x1 + x2 + x3 + x4 + x5 + x6, stepCriterion = "SigLevel"))

Linear Regression Results for: y ~ x1 + x2 + x3 + x4 + x5 + x6
File name: [YourDir]\TestStepwise.xdf
Dependent variable(s): y
Total independent variables: 4
Number of valid observations: 1e+06
Number of missing observations: 0

Coefficients:
     Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.2357     0.3003 -0.785  0.433
x1           1.5454     0.3003   5.146 2.67e-07 ***
x5           3.6523     0.3006  12.151 2.22e-16 ***
x3           3.4185     0.3001  11.391 2.22e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 300.3 on 999996 degrees of freedom
Multiple R-squared: 0.0003041
Adjusted R-squared: 0.0003011
F-statistic: 101.4 on 3 and 999996 DF,  p-value: < 2.2e-16
Condition number: 1.0042

In this case (and with many other well-behaved models), the results of using the SigLevel step criterion are identical to those using AIC.

You can control the significance levels for adding and dropping models using the maxSigLevelToAdd and minSigLevelToDrop. The maxSigLevelToAdd specifies a significance level below which a variable can be considered for inclusion in the model; the minSigLevelToDrop specifies a significance level above which a variable currently included can be considered for dropping. By default, for method="stepwise", both levels are set to .15. We can tighten the selection criterion to the .10 level as follows:

linModOut3 <- rxLinMod(y ~x1, data = testFile,
    variableSelection = rxStepControl( method = "stepwise",
        scope = ~ x1 + x2 + x3 + x4 + x5 + x6,
        stepCriterion="SigLevel",
        maxSigLevelToAdd=.10, minSigLevelToDrop=.10))

6.2 Stepwise Variable Selection with rxLogit

Stepwise variable selection is not restricted to linear models. The basic idea extends to logistic regression models fit with rxLogit, and to generalized linear models.

As an example, consider the kyphosis data set in the rpart package. This data set consists of 81 observations of four variables (Age, Number, Kyphosis, Start) in children following corrective spinal surgery; it is used as the initial example of glm in the White
Stepwise Variable Selection

The variable Kyphosis reports the absence or presence of this deformity.

An initial model can be fit as follows:

```r
library(rpart)
initModel <- rxLogit(Kyphosis ~ Age, data=kyphosis)
initModel
```

**Logistic Regression Results for: Kyphosis ~ Age**

Data: kyphosis
Dependent variable(s): Kyphosis
Total independent variables: 2
Number of valid observations: 81
Number of missing observations: 0

Coefficients:

<table>
<thead>
<tr>
<th></th>
<th>Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Intercept)</td>
<td>-1.809351230</td>
</tr>
<tr>
<td>Age</td>
<td>0.005441758</td>
</tr>
</tbody>
</table>

We can specify a stepwise model using `rxLogit` and `rxStepControl` as follows:

```r
KyphStepModel <- rxLogit(Kyphosis ~ Age, 
data = kyphosis, 
variableSelection = rxStepControl(method="stepwise", 
scope = ~ Age + Start + Number ))
KyphStepModel
```

**Logistic Regression Results for: Kyphosis ~ Age + Start + Number**

Data: kyphosis
Dependent variable(s): Kyphosis
Total independent variables: 4
Number of valid observations: 81
Number of missing observations: 0

Coefficients:

<table>
<thead>
<tr>
<th></th>
<th>Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Intercept)</td>
<td>-2.03693354</td>
</tr>
<tr>
<td>Age</td>
<td>0.01093048</td>
</tr>
<tr>
<td>Start</td>
<td>-0.20651005</td>
</tr>
<tr>
<td>Number</td>
<td>0.41060119</td>
</tr>
</tbody>
</table>

Note that the above examples use a small model with a very small data set. By default, stepwise `rxLogit` estimation does a full, iterative re-estimation of the model for each step. For large models with large data sets, we recommend setting the `refitEachStep` argument to `FALSE` in `rxStepControl`. With this option, the full model is fitted and terms are then added or dropped based on the weighted least squares fit. This is an efficient way to approximate a full refitting for each step, and is similar to the FAST option of Logistic Proc in SAS. We can see an example of this using the million-row data set we created in the previous section:

```r
logitOut <- rxLogit(yLogical ~x1, data = testFile, 
variableSelection = rxStepControl( method = "stepwise", 
```
Stepwise Variable Selection

```r
scope = ~ x1 + x2 + x3 + x4 + x5 + x6,
refitEachStep = FALSE)
summary(logitOut)
```

> summary(logitOut)
Call:
rxLogit(formula = yLogical ~ x1, data = testFile, variableSelection =
  rxStepControl(method = "stepwise",
    scope = ~x1 + x2 + x3 + x4 + x5 + x6, refitEachStep = FALSE))

Logistic Regression Results for: yLogical ~ x2 + x4 + x6
File name: [YourDir]\TestStepwise.xdf
Dependent variable(s): yLogical
Total independent variables: 4
Number of valid observations: 1e+06
Number of missing observations: 0
-2*LogLikelihood: 717299.4151 (Residual deviance on 999996 degrees of
freedom)

Coefficients:
   Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.006970   0.002959   2.356   0.0185 *
x2          1.746061   0.004247 411.095 2.22e-16 ***
x4          1.744448   0.004248 410.673 2.22e-16 ***
x6          1.746682   0.004243 411.698 2.22e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Condition number of final variance-covariance matrix: 5.2431
Number of iterations: 7

Since in creating the data, yLogical was not dependent on x1, x3, or x5, it is not surprising that these variables were dropped from the model.

6.3 Stepwise Variable Selection with rxGlm

As an example of stepwise variable selection, consider the classic example of automobile insurance claims. This is an example of data containing positive values with a positive skew, making it a candidate for the Gamma family of generalized linear models. Using the claims.xdf data set, we can proceed as follows:

```r
claimsXdf <- file.path(rxGetOption("sampleDataDir"),"claims.xdf")
claimsGlm <- rxGlm(cost ~ age + car.age + type, family = Gamma,
  dropFirst = TRUE, data = claimsXdf)
summary(claimsGlm)
```

We can recast this as a stepwise model by specifying a variableSelection argument using the rxStepControl function to provide our stepwise arguments:

```r
claimsGlmStep <- rxGlm(cost ~ age, family = Gamma, dropFirst=TRUE,
  data=claimsXdf, variableSelection = rxStepControl(
```
Stepwise Variable Selection

```
summary(claimsGlmStep)
```

Call:
```
rxGlm(formula = cost ~ age, data = claimsXdf, family = Gamma,
variableSelection = rxStepControl(scope = ~age + car.age +
type), dropFirst = TRUE)
```

Generalized Linear Model Results for: cost ~ car.age + type
File name:
```
C:\PROGRA~1\R\R-3.1.3\library\RevoScaleR\SampleData\claims.xdf
```
Dependent variable(s): cost
Total independent variables: 9 (Including number dropped: 2)
Number of valid observations: 123
Number of missing observations: 5
Family-link: Gamma-inverse

Residual deviance: 18.0433 (on 116 degrees of freedom)

Coefficients:

| Parameter | Estimate | Std. Error | t value | Pr(>|t|) |
|-----------|----------|------------|---------|----------|
| (Intercept) | 0.0040354 | 0.0004661 | 8.657 | 3.24e-14 *** |
| car.age=0-3 | Dropped | Dropped | Dropped | Dropped |
| car.age=4-7 | 0.0003568 | 0.0004037 | 0.884 | 0.37868 |
| car.age=8-9 | 0.0011825 | 0.0004688 | 2.522 | 0.01302 * |
| car.age=10+ | 0.0035478 | 0.0006853 | 5.177 | 9.57e-07 *** |
| type=A | Dropped | Dropped | Dropped | Dropped |
| type=B | -0.0004512 | 0.0005519 | -0.818 | 0.41528 |
| type=C | -0.0004135 | 0.0005558 | -0.744 | 0.45837 |
| type=D | -0.0016307 | 0.0004923 | -3.313 | 0.00123 ** |

---

Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for Gamma family taken to be 0.2249467)

Condition number of final variance-covariance matrix: 9.1775
Number of iterations: 5

We see that in the stepwise model fit, age no longer appears in the final model.

Like stepwise `rxLogit`, stepwise `rxGlm` estimation does a full, iterative re-estimation of the model for each step by default. For large models with large data sets, we recommend setting the `refitEachStep` argument to `FALSE` in `rxStepControl` for a much faster approximation.

### 6.4 Plotting Model Coefficients

By default, the values of the parameters at each step of the stepwise selection are not preserved. Using an additional argument, `keepStepCoefs`, in your `rxStepControl` statement saves the values of the coefficients from each step of the regression. This coefficient data can then be plotted using another function, `rxStepPlot`.

Consider the stepwise logistic regression on the kyphosis data in section 5.2:
Notice the addition of the argument `keepStepCoefs = TRUE` to the `rxStepContol` call. This produces an extra piece of output in the `rxLogit` object, a dataframe containing the values of the coefficients at each step of the regression. This dataframe, `stepCoefs`, can be accessed as follows:

```
KyphStepModel$stepCoefs
```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Intercept)</td>
<td>-1.809351230</td>
<td>0.224995554</td>
<td>-2.03693354</td>
</tr>
<tr>
<td>Age</td>
<td>0.005441758</td>
<td>0.009508503</td>
<td>0.01093048</td>
</tr>
<tr>
<td>Start</td>
<td>0.000000000</td>
<td>-0.237938720</td>
<td>-0.20651005</td>
</tr>
<tr>
<td>Number</td>
<td>0.000000000</td>
<td>0.000000000</td>
<td>0.41060119</td>
</tr>
</tbody>
</table>

Trying to glean patterns and information from a table can be difficult. So we've added another function, `rxStepPlot`, which allows the user to plot the parameter values at each step. Using the kyphosis model object, we plot the coefficients:

```
rxStepPlot(KyphStepModel)
```
From this plot, we can tell when a variable enters the model by noting the step when it becomes non-zero. Lines are labelled with the numbers on the right axis to indicate the parameter. The numbers correspond to the order they appear in the data frame `stepCoef`.

These plots can be easily customized by using additional graphical parameters from the `matplot` function. By default the `rxStepPlot` function uses 7 line colors. If the number of parameters exceeds the number of colors, they will be reused in the same order. These colors can be customized by adding the `col` argument to your `rxStepPlot` call. The line types are set to vary from 1 to 5, so lines that have the same color may differ in line type. The line types can also be specified using the `lty` argument in the `rxStepPlot` call.

We demonstrate the `rxStepPlot` for `rxLogit` above, but this method also works with `rxLinMod` and `rxGlm` objects.
6.5 Variable Selection with Wide Data

We’ve found that generalized linear models do not converge if the number of predictors is greater than the number of observations. If your data has more variables, it won’t be possible to include all of them in the maximal model for stepwise selection. We recommend you use domain experience and insights from initial data explorations to choose a subset of the variables to serve as the maximal model before performing stepwise selection.

There are a few things you can do to reduce the number of predictors. If there are a lot of variables that measure the same quantitative or qualitative entity, try to select one variable that represents the entity best. For example, include a variable that identifies a person's political party affiliation instead of including many variables representing how the person feels about individual issues. If your goal with linear modeling is the interpretation of individual predictors, you want to ensure that correlation between variables in the model is minimal to avoid multicollinearity. This means checking for these correlations before modeling. Sometimes it is useful to combine correlated variables into a composite variable. Height and weight are often correlated, but can be transformed into BMI. Combining variables allows you to reduce the number of variables without losing any information. Ultimately, the variables you select will depend on how you plan to use the results of your linear model.
7 Performance Improvements

7.1 Wide Dataset Import Improvements

A bug impacting the XDF column metadata was fixed and importing wide data goes much faster as a result. No syntax changes are needed to achieve this speedup. However, if you are importing wide data and have many character variables that you wish to turn into factors either read them in as strings and transform to factor after import or import using colInfo making sure to pre-specify the factor levels. Another recommendation is to turn off compression using xdfCompressionLevel=0 when importing wide data into XDF format.

7.2 Modeling Performance for Tree Based Algorithms

RevoScaleR’s tree based algorithms, rxBTrees, rxDForest, and rxDTree, now search for the best splits in parallel by default. No syntax change is needed for this improvement.

7.3 Prediction Performance for rxDForest and rxBTrees

To improve the time it takes to perform prediction for rxDForest and rxBTrees, the user can specify scheduleOnce = 2 so the rxPredict algorithm only does a single pass through the data.

8 Stochastic Gradient Boosting

The rxBTrees function in RevoScaleR, like rxDForest, fits a decision forest to your data, but the forest is generated using a stochastic gradient boosting algorithm. This is similar to the decision forest algorithm in that each tree is fitted to a subsample of the training set (sampling without replacement) and predictions are made by aggregating over the predictions of all trees. Unlike the rxDForest algorithm, the boosted trees are added one at a time, and at each iteration, a regression tree is fitted to the current pseudo-residuals, i.e., the gradient of the loss functional being minimized.

Different model types are supported by specifying different loss functions, as follows:

- "gaussian", for regression models
- "bernoulli", for binary classification models
- "multinomial", for multi-class classification models

8.1 A Simple Binary Classification Forest

We fit a simple classification decision forest model to rpart’s kyphosis data using rxBTrees as follows (we set the seed argument to ensure reproducibility; in most cases you can omit this):
data("kyphosis", package="rpart")
kyphBTrees <- rxBTrees(Kyphosis ~ Age + Start + Number, seed = 10,
data = kyphosis, cp=0.01, nTree=500, mTry=3,
lossFunction="bernoulli")
kyphBTrees

Call:
rxBTrees(formula = Kyphosis ~ Age + Start + Number, data = kyphosis,
cp = 0.01, nTree = 500, mTry = 3, seed = 10, lossFunction = "bernoulli")

Loss function of boosted trees: bernoulli
Number of iterations: 500
OOB estimate of deviance: 0.1441952

In this case, we don't need to explicitly specify the loss function; "bernoulli" is the default.

8.2 A Simple Regression Forest

As a simple example of a regression forest, consider the classic stackloss data set, containing observations from a chemical plant producing nitric acid by the oxidation of ammonia, and let's fit the stack loss (stack.loss) using air flow (Air.Flow), water temperature (Water.Temp), and acid concentration (Acid.Conc.) as predictors:

stackBTrees <- rxDForest(stack.loss ~ Air.Flow + Water.Temp + Acid.Conc.,
data=stackloss, nTree=200, mTry=2, lossFunction="gaussian")
stackBTrees

Call:
rxDForest(formula = stack.loss ~ Air.Flow + Water.Temp + Acid.Conc.,
data = stackloss, nTree = 200, mTry = 2, lossFunction = "gaussian")

Loss function of boosted trees: gaussian
Number of iterations: 200
OOB estimate of deviance: 1.46797

8.3 A Simple Multinomial Forest Model

As a simple multinomial example, we will fit an rxBTrees model to the iris data:

irisBTrees <- rxBTrees(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width, data=iris,
nTree=50, seed=0, maxDepth=3, lossFunction="multinomial")
irisBTrees

Call:
rxBTrees(formula = Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width, data = iris, maxDepth = 3, nTree = 50, seed = 0,
lossFunction = "multinomial")
Loss function of boosted trees: multinomial
Number of boosting iterations: 50
No. of variables tried at each split: 1

OOB estimate of deviance: 0.02720805

9 Decision Forests

A decision forest is an ensemble of decision trees. Each tree is fitted to a bootstrap sample of the original data, which leaves about 1/3 of the data unused in the fitting of each tree. Each data point in the original data is fed through each of the trees for which it was unused; the decision forest prediction for that data point is the statistical mode of the individual tree predictions, that is, the majority prediction (for classification; for regression problems, the prediction is the mean of the individual predictions).

Unlike individual decision trees, decision forests are not prone to overfitting, and they are consistently shown to be among the best classification algorithms. RevoScaleR implements decision forests in the rxDForest function, which uses the same basic tree-fitting algorithm as rxDTree (see Section 10.1, The rxDTree Algorithm, in the RevoScaleR User’s Guide). To create the forest, you specify the number of trees using the nTree argument and the number of variables to consider for splitting in each tree using the mTry argument. In most cases, you will also want to specify the maximum depth to grow the individual trees: greater depth typically results in greater accuracy, but as with rxDTree, also results in significantly longer fitting times.

9.1 A Simple Classification Forest

We have already used the kyphosis data set in fitting a logistic regression model; it is easy to use it also for a decision forest:

```r
data("kyphosis", package="rpart")
kyphForest <- rxDForest(Kyphosis ~ Age + Start + Number,
data = kyphosis, nTree=500, mTry=3, maxDepth=6)
kyphForest
```

We obtain the following results:

```
Call:
rxDForest(formula = Kyphosis ~ Age + Start + Number, data = kyphosis,
maxDepth = 6, nTree = 500, mTry = 3)

Type of decision forest: class
Number of trees: 500
No. of variables tried at each split: 3

OOB estimate of error rate: 19.75%
Confusion matrix:
```

```
While decision forests do not produce a unified model, as logistic regression and decision trees do, they do produce reasonable predictions for each data point. In this case, we can obtain predictions using rxPredict as follows:

```r
dfPreds <- rxPredict(kyphForest, data=kyphosis)
```

Comparing this to the Kyphosis variable in the original kyphosis data, we see that approximately 88 percent of cases are classified correctly:

```r
sum(as.character(dfPreds[,1]) == as.character(kyphosis$Kyphosis))/81
[1] 0.8765432
```

### 9.2 A Simple Regression Forest

Using rxDForest for regression is also straightforward; here we work again with the claims data that we used for our initial rxGlm model.

```r
claimsXdf <- file.path(rxGetOption("sampleDataDir"),"claims.xdf")
claimsForest <- rxDForest(cost ~ age + car.age + type, nTree=500, mTry=2, maxDepth=5, data = claimsXdf)

claimsForest
```

We get the following results:

```
Call:
rxDForest(formula = cost ~ age + car.age + type, data = claimsXdf, 
  maxDepth = 5, nTree = 500, mTry = 2)

Type of decision forest: anova
  Number of trees: 500
No. of variables tried at each split: 2
  Mean of squared residuals: 11545.78
    % Var explained: 29
```

With only 29% of the variance explained, the model can clearly be improved.

### 10 Removing Duplicates While Sorting

In many situations, you are sorting a large data set by a particular key, for example, userID, but are looking for a sorted list of unique userIDs. The `removeDupKeys` argument to `rxSort` allows you to remove the duplicate entries from a sorted list. This argument is supported only for `type="auto"` and `type="mergeSort"`; it is ignored for `type="varByVar"`. When
you use `removeDupKeys=TRUE`, the first record containing a unique combination of the `sortByVars` is retained; subsequent matching records are omitted from the sorted results, but, if desired, a count of the matching records is maintained in a new `dupFreqVar` output column. For example, the following artificial data set simulates a small amount of transaction data, with a user name, a state, and a transaction amount. When we sort by the variables `users` and `state` and specify `removeDupKeys=TRUE`, the `transAmt` shown for duplicate entries is the transaction amount for the first transaction encountered:

```r
set.seed(17)
users <- sample(c("Aiden", "Ella", "Jayden", "Ava", "Max", 
                   "Grace", "Riley", "Lolita", "Liam", "Emma", "Ethan", 
                   "Isabella", "Caleb", "Bella"), 100, replace=TRUE)
state <- sample(c("Washington", "California", "Texas", 
                   "North Carolina", "New York", "Massachusetts"), 100, 
                   replace=TRUE)
transAmt <- round(runif(100) * 100, digits=3)
df <- data.frame(users=users, state=state, transAmt=transAmt)

rxSort(df, sortByVars=c("users", "state"), removeDupKeys=TRUE, 
       dupFreqVar = "DUP_COUNT")
```

Number of rows written to file: 66, Variable(s): users, state, transAmt, 
DUP_COUNT, Total number of rows in file: 66

Time to sort data file: 0.004 seconds

<table>
<thead>
<tr>
<th>users</th>
<th>state</th>
<th>transAmt</th>
<th>DUP_COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aiden</td>
<td>New York</td>
<td>11.010</td>
<td>1</td>
</tr>
<tr>
<td>Aiden</td>
<td>North Carolina</td>
<td>73.307</td>
<td>1</td>
</tr>
<tr>
<td>Aiden</td>
<td>Texas</td>
<td>8.037</td>
<td>2</td>
</tr>
<tr>
<td>Aurora</td>
<td>California</td>
<td>8.787</td>
<td>1</td>
</tr>
<tr>
<td>Aurora</td>
<td>Massachusetts</td>
<td>55.187</td>
<td>1</td>
</tr>
<tr>
<td>Aurora</td>
<td>New York</td>
<td>91.648</td>
<td>1</td>
</tr>
<tr>
<td>Aurora</td>
<td>Texas</td>
<td>30.566</td>
<td>1</td>
</tr>
<tr>
<td>Aurora</td>
<td>Washington</td>
<td>27.374</td>
<td>1</td>
</tr>
<tr>
<td>Ava</td>
<td>California</td>
<td>70.638</td>
<td>2</td>
</tr>
<tr>
<td>Ava</td>
<td>Massachusetts</td>
<td>82.916</td>
<td>2</td>
</tr>
<tr>
<td>Ava</td>
<td>New York</td>
<td>45.683</td>
<td>2</td>
</tr>
<tr>
<td>Ava</td>
<td>North Carolina</td>
<td>51.748</td>
<td>1</td>
</tr>
<tr>
<td>Ava</td>
<td>Texas</td>
<td>52.674</td>
<td>1</td>
</tr>
<tr>
<td>Avery</td>
<td>California</td>
<td>9.756</td>
<td>4</td>
</tr>
<tr>
<td>Avery</td>
<td>Massachusetts</td>
<td>63.715</td>
<td>1</td>
</tr>
<tr>
<td>Avery</td>
<td>New York</td>
<td>93.430</td>
<td>1</td>
</tr>
<tr>
<td>Avery</td>
<td>North Carolina</td>
<td>1.889</td>
<td>2</td>
</tr>
<tr>
<td>Bella</td>
<td>California</td>
<td>60.258</td>
<td>2</td>
</tr>
<tr>
<td>Bella</td>
<td>Texas</td>
<td>32.684</td>
<td>1</td>
</tr>
<tr>
<td>Bella</td>
<td>Washington</td>
<td>60.230</td>
<td>2</td>
</tr>
<tr>
<td>Caleb</td>
<td>Massachusetts</td>
<td>94.527</td>
<td>1</td>
</tr>
<tr>
<td>Caleb</td>
<td>North Carolina</td>
<td>89.259</td>
<td>2</td>
</tr>
<tr>
<td>Dylan</td>
<td>California</td>
<td>73.665</td>
<td>1</td>
</tr>
<tr>
<td>Dylan</td>
<td>North Carolina</td>
<td>98.384</td>
<td>1</td>
</tr>
<tr>
<td>Dylan</td>
<td>Texas</td>
<td>27.067</td>
<td>1</td>
</tr>
<tr>
<td>Dylan</td>
<td>Washington</td>
<td>82.141</td>
<td>3</td>
</tr>
<tr>
<td>Elizabeth</td>
<td>California</td>
<td>95.497</td>
<td>2</td>
</tr>
<tr>
<td>Elizabeth</td>
<td>New York</td>
<td>35.546</td>
<td>3</td>
</tr>
</tbody>
</table>
Removing duplicates can be a useful way to reduce the size of a data set without losing information of interest. For example, consider an analysis of using data from the sample AirlineDemoSmall xdf file. It has 600,000 observations and contains the variables DayOfWeek, CRSDepTime, and ArrDelay. We can create a smaller data set reducing the number of observations, and adding a variable that contains the frequency of duplicated observations.

```r
sampleDataDir <- rxGetOption("sampleDataDir")
airDemo <- file.path(sampleDataDir, "AirlineDemoSmall.xdf")
airDedup <- file.path(tempdir(), "rxAirDedup.xdf")
rxSort(inData = airDemo, outFile = airDedup,
   sortByVars = c("DayOfWeek", "CRSDepTime", "ArrDelay"),
   removeDupKeys = TRUE, dupFreqVar = "FreqWt")
rxGetInfo(airDedup)
```
The new data file contains about 1/3 of the observations and one additional variable:

File name: C:\YourTempDir\rxAirDedup.xdf
Number of observations: 232451
Number of variables: 4
Number of blocks: 2

By using the frequency weights argument, we can use many of the RevoScaleR analysis functions on this smaller data set and get same results as we would using the full data set. For example, a linear model for Arrival Delay can be specified as follows, using the fweights argument:

```
linModObj <- rxLinMod(ArrDelay~CRSDepTime + DayOfWeek, data = airDedup,
  fweights = "FreqWt")
summary(linModObj)
```

```
Call:
rxLinMod(formula = ArrDelay ~ CRSDepTime + DayOfWeek, data =
  airDedup,
  fweights = "FreqWt")

Linear Regression Results for: ArrDelay ~ CRSDepTime + DayOfWeek
File name: C:\YourTempDir\rxAirDedup.xdf
Frequency weights: FreqWt
Dependent variable(s): ArrDelay
Total independent variables: 9 (Including number dropped: 1)
Sum of weights of valid observations: 582628
Number of missing observations: 3503

Coefficients: (1 not defined because of singularities)
               Estimate  Std. Error t value Pr(>|t|)
(Intercept)    -3.19458    0.20413 15.650  2.22e-16 ***
CRSDepTime      0.97862    0.01126 86.948  2.22e-16 ***
DayOfWeek=Monday 2.08100    0.18602 11.187  2.22e-16 ***
DayOfWeek=Tuesday 1.34015    0.19881  6.741  1.58e-11 ***
DayOfWeek=Wednesday 0.15155    0.19679  0.770    0.441
DayOfWeek=Thursday -1.32301    0.19518 -6.778  1.22e-11 ***
DayOfWeek=Friday   4.80042    0.19452 24.679  2.22e-16 ***
DayOfWeek=Saturday 2.18965    0.19229 11.387  2.22e-16 ***
DayOfWeek=Sunday   Dropped
DayOfWeek=Su       Dropped
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 40.39 on 582620 degrees of freedom
Multiple R-squared: 0.01465
Adjusted R-squared: 0.01464
F-statistic: 1238 on 7 and 582620 DF, p-value: < 2.2e-16
Condition number: 10.6542
```

Using the full data set, we get the following results:
Removing Duplicates While Sorting

```r
linModObjBig <- rxlInMod(ArrDelay~CRSDepTime + DayOfWeek, data = airDemo)
summary(linModObjBig)

Call:
rxLinMod(formula = ArrDelay ~ CRSDepTime + DayOfWeek, data = airDemo)

Linear Regression Results for: ArrDelay ~ CRSDepTime + DayOfWeek
File name:  C:\YourSampleDir\AirlineDemoSmall.xdf
Dependent variable(s): ArrDelay
Total independent variables: 9 (Including number dropped: 1)
Number of valid observations: 582628
Number of missing observations: 17372

Coefficients: (1 not defined because of singularities)
              Estimate Std. Error   t value Pr(>|t|)
(Intercept)  -3.19458  0.20413  -15.650   2.22e-16 ***
CRSDepTime   0.97862  0.01126   86.948   2.22e-16 ***
DayOfWeek=Monday    2.08100  0.18602   11.187   2.22e-16 ***
DayOfWeek=Tuesday   1.34015  0.19881    6.741   1.58e-11 ***
DayOfWeek=Wednesday  0.15155  0.19679    0.770    0.441
DayOfWeek=Thursday  -1.32301  0.19518   -6.778   2.22e-11 ***
DayOfWeek=Friday     4.80042  0.19452  24.679   2.22e-16 ***
DayOfWeek=Saturday   2.18965  0.19229  11.387   2.22e-16 ***
DayOfWeek=Sunday     Dropped  Dropped  Dropped  Dropped
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 40.39 on 582620 degrees of freedom
Multiple R-squared:  0.01465,  Adjusted R-squared:  0.01464
F-statistic: 1238 on 7 and 582620 DF,  p-value: < 2.2e-16
Condition number: 10.6542
```
11 Using rxDataStep and rxPredict to Create Delimited Text Files

The rxDataStep function can be used to “batch process” data transformations and subset rows and columns. It can take an .xdf file or a data frame as input. Beginning with Revolution R Enterprise 7.0, it can also output data not only to an .xdf file or data frame in memory, but to a delimited text file in a local file system. This makes it much easier to transfer data from an .xdf to a file that can be read by many other programs.

As an example, we can read data from a test .xdf file that has 100 rows stored in 5 blocks. We can take a quick look at the variables in the data file:

```
inTestData <- file.path(rxGetOption("unitTestDataDir"),
                      "test100r5b.xdf")
rxGetVarInfo(inTestData)

> rxGetVarInfo(inTestData)
Var 1: y, Type: numeric, Storage: float32, Low/High: (15.3616, 64.3370)
Var 2: ylogical
  2 factor levels: TRUE FALSE
Var 3: ybinint, Type: integer, Low/High: (0, 1)
Var 4: xnum1, Type: numeric, Storage: float32, Low/High: (-2.2719, 2.5820)
Var 5: xnum2, Type: numeric, Storage: float32, Low/High: (-2.1365, 2.1686)
Var 6: xnum2miss, Type: numeric, Storage: float32, Low/High: (-2.1365, 0.4561)
Var 7: xint1, Type: integer, Low/High: (5, 14)
Var 8: xfac1
  10 factor levels: 10 7 8 9 13 11 6 14 5 12
Var 9: xfac2
  10 factor levels: f c d e i g b j a h
```

Let’s write a subset of variables in the existing data set, plus one variable created with a transformation, to a comma separated data file. We need to specify the information for the output data source. In this case, we’ll specify the output file name, the precision to use when writing numeric variables, and set the stripZeros flag to TRUE in order to write numeric data values that only have zeros after the decimal as integers.

```
outDS <- RxTextData("testOut.csv", writePrecision = 2,
                           stripZeros = TRUE )
```

Last we’ll call the rxDataStep function, specifying the variables to read from the .xdf file and the data transformation we’d like to perform:

```
rxDataStep(inData = inTestData, outFile = outDS,
           varsToKeep = c("y", "xnum1", "xint1", "xfac1"),
           transforms = list(ytrunc = trunc(y*100)), overwrite = TRUE )
```
If you look at the testOut.csv file in your working directory, the first 10 lines will look as follows:

```
"y","xnum1","xint1","xfac1","ytrunc"
38.23,-0.50,10,"10",3822
32.85,0.13,7,"7",3284
31.16,-0.08,8,"8",3116
42.05,0.89,9,"9",4205
49.53,0.12,13,"13",4952
39.98,0.32,10,"10",3998
39.58,-0.58,11,"11",3957
55.20,0.71,13,"13",5520
53.82,-0.83,13,"13",5382
```

Similarly, we can have the output from rxPredict directed to a delimited text file. Let's create a simple linear model using the test data set:

```r
glm1 <- rxLinMod(y~xnum1 + xfac1, data = inTest)
summary(glm1)
```

```r
> summary(glm1)
Call:
rxLinMod(formula = y ~ xnum1 + xfac1, data = inTest)

Linear Regression Results for: y ~ xnum1 + xfac1
File name: 
C:\PROGRA~1\R\R-3.1.3\library\RevoScaleR\unitTest\test100r5b.xdf
Dependent variable(s): y
Total independent variables: 12 (Including number dropped: 1)
Number of valid observations: 100
Number of missing observations: 0

Coefficients: (1 not defined because of singularities)

|                | Estimate | Std. Error | t value | Pr(>|t|) |
|----------------|----------|------------|---------|---------|
| (Intercept)    | 48.9879  | 0.9176     | 53.387  | 2.22e-16 *** |
| xnum1          | 1.7360   | 0.2616     | 6.636   | 2.45e-09 *** |
| xfac1=10       | -7.1529  | 1.2543     | -5.703  | 1.52e-07 *** |
| xfac1=7        | -20.9900 | 1.1915     | -17.616 | 2.22e-16 *** |
| xfac1=8        | -15.7206 | 1.1134     | -14.119 | 2.22e-16 *** |
| xfac1=9        | -10.8970 | 1.4761     | -7.382  | 7.89e-11 *** |
| xfac1=13       | 3.7270   | 1.2613     | 2.955   | 0.000402 **  |
| xfac1=11       | -4.3737  | 1.2130     | -3.606  | 0.000513 *** |
| xfac1=6        | -23.1175 | 1.2993     | -17.792 | 2.22e-16 *** |
| xfac1=14       | 8.0012   | 1.1416     | 7.009   | 4.45e-10 *** |
| xfac1=5        | -28.2305 | 1.2627     | -22.357 | 2.22e-16 *** |
| xfac1=12       | Dropped  | Dropped    | Dropped | Dropped   |

Signif. codes:  '***' 0.001  '**' 0.01  '*' 0.05  '. 0.1 ' 1

Residual standard error: 2.554 on 89 degrees of freedom
Multiple R-squared: 0.9612
Adjusted R-squared: 0.9569
F-statistic: 220.6 on 10 and 89 DF,  p-value: < 2.2e-16
Condition number: 31.8155

Now we can call the `rxPredict` function, overwriting the testOut.csv file with the variables used in our model and the predicted values:

```r
rxPredict(myModel, data = inTestData, outData = outDS, writeModelVars = TRUE, overwrite = TRUE)
```

A look at the testOut.csv file in the working directory shows the following first 10 rows:

```
"y_Pred","y","xnum1","xfac1"
40.96,38.23,-0.50,"10"
28.23,32.85,0.13,"7"
33.13,31.16,-0.08,"8"
39.63,42.05,0.89,"9"
52.92,49.53,0.12,"13"
42.39,39.98,0.32,"10"
43.60,39.58,-0.58,"11"
53.96,55.20,0.71,"13"
51.28,53.82,-0.83,"13"
```
12 Converting RevoScaleR Objects for Use in PMML

The objects returned by RevoScaleR predictive analytics functions have similarities to objects returned by the predictive analytics functions provided by base and recommended packages in R. A key difference between these two types of model objects is that RevoScaleR model objects typically do not contain any components that have the same length as the number of rows in the original data set. For example, if you use base R's \texttt{lm} function to estimate a linear model, the result object contains not only all of the data used to estimate the model, but components such as \texttt{residuals} and \texttt{fitted.values} that contain values corresponding to every observation in the data set. For estimating models using big data, this is not appropriate.

However, there is overlap in the model object components that can be very useful when working with other packages. For example, the \texttt{pmml} package will generate PMML (Predictive Model Markup Language) code for a number of R model types. PMML is an XML-base language which allows users to share models between PMML compliant applications. For more information about PMML, visit \url{http://www.dmg.org}.

RevoScaleR provides a set of coercion methods to convert a RevoScaleR model object to a standard R model object: \texttt{as.lm}, \texttt{as.glm}, \texttt{as.rpart}, \texttt{as.kmeans}, \texttt{as.randomForest}, and \texttt{as.gbm}. As suggested above, these coerced model objects do not have all of the information available in a standard R model object, but do contain information about the fitted model that is similar to standard R.

For example, in many cases you can create PMML code for a RevoScaleR \texttt{rxLinMod} object in the following way. First, the R \texttt{pmml} package should be downloaded and installed from CRAN. After you have done so, generate the RevoScaleR \texttt{rxLinMod} object, and use the \texttt{as.lm} method when generating the PMML output:

```R
library(pmml)
rxLinModObj <- rxLinMod(Sepal.Length ~ Sepal.Width + Species, 
dropFirst = TRUE, data=iris)
pmml(as.lm(rxlInModObj))
```

Note that the \texttt{dropFirst} argument is used so that the contrasts for factor variables are the same as the default for \texttt{lm}. The generated output looks as follows:

```xml
<PMML version="4.1" xmlns="http://www.dmg.org/PMML-4_1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmg.org/PMML-4_1 http://www.dmg.org/v4-1/pmml-4-1.xsd">
  <Header copyright="Copyright (c) 2013 x" description="Linear Regression Model">
    <Extension name="user" value="sue" extender="Rattle/PMML"/>
    <Application name="Rattle/PMML" version="1.3"/>
    <Timestamp>2013-08-26 14:10:10</Timestamp>
  </Header>
  <DataDictionary numberOfFields="3">
    <DataField name="Sepal.Length" optype="continuous" dataType="double"/>
    <DataField name="Sepal.Width" optype="continuous" dataType="double"/>
    <DataField name="Species" optype="categorical" dataType="string">
      <Value value="setosa"/>
      <Value value="versicolor"/>
    </DataField>
  </DataDictionary>
</PMML>
```
Converting RevoScaleR Objects for Use in PMML

Similarly, rxLogit and rxGlm functions have as.glm methods:

```r
t <- case ~ age + parity + spontaneous + induced
rxLogitObj <- rxLogit(t, data = infert)
pmml(as.glm(rxLogitObj))

rxGlmObj <- rxGlm(t, data = infert, family = binomial())
pmml(as.glm(rxGlmObj))
```

RevoScaleR's `rxKmeans` objects can be coerced to `kmeans` objects:

```r
set.seed(17)
irow <- unique(sample.int(nrow(women), 4L, replace = TRUE))[seq(2)]
centers <- women[irow, , drop = FALSE]
rxKmeansObj <- rxKmeans(~height + weight, data = women,
centers = centers)
pmml(as.kmeans(rxKmeansObj))
```

And RevoScaleR's `rxDTree` objects can be coerced to `rpart` objects. (Note that `rpart` is a recommended package that you may need to load.)

```r
library(rpart)
method <- "class"
t <- Kyphosis ~ Number + Start
parms <- list(prior = c(0.8, 0.2), loss = c(0, 2, 3, 0),
split = "gini")
control <- rpart.control(minsplit = 5, minbucket = 2, cp = 0.01,
maxdepth = 10, maxcompete = 4, maxsurrogate = 5,
```
usesurrogate = 2, surrogatestyle = 0, xval = 0)
cost <- 1 + seq(length(attr(terms(form), "term.labels")))

rxDTreObj <- rxDTree(formula = Kyphosis ~ Number + Start,
data = kyphosis, method = method, parms = parms,
control = control, cost = cost)

pmml(as.rpart(rxDTreeObj))
13 The RevoTreeView Package

The new RevoTreeView package can be used to plot decision trees from rxDTree or rpart in an HTML page. Both classification and regression trees are supported. By plotting the tree objects returned by RevoTreeView’s createTreeView function in a browser, you can interact with your decision tree. The resulting tree’s HTML page can also be shared with other people or displayed on different machines using the package’s zipTreeView function.

As an example, consider a classification tree built from the kyphosis data that is included in the rpart package. It produces the following text output:

```r
data("kyphosis", package="rpart")
kyphTree <- rxDTree(Kyphosis ~ Age + Start + Number,
data = kyphosis, cp=0.01)
kyphTree

Call:
rxDTree(formula = Kyphosis ~ Age + Start + Number, data = kyphosis,
cmp = 0.01)
Data: kyphosis
Number of valid observations: 81
Number of missing observations: 0

Tree representation:
n= 81

node), split, n, loss, yval, (yprob)
   * denotes terminal node
1) root 81 17 absent (0.79012346 0.20987654)
   2) Start>=8.5 62 6 absent (0.90322581 0.09677419)
   4) Start>=14.5 29 0 absent (1.00000000 0.00000000) *
   5) Start< 14.5 33 6 absent (0.81818182 0.18181818)
   10) Age< 55 12 0 absent (1.00000000 0.00000000) *
   11) Age>=55 21 6 absent (0.71428571 0.28571429)
   22) Age>=111 14 2 absent (0.85714286 0.14285714) *
   23) Age< 111 7 3 present (0.42857143 0.57142857) *
   3) Start< 8.5 19 8 present (0.42105263 0.57894737) *
```

Now, you can display a HTML version of the tree output by plotting the object produced by the createTreeView function. After running the preceding R code, run the following to load the RevoTreeView package and display an interactive decision tree in your browser.

```r
library(RevoTreeView)
plot(createTreeView(kyphTree))
```
In this interactive tree, click on the circular split nodes to expand or collapse the tree branch. Clicking a node will expand and collapse the node to the last view of that branch. If you use a CTRL + Click, the tree will display only the children of the selected node. If you click ALT + Click, the tree will display all levels below the selected node. The square-shaped nodes, called leaf or terminal nodes, cannot be expanded.

To get additional information, hover over the node to expose the node details such as its name, the next split variable, its value, the $n$, the predicted value, and other details such as loss or deviance.

**Note:** Additional information is available in the RevoTreeView package help file, which can be seen by entering `??RevoTreeView` at the R command line, as well as in the resulting HTML page Help menu. See Chapter 10 of the RevoScaleR User’s Guide for examples on how to create decision tree models with `rxDTree`.