

Introduction to R

Monday, August 22, 2016 7:05 PM

Understanding Analytics

Understanding Analytics

Data Analysis Vs. Reporting

Reporting	Analysis
Provides data	Provides answers
Provides what is asked for	Provides what is needed
Is typically standardised	Is typically customised
Does not involve a person	Involves a person leading the process
Is fairly inflexible	Is extremely flexible

Understanding Analytics

Data Analysis Vs. Reporting

Interplay between reporting and analysis is common and necessary.

For Example :

Report : A company generates basic sales summary report showing monthly sales by region.

Analysis : You observe what is the trend in sales over a period of time and you find out that Sales peaks on specific holidays or weekends

© Jigsaw Academy Education Pvt Ltd

Understanding Analytics

Features of basic and advanced analytics

Basic Analytics

Not sure what you have? But you think something is of value ?

- Slicing and Dicing of the data
- Monitor large volumes of data in real time
- Anomaly Identification

Advanced Analytics

What happened, what caused it to happen, when it happened, and what the impact was?

- Predictive Modelling
- Text Analytics
- Other Statistical and Data-Mining Algorithms

Understanding Analytics

Pre-requisites - Analytics Process



Analytical Tools

Why do we need specialized analytical tools?

Depends on volume of data and complexity of analysis

For example, maximum number of rows of data allowed in Excel
– 1 MM records

- Advanced data mining algorithms are not implemented in Excel
- Specialized data mining, predictive analytics, and visualization tools have been built to effectively analyse large volumes of data

Analytical Tools

Multiple types of Analytical Tools:

- **GUI based tools:** Excel, SPSS, SAS ,RStudio
- **Visualization tools:** Tableau, MicroStrategy
- **Coding based tools:** SAS, R

Analytical Approaches and Tools

Categories of Analytical Tools

Point Solutions

- Analytic point solutions are software packages that address a very specific, narrow set of problems(business) and they often sit on top of analytical tool suites.
- **Limitations :** Point solutions are expensive

Examples : Price optimization applications, fraud applications, and demand forecasting Applications,etc

Analytical Approaches and Tools

Categories of Analytical Tools

Data Visualization Tools

- In the world of analytics, visualization refers to charts, graphs, and tables that display data and help in data analysis

Example : In a retail store ,find out which parts of the store are most frequently visited by the customers.

- **Option 1 :** Create a bunch of spreadsheets, lay them out on a table, and try to figure out the patterns.
- **Option 2 :**Produce a map of the store floor plan where colour represents the level of activity.

Analytical Approaches and Tools

Popular Analytical Tools

R Project for Statistical Computing

Features:

- Open Source Software
- Object Oriented Design.
- Can be linked with common programming platforms like C++ and Java, which makes it possible to embed R within applications.
- Commercial analytic tools like SAS,etc have even enabled R to be executed within their toolsets.
- Extensibility : Since R codes can be easily written and packaged, its easy to create and distribute them as a package

Analytical Approaches and Tools

Popular Analytical Tools

R Project for Statistical Computing

Disadvantages:

- **Scalability** : Since the base R software runs in memory ,it can only handle datasets of the size of memory available on a machine.

Analytical Approaches and Tools

Popular Analytical Tools

IBM SPSS

Syntax Language + graphic interface of scrolling menus

Important Features :

- Commands are executed line by line to update tables or add results to the Output Editor window
- Statistics can read from and write to ASCII files, databases and tables of other statistical software
- Provides basic data management functions, such as sorting, aggregation, transposition, and table merge
- The file can be in various formats
- Can integrate R functions

Analytical Approaches and Tools

Popular Analytical Tools

SAS

A SAS program consists of **DATA steps**, **procedure steps**, and **macros**

It can be used for the following:

- Statistics
- Data and Text mining
- Data Visualization
- Forecasting
- Optimization
- Model Management and Deployment
- Quality Improvement

Analytical Approaches and Tools

Popular Analytical Tools

Comparisons

	SAS	SPSS	R
User Interface	Has the SAS Enterprise Guide	Graphic interface with scrolling menus	R Graphic interfaces like Rattle, Rstudio, etc
Decision Trees	Expensive, Purchase SAS Miner	Less Expensive, Does not require Data mining suite, offers a wide variety of algorithms	Free, Does not offer many tree algorithms
File Management and Stability	More flexible	Freezes when processing large volumes	Not suitable for large volumes of data
Data Management	No memory issues	No memory issues	Memory issues
Documentation	Extensive	Not so extensive	Extensive (CRAN)

The screenshot shows the RStudio interface. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Tools, Help, and a search bar. The left sidebar has tabs for Source on Stack Overflow, Run, and Source. The main area contains two panes: a Script pane with R code and an Environment pane showing global variables. The bottom pane is a file browser.

Script Pane (R Script 1):

```
20 letters
21 letters<-c("a","b","c")
22 letters
23 letters.factors<-as.factor(letters)
24
25 ls() #list all objects in R
26 rm(letters) #removes the object letter
27
28
29
30
```

Environment Pane:

Values	numbers	num [1:3]	1 2 3
x			2

File Browser:

- D - Data Manipulation with R
 - Name
 - Size
 - Modified
 - Rhistory
 - 304 B
 - Jun 13, 2014, 2:33 PM
 - Data Analysis in R.r
 - 871 B
 - Jun 12, 2014, 2:40 PM
 - Data Manipulation in R.r
 - 6.6 KB
 - Jun 13, 2014, 2:24 PM
 - Data Manipulation

The `ls()` command is used to list all the objects created in R so far
The `rm(object name)` is used to remove the object

Manipulating objects and saving and loading an object

Data Structure in R:

Manipulating and Data Processing in R

Data Structures in R

Vectors

- Most Simplest structure in R
- If data has only one dimension, like a set of digits, then vectors can be used to represent it.

Matrices

- Used when data is a higher dimensional array
- But contains only data of a single class Eg : only character or numeric

Manipulating and Data Processing in R

Data Structures in R

Data Frames

- It is like a single table with rows and columns of data
- Contains columns or lists of different data

Lists

- Used when data cannot be represented by data frames
- It contain all kinds of other objects, including other lists or data frames
- Very Flexible

Exploring a dataset in R:

To load a dataset , that is provided in R, you need to access the dataset library

- `library(datasets)`
- To list all the dataset in this library --> `data()`
- To load a dataset from the library to the current session --> `data(iris)`
- Once the data set is loaded, to explore the structure --> `str(iris)`
- To get a gist of top rows --> `head(iris)`
- To get a gist of bottom rows --> `tail(iris)`
- To know more about the preloaded dataset --> `?iris`
- To know about the class of the dataset --> `class(iris)`
- Attach and detach --> these functions are used to directly call the columns as objects. Here is how we use it :
 - `attach(iris)`
 - `Head(Species)`
- The enxt step is to look at importing a file in R
- `Test1 <- read.csv("file name", header=T, sep=",")`

- If we want to import a text file, we use the read.table command
 - `read.table("filename", header = T, sep=\t)`
 - `\t` --> this means tab separated
 - If we want to see a specific number of top rows, like 10 rows, then use --> `head(iris,10)`
- `Summary(iris0` --> this will give us the descriptive statistics of the column in that dataset
- The `is.function` is used to find out what is the type of column in a dataset
 - `is.character(dataset$column)` or `is.factor(dataset$column)`

Data Import / Export overview

AGENDA

- ➔ • Data import/export overview
 - Working with plain text files
 - Working with large text files
 - Working with excel workbooks
 - Working with statistical system outputs
 - Working with databases
 - Working with web data

© Jigsaw Academy Education Pvt Ltd



Data import/export overview

Need for Data import/export in R

```
> personId<-c(1,2,3,4,5)
> personwt<-c(60,70,80,65,75)

> person<-data.frame(personId,personwt)
> person
  personId personwt
1         1       60
2         2       70
3         3       80
4         4       65
5         5       75

> plot(person)
```

✓ Approach not practical
 ✓ Issues of scalability and typos

© Jigsaw Academy Education Pvt Ltd

Data import/export overview

R Benefits

- Data science projects start with a bunch of data
- Common way of data sharing is through external files
- Structured format necessary for application of R functions
- data.frame format provides spread sheet like look
- Presence of R adapters to read data from various formats
- Facilities available either in R itself or via packages

personId	personWt
1	68
2	78
3	98
4	65
5	75

Analyzing data in R after loading is easier with the help of data.frame structure which is ideal for working with structured data

© Jigsaw Academy Education Pvt Ltd



Data import/export overview

Importing Data in R

- Most of the file formats are supported by R
- Popular ones are delimited files like csv, tab etc.
- These files generally have an extension .txt
- More recent formats like xml, json etc.
- Excel files with extensions like .xls, .xlsx etc.
- Other proprietary formats related to SAS, SPSS
- Files from Relational Database Management Systems (SQL)
- Data retrieval from API's like Twitter API, Facebook API etc.



To deal with different file formats, often data can be first exported as a text file like csv and then imported into R

© Jigsaw Academy Education Pvt Ltd



Data import/export overview

Checks before Data Import

When reading data from text files, it is necessary to know and understand more about the inherent characteristics

- Presence of header line
- Kind of value separator
- Representation of missing values
- Notation of comment characters or quotes
- Existence of any unfilled or blank lines
- Classes of the variables

© Jigsaw Academy Education Pvt Ltd



Data import/export overview

What will be covered?

Data Science projects involve dealing with data from various sources based on business context

- ✓ plain text files
- ✓ excel spreadsheets
- ✓ xml & json documents
- ✓ RDMS and NoSQL databases
- ✓ Other software file outputs
- ✓ API data extraction

And further, we will also look at exporting data from R into various formats

© Jigsaw Academy Education Pvt Ltd



Working with Plain text files:

Data from plain text files

Delimiter Formats

A delimiter is a sequence of one or more characters used to specify the boundary between independent regions in plain text streams

- Plain text files use delimiter-separated values
- To store two-dimensional arrays of data
- Separate values in each row with specific delimiter
- Any character can be used to separate the values
- Common delimiter formats
- ✓ comma, tab, colon, pipe (or vertical bar), space

Delimiter format files are supported by most database and spreadsheet programs

© Jigsaw Academy Education Pvt Ltd



Data from plain text files

Import functions in R

R has functions for reading in the standard types of plain text formats especially delimited formats

Different methods of reading data in R:

- ✓ **scan**: reads vectors of data which all have the same mode
- ✓ **read.table**: create data frames with different mode columns (variables)
- ✓ **read.delim**: reads delimited files into data frame objects
- ✓ **read.csv**: reads csv files into data frame objects
- ✓ **readLines**: reads the input files line by line

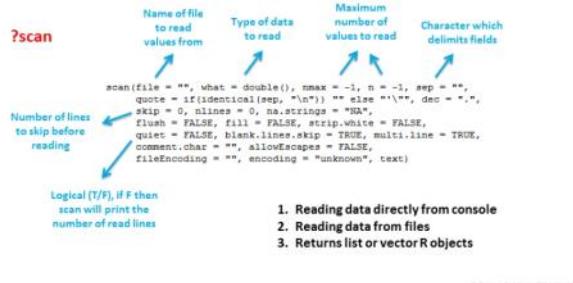
© Jigsaw Academy Education Pvt Ltd



Data from plain text files

scan function

The scan function reads the fields of data in the file as specified by the what option, with the default being numeric

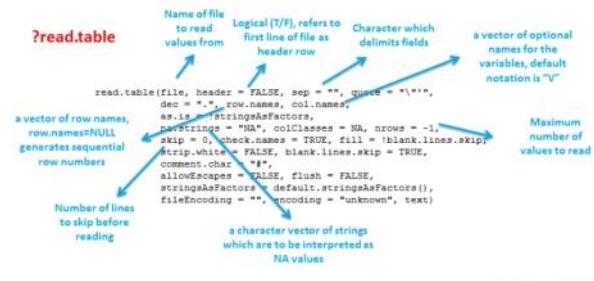


© Jigsaw Academy Education Pvt Ltd

Data from plain text files

read.table function

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables in the file

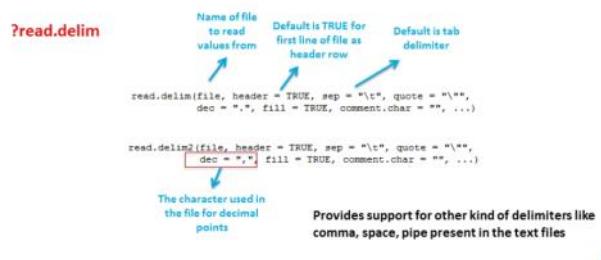


© Jigsaw Academy Education Pvt Ltd

Data from plain text files

read.delim function

Simply a wrapper function for `read.table()` with default argument values useful for reading in tab-separated data

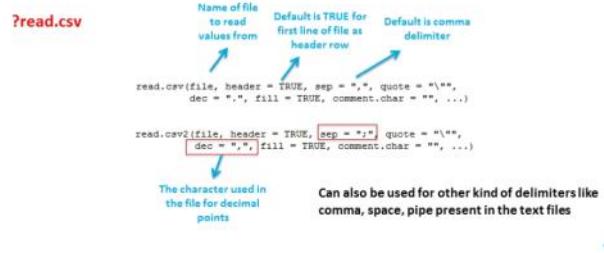


© Jigsaw Academy Education Pvt Ltd

Data from plain text files

read.csv function

Simply a wrapper function for read.table() with default argument values useful for reading in comma-separated data



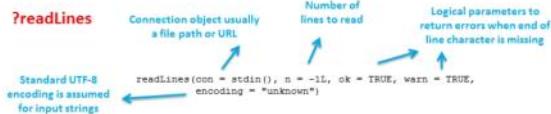
© Jigsaw Academy Education Pvt Ltd

Data from plain text files

readLines function

readLines function returns a vector with as many elements as there are lines from the input source

- Useful for processing text kind of data
- Returns each line as a character string
- read.table and others inputs data into data.frames, matrices etc



© Jigsaw Academy Education Pvt Ltd

Data from plain text files

R Import function parameters

read.table and its wrapper functions are the most convenient ways to read in a rectangular grid of data

- ✓ **encoding:** default is UTF-8, use appropriate parameters for other formats
- ✓ **header:** default argument is TRUE, but explicit declaration is recommended
- ✓ **separator:** use the delimiter as per the input file
- ✓ **column names:** use col.names() explicitly if header is not present
- ✓ **column classes:** default Import functions converts character types to factors which can be suppressed by using as.is=T or stringsAsFactors=F options. Further, colClasses() can be used for custom declaration of variable types

Other input parameters deal with missing values and few special cases

© Jigsaw Academy Education Pvt Ltd

Data from plain text files

Dealing with missing values

R Import functions read any missing values in input source file with a NA representation

- ✓ Empty fields in numeric variables are treated as missing
- ✓ Other formats like NaN, Inf or -Inf are also accepted
- ✓ For character variables, empty values are not considered as missing
- ✓ Input source files should have NA coded in place of empty values
- ✓ This option can be customized using `na.strings()` parameter

© Jigsaw Academy Education Pvt Ltd



Data from plain text files

Input parameters for special cases

Generally, R Import functions have default options set for dealing with certain special string characters present in input source file

- ✓ **quoting:** default character strings are quoted by " or ', for disabling this option use `quote = ""`
- ✓ **unfilled lines:** to deal with files having rows of unequal length using the option `fill = TRUE`
- ✓ **white spaces:** to deal with extra white spaces present in character strings using `strip.white = TRUE`
- ✓ **blank lines:** default input function ignore blank lines, change this using `blank.lines.skip = FALSE`
- ✓ **comments:** default '#' is used as a comment character, for disabling this option use `comment.char= ""`

© Jigsaw Academy Education Pvt Ltd



Data from plain text files

Export functions in R

Exporting results from R is usually straight forward, and normally a text file would be the most convenient output choice

Different methods of writing data in R:

- ✓ **write.table:** writes data frames into text files of different delimiter formats
- ✓ **write.csv:** writes data frame objects into csv files
- ✓ **cat:** writes the input files line by line
- ✓ **writeLines:** complimentary function of `readLines()`

© Jigsaw Academy Education Pvt Ltd



Data from plain text files

write.table function

write.table exports R objects like data.frame, matrix into an external text file

?write.table

```
## S3 method for class 'data.frame'  
## write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",  
##               col.names = NA, dec = ".", row.names = TRUE,  
##               col.names = TRUE, qmethod = c("escape", "double"),  
##               fileEncoding = "")
```

The R object
to be written
character string
naming a file path

Character which
delimits fields

Logical (T/F),
whether row
names to be
returned in file

© Jigsaw Academy Education Pvt Ltd

Data from plain text files

cat and writeLines functions

cat is useful for producing output in user-defined functions
and works much like a print function

?cat

```
## S3 method for class 'connection'  
## cat(..., file = "", sep = " ", fill = FALSE, labels = NULL,  
##      append = FALSE)
```

Output connection
for writing

Character which
delimits fields

Opposite to readLines, writeLines function exports
text output into an external file

?writeLines

```
## S3 method for class 'character'  
## writeLines(text, con = stdout(), sep = "\n", useBytes = FALSE)
```

A character vector

Output
connection or
file path

© Jigsaw Academy Education Pvt Ltd

Lab :

First function is scan()

Scan()

This will help in loading data from screen, i.e. manual input

By default the data type is numeric

If we want to load character data type we will need to use What

➤ Scan(what="")

```
Customers<- scan("customer.csv",  
what=list(fst_nm="", lst_nm="", city = NULL, county= NULL, state="", zip=0),  
          Sep=",",  
          Nmax=400,  
          Skip=1)
```

```
Read.table("customer.csv",  
          Header=True,  
          sep=","")
```

To give the path of the file from where you would like to read it or write to, make use of "\\\" or "/"

using the variables --> StringsAsFactors=F or as.is=T

Will retain the original data types

If we want to have a different data types for different columns

- Vartype <- c(fst_name="character", lst_nm="character", city="character", state="character", zip="numeric")
- Customers <- read.table("customer_noheader.csv", header=F, sep=",", col.names=names(varTypes), colClasses=varTypes)
- Readlines function --> reads the entire line in a row as a single output.
- Very useful while dealing with test data, comments, tweets etc
- Readlines("customer.csv", n=5, ok=T, warn=T)

Working with plain text files 2.2

Example for importing data for tweets:

```
Customers <- read.table("tweet_data.txt", sep="\n", header=F, StringsAsFactors=F, quote="", comment.char="", strip.white=T, fill=T)
```

To write the data into a file in the working directory :

```
Write.data(customers, "customers_new.csv", sep=",", row.names=F)
```

CAT function:

Deals only with character vectors

Useful for printing output from custom functions

```
Cat(customers$fst_name, sep=",")
```

To save this to external file in WD:

```
Cat(customers$fst_name, file = "new.txt", sep = ",")
```

Working with large text files part 1

Data from large text files

Problem to read large text files

R uses system's virtual memory for reading data, and often this causes problems while dealing with large files

- ✓ Easy to exhaust memory by storing unnecessary data
- ✓ OS and 32-bit system architectures can only access 4GB of memory
- ✓ Not wise to use all the system memory available

A numeric matrix containing 100 million rows and 5 columns consumes about 4 gigabytes (GB) of memory in the R statistical programming environment.



© Jigsaw Academy Education Pte Ltd

Data from large text files

Memory limitations in R

R memory limitations on large objects differ between 32-bit and 64-bit builds of R

- R built to run in both 32-bit and 64-bit systems
- Mostly memory limits are based on how R is built
- Sometimes type of OS would also have an influence
 - ✓ Windows OS on 32-bit R systems
- Unix and Mac systems perform better with 32-bit R builds
- Higher system RAM provides better processing (16GB>8GB>4GB)
- Errors related to memory in R
 - ✓ Cannot allocate vector of size

© Jigsaw Academy Education Pvt Ltd



Data from large text files

How to read large text files

Workarounds exist in R as functions, packages, or through better memory management or through using external databases

Different functions for reading large text files in R:

- ✓ **fread**: similar to **read.table** but faster and more convenient
- ✓ **read.csv.sql**: reads csv files into data frame objects
- ✓ **cbc.read.table**: reads the input files column by column
- ✓ **read.table.ffdf**: imports the text files as ff data frames
- ✓ **read.big.matrix**: imports the text files as big matrices

© Jigsaw Academy Education Pvt Ltd



Data from large text files

fread function

fread function exists as part of **data.table** R package which is similar to **read.table** in terms of functionality

```
Name of file  
to read  
values from  
Character which  
delimits fields  
Number of  
rows to be read  
?data.table::fread  
  
fread(input, sep="auto", sep2="auto", nrow=-1L, header="auto", na.strings="NA",  
stringsAsFactors=FALSE, verbose=getOption("data.table.verbose"), autostart=30L,  
skip=-1L, select=NULL, drop=NULL, colClasses=NULL,  
integer64=getOption("data.table.integer64"), # default: "integer64"  
showProgress=getOption("data.table.showProgress"), # default: TRUE  
data.table=getOption("data.table.fread.data.table"), # default: TRUE  
)
```

- ✓ All controls such as **sep**, **colClasses** and **nrows** are automatically detected
- ✓ Integer types are also detected and read directly
- ✓ Dates are read as character which can be converted later

© Jigsaw Academy Education Pvt Ltd



Data from large text files

read.csv.sql function

read.csv.sql function exists as part of **sqldf** R package which reads a file into R by filtering it with an sql statement

```
read.csv.sql(file, sql = "select * from file", header = TRUE, sep = ",",
row.names, col, skip, filter, nrow, field.types, comment.char,
colClasses, dbname = tempfile(), drv = "SQLite", ...)
```

```
read.csv2.sql(file, sql = "select * from file", header = TRUE, sep = ";",
row.names, col, skip, filter, nrow, field.types, comment.char,
colClasses, dbname = tempfile(), drv = "SQLite", ...)
```

?sqldf::read.csv.sql

- ✓ Default option for dbname is tempfile()
- ✓ Specifying NULL will improve the speed of data loading
- ✓ Reads as a database in memory but limited by system RAM

© Jigsaw Academy Education Pvt Ltd



Data from large text files

cbc.read.table function

cbc.read.table function exists as part of **colbycol** R package which makes it easier to read only the required columns

?colbycol::cbc.read.table

```
cbc.read.table(xrs, filehash.name = tempfile(pattern = "filehash_"),
tmp.dir = tempfile(pattern = "dir"),
just.read = NULL, sample.pct = NULL, sep = "\t",
header = TRUE, ...)
```

- ✓ Handles huge text files beyond restrictions of read.table function
- ✓ It reads the file line by line, breaks it into as many physical files as columns
- ✓ Reads them back into R one by one and saves them in efficient, native R data files
- ✓ Latest releases of this package are not present in CRAN and can make use of old files

© Jigsaw Academy Education Pvt Ltd



The colbycol package is not present in the latest CRAN servers, so need to follow old approach.

Data from large text files

read.table.ffdf function

read.table.ffdf function exists as part of **ff** R package which reads separated flat files into ffdf objects

?ff::read.table.ffdf

```
read.table.ffdf(
  , n = NULL,
  , file, filehash.binding = "",
  , nrow = 1, first.rows = NULL, next.rows = NULL,
  , levels = NULL, append.levels = TRUE
  , for = c("read.table", ...
  , "read.FFDF", ...
  , "as.ffdf", ...
  , "asffdf", args = list()
  , BATCHBYTES =getOption("ffbatchbytes")
  , VERBOSE = FALSE
  )
```

Number of rows to be read

ff data.frames are stored on disk and very similar to 'data.frame'

Logical: TRUE to verbose
timings for each processed chunk

Number of bytes allowed for the size of the data.frame storing the result of reading one chunk

- ✓ Reads very large files in row-chunks and store the result in a ffdf object on disk
- ✓ The first chunk is read with a default of 1000 rows, for subsequent chunks adjusts to RAM
- ✓ Provides wrapper functions specific to csv files through read.csv.ffdf similar to read.table

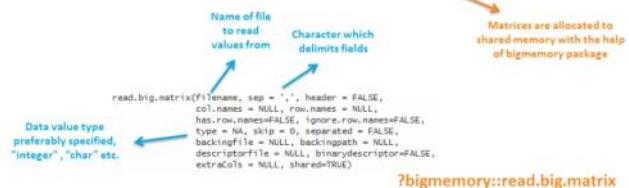
© Jigsaw Academy Education Pvt Ltd



Data from large text files

read.big.matrix function

read.big.matrix function exists as part of **bigmemory** R package which reads huge text files into a big.matrix objects



- ✓ Input files must contain only one atomic type i.e. integer, character etc.
- ✓ CRAN support is only available only for Linux and OSX machines
- ✓ Not possible to work with latest R versions on Windows machines



© Jigsaw Academy Education Pvt Ltd

Data from large text files

Optimized read.table for large text files

read.table function can also be efficiently used for reading large text files into R with help of specific options

- **nrows:** specifying total lines of file to avoid block wise reading
- **colClasses:** use of column declaration reduces overload on R
 - ✓ NULL type for columns to be omitted
- **comment.char:** use of blank option turns off interpretation of comments
- **Tips for data processing**
 - ✓ Saving intermediate files in R native format for later reuse
 - ✓ Avoiding large vector operations



© Jigsaw Academy Education Pvt Ltd

Data from large text files

Writing R data into large text files

Similar to importing, even exporting of huge R data.frame objects into text files can sometimes be time consuming

Different functions for writing large R objects:

- ✓ **write.table:** writes data frame objects into text files
- ✓ **write.csv:** writes data frame objects into csv files
- ✓ **write.table.ffdf:** exports the ff data frames as text files
- ✓ **write.big.matrix:** part of **bigmemory** package, writes huge big.matrix objects into text files



© Jigsaw Academy Education Pvt Ltd

Working with large text files - part 2 - writing to files

To find the processing time for loading the file:

```
System.time(customer.big<-read.csv("customer_million.csv", header = T, sep = "," ))
```

```
System.time(customer.big<-fread("customer_million.csv", header = T, sep = ","))
```

The screenshot shows an RStudio interface with the following details:

- File Edit Code View Plots Session Build Debug Tools Help**
- R Data Science with R - RStudio**
- Working with large test files** (selected tab)
- Source on Save**
- Run** button
- Script** tab
- Code** pane:

```
varTypes =  
  c(first_name="factor",last_name="factor",  
    city="factor",county="factor",  
    state="factor",zip="factor")  
  
system.time(customers.ffd<-read.csv.ffdf(file="customers_million.csv",  
  header=F,  
  VERBOSE=T,  
  first.rows=500000,  
  next.rows=500000,  
  col.names=names(varTypes),  
  colClasses=varTypes))  
  
write.csv, write.csv2  
  
The following objects are masked from 'package:base':  
  
  is.factor, is.ordered
```
- Console** pane:

```
> varTypes =  
>   c(first_name="factor",last_name="factor",  
>     city="factor",county="factor",  
>     state="factor",zip="factor")  
>
```

```
Customers.ffdf <- Read.csv.ffdf("customer_million.csv", header = F, verbose = T, first.rows=500000, next.rows = 500000, col.names=names(varTypes), colClasses=varTypes)
```

The screenshot shows the RStudio interface with two panes. The top pane displays R code for reading a large CSV file, and the bottom pane shows the resulting output from the R console.

```
File Edit Code View Plots Session Build Debug Tools Help  
Working with large flat files.R  
Source on Save Run Source  
varTypes =  
c(first_name="character", last_name="character",  
  city="character", county="character",  
  state="character", zip="character")  
  
system.time(customers.big<-read.table("customers_million.csv",  
  nrows=1000000,  
  header=F,  
  sep=",",  
  fill=T,  
  col.names=names(varTypes),  
  colClasses=varTypes,  
  comment.char="")  
  
#Using read.csv.sql function
```

The bottom pane shows the R console output:

```
1: + VERBOSE=1,  
2: + first.rows=500000,  
3: + next.rows=500000,  
4: + col.names=names(varTypes),  
5: + colClasses=varTypes})  
6:  
read.table.ffdf 1.500000 (500000) csv-read=3.75sec ffdff-write=0.39sec  
read.table.ffdf 500000.1.1000000 (500000) csv-read=4.05sec ffdff-write=0.07sec  
read.table.ffdf 1000001..1000001 (1) csv-read=0.01sec ffdff-write=0.07sec  
csv-read=7.79sec ffdff-write=0.84sec TOTAL=8.63sec  
  user system elapsed  
  8.09   0.27   8.63  
>
```

Using SQLDf package:

The screenshot shows an RStudio interface with the following code in the source editor:

```
header=F,
sep=",",
as.is=T,
col.names=names(varTypes),
colClasses=varTypes,
comment.char="")}

#Using read.csv.sql function
library(sqldf)
system.time(customers.big<-read.csv.sql("customers_million.csv",
header=T,
sep=","))

#Using read.csv.sql function with SQL option
system.time(customers.NY<-read.csv.sql("customers_million.csv",
SQL select * from file where state='NY'",
header=T,
sep=","))


```

The console output shows the execution of the code:

```
Loading required package: proto
Loading required package: RSQLite
Loading required package: DBI
> system.time(customers.big<-read.csv.sql("customers_million.csv",
+           header=T,
+           sep=","))

Loading required package: tcltk
  user  system elapsed
 6.61   0.54  7.23
warning message:
closing unused connection 3 (customers_million.csv)
>
```

Using the SQL statement to filter data to load:

The screenshot shows an RStudio interface with the following code in the source editor:

```
header=F,
sep=",",
colClasses=varTypes,
comment.char="")}

#Using read.csv.sql function
library(sqldf)
system.time(customers.big<-read.csv.sql("customers_million.csv",
header=T,
sep=","))

#Using read.csv.sql function with SQL option
system.time(customers.NY<-read.csv.sql("customers_million.csv",
sql=" select * from file where state='NY'",
header=T,
sep=","))


```

The console output shows the execution of the code:

```
Loading required package: proto
Loading required package: RSQLite
Loading required package: DBI
> system.time(customers.big<-read.csv.sql("customers_million.csv",
+           header=T,
+           sep=","))

Loading required package: tcltk
  user  system elapsed
 6.61   0.54  7.23
warning message:
closing unused connection 3 (customers_million.csv)
>
```

Data from large text files

Runtime comparisons of import functions

System setup includes Windows 8 64-bit
version fitted with 4 GB RAM

File type: csv text file
File size: 42 MB
Rows: 1000000
Columns: 6

Package	Function	Run Time
Base R	read.table	~6.37 sec
data.table	fread	~1 sec
sqldf	read.csv.sql	~11 sec
ff	read.table.ffdf	~8 sec
Base R	optimized read.table	~4 sec

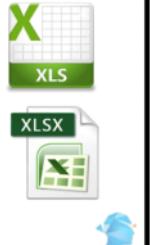
Working with Excel Workbooks:

Data from excel files

Excel spreadsheets

Easy accessible tool for organizing, analyzing and storing data in tables and has widespread applications

- Commonly occurring file formats: .xls or .xlsx
- Widely used across industry for reporting and BI
- Provides support for database connectivity
- Does not support advanced analytics functions
- Slow processing performance on large data
- R has support to read, write and manipulate excel files



Often using R, It is recommended to convert the excel files into plain text formats like csv, tab delimited forms

Data from excel files

R solutions for excel files

R support comes in the form of supported packages for reading and writing excel files

Different methods of handling excel files in R:

- ✓ [XLConnect](#): R package for manipulating excel files
- ✓ [xlsx](#): reads and writes excel files in R using java
- ✓ [gdata](#): similar to xlsx package with perl dependencies for windows

Care should be taken while using these existing methods because of their dependency on other software's and limitations with respect to OS platforms

© Jigsaw Academy Education Pvt Ltd

Data from excel files

XLConnect R package

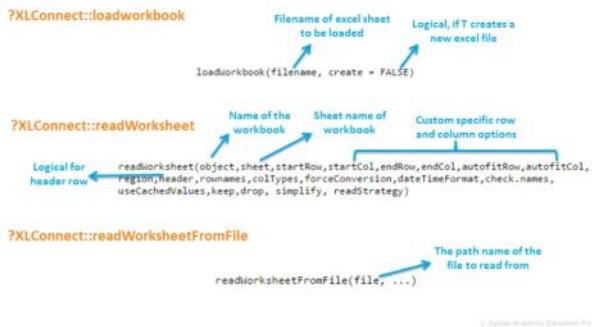
XLConnect is a package that allows for reading, writing and manipulating Microsoft Excel files from within R

- Provides support for both .xls and .xlsx files
- Cross-platform compatible and runs on Windows/Linux/Mac
- Would require java runtime environment (JRE, version 6 or higher)
- Installation directly from CRAN using `install.packages()`
- Limitation for handling large data files
- Import functions for reading from excel files
 - ✓ `loadWorkbook()`, `readWorksheet()`, `readWorksheetFromFile()`
- Export functions for writing to excel files
 - ✓ `createSheet()`, `writeWorksheet()`, `saveWorkbook()`

© Jigsaw Academy Education Pvt Ltd

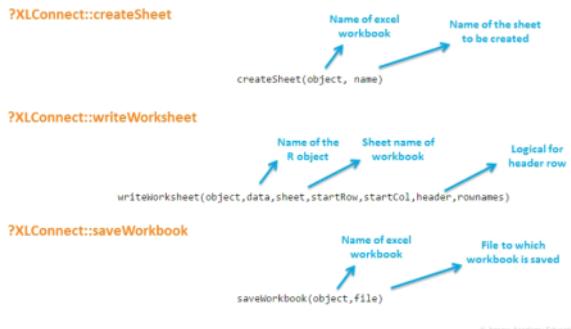
Data from excel files

XLConnect: Reading data



Data from excel files

XLConnect: Writing data



Data from excel files

xlsx R package

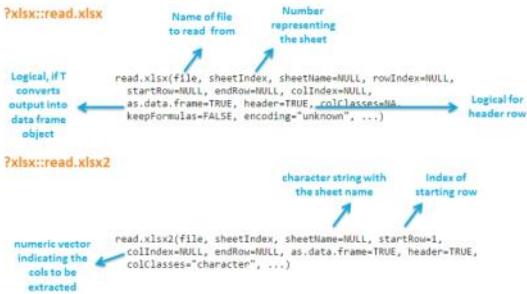
Provides high level API allows the user to read a sheet of an xlsx document into a data.frame and write a data.frame to a file

- Makes use of R and java integration
- Cross-platform compatible and runs on Windows/Linux/Mac
- Installation directly from CRAN using `install.packages()`
- Import functions for reading from excel files
 - ✓ `read.xlsx()`, `read.xlsx2()`
- Export functions for writing to excel files
 - ✓ `write.xlsx()`

© Jigsaw Academy Education Pvt Ltd

Data from excel files

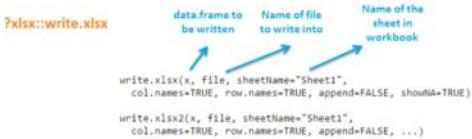
xlsx: Reading data



© Jigsaw Academy Education Pvt Ltd

Data from excel files

xlsx: Writing data



© Jigsaw Academy Education Pvt Ltd

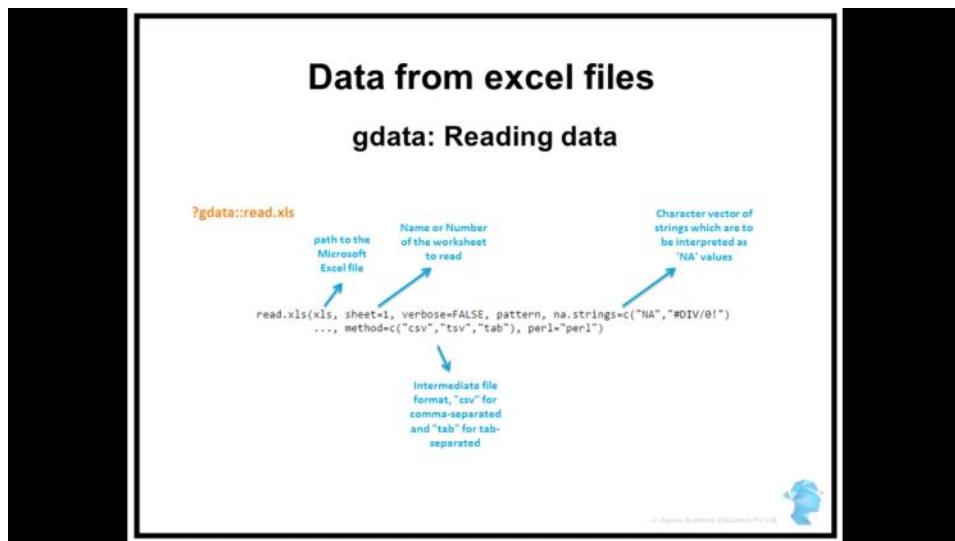
Data from excel files

gdata R package

gdata is a general package that provides various R programming tools for data manipulations tasks

- Provides support for both .xls and .xlsx files
- Cross-platform compatible and runs on Windows/Linux/Mac
- Requires additional perl libraries on Windows systems
- Installation directly from CRAN using `install.packages()`
- Supports only import functions for excel files
- Import functions for reading from excel files
- ✓ `read.xls()`

© Jigsaw Academy Education Pvt Ltd



Working with Excel Workbooks Part 2:

```
File Edit Code View Plots Session Build Debug Tools Help
R Data Science with R - RStudio
Working with excel workbooks.R
Source on Save E Z-
Floating Library library(xLConnect)

#Loading Excel workbook
wb<-loadWorkbook("customers.xlsx")

#Reading "newyork" sheet from workbook connection
newyork<-readWorksheet(wb,
  "newyork",
  header=T)

#Reading "california" sheet from workbook connection
california<-readWorksheet(wb,
  "california",
  header=T)

Console 1 Data Science with R
> library(xLConnect)
> #Loading Excel workbook
> wb<-loadWorkbook("customers.xlsx")
> wb
[1] "customers.xlsx"
> #Reading "newyork" sheet from workbook connection
> newyork<-readWorksheet(wb,
+   "newyork",
+   header=T)
> class(newyork)
[1] "data.frame"
```

```
File Edit Code View Plots Session Build Debug Tools Help
R Data Science with R - RStudio
Working with excel workbooks.R
Source on Save E Z-
Reading "california" sheet from workbook connection
california<-readWorksheet(wb,
  "california",
  header=T)

#Reading "newyork" sheet directly from excel file
newyork<-readWorksheetFromFile("customers.xlsx",
  "newyork",
  header=T)

#using .xls excel file
newyork<-readWorksheetFromFile("customers.xls",
  "newyork",
  header=T)

Console 1 Data Science with R
> header(T)
> class(california)
[1] "data.frame"
> head(california)
  first_name last_name      city county state zip
1    Leota Dillard  San Jose Santa Clara CA 95111
2    Kitley Calderera Los Angeles Los Angeles CA 90034
3  Veronika Induye  San Jose Santa Clara CA 95111
4  Rozella Ostrosky Camarillo Ventura CA 93012
5   Kanisha Waycott Los Angeles Los Angeles CA 90006
6   Shenika Seewald Van Nuys Los Angeles CA 91405
```

The screenshot shows the RStudio interface with the following R code:

```
#Writing to excel files
##using createSheet for creating a new worksheet in excel file
createSheet(wb,"new sheet")

##writing R data frame object
writeworksheet(wb,newyork,"new sheet")

##Saving the workbook
saveworkbook(wb,"customers.xlsx")

##using xlsx R package
##Reading from excel files
#Loading Library
library(xlsx)
```

The console output shows the head of the 'newyork' data frame:

```
> head(newyork)
   first_name last_name      city county state    zip
1      Abel     Macleod  Middle Island Suffolk    NY 11953
2  Maryann     Royster      Albany  Albany    NY 12204
3      willow      Kusko      New York  New York    NY 10011
4     Alishia      Sergi      New York  New York    NY 10025
5      Jose    Stockham      New York  New York    NY 10011
6     Brock   Bolognia      New York  New York    NY 10003
```

The screenshot shows the RStudio interface with the following R code:

```
#Writing to excel files
##using createSheet for creating a new worksheet in excel file
createSheet(wb,"new sheet")

##writing R data frame object
writeworksheet(wb,newyork,"new sheet")
```

The console output shows the head of the 'newyork' data frame:

```
#Saving the workbook
saveworkbook(wb,"customers.xlsx")

##using xlsx R package
##Reading from excel files
#Loading Library
library(xlsx)
```

```
> head(newyork)
   first_name last_name      city county state    zip
1      willow      Kusko      New York  New York    NY 10011
2     Alishia      Sergi      New York  New York    NY 10025
3      Jose    Stockham      New York  New York    NY 10011
4     Brock   Bolognia      New York  New York    NY 10003
```

The screenshot shows the RStudio interface with the following R code:

```
#Reading a sheet from .xlsx excel file into R data frame
newyork<-read.xlsx("customers.xlsx",
                     sheetName="newyork")
```

The console output shows the head of the 'newyork' data frame:

```
#Reading a sheet from .xls excel file into R data frame
newyork<-read.xlsx("customers.xls",
                     sheetName="newyork")
```

The code then exports the data frame back to an Excel file:

```
#Writing to excel files
#Export a data frame into excel file
write.xlsx(newyork,
```

The console output shows the attached package:

```
Attaching package: 'xlsx'
```

The following objects are masked from 'package:XLConnect':

```
  createFreezePane, createSheet, createSplitPane, getCellStyle, getSheets, loadWorkbook, removeSheet, saveWorkbook, setCellStyle, setColumnWidth, setRowHeight
```

The code then reads the sheet from the .xlsx file again:

```
> #Reading a sheet from .xlsx excel file into R data frame
> newyork<-read.xlsx("customers.xlsx",
+                      sheetName="newyork")
```

The screenshot shows the RStudio interface with two panes. The left pane displays R code for reading and writing Excel files:

```
#Working with excel workbooks.R
#Reading a sheet from .xls excel file into R data frame
newyork<-read.xlsx("customers.xls",
                     sheetName="newyork")

#Writing to excel files
#Export a data frame into excel file
write.xlsx(newyork,
           "customers_newyork.xlsx",
           sheetName="newyork",
           row.names=TRUE)

##Using gdata R package
##Reading from excel files
##Loading Library
library(gdata)
```

The right pane shows the output of the R code, displaying the first few rows of the 'newyork' data frame:

```
head(newyork)
#> #>   first_name last_name      city county state    zip
#> 1   Abel        Macleod Middle Island Suffolk NY 11953
#> 2 Maryanne     Royster      Albany Albany  NY 12204
#> 3   Willow       Kusko      New York New York  NY 10011
#> 4 Alishia       Sergi      New York New York  NY 10025
#> 5   Jose        Stockham    New York New York  NY 10011
#> 6 Brock       Bolognia    New York New York  NY 10003
```

Working with Statistical System Formats:

Data from statistical systems

Statistical Systems

R offers support to directly read and write data formats used by other statistical systems like SAS, SPSS etc.

- foreign package offers features for these formats
- Majorly provides data import facilities
- Exporting R data to these formats is possible
- Import functions use less memory than read.table()

Stata **SPSS** **Sas**

© Jigsaw Academy Education Pvt Ltd

Data from statistical systems

foreign R package: Import Functions

foreign, R package provides import facilities for files produced by statistical systems like SAS, SPSS etc.

Different methods of handling statistical system files in R:

- ✓ **SAS files:** read.xport(), read.ssdf()
 - Files with extensions like .xpt, .sas7bdat
- ✓ **SPSS files:** read.spss()
 - Files with extensions like .sav
- ✓ **Stata files:** read.dta()
 - Files with extensions like .dta

© Jigsaw Academy Education Pvt Ltd

Data from statistical systems

foreign: Reading SAS files

?foreign::read.xport

character variable with the name of the file to read

read.xport(file)

1. Read a SAS XPORT Format Library
2. Files with .xpt extension

?foreign::read.spdd

This requires SAS to be available

character string defining the SAS library

files in the libname directory

character string giving full path to SAS executable

read.spss(libname, sectionnames,
tmpXport=tempfile(), tmpProgLoc=tempfile(), sascmd="sas")

© Jigsaw Academy Education Pvt Ltd

Data from statistical systems

foreign: Reading SPSS files

?foreign::read.spss

Reads a file stored by the SPSS save or export commands

Name of the file to read

logical: convert variables with value labels into R factors with those levels

Logical, if T returns a data frame

read.spss(file, use.value.labels = TRUE, to.data.frame = FALSE,
max.value.labels = Inf, trim.factor.names = FALSE,
trim_values = TRUE, reencode = NA, use.missing = to.data.frame)

© Jigsaw Academy Education Pvt Ltd

Data from statistical systems

foreign: Reading Stata files

?foreign::read.dta

Reads a file in Stata version 5–12 binary format into a data frame

Name of the file to read

Convert Stata dates to Date class

Use Stata value labels to create factors

read.dta(file, convert.dates = TRUE, convert.factors = TRUE,
missing.type = FALSE,
convert.underscore = FALSE, warn.missing.labels = TRUE)

© Jigsaw Academy Education Pvt Ltd

Data from statistical systems

foreign R package: Export Functions

foreign, R package provides few options for writing R data.frames into statistical systems outputs

Different methods of writing statistical system files from R:

- ✓ **Stata files:** write.dta()
- ✓ **SAS, SPSS files:** write.foreign()

© Jigsaw Academy Education Pvt Ltd



Data from statistical systems

foreign: Writing Stata files

?foreign::write.dta

Writes the data frame to file in the Stata binary format

```
write.dta(dataframe, file, version = 11,  
          convert.date = TRUE, tz = "GMT",  
          convert.factors = c("labels", "string", "numeric", "codes"))
```

© Jigsaw Academy Education Pvt Ltd



Data from statistical systems

foreign: Writing SAS, SPSS files

?foreign::write.foreign

Exports simple data frames to other statistical packages by writing the data as free-format text and writing a separate file of instructions for the other package to read the data

```
write.foreign(df, datafile, codefile,  
              package = c("SPSS", "Stata", "SAS", ...))
```

© Jigsaw Academy Education Pvt Ltd



Data from databases

Need for databases

Data objects that are more than a (few) hundred megabytes in size can cause R to run out of memory

✓ **Database management systems (DBMSs) handle these things well**

- To provide fast access to selected parts of large databases
- Powerful ways to summarize and cross-tabulate columns in databases
- Concurrent access from multiple clients
- Ability to act as a server to a wide range of clients

01:44 - 13:16

300 kbps ▲ ◀ ▶ ■

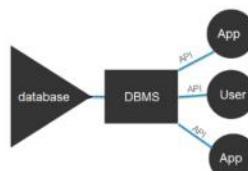
Data from databases

Overview of databases

DBMS is a software system that uses a standard method of cataloguing, retrieving, and running queries on data

The DBMS essentially serves as an interface between the database and end users or application programs, ensuring that data is consistently organized and remains easily accessible

With relational DBMSs (RDBMSs), the API is SQL, a standard programming language for defining, protecting and accessing data in a RDBMS



Over the last 30 years, the primary operational DBMSs have been relational like MySQL, PostgreSQL, SQL Server, Oracle and Teradata

03:50 - 13:16

300 kbps ▲ ◀ ▶ ■

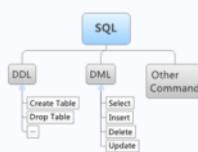
Data from databases

SQL query framework

SQL (Structured Query Language) is a standard language for accessing and manipulating databases

What can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can set permissions on tables, procedures, and views



04:17 - 13:16

300 kbps ▲ ◀ ▶ ■

Data from databases

Essence of big databases

In the era of big data, companies are leaning more towards NoSQL database alternatives compared to traditional relational databases

What is Hadoop?

Hadoop was developed as a « NoSQL » technology (more general than SQL) with specific capabilities to handle unstructured data such as text documents, XML files, images etc.

Common examples of NoSQL databases
are MongoDB, CouchDB, HBase, Cassandra and Neo4J

Data from databases

Interacting with databases

R consists of packages for connecting directly to several major database management systems

Different DBMS interface packages in R:

- ✓ **RODBC**: general purpose database connectivity package
- ✓ **DBI**: connectivity with relational DBMSs
- ✓ **RMySQL**: DBI-compliant interface to MySQL database
- ✓ **ROracle**: Oracle database interface (DBI) driver for R
- ✓ **RSQLite**: embeds the SQLite database engine in R
- ✓ **RMongoDB**: interface to MongoDB NoSQL database
- ✓ **RHadoop**: framework providing R and Hadoop integration

Data from databases

Using RSQLite Package

SQLite (<http://www.sqlite.org/>) is a lightweight, self-contained, zero-configuration, cross-platform, open source database engine

Few useful functions under RSQLite:

- ✓ **dbConnect**: Connect to a SQLite database
- ✓ **dbGetQuery**: Execute SQL queries on SQLite database from R
- ✓ **dbReadTable**: Reads table from SQLite database to R
- ✓ **dbWriteTable**: Writes R objects to SQLite database
- ✓ **dbListTables**: List tables in specified database
- ✓ **dbListFields**: List fields in specified table
- ✓ **dbDisconnect**: Disconnect from a SQLite database

Data from databases

Using RMySQL Package

MySQL is the world's most popular open source database and RMySQL allows you to access MySQL databases from within R

Few useful functions under RMySQL:

- ✓ **dbConnect**: Connect to a MySQL database
- ✓ **dbSendQuery**: Execute SQL queries on MySQL database from R
- ✓ **dbReadTable**: Reads table from MySQL database to R
- ✓ **dbWriteTable**: Writes R objects to MySQL database
- ✓ **dbListTables**: List tables in specified database
- ✓ **dbListFields**: List fields in specified table
- ✓ **dbDisconnect**: Disconnect from a MySQL database



Data from databases

Using RODBC Package

Package RODBC on CRAN provides an interface to access different database systems across all OS platforms

Few useful functions under RODBC:

- ✓ **odbcConnect**: open ODBC connections
- ✓ **odbcGetInfo**: provides information on ODBC connection
- ✓ **sqlQuery**: query an ODBC database
- ✓ **odbcClose**: close ODBC connections

RODBC supports Microsoft SQL Server, Access, MySQL, PostgreSQL, Oracle, IBM DB2, MySQL, MariaDB, Oracle, PostgreSQL and SQLite



Data from databases

RHadoop Framework

RHadoop is an open source project that allows programmers directly use the functionality of MapReduce in R code

- Collection of R packages: rhdfs, rmr2, rhbase, plyr
- Developed by Antonio Piccolboni, with support from Revolution Analytics
- Mostly implemented in native R
- Some pieces written in C++ and Java
- Provides access to R and MapReduce without leaving R console
- Not part of official CRAN software packages, separate installation needed

More information: <https://github.com/RevolutionAnalytics/RHadoop/wiki>



Working with databases Part 2:

The screenshot shows the RStudio interface with the title bar "E/Data Science with R - RStudio". The code editor pane contains the following R script:

```
##Working with databases.R
##Loading Library
library(RSQLite)

##Database connectivity
sqlite<-dbDriver("sqlite")
customersdb<-dbConnect(sqlite,"customersDB.db")

##Listing tables
dbListTables(customersdb)

##Listing fields from a table
dbListFields(customersdb,"newyork")

##Running SQL queries
dbGetQuery(customersdb,
           "select count(*) from newyork")
```

The console pane at the bottom shows the results of the first query:

```
> ##Loading Library
> library(RSQLite)
> ##Database connectivity
> sqlite<-dbDriver("sqlite")
> customersdb<-dbConnect(sqlite,"customersDB.db")
>
```

The screenshot shows the RStudio interface with the title bar "E/Data Science with R - RStudio". The code editor pane contains the following R script:

```
##Listing Fields from a table
dbListFields(customersdb,"newyork")

##Running SQL queries
dbGetQuery(customersdb,
           "select count(*) from newyork")

dbGetQuery(customersdb,
           "select * from newyork limit 10")

dbGetQuery(customersdb,
           "select city,count(*) as cnt from newyork group by city order by cnt desc")
```

The console pane at the bottom shows the results of the second query:

```
> ##Loading Library
> library(RSQLite)
> ##Database connectivity
> sqlite<-dbDriver("sqlite")
> customersdb<-dbConnect(sqlite,"customersDB.db")
> dbListTables(customersdb)
[1] "california" "newyork" "others"
> ##Listing fields from a table
> dbListFields(customersdb,"newyork")
[1] "first_name" "last_name" "city"      "county"    "state"    "zip"
>
```

The screenshot shows the RStudio interface with the title bar "E/Data Science with R - RStudio". The code editor pane contains the following R script:

```
"select * from newyork limit 10"

dbGetQuery(customersdb,
           "select city,count(*) as cnt from newyork group by city order by cnt desc")

##Reading data from SQLITE to R
newyork<-dbReadTable(customersdb, "newyork")
head(newyork)

##Writing data from R to SQLITE
dbWriteTable(customersdb,"newyork1",newyork)

##Disconnecting database connection
```

The console pane at the bottom shows the results of the third query and the final output of the fourth query:

```
10 Timothy Mulqueen Staten Island Richmond NY 10309
> dbGetQuery(customersdb,
+             "select city,count(*) as cnt from newyork group by city order by cnt desc")
      city cnt
1   New York 14
2   Brooklyn  4
3 Garden City  2
4   Albany    1
5   Astoria   1
6   Bohemia   1
7   Bronx     1
8   Buffalo   1
```

```

##Reading data from SQLite to R
newyork<-dbrditable(customersdb, "newyork")
head(newyork)

##Writing data from R to SQLite
dbwriteTable(customersdb,"newyork1",newyork)

##For disconnecting database connection
dbDisconnect(customersdb)

##Using RODBC Package - A general purpose database package
##Interacting with SQLite database tables

```

Console output:

```

1 Abel Macleod Middle Island Suffolk NY 11953
2 Maryann Royster Albany Albany NY 12204
3 Willow Kusko New York New York NY 10011
4 Alishia Sergi New York New York NY 10025
5 Jose Stockham New York New York NY 10011
6 Brock Bologna New York New York NY 10003
> ##Writing data from R to SQLite
> dbwriteTable(customersdb,"newyork1",newyork)
[1] TRUE
> dbListTables(customersdb)
[1] "california" "newyork" "newyork1" "others"
>

```

```

##update SQLite db details under ODBC connections

##Database connectivity
##Opening ODBC connection
customers<-odbcConnect("customersdb",
believeRows = FALSE,
rows_at_time = 1)

##Information about current database connection
odbcGetInfo(customers)

##Listing the tables present in the database
sqlTables(customers)

```

Console output:

```

rows_at_time = 1)
> ##Information about current database connection
> odbcGetInfo(customers)
+   [DBMS_Name           DBMS_Ver
+     "SQLite"           "3.8.0"
+   Driver_ODBC_Ver    Data_Source_Name
+     "03.00"            "customersdb"
+   Driver_Name          Driver_Ver
+     "sqlite3odbc.dll"  "0.9992"
+   ODBC_Ver             Server_Name
+     "03.80.0000"        "E:\\Data Science with R\\customersDB.db"
>

```

```

##Listing the tables present in the database
sqlTables(customers)

##Running SQL queries
sqlQuery(customers,
"select count(*) from newyork")

sqlQuery(customers,
"select * from newyork limit 10")

sqlQuery(customers,
"select city,count(*) as cnt from newyork group by city order by cnt desc")

```

Console output:

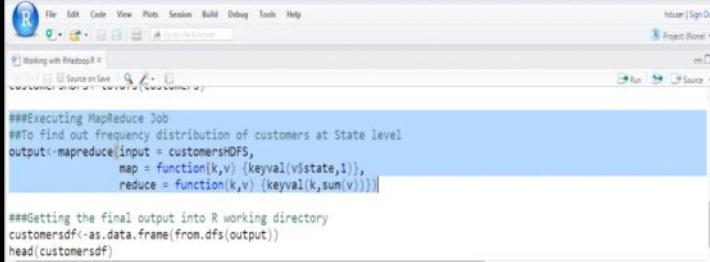
```

"03.00"           "customersdb"
Driver_Name          Driver_Ver
"sqlite3odbc.dll"  "0.9992"
ODBC_Ver             Server_Name
"03.80.0000"        "E:\\Data Science with R\\customersDB.db"
> sqlTables(customers)
TABLE_CAT TABLE_SCHEMA TABLE_NAME TABLE_TYPE REMARKS
1 <NA>      <NA>    newyork    TABLE   <NA>
2 <NA>      <NA>    california TABLE   <NA>
3 <NA>      <NA>    others    TABLE   <NA>
4 <NA>      <NA>    newyork1   TABLE   <NA>

```

Loading the dataframe object into the hadoop cluster:

Map reduce job being run here:



The screenshot shows an RStudio interface with the following details:

- Title Bar:** Studio 54.174.192.77:8767
- File Menu:** File Edit Code View Plots Session Build Debug Tools Help
- Project Bar:** hbase [Sign Out] Project Home
- Code Editor:** Working with Hadoop R
- Code Content:**

```
##Executing MapReduce Job
##To find out frequency distribution of customers at State level
output<-mapreduce(input = customersHDFS,
                   map = function(k,v) {keyval(v$state,1)},
                   reduce = function(k,v) {keyval(k,sum(v)))})
##Getting the final output into R working directory
customersdf<-as.data.frame(from.xls(output))
head(customersdf)
##Setting up hadoop environment variables
Sys.setenv("HADOOP_HOME"/opt/cloudera/parcels/CDH/bin/hadoop")
Sys.setenv("HADOOP_STREAMING"/opt/cloudera/parcels/CDH-4.7.0-1.cdh4.7.0.p0.48/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-streaming-2.0.0-mr1-cdh4.7.0.jar)
##Loading required libraries
library(rmr2)
customersHDFS<-to.xls(customers)
15/06/04 21:34:12 INFO zlib.ZlibFactory: Successfully loaded & initialized native-zlib library
15/06/04 21:34:12 INFO compress.CompressorPool: Got brand-new compressor [.deflate]
##Executing MapReduce Job
##To find out frequency distribution of customers at State level
```
- Console:**

```
> ##Setting up hadoop environment variables
> Sys.setenv("HADOOP_HOME"/opt/cloudera/parcels/CDH/bin/hadoop)
> Sys.setenv("HADOOP_STREAMING"/opt/cloudera/parcels/CDH-4.7.0-1.cdh4.7.0.p0.48/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-streaming-2.0.0-mr1-cdh4.7.0.jar)
> ##Loading required libraries
> library(rmr2)
> customersHDFS<-to.xls(customers)
15/06/04 21:34:12 INFO zlib.ZlibFactory: Successfully loaded & initialized native-zlib library
15/06/04 21:34:12 INFO compress.CompressorPool: Got brand-new compressor [.deflate]
> ##Executing MapReduce Job
> ##To find out frequency distribution of customers at State level
```

Working with Web Data Part 1:

Data from web

Web data sources

In comparison with traditional structured data formats, web data generally exists in semi-structured and un-structured formats

Different sources of web data:

- ✓ **Web pages:** static and dynamic html content
- ✓ **Information portals:** content in structured format (.txt, .csv, .xls, .xlsx)
- ✓ **APIs:** content in structured or semi-structured formats (.xml, .json)
- ✓ **Text sources:** content in unstructured format like text



Semi-Structured and unstructured data.

Gathering the webdata:

Unstructured data from social media:

Steps 1:

- Location of data:
 - Know about the data source. Is it from a single web page or spread across multiple pages
- Accessing the data -->
 - can we get it from website or do we need API
- Structure or format of data:
 - Plain text, excel, XML, json???

Data from web

Gathering web data

Web contains huge amounts of useful information and its exponential growth is challenging our data processing capabilities

Useful questions to address before web data gathering process:

- ✓ Location of data
 - Is it present in single page or spread across multiple web pages?
- ✓ Accessibility of data
 - Is it directly accessible or through API?
- ✓ Structure or format of data
 - Is it in plain text format or a spreadsheet format or other formats like xml, json?

Data from web

Reading data from URL links

Connections in R define a mechanism for handling input (reading) and output (writing) operations

Two of the important functions to create connections in R:

- ✓ **file()**: path to the file to be opened or complete URL
- ✓ **url()**: a complete URL

These functions will take the place of input parameter in the general import functions like read.table(), scan() etc.



© Jigsaw Academy Education Pvt Ltd

Data from web

R connection functions: file(), url()

?file

```
A description of the connection
Description of how to open the connection
file(description = "", open = "", blocking = TRUE,
encoding = getOption("encoding"), raw = FALSE)

A description of the connection
Description of how to open the connection
url(description, open = "", blocking = TRUE,
encoding = getOption("encoding"), method)
```

© Jigsaw Academy Education Pvt Ltd

Data from web

Common web formats

XML and JSON are general purpose, lightweight and simple formats for storing web data

✓ **XML**: Extensible Markup Language

- Designed to store and transfer data
- Data representation is done with the help of tags
- Package supportin R for xml data parsing

✓ **JSON**: JavaScript Object Notation

- Common format for representing data
- Web APIs provide data in json format
- Package supportin R for json data processing

JSON

```
{
  "siblings": [
    {"firstName": "Anna", "lastName": "Clayton"},
    {"lastName": "Alex", "lastName": "Clayton"}
  ]
}
```

XML

```

<siblings>
  < sibling>
    <firstName>Anna</firstName>
    <lastName>Clayton</lastName>
  </ sibling>
  < sibling>
    <firstName>Alex</firstName>
    <lastName>Clayton</lastName>
  </ sibling>
</siblings>
```

© Jigsaw Academy Education Pvt Ltd

Steps involved for XML extractions:

Data from web

Handling XML data

R package widely used for parsing XML documents:

- ✓ **XML**: provides approaches for both reading and creating XML documents

Steps involved in using XML package functions:

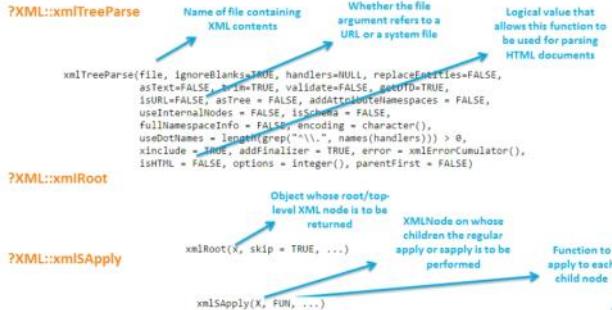
- Read an XML document into R
- Access elements of XML tree structure
- Extract structured contents of XML document
- Converting extracted contents to R data.frame object

© Jigsaw Academy Education Pvt Ltd



Data from web

XML package: parsing functions



Data from web

Handling JSON data

R has 3 packages for working with json data:

- ✓ **RJSONIO**: allows conversion to and from data in JSON format
- ✓ **rjson**: converts R object into JSON objects and vice-versa
- ✓ **jsonlite**: offers flexible, robust, high performance tools for working with JSON in R; and easier to convert json objects directly into R data.frames

All packages provide 2 main functions: **toJSON()** and **fromJSON()**; that allow conversion to and from data in JSON format, respectively

© Jigsaw Academy Education Pvt Ltd



Data from web

Reading and Writing json files

`?jsonlite::fromJSON`

a JSON string, URL or file

automatically flatten nested data frames into a single non-nested data frame

`fromJSON(txt, simplifyVector = TRUE, simplifyDataFrame = simplifyVector, simplifyMatrix = simplifyVector, flatten = FALSE, ...)`

`?jsonlite::toJSON`

Object to be encoded

how to encode data.frame objects

how to print NA values:
must be one of
'null' or 'string'

`toJSON(x, dataframe = c("rows", "columns", "values"), matrix = c("rowmajor", "columnmajor"), Date = c("ISO8601", "epoch"), POSIXct = c("string", "ISO8601", "epoch", "mongo"), factor = c("string", "integer"), complex = c("string", "list"), raw = c("base64", "hex", "mongo"), null = c("list", "null"), na = c("null", "string"), auto.unbox = FALSE, digits = 4, pretty = FALSE, force = FALSE, ...)`

© Jigsaw Academy Education Pvt Ltd

`fromJSON(URL, flatten = False)`

Using Web API:

Data from web

Web APIs

In computer programming, an application programming interface (API) specifies how some software components should interact with each other

What is a Web API?

A Web API uses HTTP requests to access information from Web-based software application

- ✓ Communication between applications is handled through a collection of standards and protocols
- ✓ Lot of companies allow users to freely access data from their websites by means of APIs
- ✓ **Examples:** Google APIs, Facebook API, Twitter API, Amazon API, NYTimes API

© Jigsaw Academy Education Pvt Ltd

Checklist for working with APIs:

Data from web

Checklist for working with APIs

- ✓ Read the API's documentation
- ✓ Create a developer account
- ✓ Gather account specific API key
- ✓ Identify data request calls limited per day
- ✓ Initiate authentication for data access
- ✓ See what kind of data you have access to
- ✓ See what kind of formats are available



Data from web

Web scraping in R

Majority of online data exists as web content such as blogs and news stories which are not easily accessible or downloadable

Advantages of web scraping:

- ✓ Access semi-structured or unstructured data formats
- ✓ Provide structure to the web content for analyzing in R

Web-scraping should always be a last resort:

- ✓ Constant changes made by the site owner
- ✓ More efforts involved in building the code

© Jigsaw Academy Education Pvt Ltd



Data from web

Web scraping using readLines() & RCurl

readLines() can be used for basic web scraping tasks and for more difficult procedures, RCurl functions would come to rescue

- ✓ **readLines():**
 - Normally used for reading data in R
 - Provides simple access to webpage source data
 - Requires use of regular expression for extracting structure
- ✓ **RCurl:**
 - Provides access to https web protocol
 - getURL() function helps in acquiring data from web page
 - Requires XML parsing to structure the extracted data

© Jigsaw Academy Education Pvt Ltd



Working with Web Data Part 2:

E/Data Science with R - RStudio

```

File Edit Code View Plots Session Build Debug Tools Help
File Source on Save Run Source
Working with web data.R
##using readLines function
data<-url("http://www.gutenberg.org/cache/epub/2701/pg2701.txt")
webText<-readLines(data,n=50)

##using read.table function
plants<-read.table("http://www.bio.ic.ac.uk/research/mjcrowtherbook/data/taxon.txt",
                     sep="",
                     header=1,
                     stringsAsFactors=F)

dim(plants)
str(plants)
head(plants)

```

Console | Data Science with R | R Script

```

> head(plants)
#> #> Petiole Internode Sepal Bract Petiole Leaf Fruit
#> 1 5.621498 29.48060 2.462107 18.20341 11.27910 1.128033 7.876151
#> 2 4.994617 28.36025 2.429321 17.65205 11.04084 1.197617 7.025416
#> 3 4.767505 27.25432 2.570947 19.40838 10.49072 1.003808 7.817479
#> 4 6.299446 25.92424 2.066051 18.37915 11.80182 1.614052 7.672492
#> 5 6.489375 25.21131 2.801583 17.31305 10.12159 1.813333 7.758443
#> 6 5.785868 25.52433 2.655643 17.07216 10.55816 1.955524 7.880880
#>

```

E/Data Science with R - RStudio

```

File Edit Code View Plots Session Build Debug Tools Help
File Source on Save Run Source
Working with web data.R
##Loading jsonlite library
library(jsonlite)

##Reading json files to R
customers<-fromJSON("customersJSON.txt")
class(customers)
head(customers)

##Writing json files from R
output<-toJSON(customers, pretty=TRUE)
cat(output)
writeLines(output,"customers_json.txt")


```

Console | Data Science with R | R Script

```

> head(customers)
#> #> Petiole Internode Sepal Bract Petiole Leaf Fruit
#> 1 5.621498 29.48060 2.462107 18.20341 11.27910 1.128033 7.876151
#> 2 4.994617 28.36025 2.429321 17.65205 11.04084 1.197617 7.025416
#> 3 4.767505 27.25432 2.570947 19.40838 10.49072 1.003808 7.817479
#> 4 6.299446 25.92424 2.066051 18.37915 11.80182 1.614052 7.672492
#> 5 6.489375 25.21131 2.801583 17.31305 10.12159 1.813333 7.758443
#> 6 5.785868 25.52433 2.655643 17.07216 10.55816 1.955524 7.880880
#>

```

E/Data Science with R - RStudio

```

File Edit Code View Plots Session Build Debug Tools Help
File Source on Save Run Source
Working with web data.R
library(jsonlite)

##Reading json files to R
customers<-fromJSON("customersJSON.txt")
class(customers)
head(customers)

##Writing json files from R
output<-toJSON(customers, pretty=TRUE)
cat(output)
writeLines(output,"customers_json.txt")

##Handling xml files
##Loading XML library
library(XML)

```

Console | Data Science with R | R Script

```

> customers<-fromJSON("customersJSON.txt")
> class(customers)
[1] "data.frame"
> head(customers)
  first_name last_name      city    county state  zip
1   James      Butt New Orleans     Orleans LA 70116
2 Josephine    Darakjy Brighton Livingston      MI 48116
3      Art      Venere Bridgeport Gloucester      NJ 8014
4     Lenna Paprocki Anchorage Anchorage      AK 99501
5   Donette     Foller Hamilton      Butler    OH 45011
6    Simona    Morasca Ashland      Ashland    OH 44805
#>

```

```

{
  "first_name": "James",
  "last_name": "Brett",
  "city": "New Orleans",
  "county": "Orleans",
  "state": "LA",
  "zip": 70116
},
{
  "first_name": "Josephina",
  "last_name": "Dobakly",
  "city": "Brighton",
  "county": "Livingston",
  "state": "MD",
  "zip": 48116
},
{
  "first_name": "Art",
  "last_name": "Vener",
  "city": "Bridgeport",
  "county": "Gloucester",
  "state": "NJ",
  "zip": 08114
},
{
  "first_name": "Lenna",
  "last_name": "Pospocki",
  "city": "Anchorage",
  "county": "Anchorage",
  "state": "AK",
  "zip": 99501
},
{
  "first_name": "Donetta"
}

```

Loading and Reading XML files :

First we will need to load the XML library

>library(XML)

>xmldata<- "sample.xml"

Then use the tree parser functionality to parse through the XML file that has been loaded

>xmlparse<- XMLTreeParse(xmldata)

>class(xmlparse)

After looking at the class, the class should be XMLDocument, we can look at the root node of the XML

>xmltop<- xmlroot(xmlparse)

Then to extract the values from the attributes of the nodes we will use XMLApply

>cust <- XMLApply(xmltop, function(X) XMLApply(x, XMLValue))

This will be a column by column output, but the end result we want to convert it into a data frame:

To convert it into data frame: the t function is used to transpose the dataset in columns to rows

Custoemrs_df<- data.frame(**t(customers)**, row.names=NULL)

=====

The next topic is about web scraping

```

##Web scraping using RCurl and XML packages
##website: https://stat.ethz.ch/pipermail/r-help/2009-January/date.html
##The site consists of January 2009 R help Archives sorted by date
##The goal is to extract Authors info and identify the biggest contributor

##Loading library
library(RCurl)
library(XML)

##Getting the html content from URL
htmlContent<-getURL("https://stat.ethz.ch/pipermail/r-help/2009-January/date.html",
  ssl.verifypeer = FALSE)

customers_df<-data.frame(readLines(htmlContent),row.names=NULL)
head(customers_df)

state      "DC"      "CA"      "ID"      "IN"      "NE"      "WA"      "FL"
zip       "20001"   "94945"   "83709"   "46514"   "69301"   "98104"   "32804"
> customers_df<-data.frame(t(customers),row.names=NULL)
> head(customers_df)
  first_name last_name    city    county state zip
1  James        Butt New Orleans Orleans LA 70116
2 Josephine Darakjy Brighton Livingston MI 48116
3 Art        Venere Bridgeport Gloucester NJ 8014
4 Lenna Paprocki Anchorage Anchorage AK 99501
5 Donette Poller Hamilton Butler OH 45011
6 Simona Morasca Ashland Ashland OH 44805

```

We can use Rcurl and XML packages for this functionality

```

##Getting the html content from URL
htmlContent<-getURL("https://stat.ethz.ch/pipermail/r-help/2009-January/date.html",
  ssl.verifypeer = FALSE)

##Parsing the html tree structure from html content
htmParsed<-htmlTreeParse(htmlContent,
  useInternal = TRUE)
authorsOutput<-unlist(xpathApply(htmParsed, '//i', xmlValue))
authorsOutput<-gsub("\n", "", authorsOutput)

##Converting the output to data frame object
output<-data.frame(table(authorsOutput))
output<-output[order(-output$Freq),]
head(output)

> authorsOutput[1:10]
[1] "Thu Jan 1 00:36:26 CET 2009" "Sat Jan 31 23:28:10 CET 2009" "George Chen\n"
[4] "Chris Poliquin\n"           "Ben Bolker\n"           "Felipe Carrillo\n"
[7] "Deepayan Sarkar\n"         "Barry Rowlingson\n"     "Carlos J. Gil Bellosta\n"
[10] "RON70\n"                  ""                      ""

> authorsOutput<-gsub("\n", "", authorsOutput)
> authorsOutput[1:10]
[1] "Thu Jan 1 00:36:26 CET 2009" "Sat Jan 31 23:28:10 CET 2009" "George Chen"
[4] "Chris Poliquin"           "Ben Bolker"           "Felipe Carrillo"
[7] "Deepayan Sarkar"         "Barry Rowlingson"     "Carlos J. Gil Bellosta"
[10] "RON70"                  ""

```

Use the get URL to load the HTML page to a content on R

>htmlContent<-getURL("http://www.xyz.com/test.htm", ssl.verifypeer = FALSE)

Then use HTMLTreeParse to parse the HTML page content using the internal nodes that are already present

>htmParsed<-HTMLParse(htmlContent, useInternal =TRUE)

The parsed output can now be ready for the specific nodes using the node indicators and XPathApply function

>authors <- unlist(xpathApply(htmParsed, '//i', xmlValue))

If we want to extract the href, we will use the @ symbol i.e. '//i/@href'

Once the extract is completed the next step is to remove unwanted spaces or characters from the extract

>authors<-gsub('\\n', "", authors)

Finally we will convert the output into a structured dataframe

>authorsdf<- data.frame(table(authors))

The table function used above is used for calculating frequency of the data.

```

E/Data Science with R - RStudio
File Edit Code View Plots Session Build Debug Tools Help
Working with web data.R
Source on Save https://stat.ethz.ch/pipermail/r-help/2009-january/date.html
ssl.verifyPeer = FALSE

#Parses the html tree structure from html content
htmlParsed<-htmlTreeParse(htmlContent,
                           useInternal = TRUE)
authorsOutput<-unlist(xpathApply(htmlParsed, '//i', xmlValue))
authorsOutput<-gsub('\n', '', authorsOutput)

#Converting the output to data frame object
output<-data.frame(table(authorsOutput))
output<-output[order(-output$Freq),]
head(output)

[1] "Deepayan Sarkar" "Barry Rowlingson" "Carlos J. Gil Bellosta"
[10] "RON70" > output<-data.frame(table(authorsOutput))
> head(output)
  authorsOutput Freq
1   (Ted Harding)  9
2 "Wärning, Tim (Ulf)"  5
3 "Ocaña Riola, Ricardo"  1
4   Ooalastair00  1
5     A Van Dyke  2
6 Aaron M. Swoboda  1

```

In order to put these in descending order we use the following:

Output<-output[order(-output\$freq),]



The screenshot shows a Microsoft Excel spreadsheet titled 'Data Dictionary'. The table has three columns: 'Variable name', 'Description', and 'Data type'. The data includes:

Variable name	Description	Data type
Age	Age in the year	numeric
Sex	Value ("0" = male, "1" = female).	factor
CP	Chest pain type ("1" = typical angina, "2" = atypical angina, "3" = non-angina pain, "4" = asymptomatic).	factor
trestbps	Resting blood pressure (in mm Hg on admission to the hospital)	numeric
Chol	Serum cholesterol in mg/dl	numeric
Fbs	Indicator of whether fasting blood sugar was > 120 mg/dl ("1" = yes, "0" = no)	factor
restecg	Resting electrocardiographic results ("0" = normal, "1" = ST-T wave abnormality, "2" = probable or definite left ventricular hypertrophy).	factor
thalach	Maximum heart rate achieved	numeric
exang	Indicator of whether the angina is exercise induced ("1" = yes, "0" = no).	factor
oldpeak	ST depression induced by exercise relative to rest.	numeric
slope	The slope of the peak exercise ST segment ("1" = up sloping, "2" = flat, "3" = down sloping).	factor
ca	Number of major vessels colored by fluoroscopy	numeric
thal	Summary of heart condition ("3" = normal, "6" = fixed defect, "7" = reversible defect).	factor
DV	The Disease Diagnosis" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 1(Presence).	factor

Steps

- Read the dataset
- Check if all the columns match the necessary data types
- Understand each variable and create a data Dictionary
- Understand the summary of the data
- Look for wrong or suspicious values

© Jigsaw Academy Education Pvt Ltd

We can use the `as.factor` function to convert a variable in a `Data.frame` to a factor after checking the structure of the `dataframe`

```
## $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
## $ slope   : int  3 2 2 3 1 1 3 1 2 3 ...
## $ ca      : Factor w/ 5 levels "?","0","1","2",..: 2 5 4 2 2 2 4 2 3 2 ...
## $ thal    : Factor w/ 4 levels "?","3","6","7": 3 2 4 2 2 2 2 4 4 ...
## $ num     : int  0 2 1 0 0 0 3 0 2 1 ...
## $ dv      : int  0 0 0 0 0 0 0 0 0 0 ...

## 1=has heart disease, 0=no heart disease

#Looking at the summary of the data for numeric variables
summary(HD)

##      Age          Sex          cp          trestbps
##  Min. :29.00  Min. :0.0000  Min. :1.000  Min. : 94.0
##  1st Qu.:48.00  1st Qu.:0.0000  1st Qu.:3.000  1st Qu.:120.0
##  Median :56.00  Median :1.0000  Median :3.000  Median :130.0
##  Mean   :54.44  Mean   :0.6799  Mean   :3.158  Mean   :131.7
##  3rd Qu.:61.00  3rd Qu.:1.0000  3rd Qu.:4.000  3rd Qu.:140.0
##  Max.  :77.00  Max.  :1.0000  Max.  :4.000  Max.  :200.0
##      chol         fbs         restecg        thalach
##  Min. :126.0  Min. :0.0000  Min. :0.0000  Min. : 71.0
##  1st Qu.:211.0 1st Qu.:0.0000  1st Qu.:0.0000  1st Qu.:133.5
##  Median :241.0  Median :0.0000  Median :1.0000  Median :153.0
##  Mean   :246.7  Mean   :0.1485  Mean   :0.9901  Mean   :149.6
##  3rd Qu.:275.0 3rd Qu.:0.0000  3rd Qu.:2.0000  3rd Qu.:166.0
##  Max.  :564.0  Max.  :1.0000  Max.  :2.0000  Max.  :202.0
##      exang        oldpeak        slope        ca        thal
##  Min. :0.0000  Min. :0.00  Min. :1.000 ?: 4 ?: 2
##  1st Qu.:0.0000 1st Qu.:0.00  1st Qu.:1.000 0:176 3:166
##  Median :0.0000  Median :0.80  Median :2.000  1: 65 6: 18
##  Mean   :0.3227  Mean   :1.08  Mean   :1.601 ? 78 3:117
```

Run a summary to get the descriptive stats of these variables

For categorical values, use the `table` function to get a frequency distribution