Predictive Analytics Accelerator for R™

Document Version: 1.0

Document Author: Theodor S. Klemming, Director, Solution Enablement, Theodor_Klemming@datawatch.com

For a better result when printing 2-sided, this page has intentionally been left blank.

Contents

Introduction

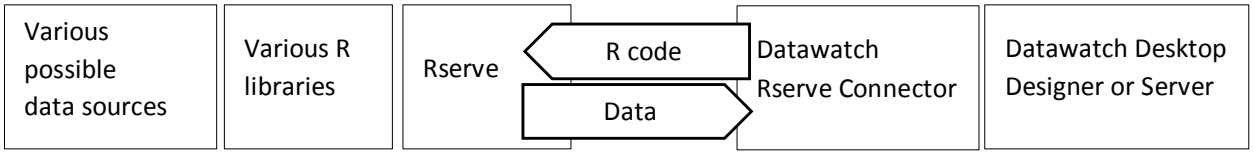
R and Datawatch

From release 12.4 onward, Datawatch supports the use of Rserve as a primary data source, using a dedicated Rserve connector. This means R libraries can be leveraged that provide connectivity to various data sources. Datawatch can be used with Rserve in the role as a universal data connectivity adapter. In addition, R can be leveraged for data preparation, calculations and transformations.

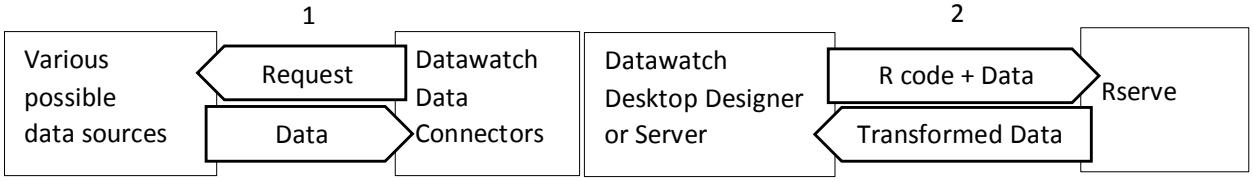
Using R in combination with Datawatch can be done in two different ways

a) Using Rserve as the primary data source with the Datawatch Rserve connector, or

b) Using Rserve as a calculations and transformations engine for data sets loaded using any of the Datawatch data connectors.



a) Rserve is used as the primary data source through the Rserve Connector.



b) Rserve is used as a data transformation engine for data sets loaded with other connectors

Rserve as a "universal" data connector

Rserve can be viewed as a universal data connector.

Datawatch provides purpose-built, high-performance data connectors "in the box" for a wide variety of Data in Motion and Data at Rest sources.

With the Rserve connector, anything that you can load into R can be loaded into Datawatch. Because the R platform is so open and extensible, and the global community has created packages to connect to many, many data sources – the odds are very good that if you need to access a particular data source, an R package (sometimes several) exists for it.

Typically, loading data from source X via R is a 2-line R script, where the second line is just the name of the data frame that you want R to send to Datawatch. With just a few short lines of R code, you can often achieve things that would otherwise be be near-impossible.

These are some examples of data sources from which R can load data, using built-in R functions or functions delivered by add-on R packages. Again, many of these are sources that Datawatch also supports natively:

1. CSV (with any character separator etc.) (function: read.csv and many others)

2. Excel (package: gdata, XLConnect and others)
3. XML
4. JSON (package: RJSON)
5. SAS (file type .xpt)
6. IBM/SPSS (file type .sav) (function: read.spss)
7. Stata (file type .dta) (function: read.dta)
8. Systat (file type .sys, .syd) (function: read.systat)
9. Minitab (function: read.mtb)
10. S-PLUS (function: read.S)
11. EpiInfo, EpiData (file type .rec) (function: read.epiinfo)
12. Databases with ODBC driver via Data Source Name, using SQL (package: RODBC) (package officially tested on Microsoft SQL Server, Access, MySQL, PostgreSQL, Oracle and IBM DB2 on Windows and MySQL, MariaDB, Oracle, PostgreSQL and SQLite on Linux)
13. Databases with JDBC driver (package: RJDBC)
14. MySQL (package: RMySQL with DBI)
15. Oracle (package: ROracle with DBI)
16. Oracle Data Mining (ODM) in-database analytics (package: RODM with RODBC)
17. PostgreSQL (package: RPostgreSQL with DBI, RpgSQL)
18. SQLite (package: RSQLite with DBI)
19. MongoDB (package: RMongo, rmongodb)
20. NASA's HDF5 file format (package: hdf5, h5r, rhdf5, RNetCDF, ncdf, ncdf4)
21. UCAR's netCDF file format (package: hdf5, h5r, rhdf5, RNetCDF, ncdf, ncdf4)
22. dBase files (dBase, Clipper, FoxPro, Visual dBase, Visual Objeccts, Visual FoxPro) (function: read.dbf)
23. Teradata (package: teradataR (unsupported, made open source)
24. Twitter API (package: twitteR, http://cran.r-project.org/web/packages/twitteR/twitteR.pdf or streamR, http://cran.r-project.org/web/packages/streamR/streamR.pdf)
25. Google Analytics (package: RGoogleAnalytics or rga on GitHub)

The web site http://ropensci.org/related/index.html maintains a "growing list of R packages that collect open data from the web, or are tools for doing weby things". Packages are grouped by field.

On the topic of using R to connect to different data sources, this official documentation from the R core team is very relevant: http://cran.r-project.org/doc/manuals/r-release/R-data.pdf

Preparations

To use R transforms from Datawatch, you need R and the Rserve package installed on your local system, or on a remote system to which you have the credentials to connect and make calls to Rserve.

Installing R

R has a basic set of packages that always come with R. These include "base", "utils", "graphics", and "stats". In addition to that, there is a large collection of other packages that can be downloaded and added. A package is a library that slot into R and provide additional functionality. Since Rserve is an extension package for R, you will always need to install R as a first step, and then the Rserve package on top of that. Rserve provides a server that responds to requests from clients, and hands the request to R for execution.

This page holds the latest release of R for Windows, with installation instructions:

http://cran.r-project.org/bin/windows/base/

There are special distributions for a few different versions of Linux, as listed here:

http://cran.r-project.org/bin/linux/

Installing and starting Rserve

After installing R, you run the following commands in the R application to download and install the Rserve library (package) and then start it. Type each row, hit Enter and wait for the ready-prompt before the next one.

```
install.packages("Rserve")
library(Rserve)
Rserve()
```

Note: When asked to select a CRAN mirror, choose the location closest to you.

After the Rserve() command, R will say something like

```
Starting Rserve…
"C:\Users+JohnDoe\DOCUME~1\R\WIN-LI~1\3.1\Rserve\libs\x64\Rserve.exe"
```

This means Rserve is now running, responding to requests over TCP.

Create a shortcut for starting Rserve

Once you have R and Rserve installed, you might find it convenient to have a shortcut icon on your desktop that starts Rserve. You can create that by following these steps:

1. Check the location of R.exe on your computer. If you installed R 64-bit, the location will be something similar to C:\Program Files\R\R-3.1.0\bin\x64
2. Add this location to the system environment variable *Path* by appending it to the end of the current value, following a semicolon.

3. From the installation folder of Rserve (which is likely to be Documents\R\win-library\3.1\Rserve) copy the files libs\x64\Rserve.exe and libs\x64\Rserve_d.exe and place them in the R path folder (they should be in the same folder as R.exe 64-bit).
4. Create a text file with Notepad, and enter the following line
   R CMD Rserve
   Save this file as "Lauch_Rserve.bat" and save it on your desktop, or save it somewhere else and place a shortcut on the desktop.
5. Double-click the file Launch_Rserve.bat. You will see:
   Rserve: OK, ready to answer queries.
   Rserve will be up and running, responding to requests, for as long as the command prompt window is open.
6. Closing the command prompt window means you terminate Rserve.

NOTE! If you upgrade your R installation, say for example from version 3.10 to 3.11, you will have to update the R location in your Path environment variable. You will also have to copy the Rserve.exe and Rserve_d.exe files into the new installation.

Additional development tools

Although R has a graphic user interface ("RGui"), it is a bit rudimentary. RStudio is an alternative interface to R. You need to install R regardless of whether you use RStudio or not.

Get RStudio here: http://www.rstudio.com/ide/download/desktop

For additional help with getting and installing R, see the section "About R", below.

Debugging and troubleshooting your R scripts

While R and RStudio provide very good error messages, those error messages will not reach you through Desktop Designer. Instead, all you will see is something like "Unable to load data: R threw an error".

Therefore, if you want to debug your R script, do it in RGui or RStudio. To be able to do that, you will also need a data set sample which should be as close to the final, real data as possible. You can connect to various databases and load data from various file formats from R. The quickest and easiest is probably to load data from csv.

When you have a script that works, you can save that as a file, and then use it in Desktop Designer by pressing the Browse button in the transforms dialog, and load the content of the file into the script window. Or, just copy-paste it into the R Script window in the Rserve connector.

Using Rserve from Desktop Designer

To be able to get anything done with Rserve, you need to know some R language syntax, and you need to understand a few basic concepts about R.

R performs fast transformations and calculations on datasets after loading them into memory. When R loads a tabular data set into memory it creates a so called Data Frame, which is a list of vectors of equal length.

You can also load several data sets into different data frames, and combine them in whole or in part.

Note! While R is very fast when doing data transformations and calculations on data frames already loaded into memory, R can never speed up the loading of data into memory from a slow data source.

When to use Rserve connector or just R transforms

As a rule of thumb, you can use the diagram below in order to get some help deciding which approach you should take when using R with Datawatch.

	Data transformation made as part of connection R code	**Data transformation made as an R trans-form after loading data**	**No data transformation made**
Any other data connector		Case A	*Default use case*
Data Connector for Rserve	Case B		Case C

	Use case	**Pro**	**Con**
Case A: **Any data connector** **+ R transform**	You want to use a Datawatch data connector, but also apply the power or R to the data before using it in your dashboards.	You can keep the R-script to the bare necessities, leaving the data loading to the Datawatch connectors.	Compared to using R for both connecting to data and transforming the data, there may be longer load times and more memory usage.
Case B: **Rserve connector** **+ data transformation**	You prefer handling data connection and data transformation in one place, that is your own R script	You get a high degree of control and you might save a bit of load time and memory usage.	To load data from your sources using R, suitable libraries and R code requires the right skills.
Case C: **Rserve connector** **but no data transformation**	You require data from a source for which Datawatch does not have a connector.	Provided there are R libraries available, you can get data from an even larger variety of sources.	Connecting to data will require R programming.

Datawatch's approach to R integration compared to competing visualization products

There is direct connectivity from Datawatch to R: anything you can write in R can be run from within Datawatch.

If you are already using R, you can paste your existing scripts right into Desktop Designer and have the resulting data immediately accessible for visualization in a dashboard. You can also pass Datawatch parameters directly into an R script by enclosing the parameter within {brace brackets}, which in practice means non-technical business users can directly interact with a sophisticated R script from a graphical dashboard interface.

In short, using R with Datawatch is no different from using R in a standalone mode. You don't have to wrap you R code in anything, or modify the R syntax - just use it. In this respect, Datawatch has a major advantage over some competing visualization tools that offer R integration.

Unlike in Datawatch, where the full scope of R is available to you and you apply R to your data directly, the R integration in some competing products is implemented through a limited set of particular functions in the product's "Calculated Columns" section. Each function can only return a single value of a particular data type, such as Integer or String, per row in the data set, after a single-line R script is executed.

This means that you will have to learn the proprietary vagaries of how to apply R scripts to your data from within that product. The entire R script you want to apply to the data must be written as a single-line argument to a function, using semicolon as a line break character. Readable code is vital for maintaining and evolving advanced business applications, but code written in this way is difficult or even impossible to understand. To compound the problem, there is no mechanism to provide human-readable comments in the R code.

With a competing product, each data column or parameter passed to R must be referenced as ".arg" and a number, for example .arg1, arg2, .arg3 etc. Any parameter that you use in an R-powered calculated column is forced into a vector format, which means that it must be referenced with a row number predicate, for example ".arg7[1]". In comparison, Datawatch transfers all of your data set columns to corresponding columns in an R data frame, automatically maintaining the same column headers. Parameters are referenced by the name of your choice, and their values will be stored in the format you choose.

Further, there is no possibility to leverage R as a data connection conduct in the competing product. In Datawatch, you can use the Rserve connector, write a script that uses any R library to connect to any data source and get a complete data set returned. This cannot be done at all in some competing products. They are limited to asking R for a single calculated column.

Finally, whereas Datawatch's R integration aligns perfectly with existing user interfaces and the internal data architecture, in some competing products, R integration is implemented as a crude "bolt-on" that undermines ease-of-use and the user experience, particularly for developers who have to learn multiple ways to work with the product.

Example: R in a competing product compared to R in Datawatch

In a csv file, there is data for 29 consecutive years, describing college enrollment along with the number of high school graduates, the per capita income, and the unemployment rate. The enrollment number is missing for the last year, and this needs to be predicted, based on what is entered for number of graduates, per capita income, and unemployment rate.

In a competing product that number is obtained by producing a calculated column, using a product specific, built-in function called SCRIPT_REAL, like so (all in one line when in the product)

```
SCRIPT_REAL("mydata <- data.frame(cbind(Enrollment=.arg1,
HighSchoolGrads=.arg2, PerCapitaIncome=.arg3, Unemployment = .arg4)) ; fit
<- lm(Enrollment[-29] ~ HighSchoolGrads[-29] + PerCapitaIncome[-29]+
Unemployment[-29],data=mydata) ;  mydata$Enrollment[29]<-
predict(fit,list(HighSchoolGrads= .arg7[1], PerCapitaIncome =
.arg6[1],Unemployment = .arg5[1]))
;mydata$Enrollment",SUM([Enrollment]),SUM([HighSchoolGrads]),SUM([PerCapita
Income]),SUM([Unemployment]),[What If - Unemployment],[What If -
Income],[What If - Graduates])
```

To get the same job done in Datawatch, you apply an R transform to the data set, and enter this script (here, with explanatory code comments):

```
# we use the lm() function to fit a linear model to the historic data
# (leaving out 2014, row 29)

LinearModelFit <- lm(Enrollment[-29] ~ HighSchoolGrads[-29] + PerCapitaIncome[-29]
+ Unemployment[-29],data=Historic)

# In the column Enrollment, on row 29, we use the function predict()
# and enter a predicted value where the input values are provided by
# Datawatch parameters

Historic$Enrollment[29]<-predict(LinearModelFit, list(HighSchoolGrads={DWP_hsg},
PerCapitaIncome={DWP_pci}, Unemployment={DWP_ue}))

# We ask R to return the Historic data set,
# which now contains a predicted valued for the last year

Historic
```

As you can see, in the competing product, you are forced to learn a lot of details about the product specific function SCRIPT_REAL() and the arguments it needs, in order to use R for your data analysis. This is not the case with Datawatch, where a direct copy-paste from your R window will work without additional adaptation.

Reasons for learning R

R is a product, and a programming language. The things you can achieve with R could also be done with other tools and programming languages.

However, there are a few big advantages of using R to solve statistical problems:

- It is free. Other statistical languages require paid license fees
- It is available on every major platform
- it is open source, so there is a thriving user community around R that is continuously improving and extending it
- It is open source, so you don't have to ask permission from anyone to do anything
- There is direct, seamless connectivity from Datawatch to R - anything you can write in R can be run from within Datawatch.

R is case sensitive

To repeat: R is case sensitive!

This means that if you use a variable name like “myData”, it will be regarded by R as a separate thing from “MyData”, for example. This can be used to your advantage, but can also cause a problem if you happen to forget about it. Similarly, lower case and upper case letters matter when you call functions and packages.

The best approach is to decide very early on what capitalization convention you will use and stick to it religiously to avoid baffling behavior i.e. everything looks correct yet doesn’t work as expected.

The most common convention is known as mixed-case, or sometimes “camel case”, because TheObjectNameHasHumpsInItLikeACamel. This style has the advantage of eliminating redundant spaces or separators like “_”, while still being highly readable.

While on the subject of coding style and comprehension, VeryLongNamesForObjectsLike ThisAreDiscouraged - object names should be more symbolic versus descriptive. Also, some form of short easily-recognizable object prefix is encouraged e.g. “varYearsSinceRenewal” to denote a variable, “df_ResultFromStep10” for a data frame, and so on. Again, pick a convention and stick to it.

Value assignment in R

In most other software contexts, you use a single equal sign when assigning values, for example:

```
Myvariable = 5
```

This is also valid in R. However, you will often see another notation for value assignment, which is very specific to R: the little arrow or pointer. In the early days of R, the idea was that assignment should be possible to write in the direction of your choice: You should not necessarily have to write that “left thing gets the value of right thing”. The opposite should be possible as well. To achieve that, the following notation was introduced:

```
Myvariable <- 5
5 -> Myvariable
```

In the examples of this document, you will see both "=" and "<-" in use. One argument in favor of using the more old-school "<-" is that it reduces the risk of confusing the assignment notation "=" with the evaluation notation "=="

R is an interpreted language

When describing programming languages, it is common to distinguish between interpreted languages and compiled languages. While this is not an exact and unambiguous classification of programming languages, it gives a hint about the nature of the programming language – or at least a hint about the typical way of using the code written in that language.

When you have written code in R, you typically execute it immediately, before compiling your source code into machine code. Your R code is understood by a piece of software referred to as an interpreter, which is the R software. This means that when you write a program in R, you normally do not compile the code to create a program that runs stand-alone like any executable.

Other examples of interpreted languages are JavaScript, VBScript, Perl, and Python.

Examples

Below are a few examples aimed at showing the basics of how to use the Rserve connector in Desktop Designer.

Example 101: A plain csv file

As an initial, simple example of how to use Rserve via the Datawatch Rserve connector, we will use a code snippet that makes R load data from a csv file, and return the data to Datawatch. Obviously, this is a task performed more easily with the Datawatch text connector, but it serves a nice and easy example. The data set below is available in the Datawatch Enablement Kit for R, saved as C:\DWCH\DEK\R\Example101.csv

ColA	ColB	ColC	ColD
RowA	1	1	9
RowB	2	4	8
RowC	3	9	7
RowD	4	16	6
RowE	5	25	5
RowF	6	36	4
RowG	7	49	3
RowH	8	64	2
RowI	9	81	1

Alternatively, you can use any csv file of your choice. Below, we will assume the file is named "Example101.csv". We will also assume that the file is saved in this location:

C:\DWCH\DEK\R\

Also, it is assumed that you have installed and started Rserve() as described earlier in this document.

In the R script field of the Rserve Connector, enter this code snippet:

```
# set the working directory path
setwd("C:/DWCH/DEK/R")

# load csv file into a data frame
Example101 <- read.csv("Example101.csv", header = T, sep=",")

# command R to return the content of the data frame
Example101
```

In case you want to load a subset of a csv file, you can use an R script like the one below. Again, we are using read.csv, but with the addition of the nrow and skip arguments.

```
# set the working directory
setwd("C:/DWCH/DEK/R")

# load the headers and the first row of data into a data frame
headers <- read.csv("Example101.csv", header = T, sep=",", skip = 0, nrow =
1)

# skip the first 5 rows, including the headers,
# and load the following 3 rows into a data frame
datarows <- read.csv("Example101.csv", header = FALSE, skip = 5, nrow = 3)
```

```
# Set the columns names of the "datarows" data frame to be
# the column names of the "headers" data frame
colnames(datarows) <- c(colnames(headers))

# Throw away the headers data frame to free RAM
# (important in case using large data sets)
remove(headers)

# command Rserve to return the content of the "datarows" data frame
datarows
```

All of the above is also available in an example workbook named example101_Rserve_csv.exw

Example 102: Using R for creating synthetic data

With R, it is quick to generate synthetic data with a given density distribution. For example, to generate 1000 values with mean=100 and standard deviation=20, you use this function

```
rnorm(1000, mean=100, sd=20)
```

So, you can ask R for a dataset which is generated by R, and not loaded from any file. If you parameterize the R script, you can control the properties of the data set through navigation action controls in your dashboard. {n}, {m} and {sd} refer to parameters defined in Desktop Designer for your data table.

```
# In case you want the same data set each time
# for any given set of input values, then uncomment
# the set.seed line below. The seed number can be any number
# you want

# set.seed(42)

# Create vector with values 1 to {n}
CountID <- c(1:{n})

# Create vector with values produced by the rnorm() function
Rnorm <- c(rnorm({n}, m={m}, sd={sd}))

# Create vector with values rounded to zero decimals
Value <- c(round(Rnorm, digits=0))

# Build data frame from two vectors
output <- data.frame(CountID, Value)

# Insert another vector into data frame, with value 1 on each row
output$Count <- c(1)

# Command Rserve to return the data frame
output
```

From the dashboard, you can then quickly investigate the visual appearance of a normal distribution with different n, mean and standard deviation. For the visualization, use a Numeric Needle Graph, and right-click the column Value to create ID-buckets. Put Value Id bucketing on the Breakdown of the chart, and Value on the X-axis. Make sure you use Mean as the aggregation method. Last, put Count on the Y-axis. You then also need to create a navigation action and an action control like an action text box for each of the three parameters n, m and sd. (NOTE: Prior to creating parameters, you will get a message that R threw an error.)

When testing this on an 8 GB RAM, 4 core Windows laptop, R threw an error when n=671001, but returned data all the way up to and including 671000 rows.

All of the above is also available in an example workbook named example102_Rserve_synthetic.exw

Example 103: Load data from a SAS export file (xpt)

You can use R and for example the *SASxport* package to load data directly from SAS export files (xpt) into Datawatch. Start by installing the SASxport package, since it is not part of the basic installation of R, by typing this command in R Studio:

```
install.packages("SASxport")
```

Next, create a new data table in Desktop Designer, and use the Rserve connector. Put this code snippet in the R script window:

```
# call the SASxport library
library(SASxport)

# set the working directory
setwd("c:/dwch/dek/r/")

# load the Example103.xpt SAS export file into a data frame
SASdata <-read.xport("Example103.xpt")

# command R to return the content of the data frame
SASdata
```

In Desktop Designer, change the data type of the columns SAMPLE and REP from numeric to text.

To understand how non-trivial it would be to load the data from an XPT file unless R and the SASxport package was available, you should open the Example103.xpt file with a text editor like Notepad or Notepad++ and have a look at the data.

This data file was picked off the web, and it is not known what the meaning of the data is. The example just serves the purpose of demonstrating connectivity.

All of the above is also available in an example workbook named example103_Rserve_SAS_xpt.exw

Example 104: Load data from a SAS export file (xpt) that has several data sets

SAS export files can optionally contain more than one data set. When such a file is imported into R, the different datasets become separate vectors with attributes, in the same data frame. In order to get useful data into Datawatch, you must specify which one of the SAS datasets you want, i.e. you specify which of the vectors in the R data frame you want. This example is very similar to example 3, with the addition that here we are loading an XPT file that as 2 datasets in it: "TEST" and "Z".

Start by installing the SASxport package, since it is not part of the basic installation of R, by typing this command in R Studio:

```
install.packages("SASxport")
```

Next, put the following code in the R script window:

```
# call the SASxport library
library(SASxport)

# set the working directory
setwd("c:/dwch/dek/r/")

# load the Example104.xpt SAS export file into a data frame
SASdata <-read.xport("Example104.xpt")

# Put the "Z" vector dataset in a data frame of its own
dataset_wanted <- SASdata$Z

# Create a single column vector containing the row ids from dataset_wanted
id <- rownames(dataset_wanted)

# Extend "dataset_wanted" with a row ID column and put it all in the frame
"output"
output <- cbind(id=id, dataset_wanted)

# command R to return the content of the "output" data frame
Output
```

In Desktop Designer, change the data type of the "id" column from numeric to text, and create ID buckets for X7 and X8.

This data file was picked off the web, and it is not known what the meaning of the data is. The example just serves the purpose of demonstrating connectivity.

All of the above is also available in an example workbook named example104_Rserve_SAS_xpt_multi.exw

Example 105: Load data from an SPSS export file (sav)

This example relies on a library called memisc. Start by adding this package to your R installation:

```
install.packages("memisc")
```

Put the following code in the R script window of the Rserve connector in Desktop Designer:

```
# call the memsisc library
library(memisc)

# set the working directory
setwd("C:/DWCH/DEK/R/")

# Import an SPSS sav file into a data frame called "spssdata"
spssdata <- as.data.set(spss.system.file('Example105.sav'))

# Command R to return the "spssdata" data frame
Spssdata
```

This data file was picked off the web, and it is not known what the meaning of the data is. The example just serves the purpose of demonstrating connectivity.

All of the above is also available in an example workbook named example105_Rserve_SPSS_sav.exw

Example 106: Load data from an SPSS portable file (por)

This example relies on a library called memisc. Start by adding this package to your R installation:

```
install.packages("memisc")
```

Put the following code into the R script window of the Rserve connector in Desktop Designer:

```
# call the memsisc library
library(memisc)

# set the working directory
setwd("C:/DWCH/DEK/R/")

# Import an SPSS portable file into a data frame called "spssdata"
spssdata <- as.data.set(spss.portable.file("Example106.por"))

# Command R to return the "spssdata" data frame
spssdata
```

This data file was picked off the web, and it is not known what the meaning of the data is. The example just serves the purpose of demonstrating connectivity.

All of the above is also available in an example workbook named example106_Rserve_SPSS_por.exw

Example 107: Load data from a Stata export file (dta)

This example relies on a library called foreign. Start by adding this package to your R installation:

```
install.packages("foreign")
```

Put the following code into the R script window of the Rserve connector in Desktop Designer:

```
# call the library "foreign"
library(foreign)

# set the working directory
setwd("C:/DWCH/DEK/R/")

# load a dta file into a data frame called "stata_data"
stata_data <- read.dta("Example107.dta")

# command R to return the content of the "stata_data" data frame
stata_data
```

In Desktop Designer:

- Create an ID bucketing column from the id column.
- Create a Sign bucketing column from the Theta column.
- Do an Unpivot transform and include all columns except Id and Theta.

This data file was picked off the web, and it is not known what the meaning of the data is. The example just serves the purpose of demonstrating connectivity.

All of the above is also available in an example workbook named example107_Rserve_Stata_dta.exw

Example 108: Load data from a Systat export file (sys, syd)

This example relies on a library called foreign. Start by adding this package to your R installation:

```
install.packages("foreign")
```

Put the following code into the R script window of the Rserve connector in Desktop Designer.

Note! Replace “myfile.sys” with the name of your .sys or .syd file, as no example file is currently available.

```
# call the library "foreign"
library(foreign)

# set the working directory
setwd("C:/DWCH/DEK/R/")

# load a dta file into a data frame called "systat_data"
systat_data <- read.systat("myfile.sys")

# command R to return the content of the "systat_data" data frame
systat_data
```

There is no example workbook for this use case.

Example 109: Load data from an EpiData file (rec)

This example relies on a library called foreign. Start by adding this package to your R installation:

```
install.packages("foreign")
```

Put the following code into the R script window of the Rserve connector in Desktop Designer.

```
# call the library "foreign"
library(foreign)

# set the working directory
setwd("C:/DWCH/DEK/R")

# load a rec file into a data frame called "epidata"
epidata <- read.epiinfo("Example109.rec")

# command R to return the content of the "epidata" data frame
epidata
```

In Desktop Designer:

- Create time bucketing columns for Day, Month and Year based on *tlahir*
- Change the title of the *darah* field to *darah (blodd type)*
- For the *urut* field, change data type to text, and title to *urut (ID)*
- Change the data type of *sex* to text

This data file was picked off the web, and it is not known what the meaning of the data is. The example just serves the purpose of demonstrating connectivity.

All of the above is also available in an example workbook named example109_Rserve_Epidata_rec.exw

Example 110: Make a union of two CSV files, create a disk cache, get rid of duplicates

In this example you can see how R can be used for merging two datasets that have the same data structure and topic, but cover different time periods and have different update frequencies. The idea is that you want the full history provided by the larger and heavy-to-load dataset 1, and at the same time the frequent updates provided by dataset 2.

The datasets we use in the example are from earthquakes.usgs.gov:

- Dataset 1 is an online csv file with all registered earthquakes during the past 30 days. This is a file of about 1400 kilobytes and takes a while to download (around a minute). Also, it only updates once every 15 minutes. Reading the data from disk into a data frame in R takes about 2 seconds.
- Dataset 2 is an online csv file with all registered earthquakes during the current day. This is a file of about 40 kilobytes and downloads quickly. It is updated every 5 minutes.

This example shows how we can:

1. Avoid downloading the 30-days file from the web when we already have it, by caching it in a csv file that we refresh only once per day. So, we create our own disk-cache in the form of a csv file, as part of our script used in the Rserve connector.
2. Still download the current day file every 5 minutes, and merge it with the 30-days file
3. Get rid of all the duplicated earthquake records that we get since both data files contain data about the current day

Steps 1-3 are things we cannot achieve by using Desktop Designer alone. For example, we cannot cache Dataset 1 during a full day, while fetching Dataset 2 every five minutes, and still use all the data as a single data table in Datawatch. Further, we cannot get rid of duplicate records in the data set by using Desktop Designer alone.

This example uses no add-on packages, only “base” R commands. Put the following code into the R script window of the Rserve connector in Desktop Designer:

```
# set working directory
setwd("C:/DWCH/DEK/R")

# check if there is a file called "Example110_30days.csv"
# if not, create one
# if there is a file, but too old, then refresh it and load the data into a
data frame
# otherwise load the existing file content into a data frame

if (file.exists("Example110_30days.csv") == FALSE) {
  thirtydays <-
read.csv("http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_mont
h.csv")
  write.csv(thirtydays, file = "Example110_30days.csv", row.names=FALSE)
  } else if (format(file.info("Example110_30days.csv")$mtime, "%Y-%m-%d") <
format(Sys.Date(), "%Y-%m-%d")) {
     file.remove("Example110_30days.csv")
     thirtydays <-
read.csv("http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_mont
h.csv")
     write.csv(thirtydays, file = "Example110_30days.csv", row.names=FALSE)
   } else {
  thirtydays <- read.csv("Example110_30days.csv")
}
```

```
# load data for today
today <-
read.csv("http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_day.
csv")

# make a union set of today and thirty days
alldata <- rbind(today,thirtydays)

# remove the thirtydays from memory
remove(thirtydays)

# get rid of duplicate values
alldata = alldata[!duplicated(alldata$id), ]

# create hour bucket. In this case it is done as a string manipulation
# not as a time calculation
alldata$hourbucket <- paste0(substr(alldata$time, 1, 10),"
",substr(alldata$time, 12, 13),":00:00")

# create an event count column
alldata$count = 1

# create a time loaded column
alldata$timeloaded <- Sys.time()

# command R to return all data in the data frame "alldata"
alldata
```

Also, in Desktop Designer:

- Change the data type of the “time” column from text to time. This column can then be used in a filter.
- Add time bucketing columns such as Year, Month, Day, Hour for a frequency-by-time-bucket diagram that you can create using the vertical bar graph.
- Create a calculated column, *Depth below*, with this expression: [depth]*-1

All of the above is also available in an example workbook named example110_Rserve_union_of_two_CSVs.exw

Example 111: Load JSON data from the web, loop through multiple URLs and do a union

With Datawatch, the JSON connector makes it relatively easy to grab a JSON document off a web URL and visualize it in a dashboard. What is arguably even easier is to use the "jsonlite" package in R, which will parse the JSON data into a tabular structure completely automatically. In addition to that, we can instruct R to loop through a number of URLs holding JSON data of identical structure, and make a union of all the data rows before it is handed back to Datawatch.

In the example below, we are loading 15 JSON documents from 15 URLs into a single data set. Before you run this example, you will need to install the "jsonlite" and "httr" packages:

```
install.packages("jsonlite")
install.packages("httr")
```

```
# this requires install.packages("jsonlite")
# this requires install.packages("httr")

library("jsonlite") # "jsonlite" has a dependency on the package "httr"
startyear = 1999 # set the first year for which you want to load data
endyear = 2014   # set the last year for which you want to load data

# After loading the first year, this is the next year
# The for loop starts with this year. Don't change.
loopstartyear = startyear+1

# The main part of the URL is put in a variable
# since it is used in more than 1 place
baseurl = "http://fts.unocha.org/api/v1/Appeal/year/"

# Getting data for the first year into a data frame
# Using paste0(), we are concatenating a complete URL
# The fromJSON() function parses the JSON data.
# We put the result in a data frame
UNOCHAdata.df = fromJSON(paste0(baseurl, startyear, ".json"))

# This is a loop which steps through all remaining years up to the endyear
for(i in loopstartyear:endyear){
 anotheryear.df <- fromJSON(paste0(baseurl, i, ".json"))
 # we make a union of the previously loaded data and the last year loaded
 UNOCHAdata.df <- rbind(UNOCHAdata.df, anotheryear.df)
 # we empty the data frame named anotheryear.df
 anotheryear.df <- NULL
}

# Adding an alphabetic character in the 'year' column
# to assure it comes out as a text column in Datawatch
UNOCHAdata.df$year <- paste0("y ", UNOCHAdata.df$year)

# We command R to return the data frame UNOCHAdata.df
UNOCHAdata.df
```

If you create your own workbook and enter the script above, remember to adjust the Rserve connector timeout to more than the default of 10 seconds. Depending on your internet connection speed, the data loading will require more time than 10 seconds.

All of the above is also available in an example workbook named example111_Rserve_load_JSON_loop_multi_URLs.exw

Example 112: Load HDF5 files

The R/Bioconductor package provides an interface between HDF5 files and R. HDF5's main features are the ability to store and access very large and/or complex datasets and a wide variety of metadata on mass storage (disk) through a completely portable file format. The rhdf5 package is thus suited for the exchange of large and/or complex datasets between R and other software package, and for letting R applications work on datasets that are larger than the available RAM.

How to install the rhdf5 package:

```
source("http://bioconductor.org/biocLite.R")
biocLite("rhdf5")
```

NOTE! This installation will likely return an error:

"installed directory not writable, cannot update packages 'class', 'cluster', 'codetools', 'KernSmooth', 'MASS', 'mgcv'"

If you get this error, please ignore it for this example. The package will work as required anyway.

```
library(rhdf5)

# setting the working directory to the location of the example file
setwd("C:/DWCH/DEK/R")

# reading the data set named "transforms" in the groupnamed
# "transform/internal_to_aligned"
# from the HDF5 file "example112.hdf5" into data frame named "HDF5"
HDF5 = data.frame(h5read("Example112.hdf5", "transform/internal_to_aligned/transform"))

# NOTE!
# Before you can read the content of any hdf5 file into R,
# you will need to investigate what data sets are available in the
# file, and what the name of the data set group is.
# You do that by using the "h5ls" (h5 list) command:
# h5ls("Example112.hdf5")

HDF5$rowid = c(LETTERS[1:4])

# command R to return the content of the data frame "HDF5"
HDF5
```

This data file was picked off the web, and it is not known what the meaning of the data is. The example just serves the purpose of demonstrating connectivity.

All of the above can be tried out using the example workbook example112_Rserve_HDF5_file.exw

Example 113: A simulated prediction model and synthetic data created with R

The example below is supporting the following story:

An Internet Service Provider has segmented their clients into 4 different demographic groups or market segments that show different traits on group level. For example, the groups have different average monthly spend, different bandwidth plans on average and so on. The company has made substantial statistical analysis of large amounts of customer behavior data correlated to negative service incidents and service level drops. This has allowed them to build a prediction model that can forecast the Probability of Defection (POD) of customers in the four different demographic groups, at time horizons from 1 month to 6 months. By setting five different input variables that describe quality of service issues and service level issues, the model will deliver a prediction of the Probability of Defection.

All the data, and the simulated prediction model (which in reality is not based on serious statistical analysis, but instead on a few simple assumptions), **is produced in an R script**. As you can see, the R script contains a number of Datawatch parameters, surrounded by curly brackets. All of them have the prefix "DWP" to make it easier to find them:

```
{DWP_LatencyAvg}
{DWP_LatencyPeak}
{DWP_BandwidthDiffAvgVsAdv}
{DWP_BandwidthDiffPeakVsAdv}
{DWP_Incidents}
```

```
# Define data frame
# Create demographic groups (market segmentation groups)
data = data.frame(Group=c("A. Mobile Executive", "C. Soccer Mom", "B.
Millenial Gamer", "D. College Student"))

# Insert demographic group size numbers
data$Size = c(50000, 200000, 300000, 800000)

# Insert average monthly spend per demographic group
data$MonthSpend = c(400, 200, 300, 100)

# Insert base churn monthly per demographic group
data$BaseChurnMonthly = c(0.03, 0.01, 0.02, 0.005)

# Insert sensitivity factor: an imagined number describing the
statistically verified sensitivity to Quality of Service issues
# Value 0 means the customer does not care at all, regardless of how bad
the Quality of Service issues are
# Value 1 means the customer is very sensitive and defects following very
small disturbances in service
# In a real life implementation these values would be derived through
advanced statistical analysis using a tool like Statistica from DELL
data$Sensitivity = c(0.6, 0.2, 0.8, 0.5)

# Insert baseline or normal average latency numbers in milliseconds. This
is supposed to be the generally accepted level
data$LatencyNormalAvg = c(150)

# Insert average latency numbers in milliseconds. It is reasonable to
assume that latency is the same for all customers
data$LatencyAvg = c({DWP_LatencyAvg}) # Note! Here is a Datawatch parameter
i curly brackets

# Insert baseline or normal peak latency numbers in milliseconds. This is
```

```
supposed to be the generally accepted level
data$LatencyNormalPeak = c(1500)

# Insert peak latency numbers in milliseconds. It is reasonable to assume
that peak latency is the same for all customers
data$LatencyPeak = c({DWP_LatencyPeak})

# Insert advertised bandwidth in Mbps
data$BandwidthAdvertised = c(15, 10, 50, 25)

# Insert difference between advertised and average bandwidth in Mbps
data$BandwithDiffAvgVsAdv = c({DWP_BandwidthDiffAvgVsAdv})

# Insert average bandwidth in Mbps as a calculated field
data$BandwidthAvg = data$BandwidthAdvertised-data$BandwithDiffAvgVsAdv

# Insert average bandwidth as percent of advertised bandwidth as a
calculated field
data$BandwidthAvgPcnt = data$BandwidthAvg/data$BandwidthAdvertised

# Insert difference between advertised and peak bandwidth in Mbps
data$BandwidthDiffPeakVsAdv = c({DWP_BandwidthDiffPeakVsAdv})

# Insert peak bandwidth in Mbps as a calculated field
data$BandwidthPeak = data$BandwidthAdvertised-data$BandwidthDiffPeakVsAdv

# Insert peak bandwidth as percent of advertised bandwidth as a calculated
field
data$BandwidthPeakPcnt = data$BandwidthPeak/data$BandwidthAdvertised

# Insert number of service interruptions per month
data$Incidents = c({DWP_Incidents})

# Insert the increase in POD above BaseChurn resulting from quality-of-
service issues.
# This is a formula set up in such way that
# the probability will increase above BaseChurn with increasing service
level errors
# In the real world, this calculation would be based on a statistically
verified algorithm
# For example, the weighting of the impact of the different input
parameters could be based on statistical analysis
data$PODincrease = data$Sensitivity*(0.5*(1-data$BandwidthAvgPcnt)+0.5*(1-
data$BandwidthPeakPcnt)+0.1*data$Incidents+0.001*(data$LatencyAvg-
data$LatencyNormalAvg)+0.001*(data$LatencyPeak-data$LatencyNormalPeak))

# Insert probability of defection, for each month, for each group
# Check if POD is > 100%, and cap at 100% if so

data$POD_1_Month = data$BaseChurnMonthly + data$PODincrease
data$POD_1_Month[data$POD_1_Month > 1] = 1 # those cells in this vector
where the cell has a value > 1 gets value = 1

data$POD_2_Month = data$POD_1_Month + data$PODincrease^5
data$POD_2_Month[data$POD_2_Month > 1] = 1

data$POD_3_Month = data$POD_2_Month + data$PODincrease^4
data$POD_3_Month[data$POD_3_Month > 1] = 1

data$POD_4_Month = data$POD_3_Month + data$PODincrease^3
data$POD_4_Month[data$POD_4_Month > 1] = 1
```

```
data$POD_5_Month = data$POD_4_Month + data$PODincrease^2
data$POD_5_Month[data$POD_5_Month > 1] = 1

data$POD_6_Month = data$POD_5_Month + data$PODincrease^1.5
data$POD_6_Month[data$POD_6_Month > 1] = 1


# Create more data where each group, and each POD time frame, and POD value
is a separate row
prob1M = data.frame(Group=data$Group)
prob1M$TimeFrame = c("1M")
prob1M$POD = data$POD_1_Month

prob2M = data.frame(Group=data$Group)
prob2M$TimeFrame = c("2M")
prob2M$POD = data$POD_2_Month

prob3M = data.frame(Group=data$Group)
prob3M$TimeFrame = c("3M")
prob3M$POD = data$POD_3_Month

prob4M = data.frame(Group=data$Group)
prob4M$TimeFrame = c("4M")
prob4M$POD = data$POD_4_Month

prob5M = data.frame(Group=data$Group)
prob5M$TimeFrame = c("5M")
prob5M$POD = data$POD_5_Month

prob6M = data.frame(Group=data$Group)
prob6M$TimeFrame = c("6M")
prob6M$POD = data$POD_6_Month

# Make a union of all by group, by timeframe POD data sets
prob = rbind(prob1M, prob2M, prob3M, prob4M, prob5M, prob6M)

# Join the row-by-row POD data onto the main dataset
result = merge(prob, data, by = "Group", all.x = TRUE)

# Calculate Revenue at risk per group, per time frame
result$RevenueAtRisk = result$POD * result$Size * result$MonthSpend

# command R to hand back the results data
result
```

In Desktop Designer, create the following calculated columns with the specified expressions:

- Revenue all things alike; expression: [Size]*[MonthSpend]
- Revenue at risk (%); expression: [RevenueAtRisk]/([Size]*[MonthSpend])
- Revenue at risk (M); expression: [RevenueAtRisk]/1000000

Also, if you want to look at how the Navigation Actions are setup in this example, note that they are *Dashboard context* actions rather than Workbook context actions. This means that you can view them by right-clicking the dashboard tab and selecting Actions.

All of the above can be tried out using the example workbook example113_Rserve_Simulated_prediction_model.exw

Example 114: Predictive analysis with multiple linear regression

The subject matter expert for this example is Henrik Melin, quantitative analysis expert in the Solutions team, based in Stockholm, henrik_melin@datawatch.com

This example shows how to work with predictive analytics by learning from historic data to be able to make a qualified assessment of what future data values will be.

The historic data with actual outcome values (the *training data*) is a data set containing one *dependent variable*[1], and one or several *explanatory variables*[2]. The assumption is that the value of the dependent variable is correlated to the values of the explanatory values. The historic data is analyzed to create a *multiple linear regression model*.

ObjectID	Location	PreviousPrice	Bedrooms	Bathrooms	Size	Price.SQ.Ft	Status	SellPrice
154482	Atascadero	349000	5	3	3000	116.33	Regular	698000
154483	Cambria	825000	2	3	1840	448.37	Regular	1650000
154484	Grover Beach	255000	3	2	1189	214.47	Short Sale	510000
			Explanatory variables					*Dependent variable*

Sample from the training data set

Linear regression assumes that the relationship between the dependent variable (y) and any one of the explanatory variables (x) is linear, i.e. can be plotted as a straight line, y = mx+b.

The MLR model is later applied to a data set which contains values for only the explanatory variables, where the values of the dependent variable is still unknown. Thereby, predicted values for the dependent variable can be calculated, based on the new explanatory values and the correlations found in the historic data set.

Multiple linear regression in this example

In this example, the training data is about sold housing real estate. The data contains the prices at which these properties were sold. The data also contains:

- the number or bedrooms
- the number of bathrooms
- the total size

These are the explanatory values which are assumed to be correlated to the valuation and price of each property. I.e., we are making the fundamental assumption that the number of bedrooms, bathrooms and the total size of the property will affect the valuation and price of the property, and therefore appoint these three variables as *explanatory variables*.

The model is then applied to similar data where the final sell price outcome is still unknown. By feeding the multiple linear regression model with the values for bedrooms, bathrooms, and size, the model will calculate a predicted value of the property.

By comparing the predicted value to the listed price of the property, potential overvaluation and undervaluation can be detected, and good market opportunities can be identified.

Non-quantitative variables

[1] Other names used for dependent variable are *regressand*, *endogenous variable*, *response variable*, *measured variable*, *criterion variable*

[2] Other named used for explanatory variables are *regressors*, *exogenous variables*, *covariates*, *input variables*, *predictor variables*, *independent variables*

In this example data, there is another variable that could have been taken into account as an explanatory variable: Location. It would make a lot of sense to do so, since it is well known that area and location affects housing prices more than many other things. However, to be used in an MLR model, a variable must be quantitative, i.e. numeric. To make location a part of this analysis, the locations would have to be quantified. This could be done by for example measuring the driving distance from the property to the nearest shopping facility, or the walking distance to the nearest subway station. In this example, that exercise has been omitted.

Additional theory

Let's look at the general straight line equation[3] or linear equation again:

y = mx+b

where m is the slope of the line, and b is the intercept, i.e. the value of y at the point where the line crosses the y axis is the graph.

When doing a multiple linear regression modelling, you can think of it as being an operation where the best value of m for each variable is identified, so that each variable multiplied with its own m-value adds up to the value of the dependent (predicted) variable. These m-values are referred to as coefficients. In addition, another coefficient is calculated, which is the value for b, the intercept. This is added as-is (multiplied by 1 instead of multiplied by any of the variables).

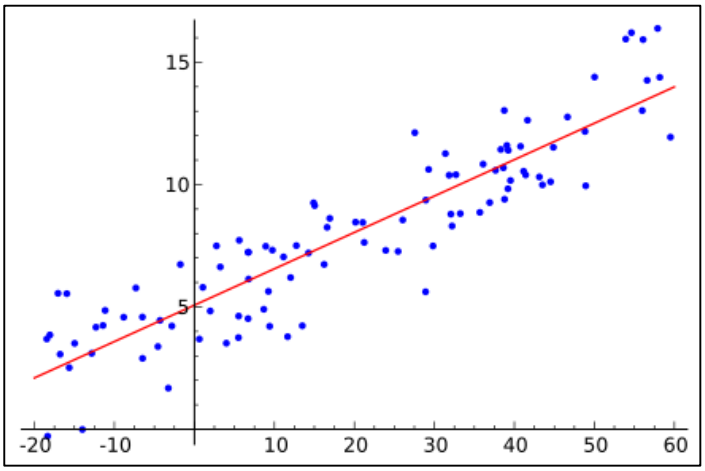


Figure 1: The red line is an example of y=mx+b. The blue points represent the training data.
Image source: Wikipedia

So, for this example, with three explanatory variables x_1, x_2, x_3, and four coefficients $A_{Intercept}$, A_1, A_2, A_3, the value of the dependent variable or predicted variable for a given set of values of the explanatory variables can be written as:

$$h\,(1, x_1, x_2, x_3) = A_{Intercept}\,1 + A_1 x_1 + A_2 x_2 + A_3 x_3$$

With several data records in a data set, the operation of multiplying each occurrence of each variable with the appropriate coefficient is a matrix multiplication[4], where a vector containing the 4 coefficients [1x4] is multiplied with a matrix containing each occurrence of the three explanatory variables, and a 1. So with *p* occurrences or records, that is a matrix of [4x*p*].

If we expand the equation above to better reflect the matrix multiplication, we can write this:

$$\text{EstimatedValue} = h\,(1, x_1, x_2, x_3) = X_{input} \times A = \begin{bmatrix} 1 & \cdots & x3,1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & x3,n \end{bmatrix} \times [A_{Intercept}\ A_1\ A_2\ A_3]$$

A good next step for optional further reading is this Wikipedia article:
http://en.wikipedia.org/wiki/Linear_regression

[3] http://www.mathsisfun.com/equation_of_line.html
[4] http://www.mathsisfun.com/algebra/matrix-multiplying.html

This is the R script used in the example:

```
##########################################
#
#   Multiple Linear Regression Modelling
#   for Predictive Analysis
#
# In this example of Multiple Linear Regression Modelling, the training set
# is a set of historic prices for real estate properties.
#
# Estimate = h (1, x1, x2, x3 )=  A(Intercept)*1+ A1*x1 +A2*x2 +A3*x3.
# The A:s are the intercept and the three coefficients for bedrooms,
# bathrooms, and size.
# The new input data is a set of 63 real estate properties for sale
# The model estimates the actual value of the properties for sale, which we
# compare to the price asked
# in order to identify market opportunity in the form of undervalued
# real estate.
#
##########################################

# Set working directory
setwd ("C:/DWCH/DEK/R")

# Read historic data set which is the training set for the
# Multiple Linear Regression analysis
soldRealEstate<-read.csv("example114_MLR_training_set.csv", header = TRUE)

# Read data about properties currently for sale for which
# the value will be predicted
unSoldRealEstate<-read.csv("example114_MLR_prediction.csv", header = TRUE)

# We use the lm() function from the stats package to create a
# Multiple Linear Regression Model which we name modelQ
# We are specifying the price of previously sold real estate as the
# dependent variable, and as explanatory variables we are
# specifying the number of bedrooms, the number of bathrooms,
# and the size is square feet
modelQ <- lm (soldRealEstate$SellPrice ~ soldRealEstate$Bedrooms +
soldRealEstate$Bathrooms + soldRealEstate$Size)

# Here, we are extracting the coefficients calculated through the
# Multiple Linear Regression modelling above.
# The first coefficient is the intercept, or "b" in the
# straight line equation y=mx+b
# The second, third, and fourth coefficents are the three slope values,
# "m", for each of the three explanatory variables.
A <-coefficients (modelQ,level=0.95)

# We create a matrix where column 1 is a vector of all 1:s,
# and the 2nd, 3rd, and 4thcolumns are vectors containing the existing
values
# of the three explanatory variables
# that we have in the data set of unsold real estate
x <-cbind(1, unSoldRealEstate$Bedrooms ,unSoldRealEstate$Bathrooms ,
unSoldRealEstate$Size)

# We do a matrix multiplication of the vector A (coefficients)
# and the matrix X (explanatory variables).
```

```
# The matrix multiplication is achieved using the crossprod() function.
# note that in matrix multiplication there must be a match between column
# count in the first matrix and row count in the second matrix,
# so therefore we are doing a transpose t() of the X matrix which achieves
# this: A [1x4] and X [4x63]
# crossprod() will automatically turn the vector A into a single row
matrix.
# For the calculation, it does not matter if we write "crossprod(t(x),A)"
or
# "crossprod(A, t(x))". However, in the latter
# case, we will get a result which is a single row, 63 column matrix, while
# what we need is a 63 row, single column matrix
# Therefore, we will write "crossprod(t(x),A)"
EstimatedValue  <- crossprod (t(x),A)

# We calculate the estimated overpricing or underpricing
Opportunity <- (EstimatedValue - unSoldRealEstate$CurrentListPrice)

# We calculate the estimated overpricing or underpricing
# as a percentage of the list price
OpportunityPercent<- (Opportunity/unSoldRealEstate$CurrentListPrice)

# We remove unnecessary columns from the data frame
unSoldRealEstate$Status <- NULL
unSoldRealEstate$PreviousPrice <- NULL
unSoldRealEstate$Price.SQ.Ft  <- NULL

# We put together a new data frame containing the remaining columns
# and the new estimated value
unSoldRealEstate = cbind
(unSoldRealEstate,EstimatedValue,Opportunity,OpportunityPercent)

# We add a categorization based on the opportunity identified.
unSoldRealEstate$Group[unSoldRealEstate$Opportunity > 0] <- 'Underpriced'
unSoldRealEstate$Group[unSoldRealEstate$Opportunity < 0] <- 'Overpriced'
unSoldRealEstate$Group[unSoldRealEstate$Opportunity == 0] <- 'Neutral'

# We add an 'id' prefix to the ObjectID column to assure it is
# loaded as text in Datawatch
unSoldRealEstate$ObjectID <- paste0('id',unSoldRealEstate$ObjectID)

# we command R to return the data frame content
unSoldRealEstate
```

All of the above can be tried out using the example workbook example114_Rserve_MLR_prediction.exw

Example 115: Scraping data from HTML tables and joining

This example shows how to load data from HTML tables on the web, and join the data sets to create a combined, more informative data set.

We are loading stock prices and stock performance numbers for the large cap list of the Stockholm Stock Exchange which are published on the web site of the daily newspaper Dagens Industri on http://www.Di.se. This data set is combined with a list of company names, symbols, ISIN codes, and market sectors published on http://www.nasdaqomxnordic.com/. We are also loading the current exchange rate for SEK to USD from the Swedish bank SEB on http://www.SEB.se to use in the calculation of USD dollar values.

The example used the library "XML", which you must first install by using this command in R:

```
install.packages("XML")
```

```
library(XML)

URL.NasdaqStockholm <- "http://www.nasdaqomxnordic.com/shares/listed-
companies/stockholm"
URL.sthlmLargeCapNumbers <- "http://www.di.se/borssidor/large-cap/"
URL.currency <- "http://www.seb.se/pow/apps/valutakurser/avista_tot.asp"


# loading a html table on a URL into a data frame. It is the 1st html
numbers in the page
stocks = readHTMLTable(URL.NasdaqStockholm, header=T,
which=1,stringsAsFactors=F)

# loading a html table on a URL into a data frame. It is the 1st html
numbers in the page
numbers = readHTMLTable(URL.sthlmLargeCapNumbers, header=T,
which=1,stringsAsFactors=F)

# loading a html table on a URL into a data frame. It is the 2nd html
numbers in the page
currency = readHTMLTable(URL.currency, header=T,
which=2,stringsAsFactors=F)

# replacing decimal comma with decimal dot
currency <-
as.data.frame(sapply(currency,gsub,pattern=",",replacement="."))

# inserting new column names
colnames(currency) <- c("Country", "Currency", "Buy", "Sell", "Date")

# removing the row containing the column names from the html table
currency <- currency[!currency$Country == "Land",]

# removing all NA data
currency <- na.omit(currency)

# resetting the rownames
rownames(currency) <- NULL

# change from factors to numerics to be able to work with the data
currency$Currency <- as.character(currency$Currency)
currency$Buy <- as.numeric(as.character(currency$Buy))
currency$Sell <- as.numeric(as.character(currency$Sell))
```

```
# calculating the average rate between buy and sell
currency$BuySellAvg <- (currency$Buy+currency$Sell)/2

# putting the exch rate for SEK to USD in a vector and resetting the
rowname
currencyUSD <- currency[currency$Currency == "USD",]
rownames(currencyUSD) <- NULL


# assigning new column titles
colnames(numbers) <- c("Company", "Latest", "Percent", "Change", "Buy",
"Sell", "High", "Low", "YTD_pcnt", "AllTimeHigh", "ATH_Date", "Time")

# The html numbers contains repeated column titles inside the numbers.
Removing these.
numbers <- numbers[numbers$Company != "Aktie", ]

# there may be rows in the data which reflect errors in the HTML numbers
# this removes any rows where there is not a value for date
numbers <- numbers[numbers$ATH_Date > 0, ]
numbers <- numbers[!rownames(numbers) == 'NA', ]

# replacing decimal comma with decimal dot
numbers <- as.data.frame(sapply(numbers,gsub,pattern=",",replacement="."))

# resetting the row labels
rownames(numbers) <- NULL

# fixing come company names to make the names match in the join
numbers$Company <- as.character(numbers$Company)
numbers$JoinNames <- numbers$Company # creating a copy of the Company
column to use for joining
numbers$JoinNames[numbers$JoinNames == "ABB"] <- "ABB Ltd"
numbers$JoinNames[numbers$JoinNames == "Africa Oil Corp."] <- "Africa Oil"
numbers$JoinNames[numbers$JoinNames == "Assa Abloy B"] <- "ASSA ABLOY B"
numbers$JoinNames[numbers$JoinNames == "Castellum B"] <- "Castellum"
numbers$JoinNames[numbers$JoinNames == "H&M B"] <- "Hennes & Mauritz B"
numbers$JoinNames[numbers$JoinNames == "Handelsbanken A"] <- "Sv.
Handelsbanken A"
numbers$JoinNames[numbers$JoinNames == "Handelsbanken B"] <- "Sv.
Handelsbanken B"
numbers$JoinNames[numbers$JoinNames == "Hexpol B"] <- "HEXPOL B"
numbers$JoinNames[numbers$JoinNames == "Lundberg B"] <- "Lundbergföretagen
B"
numbers$JoinNames[numbers$JoinNames == "Lundberg B"] <- "Lundbergföretagen
B"
numbers$JoinNames[numbers$JoinNames == "Meda A"] <- "Meda  A"
numbers$JoinNames[numbers$JoinNames == "Millicom International Cellular
SDB"] <- "Millicom Int. Cellular SDB"
numbers$JoinNames[numbers$JoinNames == "MTG A"] <- "Modern Times Group A"
numbers$JoinNames[numbers$JoinNames == "MTG B"] <- "Modern Times Group B"
numbers$JoinNames[numbers$JoinNames == "Nibe Industrier B"] <- "NIBE
Industrier B"
numbers$JoinNames[numbers$JoinNames == "Oriflame Cosmetics SDB"] <-
"Oriflame, SDB"
numbers$JoinNames[numbers$JoinNames == "Tieto"] <- "Tieto Oyj"

# renaming the first column in the stocks data frame
colnames(stocks)[1] <- "JoinNames"
```

```
# joining the company details onto the stock performance data as a left
outer join (keeping all in the numbers data frame)
DataCombo <- merge(numbers, stocks, by = "JoinNames", all.x = TRUE)

# getting rid of the JoinNames column
DataCombo$JoinNames <- NULL

# adding an exchange rate column SEK to USD
DataCombo$ExchRate <- currencyUSD$BuySellAvg

# casting columns to numeric from "factor"
DataCombo$Latest <- as.numeric(as.character(DataCombo$Latest))
DataCombo$Percent <- as.numeric(as.character(DataCombo$Percent))
DataCombo$Change <- as.numeric(as.character(DataCombo$Change))
DataCombo$Buy <- as.numeric(as.character(DataCombo$Buy))
DataCombo$Sell <- as.numeric(as.character(DataCombo$Sell))
DataCombo$High <- as.numeric(as.character(DataCombo$High))
DataCombo$Low <- as.numeric(as.character(DataCombo$Low))
DataCombo$YTD_pcnt <- as.numeric(as.character(DataCombo$YTD_pcnt))
DataCombo$AllTimeHigh <- as.numeric(as.character(DataCombo$AllTimeHigh))

# calculating USD versions of the price columns
DataCombo$Latest.USD <- DataCombo$Latest/DataCombo$ExchRate
DataCombo$Buy.USD <- DataCombo$Buy/DataCombo$ExchRate
DataCombo$Sell.USD <- DataCombo$Sell/DataCombo$ExchRate
DataCombo$High.USD <- DataCombo$High/DataCombo$ExchRate
DataCombo$Low.USD <- DataCombo$Low/DataCombo$ExchRate

# building a proper date format of the ATH_Date
DataCombo$ATH_Date <- paste0('20', substr(DataCombo$ATH_Date, 1, 2), '-',
substr(DataCombo$ATH_Date, 3, 4), '-', substr(DataCombo$ATH_Date, 5, 6))

# creating a column reflecting the time of loading the data
DataCombo$DataLoadTime <- Sys.time()

# renaming the data set
StockholmLargeCaps <- DataCombo
remove(DataCombo)

# commanding R to return the data
StockholmLargeCaps
```

All of the above can be tried out using the example workbook example115_Rserve_web_scraping.exw

Using Rserve for transforming data loaded with other data connectors

The examples that follow below focus on things you can do to transform and prepare data loaded from any other connector in Datawatch (except streaming connectors).

When loading data from a web API, you typically have limited or no options for modifying or transforming the data. Such data sources could be accessible with the Datawatch XML connector or JSON connector, and the data set obtained could be modified through an R transform after loading the data from the source.

When applying an R transform to a data set loaded in Datawatch, you therefore specify a name for this data frame which will be created in R when R loads the data set.

After you have done what you want in your R script, the final command should be that you request Rserve to return the data frame which you intend to use in your dashboard.

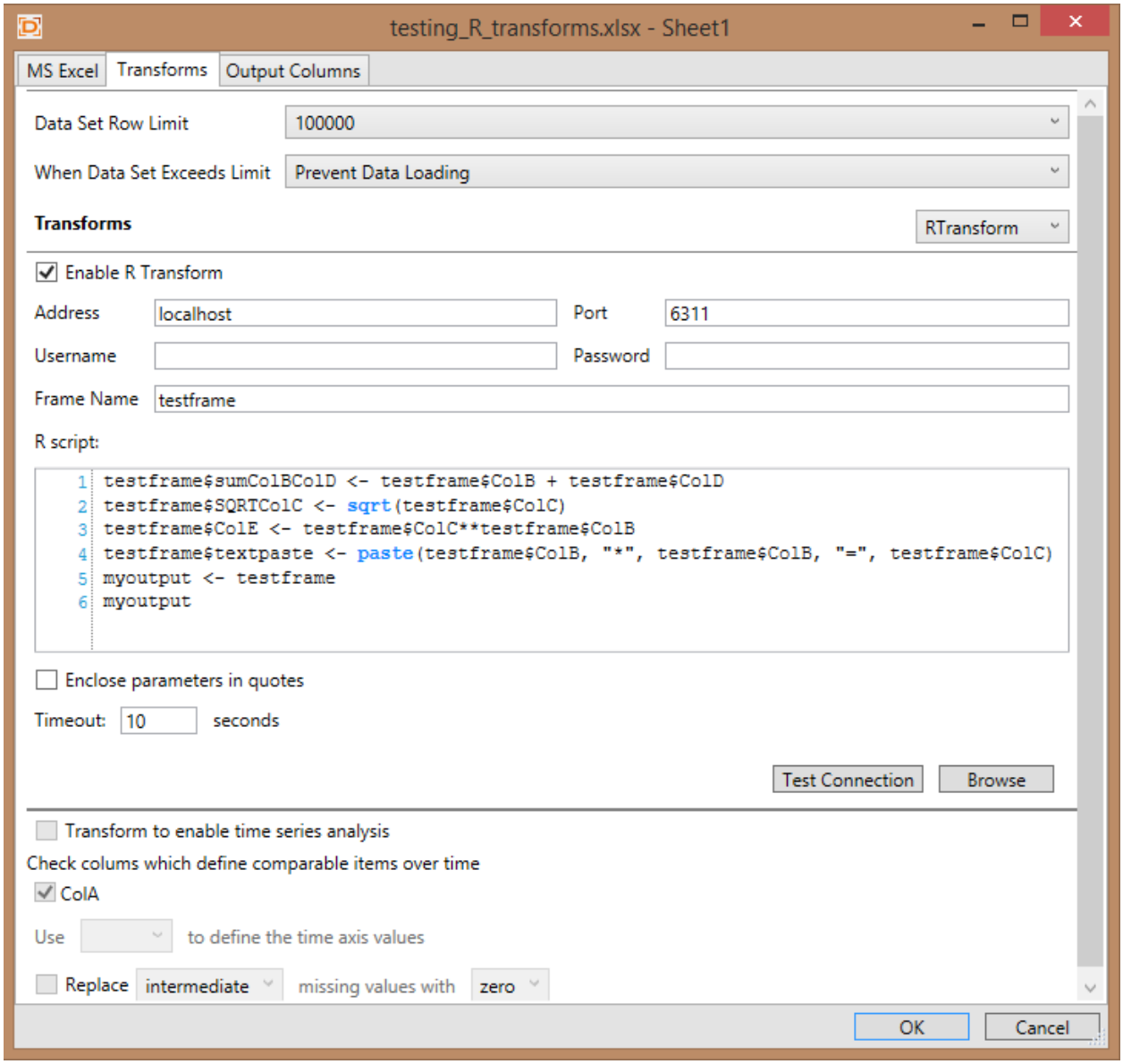
This means that the data frame which you specify in the settings for the R transform can be named differently from the frame name that you finally bring back into Datawatch.

Example 201: Simple calculations and string manipulation

Consider this data set, which could be in an Excel file or a CSV file. The column names and Row labels were chosen arbitrarily to make a simple example.

ColA	**ColB**	**ColC**	**ColD**
RowA	1	1	9
RowB	2	4	8
RowC	3	9	7
RowD	4	16	6
RowE	5	25	5
RowF	6	36	4
RowG	7	49	3
RowH	8	64	2
RowI	9	81	1

After loading this small data set using any of the data connectors, you can perform transforms and calculations on it using R, by sending data and R code to Rserve. In the transformation settings dialog below, you can see that we have Rserve running on our local system (localhost).



Here is the R script for copy-paste purposes:

```
testframe$sumColBColD <- testframe$ColB + testframe$ColD
testframe$SQRTColC <- sqrt(testframe$ColC)
testframe$ColE <- testframe$ColC**testframe$ColB
testframe$textpaste <- paste(testframe$ColB, "*", testframe$ColB, "=",
testframe$ColC)
myoutput <- testframe
myoutput
```

When this R script runs on the sample data set shown above, the result (which in this example is finally put in a data frame called myoutput) contains this:

ColA	ColB	ColC	ColD	sumColBColD	SQRTColC	ColE	textpaste
RowA	1	1	9	10	1	1	1 * 1 = 1
RowB	2	4	8	10	2	16	2 * 2 = 4

RowC	3	9	7	10	3	729	3 * 3 = 9
RowD	4	16	6	10	4	65,536.00	4 * 4 = 16
RowE	5	25	5	10	5	9,765,625.00	5 * 5 = 25
RowF	6	36	4	10	6	2,176,782,336.00	6 * 6 = 36
RowG	7	49	3	10	7	678,223,072,849.00	7 * 7 = 49
RowH	8	64	2	10	8	281,474,976,710,656.00	8 * 8 = 64
RowI	9	81	1	10	9	150,094,635,296,999,000.00	9 * 9 = 81

Summary and repetition

The dialog shown in this document is what you see when you choose to do a so called “transform” on a data set you have loaded into Desktop Designer. The data connector can be any of the ones available.

The data set loaded through the connector is sent to Rserve, where it is at once loaded into a so called data frame, in memory, in Rserve. The data frame will be named as you specify in the settings in the dialog.

While the data is in Rserve, you can move it in and out of any number of data frames. One reason for moving the data into another data frame can be to continue working with a changed version of the data set, while keeping the original version untouched. To terminate your R transforms and get data back from Rserve, you specify the desired data frame name on the last row of your R-script. In the example above, that is the data frame called “myoutput”.

The data at this point is still in memory in Rserve, in a format called a data frame, which you can think of as a table. The data set is transferred into the Datawatch in-memory data model. This means that the output from Rserve to Datawatch does not have a file format – is it a data object that exists in RAM only.

Tips and hints for those used to SQL

When using SQL to work with data, you frequently apply various types of joins to merge data sets or tables.

Below is what you do in R to achieve the equivalence of various SQL joins. The examples assume we have two data sets “Adata” and “Bdata”. We assume we are using vectors (columns) called “key”, “keyA”, and “keyB” as the join keys depending on example. With the R function *merge* which we are using, the first data set (in this case “Adata”) is referred to as x, and the second as y.

Join type	**R syntax**
INNER JOIN	`merge(Adata, Bdata, by = "key")`
	`merge(Adata, Bdata, by.x = "keyA", by.y = "keyB")`
OUTER JOIN	`merge(Adata, Bdata, by = "key", all = TRUE)`
LEFT OUTER JOIN	`merge(Adata, Bdata, by = "key", all.x = TRUE)`
RIGHT OUTER JOIN	`merge(Adata, Bdata, by = "key", all.y = TRUE)`
CROSS JOIN	`merge(Adata, Bdata, by = NULL)`
	`expand.grid(Adata, Bdata)`
UNION	`rbind(Adata, Bdata)`

Tips and hints for producing synthetic data

There are a number of easy commands or functions in R that quickly produces sample data.

The following is part of R base, i.e. it comes with R without the need for additional packages:

X = c(1:100)	Create a list (vector) with all numbers from 1 to 100
X = seq(10, 1000, 2)	Create a list (vector) going from 10 to 1000 in steps of 2
X = rep(LETTERS[1:4], 4)	Create a list (vector) of the letters A, B, C, D repeated 4 times (16 rows)
X = expand.grid(y,z)	Creates a list of two columns with one row for each unique combination of values from the vectors y and z.
X = seq(from=as.Date("1975-01-01"), to=as.Date("1975-12-31"), by="days")	Creates a list of dates for each day from 1975-01-01 to 1975-12-31
X = seq(from=as.Date("2014/10/1"), by="quarters", length.out=16)	Creates a list of dates, quarterly, for 16 quarters ahead starting on October 1, 2014

X = rnorm(1000, mean = 50, sd = 20)	Creates 1000 random numbers that follow the normal distribution with mean 50 and standard deviation 20. The numbers will have decimals.
X = runif(1000, 0, 100)	Creates 1000 random numbers that follow the uniform pasdistribution (same probability for all numbers) between 0 and 100. The numbers will have decimals.
X = sample(0:100, 1000, replace = T)	Creates 1000 random numbers that follow the uniform distribution between 0 and 100. The numbers will be integer values. Replace = T means the same value can occur more than once. Note! In this case replace = T (true) is necessary to achieve a sample of 1000 from 100 unique values. Entering Replace = F will return an error.
X = sample(0:100, 33, replace = F)	Creates 33 random numbers that follow the uniform distribution between 0 and 100. The numbers will be integer values. Replace = F (false) means each value can occur only once.
X = sample(y, 10)	Creates a list of 10 sample from the values held in the vector y. If you sample the same number of values that exist in the vector, this is a handy way of *randomizing the order* of the values.
set.seed(N)	This command is useful when you want the same result each time you run a script that contains (pseudo)-randomly generated numbers. Set N as any integer value. As long as you use the same value for N, the random number generation will return the same results each time.

Tips and hints for working with date and time data

More often than not, you find yourself in need of converting between different date-time formats, from date-time to string values, from integers to date-time, and from strings to date-time. For any such tasks, look up the following classes and functions, either by Googling, or by the built-in help in R.

weekdays(Sys.monthsDate())	Today's day of the week
months()	
quarters()	
strptime()	For converting from string to date-time
as.POSIXct	
as.POSIXct	
format.Date	
cut.Date	
as.Date	
round.Date	
difftime	

You should also look into the package "chron" which is specifically for date-time manipulation.

Good web resources for loading data into R

http://theodi.org/blog/how-to-use-r-to-access-data-on-the-web

Big Data

R is often referenced in "Big data" discussions. However, recall that R is a pure in-memory calculation product, meaning that there is a natural limit to how much data you can handle at once. Below is some good information for when you need to deal with large data sets that stretch the capabilities of your hardware in terms of RAM and CPU.

The Open Data Institute published a blog post with 11 tips about how to handle big data with R. It is quoted in full below.

1. Think in vectors. This means that for-loops ("do this for x times") are generally a bad idea in R. The R Inferno has a chapter on why and how to "vectorise". Especially if you're a beginner or come from another programming language for-loops might be tempting. Resist and try to speak R. The apply family may be a good starting point, but of course do not avoid loops simply for the sake of avoiding loops.
2. Use the fantastic data.table package. Its advantage lies in speed and efficiency. Developed by Matthew Dowle it introduces a way of handling data, similar to the data.frame class. In fact, it includes data.frame, but some of the syntax is different. Luckily, the documentation (and the FAQ) are excellent.
3. Read csv-files with the fread function instead of read.csv (read.table). It is faster in reading a file in table format and gives you feedback on progress. However, it comes with a big warning sign

"not for production use yet". One trick it uses is to read the first, middle, and last 5 rows to determine column types. Side note: ALL functions that take longer than 5 seconds should have progress bars. Writing your own function? –> txtProgressBar.

4. Parse POSIX dates with the very fast package fasttime. Let me say that again: very fast. (Though the dates have to be in a standard format.)
5. Avoid copying data.frames and remove, rm(yourdatacopy), some in your workflow. You'll figure this out anyway when you run out of space.
6. Merge data.frames with the superior rbindlist – data.table we meet again. As the community knows: "rbindlist is an optimized version of do.call(rbind, list(...)), which is known for being slow when using rbind.data.frame."
7. Regular expressions can be slow, too. On one occasion I found a simpler way and used the stringr package instead; it was a lot faster.
8. No R tips collection would be complete without a hat tip to Hadley Wickham. Besides stringr, I point you to bigvis, a package for visualising big data sets.
9. Use a random sample for your exploratory analysis or to test code. Here is a code snippet that will give you a convenient function: row.sample(yourdata, 1000) will reduce your massive file to a random sample of 1,000 observations.
10. Related to the previous point is reading only a subset of the data you want to analyze. read.csv() for example has a nrows option, which only reads the first x number of lines. This is also a good idea of getting your header names. The preferred option, a random sample, is more difficult and probably needs a 'workaround' as described here.
11. Export your data set directly as gzip. Writing a compressed csv file is not entirepastely trivial, but stackoverflow has the answer. Revolution Analytics also has some benchmark times for compressions in R.

http://theodi.org/blog/fig-data-11-tips-how-handle-big-data-r-and-1-bad-pun (July 18, 2013)

Good web resources for learning R syntax

http://www.statmethods.net/

http://www.statmethods.net/input/importingdata.html

http://www.statmethods.net/management/index.html

http://www.statmethods.net/stats/index.html

http://www.statmethods.net/advstats/index.html

http://www.r-tutor.com/

http://www.r-tutor.com/r-introduction

http://www.r-tutor.com/r-introduction/data-frame/data-frame-column-vector

http://www.r-tutor.com/r-introduction/data-frame/data-frame-column-slice

http://www.r-tutor.com/r-introduction/data-frame/data-frame-row-slice

http://www.r-tutor.com/r-introduction/data-frame/data-import

http://www.r-tutor.com/elementary-statistics

http://www.r-tutor.com/elementary-statistics/numerical-measures

http://www.nceas.ucsb.edu/files/scicomp/Dloads/RProgramming/BestFirstRTutorial.pdf

(17 pages)

About R

R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.

R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.

One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.

R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

Many users think of R as a statistics system. Another way of thinking of R is as an environment within which statistical techniques are implemented. R can be extended (easily) via packages. There are about eight packages supplied with the R distribution and many more are available through the CRAN family of Internet sites covering a very wide range of modern statistics.

Read more on: http://www.r-project.org/

About Rserve

The main web site of RServe is https://rforge.net/Rserve/

Rserve is a TCP/IP server which allows other programs to use facilities of R. Every connection has a separate workspace and working directory (unless you are using Rserve on Windows – see below).

The documentation for Rserve is found on https://rforge.net/Rserve/doc.html

Rserve itself is the server, that is, a program that responds to requests from clients. It listens for any incoming connections and processes incoming requests. You need to have R-1.5.0 or higher installed on your system in order to be able to use Rserve. NOTE! In relation to Rserve, Datawatch Server is a CLIENT of Rserve.

Using Rserve on Windows

If you have more versions of R, make sure that you use the one **most recently installed**, because Rserve detects the location of $RHOME from the registry. Although Rserve works on Windows it is not recommended to use it on that platform. Windows lacks important features that make the separation of namespaces possible, therefore Rserve for Windows works in cooperative mode only, that is, only one connection at a time is allowed and all subsequent connections share the same namespace. One consequence of this is that if one user creates a variable or a data frame with a certain name, that object can potentially be over-written by another user assigning the same name. **Briefly: don't use Windows as the platform for Rserve unless you really have to**.

This means that it is fine running Rserve on Windows for demonstration and development purposes. However, in production installation, when shared by multiple users and multiple applications, Rserve will run better on Unix/Linux.

Configuration of Rserve on Linux / Unix

If no config file is supplied, Rserve accepts no remote connections, requires no authentication and file transfer is enabled.

On Linux/Unix, Rserve is configured by the configuration file /etc/Rserv.conf

Additional configuration files can be added by the --RS-conf command line argument. The possible configuration entries are as follows (all entries are optional; default values are in angled brackets):

```
workdir <path> [/tmp/Rserv]
pwdfile <file> [none=disabled]
remote enable|disable [disable]
auth required|disable [disable]
plaintext enable|disable [disable]
fileio enable|disable [enable]
interactive yes|no [yes]
socket <socket> [none=disabled]
port <port> [6311]
maxinbuf <size in kb> [262144]
maxsendbuf <size in kb> [0=unlimited]
su now|server|client [none
source <file>
eval <expressions>
sockmod <mode> [0=default]
encoding native|utf8|latin1 [native]
```

Supported on Unix only:

```
chroot <directory> [none]
uid <uid> [none]
gid <gid> [none]
umask <mask> [0]
```

The following configuration options can use either hexadecimal (0x..), octal (0..) or decimal values:

`Port`
`Uid`
`Gid`
`Umask`
`sockmode`.
All other options and command-line options always assume decimal notation.

Most entries are self-explaining and their command line equivalents are described below.

A note on buffer sizes

`maxinbuf` and `maxsendbuf` are rather special. Previous versions of Rserve had fixed buffer sizes. Since 0.1-9 internal buffers change per-connection automatically. The maxinbuf specifies (in kilobytes) the maximal allowable size of the input buffer, that is, the maximal size of data transported from the client to the server. Analogously maxsendbuf sets the maximum size of the send buffer, that is, the size of data sent from Rserve to the client. If your server is likely to process very many parallel connections you may

want to lower this setting for security reasons. On the other hand if the server will process only few connections in parallel and you expect very large data, raise the value according to your computer's memory. Basically the settings are present to prevent malicious users from *crashing your server by supplying too large data* and thereby cause an out-of-memory error. 0 has a special meaning telling Rserve to allow unlimited use.

Only supported by Unix

uid, gid, umask and chroot are supported on unix computers only. If Rserve is run as root user, then it switches the user/group to the specified uid/gid before starting the server. Note: The directives are processed in the same sequence in which they occur in the config file. This implies that if using both uid and gid, you MUST use gid first, otherwise if setting uid first, then the user will have no right to change gid anymore! Also chroot must be used before uid as only root can use it.

Security

Anyone with access to R has access to the shell via "system" command, so you should consider following rules:

- NEVER EVER run Rserve as root (unless uid/gid entries are set) - this compromises the box totally
- Use "remote disable" whenever you don't need remote access.
- If you need remote access use "auth required" and "plaintext disable". Consider also that anyone with the access can decipher other's passwords if he knows how to. The authentication prevents hackers from the net to break into Rserve, but it doesn't (and cannot) protect from inside attacks (since R has no security measures).
- You should also use a special, restricted user for running Rserv as a public server, so no-one can try to hack the box it runs on.
- Don't enable plaintext unless you really have to. Passing passwords in plain text over the net is not wise and not necessary since both Rserv and JRclient provide encrypted passwords with server-side challenge (thus safe from sniffing).
- On Unix, consider using `su client` and `cachepwd yes` options with a password file that is only root-readable. That will prevent client password retrieval and clients from attacking the server (see NEWS for 0.6-1).

String encoding

String encoding directive was introduced in Rserve 0.5-3. This means that strings are converted to the given encoding before being sent to the client and also all strings from the client are assumed to come from the given encoding. (Previously the strings were always passed as-is with no conversion). The currently supported encodings are "native" (same as the server session locale), "utf8" and "latin1". The server default is currently "native" for compatibility with previous versions (but may change to "utf8" in the future, so explicit use of encoding in the config file is advised).

If a server is used mainly by Java clients, it is advisable to set the server encoding to "utf8" since that is the only encoding supported by Java clients. (i.e. add encoding utf8 line to the config file).

For efficiency it is still advisable to run Rserve in the same locale as the majority of clients to minimize the necessary conversions. With diverse clients UTF-8 is the most versatile encoding for the server to run in while it can still serve latin1 clients as well.

Command line arguments

Starting with Rserve 0.1-9, special command line arguments are supported in addition to the config file. Normally Rserve passes all arguments to R, but several special ones are processed and removed from the list before initializing R. Those parameters override any settings specified in the config file.

`--RS-port` *port*	Rserve will listen on the specified TCP port.
`--RS-socket` *socket*	Use the specified local unix socket instead of TCP/IP.
`--RS-workdir` *path*	Set working directory root for connections to the specified path.
`--RS-conf` *file*	Load the specified config file (the default config file is still loaded, but its settings have lower priority).
`--RS-dumplimit` *n*	(debug version of Rserve only) specifies the number of values to print when dumping content of packages or SEXPs (default is 128).
`--RS-settings`	Print current settings of the Rserve. Useful to check the current configuration.
`--RS-encoding` *encoding*	Set default string encoding to the given value
`--help`	Prints a brief help screen and exits (R is not started)
`--version`	Prints version of Rserve. This argument is retained in the list and processed by R as well.

In order to list all currently supported arguments, type:

```
R CMD Rserve –help
```