* **[Template]**

Notes:

1. xxx

Pre-Conditions:

1. xxx

Main Success Scenario:

1. Xxx

Alternatives:

1. Xxx

* **Add Column to DataFrame**

Pre-Conditions:

1. “size” is the size of the file

Main Success Scenario:

1. Add a column to the listOfFiles data frame that you can store file size in.

listOfFiles$fileSize <- size

* **Increase Breaks in Histogram Chart**

Note:

1. Increases the # of vertical bars in the Histogram

Pre-Conditions:

1. xxx

Main Success Scenario:

1. Xxx

Alternatives:

1. Xxx

* **Add Rug to Histogram Chart**

Pre-Conditions:

1. xxx

Main Success Scenario:

1. Xxx

Alternatives:

1. Xxx

* **Change Data Frame to Table**

Note:

1. This is just like pivoting rows to columns.

Pre-Conditions:

1. [Get Database Connection](#getDatabaseConnection)
2. Get two sets of data.

**dataQuery <- sqlQuery(cn,"Select CustomerID, Country from StgCustomerGeography Group By Country");**

1. Close database connection

**close(cn);**

Main Success Scenario:

1. Change dataQuery data frame to a table

Table(dataQuery)

1. Returns a table

CustomerID United Kingdom (L7) United States (L7)

[customer ids] boolean flag boolean flag

* **Check Class of Variable**

Main Success Scenario:

1. Have data frame variable called “dataQuery”.
2. See what class the variable is in.

**class(dataQuery)**

1. It will return “data.frame”.

* **Check if object is atomic**

Notes:

1. Recursive object has a list like structure. Atomic objects do not.
2. Vector is not recursive. Vector is an atomic object.
3. DataSetName$FieldName is only valid for recursive objects.

Main Success Scenario:

1. See if object “x” is atomic.

**is.atomic(x)**

* **Check if object is recursive**

Notes:

1. Recursive object has a list like structure.
2. Vector is not recursive. Vector is an atomic object.
3. DataSetName$FieldName is only valid for recursive objects.

Main Success Scenario:

1. See if object “x” is recursive.

**is.recursive(x)**

* **Check R version**

Main Success Scenario:

1. [Start R Console](#startRConsole)
2. Go to Help menu > About

* **Choose Plot**

Pre-Conditions:

1. Format
   1. Vector
      1. Good for:
         1. Line drawings using solid colors and modest number of points
         2. Plots using solid colors and modest number of points
         3. Resizing
   2. Bitmap
      1. Good for:
         1. Plots with large numbers of points.
         2. Natural Scenes
         3. Web-based plots
2. Device to Output to
   1. Screen on computer
   2. File
3. Plot System
   1. Base
   2. Lattice
   3. Ggplot2

Main Success Scenario:

Alternatives:

1. Xxx

* **Close SQL Server Connection**

Main Success Scenario:

1. Close “cn” connection

**odbcClose(cn)**

* **Compile R with Profiler**

Pre-Conditions:

1. xxx

Main Success Scenario:

1. Xxx

Alternatives:

1. Xxx

* **Copy Plot Chart to PDF file**

Pre-Conditions:

1. Using dev.copy2pdf
2. Dev.copy may not copy exactly as expected.

Main Success Scenario:

1. Xxx

Alternatives:

1. Xxx

* **Create Sequence of Numbers in R**

Main Success Scenario:

1. Create sequence from 3 to 5

**seq(3,5)**

1. Create sequence from 3 and for a length of 3

**seq(from = 3, length = 3)**

1. Create sequence from 3, for a length of three, and by 0.5

**seq(from = 3, length = 3, by = 0.5)**

* **Drop DataFrame**

Pre-Conditions:

1. Data frame is called hamburger

Main Success Scenario:

1. Hamburger <- NULL

* **Drop Relational DB table**

Pre-Conditions:

1. “SomeTable” is an existing table in the relational DB.

Main Success Scenario:

1. [Get Database Connection](#getDatabaseConnection).
2. Drop table

**sqlDrop(cn,”SomeTable”)**

1. Close database connection

**close(cn);**

* **Enable R use in Excel**

Pre-Conditions:

1. Save each tab in Excel file as .CSV
2. xxx

Main Success Scenario:

1. Xxx

Alternatives:

1. Xxx

* **Enable R use in MongoDB**

Pre-Conditions:

1. xxx

Main Success Scenario:

1. Xxx

Alternatives:

1. Xxx

* **Enable R use in SQL Server 2008 R2**

Pre-Conditions:

1. xxx

Main Success Scenario:

1. Xxx

Alternatives:

1. Xxx

* **Enable R use in SQL Server 2012**

Pre-Conditions:

1. xxx

Main Success Scenario:

1. Xxx

Alternatives:

1. Xxx

* **Enable R use in SQL Server 2014**

Pre-Conditions:

1. xxx

Main Success Scenario:

1. Xxx

Alternatives:

1. Xxx

* **Enable Rscript.exe execution in Windows Command Prompt**

Pre-Conditions:

1. On Windows 8
2. [Install R](#installR).

Main Success Scenario:

1. Search for “environment” in start menu.
2. Choose “Edit the system environment variables”
3. “System Properties” window appears.
4. Choose “Advanced” tab.
5. Click “Environment Variables” button.
6. Select “Path” in “System variables” section.
7. Copy path of Rscript.exe on your computer.

**E:\Program Files\R\R-3.2.0\bin\**

1. Click “Edit…” button below the “Environment Variables” window.
2. Add this new path to the existing paths.

**; E:\Program Files\R\R-3.2.0\bin\**

1. Click OK button and then the other OK button.

* **ETL for Cube source**

Pre-Conditions:

1. xxx

Main Success Scenario:

1. Truncate destination table

xxx

1. Pull data from source

xxx

1. Insert data into destination table

xxx

1. Transform data

xxx

1. Update data in other table

xxx

Alternatives:

1. Xxx

* **ETL for relational DB source**

Pre-Conditions:

1. xxx

Main Success Scenario:

1. Truncate destination table

xxx

1. Pull data from source

xxx

1. Insert data into destination table

xxx

1. Transform data

xxx

1. Update data in other table

xxx

Alternatives:

1. Xxx

* **Execute R script from Windows Command Prompt**

Pre-Conditions:

1. [Enable Rscript.exe execution in Windows Command Prompt](#enableRscriptExecutionInWin).

Main Success Scenario:

1. Run my R code in Windows Command prompt

Rscript.exe d:\r\_code\mycode.r

* **Find Position of Matching Number in Series**

Main Success Scenario:

1. Find position in series where element equals 10. Returns 7.

**x <- c(4:20)**

**which(x == 10)**

* **Find Metrics born in Cube Script**

Pre-Conditions:

1. xxx

Main Success Scenario:

1. Xxx

Alternatives:

1. Xxx

* **Find Cube Metrics in Excel**

Pre-Conditions:

1. xxx

Main Success Scenario:

1. Xxx

Alternatives:

1. Xxx

* **Find Method**

Main Success Scenario:

1. Find all plot methods.

Methods(plots)

* **Find a package if it is installed**

Main Success Scenario:

1. For example, see if “devtools” package is installed.

**find.package(“devtools”)**

Alternatives:

1. Xxx

* **Get Arguments for Function**

Main Success Scenario:

1. Call args functions with name of function.

**args(“rnorm”)**

Alternatives:

1. Xxx

* **Get Cube Connection**

Pre-Conditions:

1. [Load Non-default Referenced Package](#loadNonDefaultReferencedPackage)

Main Success Scenario:

1. Get Connection to “AnO Reporting” cube on “AnO Reporting Cube FY16” DB on adcrm\_SQL05 server.

**cn <- odbcDriverConnect(**

**connection="**

**Data Source=adcrm\_SQL05;**

**Initial Catalog=AnO Reporting;**

**Provider=MSOLAP.5;**

**Integrated Security=SSPI;**

**Format=Tabular;”**

**)**

1. Specify path for MSOLAP.5 provider driver

Search for “Msolap110.dll” on a computer that has analysis services installed.

* **Get Database Connection**

Pre-Conditions:

1. [Load Non-default Referenced Package](#loadNonDefaultReferencedPackage)

Main Success Scenario:

1. Get Connection to CnO\_BI\_PowerPivot\_Dev\_Geo DB on adcrmsql server.

**cn <- odbcDriverConnect(**

**connection="**

**Driver={SQL Server Native Client 11.0};**

**server=adcrmsql;**

**database=CnO\_BI\_PowerPivot\_FY16;**

**trusted\_connection=yes;"**

**)**

* **Get Database Connection with specified credentials**

Pre-Conditions:

1. [Load Non-default Referenced Package](#loadNonDefaultReferencedPackage)

Main Success Scenario:

1. Get Connection to CnO\_BI\_PowerPivot\_Dev\_Geo DB on adcrmsql server.

**cn <- odbcDriverConnect(**

**connection="**

**Driver={SQL Server Native Client 11.0};**

**server=adcrmsql;**

**database=CnO\_BI\_PowerPivot\_FY16;**

**trusted\_connection=yes;**

**xxx;"**

**)**

* **Get MongoDB Database Connection**

Note:

1. Use RMongoDB instead of RMongo. See [here](http://www.joyofdata.de/blog/mongodb-state-of-the-r-rmongodb/).

Pre-Conditions:

1. Install rmongodb package

**install.packages(“rmongodb”);**

1. Load rmongodb package

**library(rmongodb);**

Main Success Scenario:

1. Get DB name.

**dbName <- “testdatabase”**

1. Create Connection

**mongo <- mongo.create(**

**host = “dharma.compse.io:10200”,**

**db = dbName,**

**username = “myuser”,**

**password = “mypassword”**

**);**

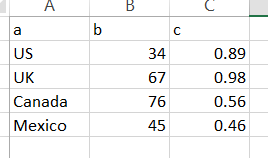
* **Get Data from CSV file**

Note:

1. Data is loaded into data frame.
2. Read.table can have errors that read.csv does not.
3. Read.csv automatically skips blank lines.

Pre-conditions:

1. abc.csv file has the format of.



1. For step 5,6

**install.packages("sqldf")**

**library(sqldf)**

Main Success Scenario:

1. Pull data from abc.csv file (that has column titles) on my C drive.

**MyData <- read.csv(file="c:/abc.csv", header=TRUE, sep=",")**

1. Pull data from abc.csv file, but only keep columns b and c.

**MyData <- read.csv(file="c:/abc.csv", header=TRUE, sep=",")[,2:3]**

1. Pull data from abc.csv file, but only keep columns a and c.

**MyData <- read.csv(file="c:/abc.csv", header=TRUE, sep=",")[,c(‘a’,’c’)]**

1. Pull data from abc.csv file on my C drive & strip leading and trailing white space on string fields.

**MyData <- read.csv(file="c:/abc.csv", header=TRUE, sep=",",strip.white=TRUE)**

1. Pull data from abc.csv file for only values of 34 in column b.

**MyData <- read.csv.sql("c:/abc.csv",sql = "select \* from file where b = 34",eol = "\n")**

1. Pull only columns a and c from abc.csv file.

**MyData <- read.csv.sql("c:/abc.csv",sql = "select a,c from file",eol = "\n")**

1. Pull data from abc.csv file for only values like UK in column a.

**MyData <- read.csv.sql("c:/abc.csv",sql = "select \* from file where a like '%UK%'",eol = "\n")**

* **Get Data Summary**

Main Success Scenario:Were

1. Get summary of numbers in dataFetch dataframe.

**summary(dataFetch)**

* **Get Working Directory**

Main Success Scenario:

1. Set my desktop as working directory

**getwd()**

* **Graph Frequency of data point in DataFrame**

Main Success Scenario:

1. hist([column in data frame])

*hist(outcom[,11])*

* **Graph y = data point & x = data point position**

Main Success Scenario:

1. plot([set of numbers])

*plot(rnorm(10,0,20))*

* **Show Bar Plot Chart**

Pre-Conditions:

1. xxx

Main Success Scenario:

1. Xxx

Alternatives:

1. Xxx

* **Show Histogram Chart**

Pre-Conditions:

1. [Get Database Connection](#getDatabaseConnection)
2. Get two sets of data.

**dataQuery <- sqlQuery(cn,"Select CustomerID, datepart(day,TransitionDate) As day from StgCustomerTransitionDate");**

1. Close database connection

**close(cn);**

Main Success Scenario:

1. Get days in month for all records

dayInMonth <- dataQuery$day

1. Create histogram chart

hist(dayInMonth,right=FALSE)

* **Show Heatmap Chart**

Pre-Conditions:

1. Use image plot to do this

Main Success Scenario:

1. Xxx

Alternatives:

1. Xxx

* **Show Lattice Plot Chart**

Pre-Conditions:

1. xxx

Main Success Scenario:

1. Xxx

Alternatives:

1. Xxx

* **Show MatPlot Chart**

Pre-Conditions:

1. Data comes from a matrix or two vectors.

Main Success Scenario:

1. Xxx

* **Show Scatter Plot Chart**

Pre-Conditions:

1. xxx

Main Success Scenario:

1. Xxx

Alternatives:

1. Xxx

* **Insert Data into Relational DB table**

Note:

1. Default for “safer” parameter is “TRUE”.
2. Default for “rownames” parameter is “TRUE”.

Main Success Scenario:

1. [Get Database Connection](#getDatabaseConnection).
2. Query data

**dataQuery <- sqlQuery(cn,"Select CustomerID,Country from StgCustomerGeography");**

1. Insert data into DB table named the same as the data frame variable.

**sqlSave(cn, dataQuery);**

1. Insert data into DB table that is named with “tablename” parameter.

**sqlSave(cn, dataQuery,tablename=“CustGeoMap”);**

1. Delete all existing rows in a table and insert new rows.

**sqlSave(cn, dataQuery,safer=FALSE);**

1. Create table named “dataQuery” if it does not already exist.

**sqlSave(cn, dataQuery,safer=TRUE);**

1. Append data to existing table

**sqlSave(cn, dataQuery,append=TRUE);**

1. Insert data into DB table without the “rownames” column.

**sqlSave(cn, dataQuery,rownames=FALSE);**

1. Close database connection

**close(cn);**

* **Install R**

Pre-Conditions:

1. Free
2. Platform independent
3. Open Source

Main Success Scenario:

1. Install R from <https://www.r-project.org/>.

* **Install Referenced Package**

Pre-Conditions:

1. Package to install is RODBC.
2. Use “dependencies=TRUE” parameter if the package is dependent on others that need to be installed.

Main Success Scenario:

1. [Check R version.](#checkRVersion)
2. Update packages R version is not at least 3.1.3
3. Click Packages menu > Updates Packages …
4. Run install.packages(“RODBC”)

* **Install mailR Package**

Note

1. mailR is for sending authenticated emails.
2. Can add attachments.
3. Can format body with HTML.

Pre-Conditions:

1. Install rJava Package.
2. Max of 20 mails per day can be sent.

Main Success Scenario:

1. Install package in R

**install.packages(“mailR”)**

1. Check if library can be loaded.

**library(mailR)**

* **Install Multiple Packages in R**

Main Success Scenario:

1. Install packages in R

**install.packages(c(“slidify”,”ggplot2”,”devtools”))**

* **Install rJava Package**

Note

1. [rJava](https://www.rforge.net/rJava/) is a simple R to Java interface.

Pre-Conditions:

1. [Install 64 bit Java](http://www.java.com/en/download/manual.jsp) if I have installed 64 bit R.
2. Set JAVA\_HOME environment variable in Windows OS.

Main Success Scenario:

1. Install package in R

**install.packages(“rJava”)**

1. Set JAVA\_HOME environment variable in R Studio.

**Sys.setenv(JAVA\_HOME="E:\\Program Files\\Java\\jre1.8.0\_60")**

1. Check if library can be loaded.

**library(rJava)**

* **Install Rtools**

Pre-Conditions:

1. Needed for building R packages in Windows.

Main Success Scenario:

1. Download from <http://cran.r-project.org/bin/windows/Rtools/>
2. Confirm version corresponds to your version of R.
3. Use default selects during install.
4. Check box to let installer edit the PATH variable.
5. Install “devtools” package

**Install.packages(“devtools”)**

1. Load “devtools” package

**Library(devtools)**

1. Run find\_rtools(). True indicates Rtools installed correctly.

Alternatives:

1. Xxx

* **Install X4R Package**

Notes:

* + Used help from [here](http://www.scriptscoop.com/t/3bc447b530b6/how-do-i-install-x4r-package-on-r.html) and [here](https://github.com/overcoil/X4R).

Pre-Conditions:

* + Get GitHub account.
  + Install GitHub Desktop from [here](https://desktop.github.com/).
  + Make sure R is installed in “C:\R” instead of “C:\Program Files”

Main Success Scenario:

1. [Check R version.](#checkRVersion)
2. Update packages R version is not at least 3.1.1.
3. Install.packages(“devtools”)
4. library(devtools)
5. install.packages(“httr”)
6. Run Library(httr) in R studio.
7. Download rtools from [here](https://cran.r-project.org/bin/windows/Rtools/) and install (& update path environment variable).
8. Run install.packages(“Rcpp”) in R studio.
9. Run library(Rcpp) in R studio.
10. Add R\_LIBS\_USER environment variable and set to “E:\R\R-3.2.0\library”
11. install\_github(“overcoil/X4R”)
12. library(X4R)

* **Iterate with For Loop**

Notes:

1. xxx

Pre-Conditions:

1. use lapply function instead if needing to call a function several times.
2. Cannot use faster, better performing iteration tools.

Main Success Scenario:

1. Concatenate vector members into a string. Where x is a vector with values from 1 to 10.

**to.string <- function(x){**

**string <- x[1]**

**for(i in 2:length(x)){**

**string <- paste(string,x[i],sep=””)**

**}**

**return(string)**

**}**

1. Another way of doing the same thing

**to.string(1:10)**

1. Yet another way of doing the same thing

**paste(1:10,collapse=””)**

* **Iterate with lapply**

Notes:

1. Applies a function to each element of a list or vector.
2. Lapply collection results in a list.
3. First parameter is a list or vector.
4. Second parameter is the function to call.

Pre-Conditions:

1. use lapply function instead if needing to call a function several times.

Main Success Scenario:

1. Daisy chaining calls to lapply

**first.step <- lapply(X, first.function)**

**second.step <- lapply(first.step, next.function)**

1. xxx

* **List Files for current directory for R console**

Main Success Scenario:

1. list.files()

* **List Functions in Package**

Main Success Scenario:

1. Open R console
2. Create a function to list all functions in RODBC package

**lsp <- function(package, all.names = FALSE, pattern)**

**{**

**package <- deparse(substitute(package))**

**ls(**

**pos = paste("package", package, sep = ":"),**

**all.names = all.names,**

**pattern = pattern**

**)**

**}**

**lsp(RODBC)**

* **Load non-default Referenced Package**

Pre-Conditions:

1. [Install Referenced Package](#installReferencedPackage) is already run.
2. Referenced package is RODBC.

Main Success Scenario:

1. library(RODBC)

Alternatives:

1. Want to use console menu instead.

1a. Package menu > Load Package …

1b. Choose package from list.

1c. Click OK button.

* **Load X4R Package**

Pre-Conditions:

1. [Install Referenced Package](#installReferencedPackage) is already run.
2. Referenced package is RODBC.

Main Success Scenario:

1. library(RODBC)

Alternatives:

1. Want to use console menu instead.

1a. Package menu > Load Package …

1b. Choose package from list.

1c. Click OK button.

* **Make .R file accessible by console**

Pre-Conditions:

1. Use back slashes
2. Escape back slash with another backslash

Main Success Scenario:

1. Use source function

**source("E:\\ScottFiles\\work\\workspaceforR\\myCode\\ano.R")**

* **Merge Two Data Frames**

Pre-Conditions:

1. [Get Database Connection](#getDatabaseConnection).
2. Get two sets of data.

* **dataQuery <- sqlQuery(cn,"Select CustomerID,Country from StgCustomerGeography");**
* **geoQuery <- sqlQuery(cn,"Select Country,WWSubRegionName from vDimGeography");**

Main Success Scenario:

1. Full Outer Join

**merge(dataQuery,geoQuery,by.x = "Country",by.y = "Country", all = TRUE)**

1. Left Outer Join (only return data that exists in the dataQuery data frame).

**merge(dataQuery,geoQuery,by.x = "Country",by.y = "Country", all.x = TRUE)**

1. Right Outer Join (only return data that exists in the geoQuery data frame).

**merge(dataQuery,geoQuery,by.x = "Country",by.y = "Country", all.y = TRUE)**

1. Inner Join (only return data that exists in both frames)

**merge(dataQuery,geoQuery,by.x = "Country",by.y = "Country", all = FALSE)**

* **Merge Two Data Frames via two columns**

Pre-Conditions:

1. [Get Database Connection](#getDatabaseConnection).
2. Get two sets of data.
3. “x” is the left data frame and “y” the right data frame.
4. **dataQuery <- sqlQuery(cn,"Select CustomerID,Country from StgCustomerGeography");**
5. **geoQuery <- sqlQuery(cn,"Select CustomerID,Country,WWSubRegionName from vDimGeography");**

Main Success Scenario:

1. Full Outer Join

**merge(dataQuery,geoQuery,by.x = c(“CustomerID”,“Country"),by.y = c(“CustomerID”,“Country"), all = TRUE)**

1. Left Outer Join (only return data that exists in the dataQuery data frame).

**merge(dataQuery,geoQuery,by.x = c(“CustomerID”,“Country"),by.y = c(“CustomerID”,“Country"), all.x = TRUE)**

1. Right Outer Join (only return data that exists in the geoQuery data frame).

**merge(dataQuery,geoQuery,by.x = c(“CustomerID”,“Country"),by.y = c(“CustomerID”,“Country"), all.y = TRUE)**

1. Inner Join (only return data that exists in both frames)

**merge(dataQuery,geoQuery,by.x = c(“CustomerID”,“Country"),by.y = c(“CustomerID”,“Country"), all = FALSE)**

* **Names of DataFrame Columns**

Main Success Scenario:

1. names([data frame name])

* **Number of columns in DataFrame**

Main Success Scenario:

1. ncol([data frame name])

* **Number of rows in DataFrame**

Main Success Scenario:

1. nrow([data frame name])

* **Operations on Time**

Pre-Conditions:

1. “myDate1” and “myDate2” variables are Date objects.
2. Date in “myDate2” comes after date in “myDate1”.

Main Success Scenario:

1. Get UTC milliseconds since Jan 1st 1970.

**time1 <- myDate1.getTimeUTC()**

**time2 <- myDate2.getTimeUTC()**

1. Get number of milliseconds between two times.

**msDiff <- time2 – time1**

1. Get number of hours between two times.

**hDiff <- ((((time2 – time1)/1000)/60)/60)**

* **Plot value distribution with Box Plot**

Pre-Conditions:

1. [Get Database Connection](#getDatabaseConnection).
2. Get data

**dataQuery <- sqlQuery(cn,"Select Country,Count(CustomerID) As Cnt from StgCustomerGeography Group By Country");**

Main Success Scenario:

1. See [here](http://www.statmethods.net/graphs/boxplot.html).

* **Plot values on y-axis & labels on x-axis**

Pre-Conditions:

1. [Get Database Connection](#getDatabaseConnection).
2. Get data

**dataQuery <- sqlQuery(cn,"Select Country,Count(CustomerID) As Cnt from StgCustomerGeography Group By Country");**

Main Success Scenario:

1. Attach data frame to be searchable by R.

**Attach(dataQuery)**

1. Plot data with Country labels on x-axis on customer count on y-axis.

**Plot(Country,Cnt)**

1. Add title to graph.

**Title(“Customer Cnt per Country”)**

* **Plot values on y & z and labels on x**

Pre-Conditions:

1. [Get Database Connection](#getDatabaseConnection).
2. Get data

**dataQuery <- sqlQuery(cn,"**

**Select**

**s.ScenarioName,**

**Count(f.AccountID) As AccountIDCnt,**

**Count(f.GeoID) As GeoIDCnt**

**From**

**FactScorecard f**

**Inner Join DimScenario s On**

**f.ScenarioId = s.ScenarioId**

**Group By**

**s.ScenarioName**

**");**

Main Success Scenario:

1. Install ScatterPlot3d package.

**install.packages(“scatterplot3d”,dependencies=TRUE)**

1. Load package

**library(scatterplot3d)**

1. Attach data frame to be searchable by R.

**attach(dataQuery)**

1. Plot ScenarioName on x, AccountIDCnt on y, and GeoIDCnt on z.

**scatterplot3d(x=dataQuery$ScenarioName,y=dataQuery$AccountIDCnt,z=dataQuery$GeoIDCnt)**

1. Add title to graph.

**title(“Scenario-AccountIDCnt-GeoIDCnt”)**

* **Pull SQL table/view with row rate**

Main Success Scenario:

1. [Get Database Connection](#getDatabaseConnection)
2. Pull from the SalesDate table 1000 rows at a time.

Xxx

1. Close database connection

**close(cn);**

* **Query Cube**

Pre-Conditions:

1. [Install X4R package](#installX4R). Not the usual package install.
2. [Query Cube – Enable HTTP SSAS access](#queryCubeEnableHTTP)

Main Success Scenario:

1. Load X4R package

**Library(X4R)**

1. Connect to SSAS

**handle <- xmlaConnect(**

**url="http://10.197.206.11/olap/msmdpump.dll",**

**uid="northamerica\v-scburn",**

**pwd="\*\*\*"**

**);**

1. Check if connected. Will return something other than false if connected.

handle

1. Xxx

* **Query Cube – Enable HTTP SSAS access**

Notes:

1. [Must enable HTTP access to SSAS instance](https://msdn.microsoft.com/en-us/library/gg492140.aspx). [Here](https://msdn.microsoft.com/en-us/library/gg492140%28v=sql.105%29.aspx) for SQL Server 2008 R2.

Pre-Conditions:

1. Done on web server that is different from the SSAS server.
2. Step 8 & 9 are not on the page referenced in the Notes section.
3. URL in connection needs to be the IP address and not the server name.

Main Success Scenario:

1. Create “110Consulting6\C$\inetpub\wwwroot\olap” folder
2. Copy contents of “\\ADCRM\_SQL05\D$\Program Files\Microsoft SQL Server\MSAS10\_50.MSSQLSERVER\OLAP\bin\isapi\” folder to “olap” folder.
3. Create “olap” application pool on 110Consulting6 server.
4. Create “olap” virtual directory on 110Consulting6 server.
5. Configure IIS authentication on 110Consulting6 server.
6. Add IIS Extension on 110Consulting6 server.
7. Edit MSMDPUMP.INI file on 110Consulting6 server.
8. On olap virtual directory > Basic Settings set “Connect As” to your credentials
9. On olap virtual directory > Directory Browsing > Enable
10. Test access with an excel file. Enter HTTP path in the server field for SSAS connection.

**http://10.177.148.230/olap/msmdpump.dll**

* **Query SQL Server**

Main Success Scenario:

1. [Get Database Connection](#getDatabaseConnection)
2. Get values for Country field in vDimGeography view.

**dataQuery <- sqlQuery(cn,”Select Country from vDimGeography”)**

1. Close database connection

**close(cn);**

* **Rank a column**

Pre-Conditions:

1. [Get Database Connection](#getDatabaseConnection)
2. Get data

**dataQuery <- sqlQuery(cn,"Select CustomerID, datepart(day,TransitionDate) As day from StgCustomerTransitionDate");**

Main Success Scenario:

1. Do a rank on day column.

**dataQuery$dayRank <- rank(dataQuery$day,na.last=TRUE,ties.method = "max")**

1. Do a row number rank on day column

**dataQuery$dayRank <- rank(dataQuery$day,na.last=TRUE,ties.method = "first")**

* **Reference Relative Folder**

Pre-Conditions:

1. xxx

Main Success Scenario:

1. Xxx

Alternatives:

1. Xxx

* **Remove Data Frame Column**

Pre-Conditions:

1. Data frame called “df” and has a column named “x”.

Main Success Scenario:

1. df$x <- NULL

* **Repeat an object**

Main Success Scenario:

1. Repeat a vector three times

**rep(c(3,4,5),3)**

1. Repeat the series of 1 to 10 three times

**rep(1:10,times = 3)**

1. Repeat each element in a vector 3 times

**x <- c(1,2,3)**

**rep(x,each = 3)**

1. Repeat each element in a vector and each vector three times

**rep(x,each = 3,times = 3)**

* **Return Top of DataFrame**

Main Success Scenario:

1. head([data frame name])

* **RmongoDB: Check if Connected**

Pre-Conditions:

1. “cn” is a variable that holds the connection.

Main Success Scenario:

1. [Get MongoDB Connection](#getMongoDBDatabaseConnection)
2. Check if connected

**mongo.is.connected(cn);**

* **RmongoDB: Close Connection**

Pre-Conditions:

1. “cn” is a variable that holds the connection.

Main Success Scenario:

1. [Get MongoDB Connection](#getMongoDBDatabaseConnection)
2. Close connection.

**mongo.destroy(cn);**

* **RmongoDB: Count Documents in Collection**

Pre-Conditions:

1. “cn” is a variable that holds the connection.

Main Success Scenario:

1. [Get MongoDB Connection](#getMongoDBDatabaseConnection)
2. Store collection name in variable.

**coll <- “myColl”;**

1. Count documents in the collection.

**mongo.count(cn,coll);**

* **RmongoDB: Count Documents in Collection that match filter**

Pre-Conditions:

1. “cn” is a variable that holds the connection.

Main Success Scenario:

1. [Get MongoDB Connection](#getMongoDBDatabaseConnection)
2. Store collection name in variable.

**coll <- “myColl”;**

1. Count documents in the collection.

**mongo.count(cn,coll,”{‘pop’:{‘$lte’:2}}”);**

* **RmongoDB: Find One Document**

Note:

1. MongoDB stores JSON objects as BSON.
2. BSON is Binary JSON.
3. Cityone variable contains a BSON object.

Pre-Conditions:

1. “cn” is a variable that holds the connection.
2. BSON object could be used instead of JSON in the “one” function.

Main Success Scenario:

1. [Get MongoDB Connection](#getMongoDBDatabaseConnection)
2. Store collection name in variable.

**coll <- “myColl”;**

1. Show document where city is Welton.

**Cityone <- mongo.find.one(cn,coll,”{‘city’:’WELTON’}”)**

**cityone**

1. Show returned doc attributes in list.

**mongo.bson.to.list(cityone)**

* **RmongoDB: Find All Document that match filter**

Pre-Conditions:

1. “cn” is a variable that holds the connection.
2. ~~BSON object could be used instead of JSON in the “one” function.~~

Main Success Scenario:

1. [Get MongoDB Connection](#getMongoDBDatabaseConnection)
2. Store collection name in variable.

**coll <- “myColl”;**

1. Get documents where pop is less than or equal to 2.

**pops <- mongo.find.all(cn,coll,”{‘pop’:{‘$lte’:2}}”)**

1. Show returned doc attributes in list.

**mongo.bson.to.list(pops)**

* **RmongoDB: Insert data into Collection**

Pre-Conditions:

1. “db” is a variable that holds the db.
2. “cn” is a variable that holds the connection.

Main Success Scenario:

1. [Get MongoDB Connection](#getMongoDBDatabaseConnection)
2. Create empty collection called “test”.

**icoll <- paste(db,”test”,sep=”.”)**

1. Create docs to go into new collection.

**a <- mongo.bson.from.JSON(“{‘name’:’Marcus’,’age’:33}”)**

**b <- mongo.bson.from.JSON(“{‘name’:’Jim’,’age’:35}”)**

1. Insert data into MongoDB.

**Mongo.insert.batch(cn,icoll,list(a,b))**

* **RmongoDB: Show Collections in DB**

Pre-Conditions:

1. “cn” is a variable that holds the connection.

Main Success Scenario:

1. [Get MongoDB Connection](#getMongoDBDatabaseConnection)
2. Store DB name in a variable.

**dbName <- “myDB”;**

1. Show databases

**mongo.get.database.collections(cn,dbName);**

* **RmongoDB: Show Distinct Values used in Collection**

Pre-Conditions:

1. “cn” is a variable that holds the connection.

Main Success Scenario:

1. [Get MongoDB Connection](#getMongoDBDatabaseConnection)
2. Store collection name in variable.

**coll <- “myColl”;**

1. Store name space in variable.

**ns <- paste(dbName,”.”,coll);**

1. Show distinct list of values for “city” key.

**mongo.distinct(cn,ns,”city”);**

* **RmongoDB: Show Keys used in Collection**

Pre-Conditions:

1. “cn” is a variable that holds the connection.

Main Success Scenario:

1. [Get MongoDB Connection](#getMongoDBDatabaseConnection)
2. Store collection name in variable.

**coll <- “myColl”;**

1. Show distinct list of keys in collection.

**mongo.get.keys(cn,coll);**

* **RmongoDB: Show DBs**

Pre-Conditions:

1. “cn” is a variable that holds the connection.

Main Success Scenario:

1. [Get MongoDB Connection](#getMongoDBDatabaseConnection)
2. Show databases

**mongo.get.databases(cn);**

1. List of databases for the connection are shown.

Alternatives:

3a. NULL is returned.

3a1. When connecting to a specific database only NULL is returned.

* **RmongoDB: Use Pretty function like in MongoDB**

Main Success Scenario:

1. Load JSON lite library

**Library(jsonlite)**

1. Get JSON document

**jsonDoc <- “{‘pop’:{‘$lite’:2},’pop’:{‘$gte’:1}}”**

1. Show pretty formatted JSON doc

**Cat(prettify(jsonDoc))**

* **RmongoDB: Update Document**

Pre-Conditions:

1. “cn” is a variable that holds the connection.
2. “ns” is a variable that holds the name space name.
3. Name space name is in the form of [DB name].[name space name]
4. “criteria” is a BSON object variable that holds the filter criteria.
5. “objNew” is a new BSON object that holds a key value pair to be added to a doc.

Main Success Scenario:

1. [Get MongoDB Connection](#getMongoDBDatabaseConnection)
2. Update existing record.

**Mongo.update(mongo,ns,criteria,objNew)**

1. Update existing record and add new doc if it does not already exist.

**Mongo.update(mongo,ns,criteria,objNew,mongo.update.upsert)**

* **RmongoDB: Validate JSON Document**

Main Success Scenario:

1. Load JSON lite library

**Library(jsonlite)**

1. Get JSON document

**jsonDoc <- “{‘pop’:{‘$lite’:2},’pop’:{‘$gte’:1}}”**

1. Show pretty formatted JSON doc

**validate(jsonDoc)**

* **Run R Profiler**

Pre-Conditions:

1. Don’t run system.time() and Rprof() at the same time.
2. Rprof() output is hard to read. SummaryRprof() is preferred.
3. Profiler gives percent of time spent in each function.
4. Breaking code into functions is to your benefit with Profiler.
5. By.self is time spent in parent function and not child functions.

Main Success Scenario:

1. Use by.total (= time spend in each function / total time).
2. Use by.self (= same as by.total, but subtracts time spent in functions above in call stack).
3. Use sampling.time to get the full run time of the R code.
4. Use sample.interval from Rprof() to get function call hierarchy.

* **Search for Values in DataFrame**

Main Success Scenario:

1. Find Texas in State field in data frame called “outcome”.

**“TX” %in% outcome[,7]**

* **Search for Values in SQL Server**

Pre-Conditions:

1. xxx

Main Success Scenario:

1. Xxx

Alternatives:

1. Xxx

* **Select Columns to return from a Data Frame**

Pre-Conditions:

1. “dataQuery” is a data frame variable
2. ”AcquisitionSortMonth” and ”Signups” are column names in the dataQuery data frame.

Main Success Scenario:

1. dataQuery[,c(”AcquisitionSortMonth”,”Signups”)];

* **Select Into Query with R**

Main Success Scenario:

1. Run Select Into query to save to myRSQLTest table in DB

**sqlSave(**

**cn,**

**dataFetch,**

**rownames=FALSE,**

**tablename="myRSQLTest",**

**colname=FALSE,**

**append=FALSE**

**)**

* **Set Working Directory**

Main Success Scenario:

1. Set my desktop as working directory

**Setwd(“//Users//v-sburn/desktop”)**

* **Show Function Help**

Main Success Scenario:

1. Show help for dbReadTable function

**?dbReadTable**

* **Show Package Help**

Main Success Scenario:

1. Show help for RODBC package

**?RODBC**

* **Start R Console**

Main Success Scenario:

1. Click on “R x64 3.2.0” in start menu

* **Update R language version**

Note:

1. Doing “R console > Packages menu > Update packages” just updates packages.

Main Success Scenario:

1. Run in R console.

if(!require(installr)) {

install.packages("installr"); require(installr)}

updateR()

* **Use R in SSIS**

Pre-Conditions:

1. Use VS 2013

Main Success Scenario:

1. Use “Integration Services Project” project in VS.
2. Add “Execute Process Task” task to DTSX file.
3. Navigate to “mongo.exe”

C:\Program Files\MongoDB\Server\3.0\bin\

1. xxx

* **View data frame data in memory**

Main Success Scenario:

1. Get data in a data frame called dataFetch
2. View data in data frame

**View(dataFetch)**

**More information:**

* Markus Schmidberger authored the RMongoDB package.
* Dmitriy Selivanov maintains the RMongoDB package.
* [CRAN project location of RMongoDB package](https://cran.r-project.org/web/packages/rmongodb/index.html).
* [Help for RMongoDB package](https://cran.r-project.org/web/packages/rmongodb/rmongodb.pdf).