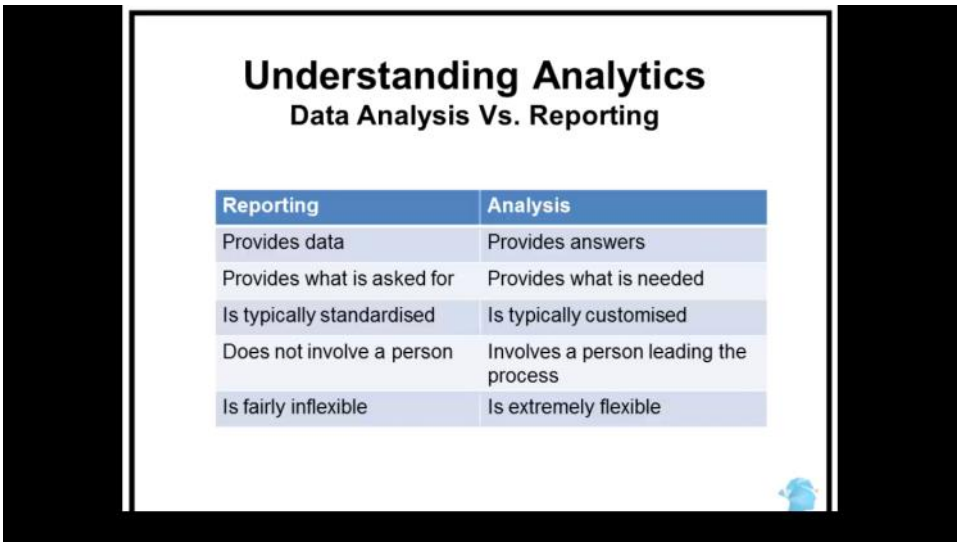
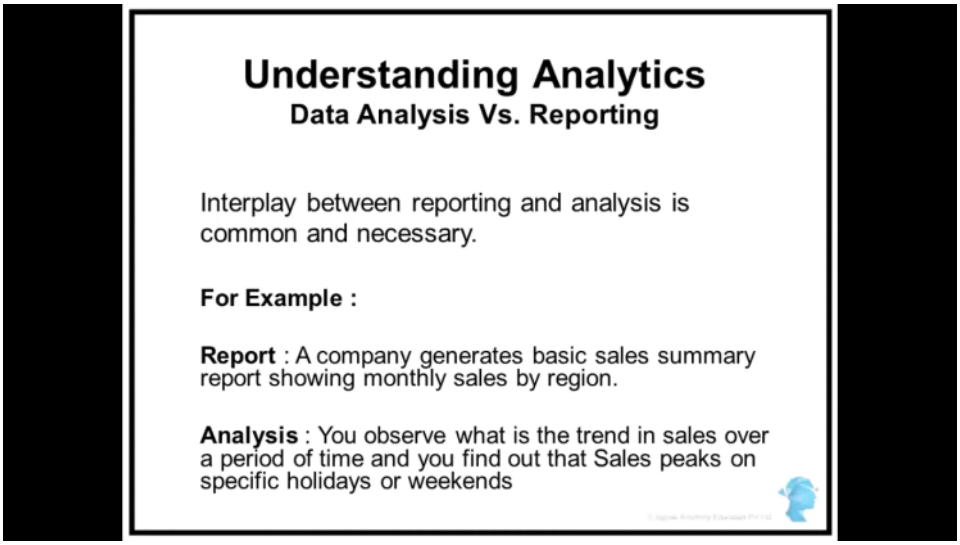
Introduction to R

Monday, August 22, 2016 7:05 PM

Understanding Analytics





Understanding Analytics
Features of basic and advanced analytics

Basic Analytics	Advanced Analytics
Not sure what you have? But you think something is of value ?	What happened, what caused it to happen, when it happened, and what the impact was?
➢ Slicing and Dicing of the data ➢ Monitor large volumes of data in real time ➢ Anomaly Identification	➢ 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:

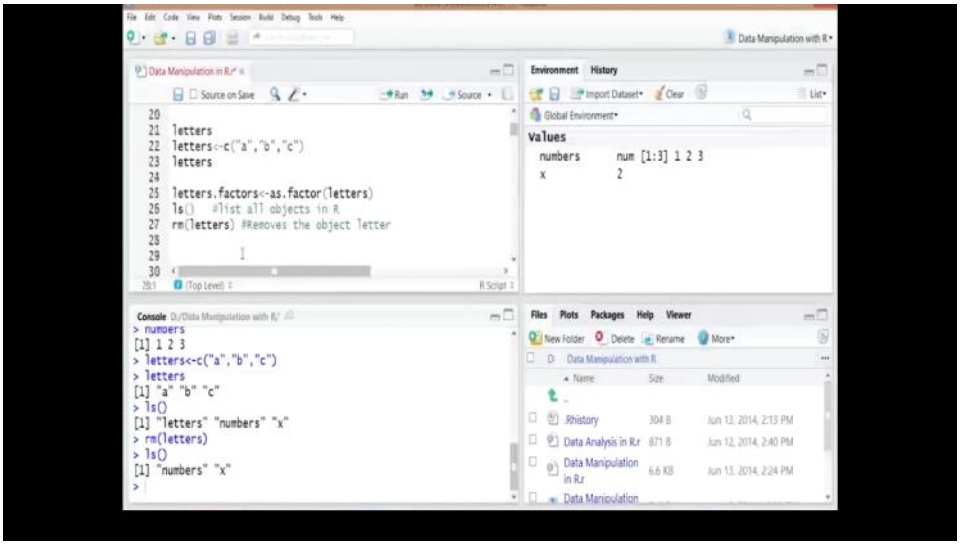
- Statistics
- Data and Text mining
- Data Visualization
- Forecasting
- Optimization
- Model Management and Deployment
- Quality Improvement

Analytical Approaches and Tools

Popular Analytical Tools

Comparisons

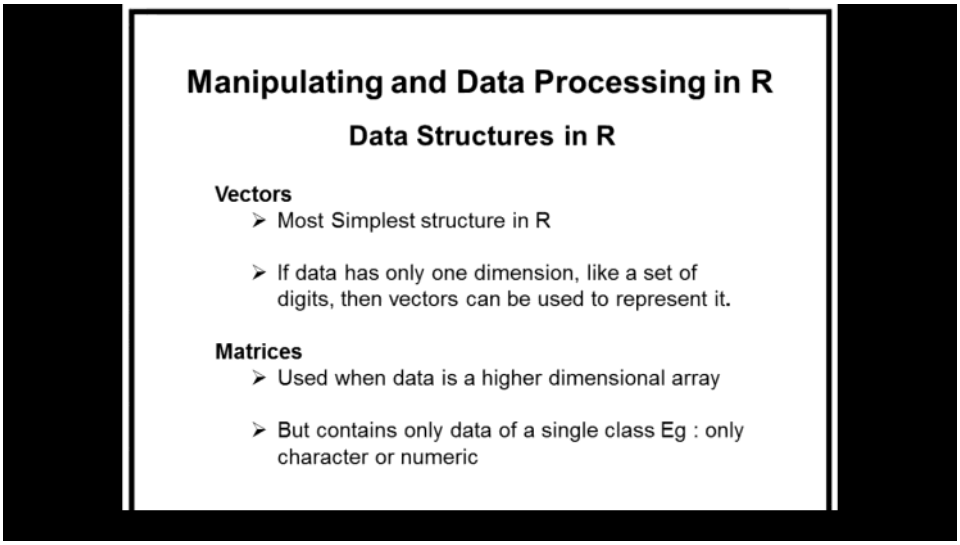
	SAS	SPSS	R
User Interface	Has the SAS Enterprise Guide	Graphic interface with scrolling menus	R Graphic interfaces like Rattle, Rstudio, etc
Decision Trees	Expensive, Purchase SAS Miner	Less Expensive, Does not require Data mining suite, offers a wide variety of algorithms	Free, Does not offer many tree 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)

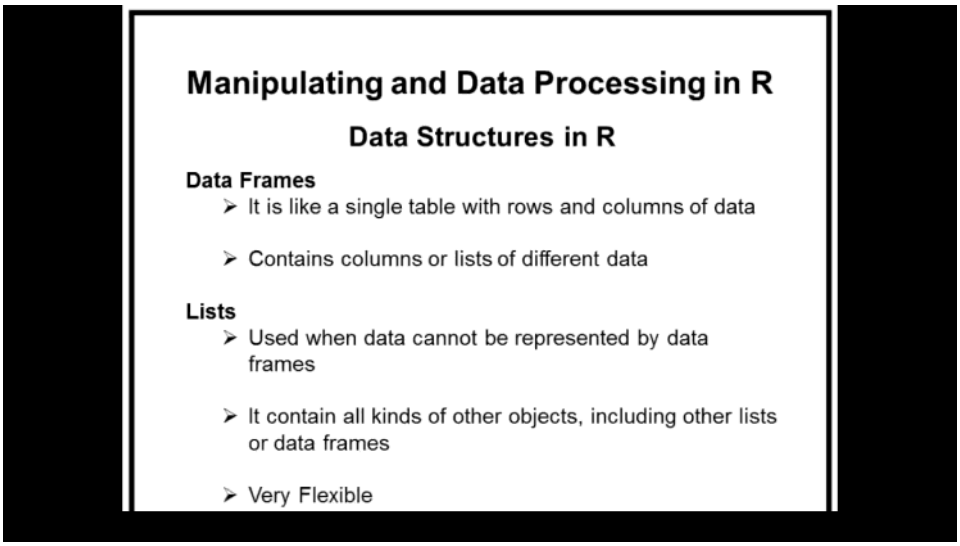


The ls() command is used to list all the objects created in R so far
The rm(object name) is used to remove the object

Manipulating objects and saving and loading an object

Data Structure in R:





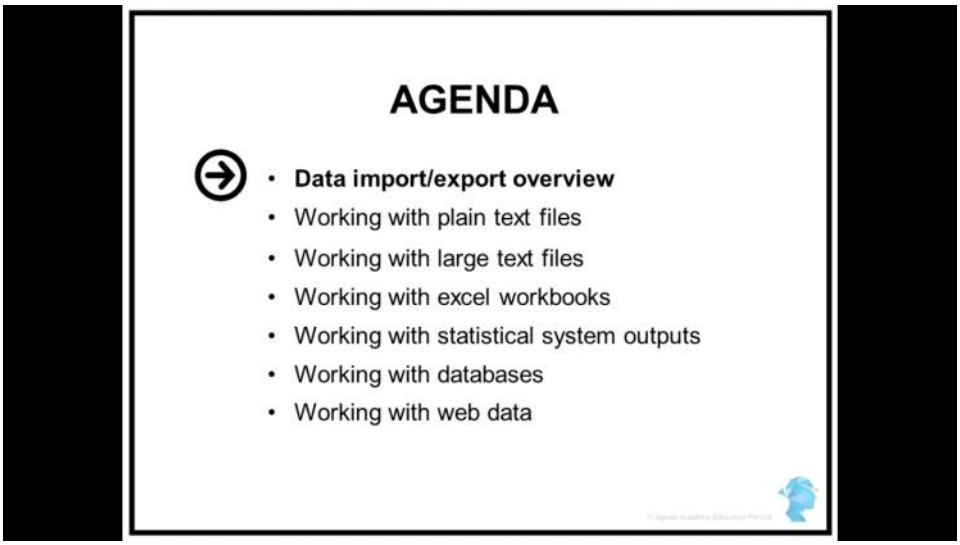
Exploring a dataset in R:

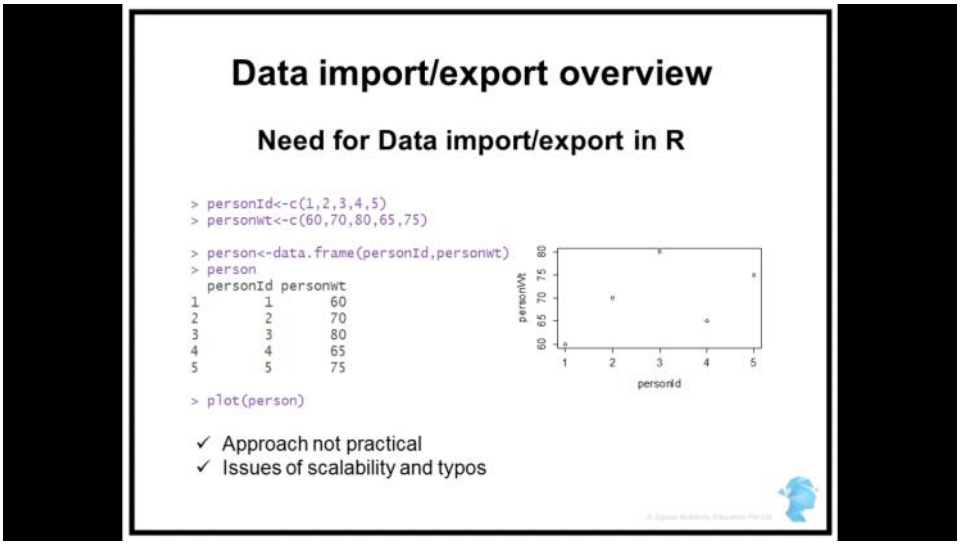
To load a dataset , that is provided in R, you need to access the dataset library

- 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 seperated
  - 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





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	60
2	70
3	80
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

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.

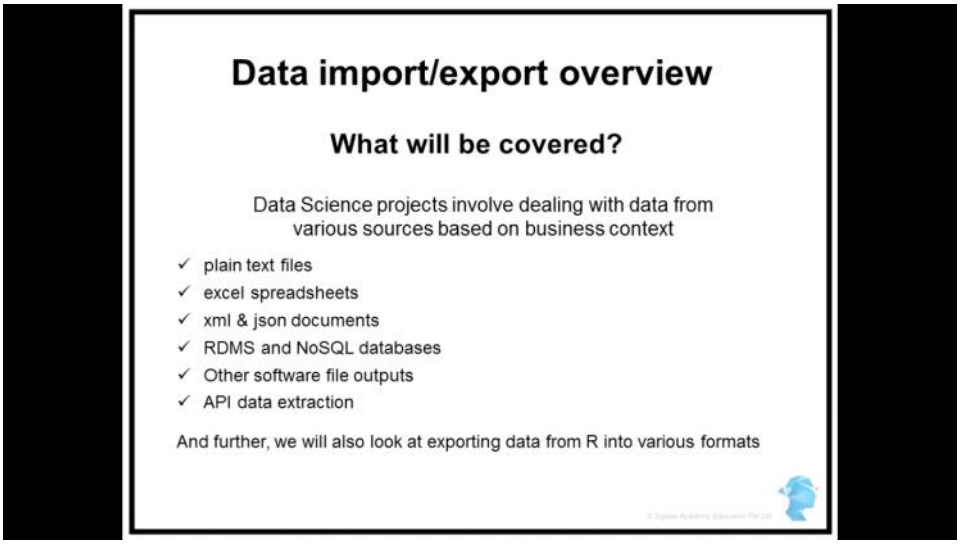
To deal with different file formats, often data can be first exported as a text file like csv and then imported into R

Data import/export overview

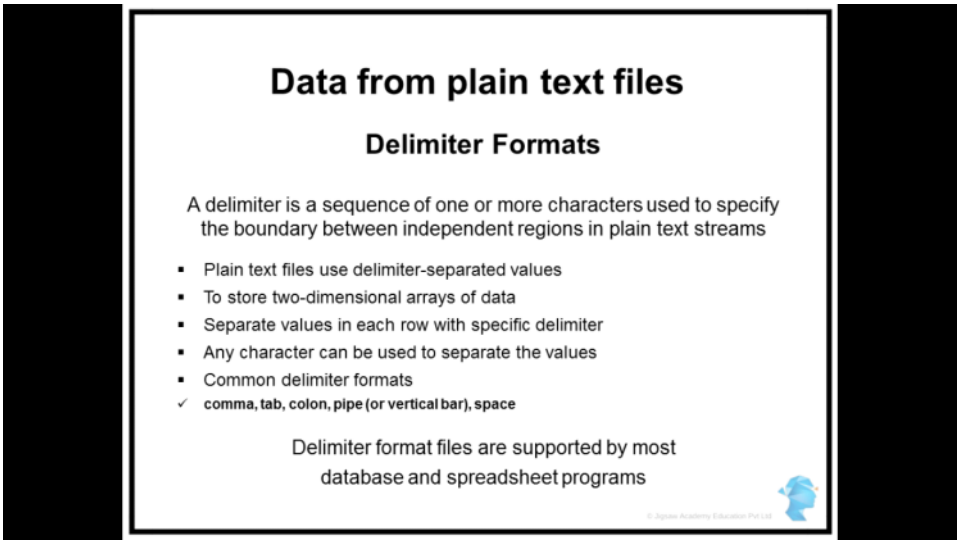
Checks before Data Import

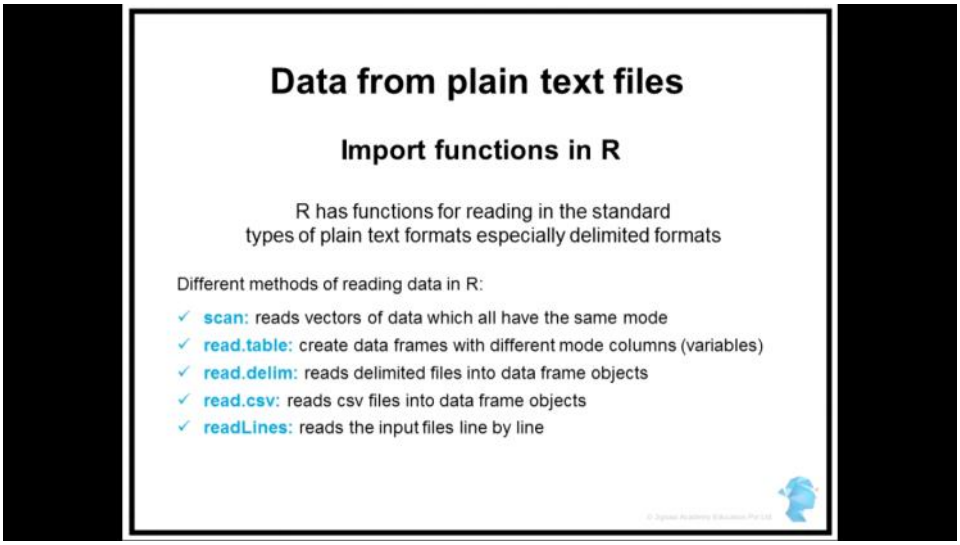
When reading data from text files, it is necessary to know and understand more about the inherent characteristics

- 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



Working with Plain text files:



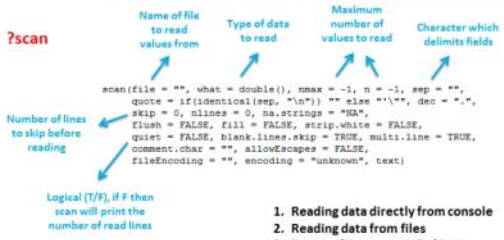


Data from plain text files

scan function

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

?scan



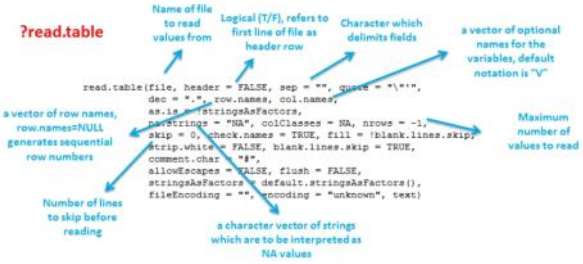
1. Reading data directly from console
2. Reading data from files
3. Returns list or vector R objects

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

?read.table

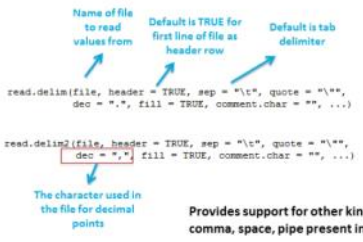


Data from plain text files

read.delim function

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

?read.delim



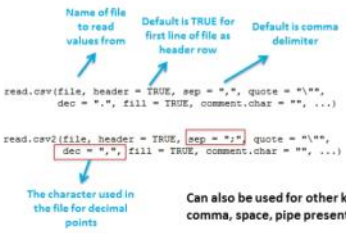
Provides support for other kind of delimiters like comma, space, pipe present in the text files

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

?read.csv



Can also be used for other kind of delimiters like comma, space, pipe present in the text files

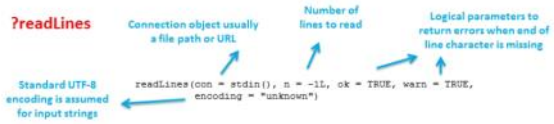
Data from plain text files

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

?readLines



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 supressed 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

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

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= ""

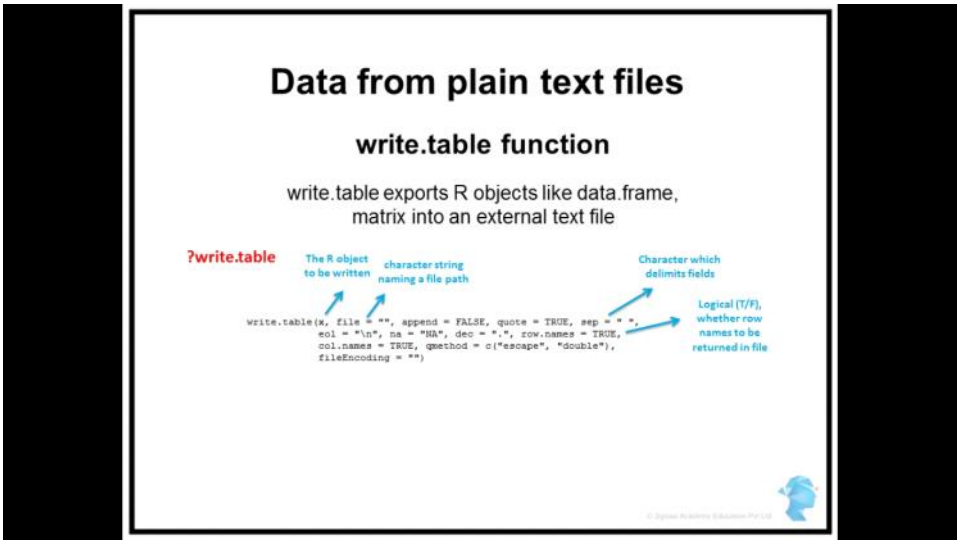
Data from plain text files

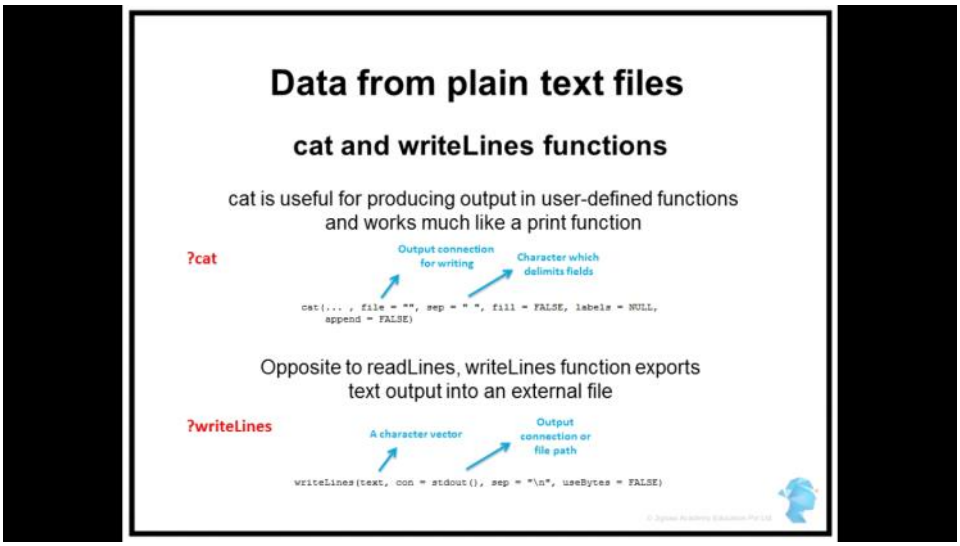
Export functions in R

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()





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 tpyes 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 eentire 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 fo 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 :

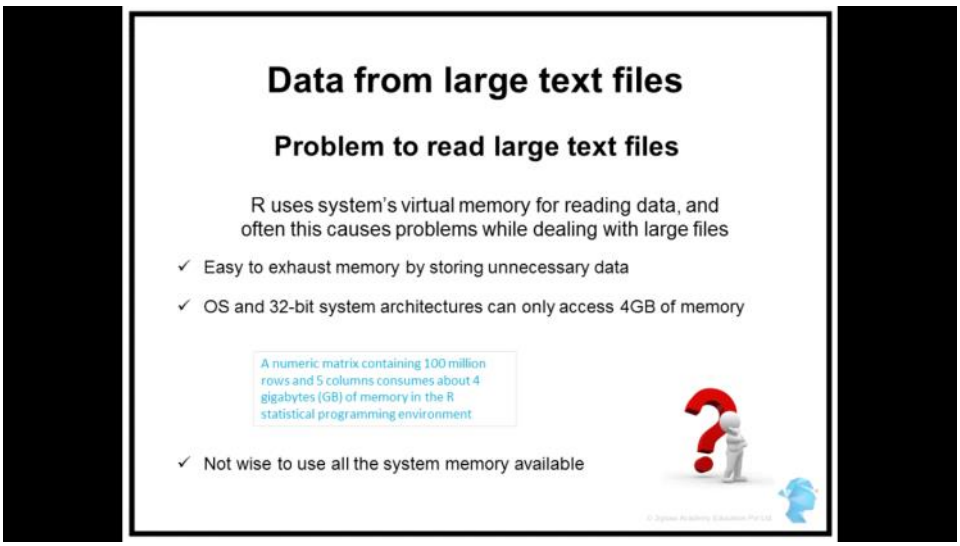
Write.data(customers, "custoemrs_new.csv", sep=",", row.names=F)

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

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**

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

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

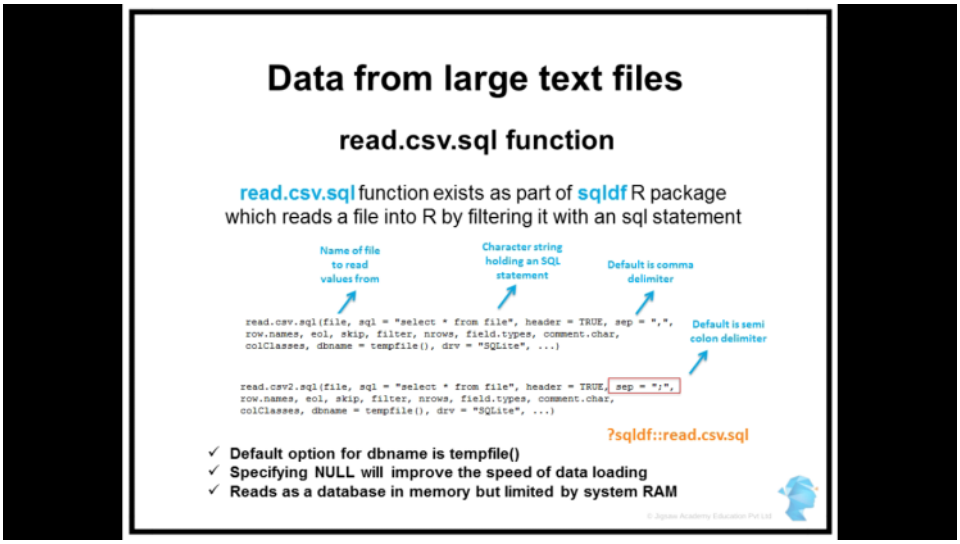
Character which delimits fields

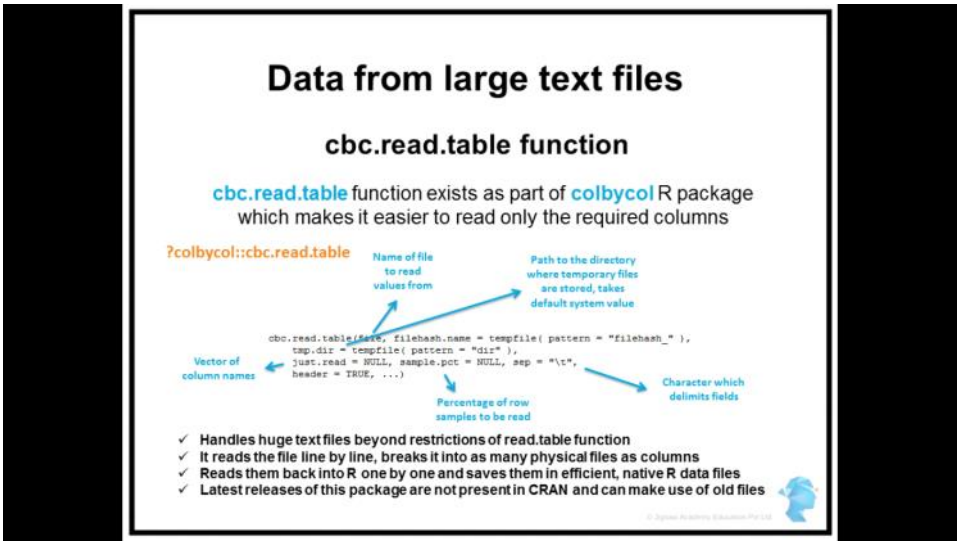
Number of rows to be read

?data.table::fread

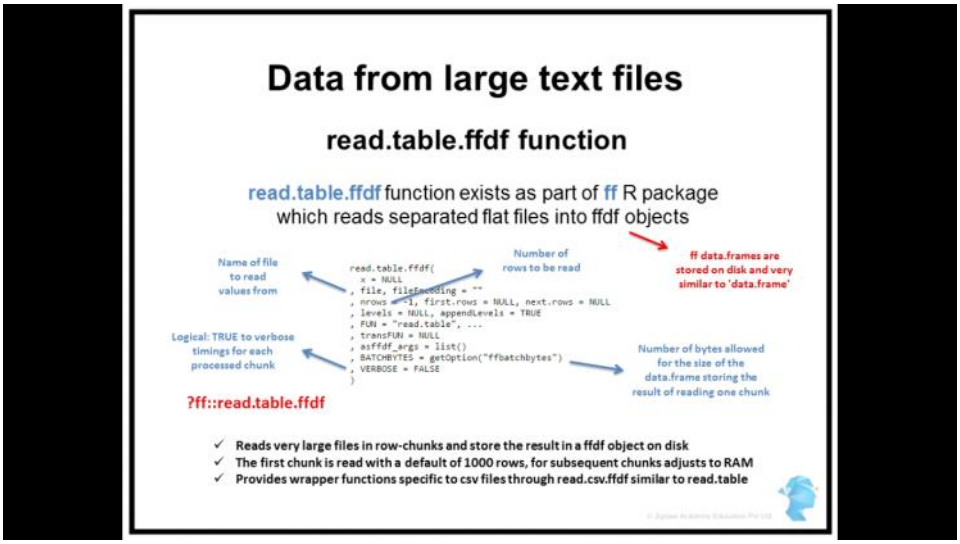
```
fread(input, sep="auto", sep2="auto", nrows=-1L, header="auto", na.strings="NA",
stringsAsFactors=FALSE, verbose=getOption("datatable.verbose"), autostart=30L,
skip=-1L, select=NULL, drop=NULL, colClasses=NULL,
integer64=getOption("datatable.integer64"),          # default: "integer64"
showProgress=getOption("datatable.showProgress"),    # default: TRUE
data.table=getOption("datatable.fread.datatable")    # default: TRUE
)
```

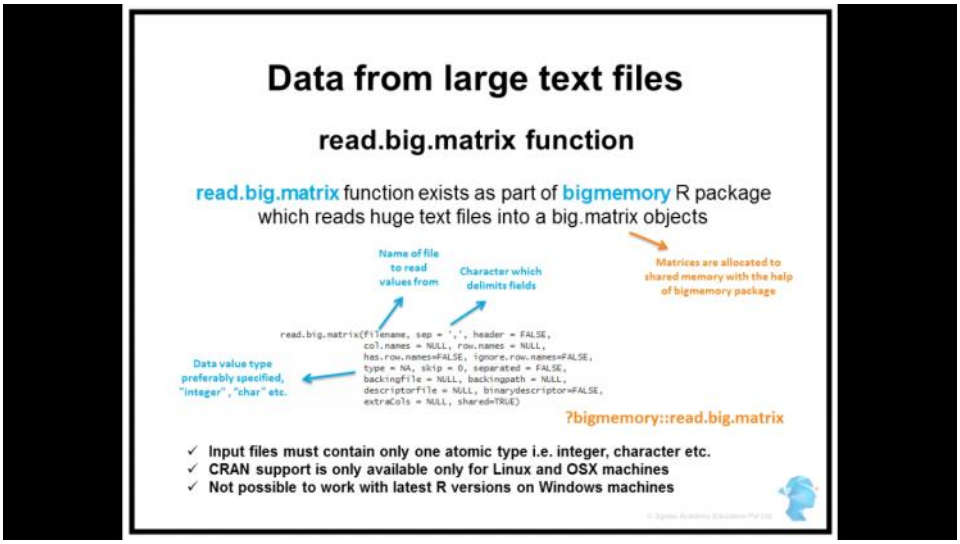
- ✓ **All controls such as sep, colClasses and nrows are automatically detected**
- ✓ **Integer types are also detected and read directly**
- ✓ **Dates are read as character which can be converted later**

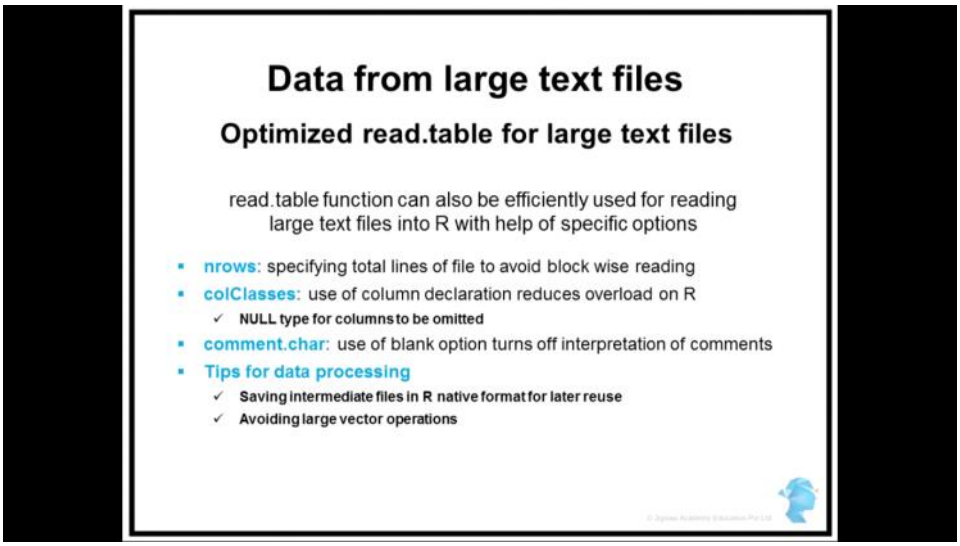


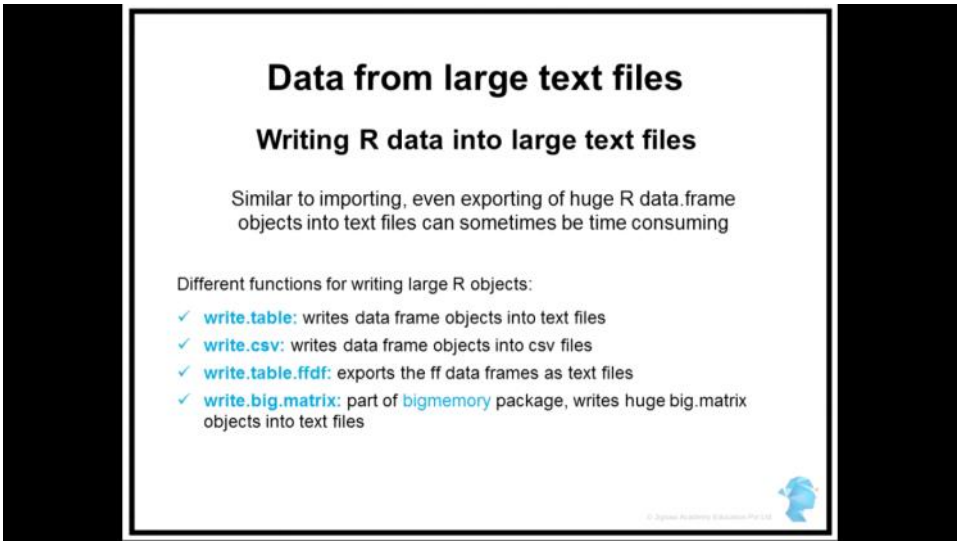


The colbycol package is not present in the latest CRAN servers, so need to follow old approach.







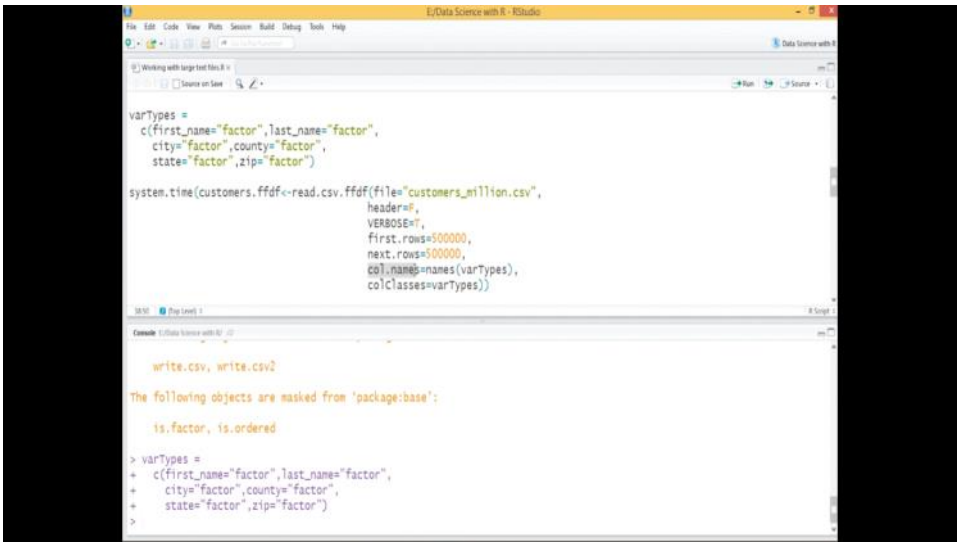


Working with large text files - part 2 - writing to files

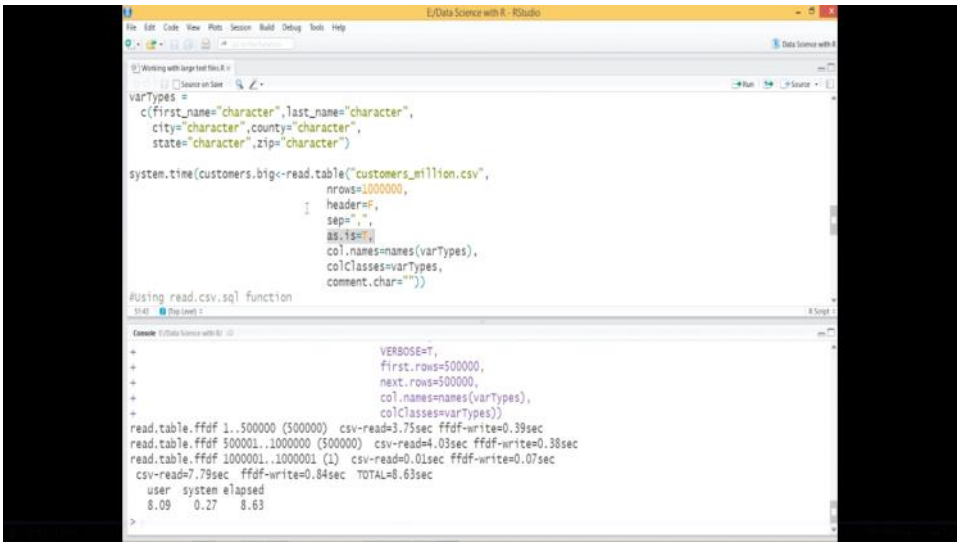
To find the processing time for loading the file:

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

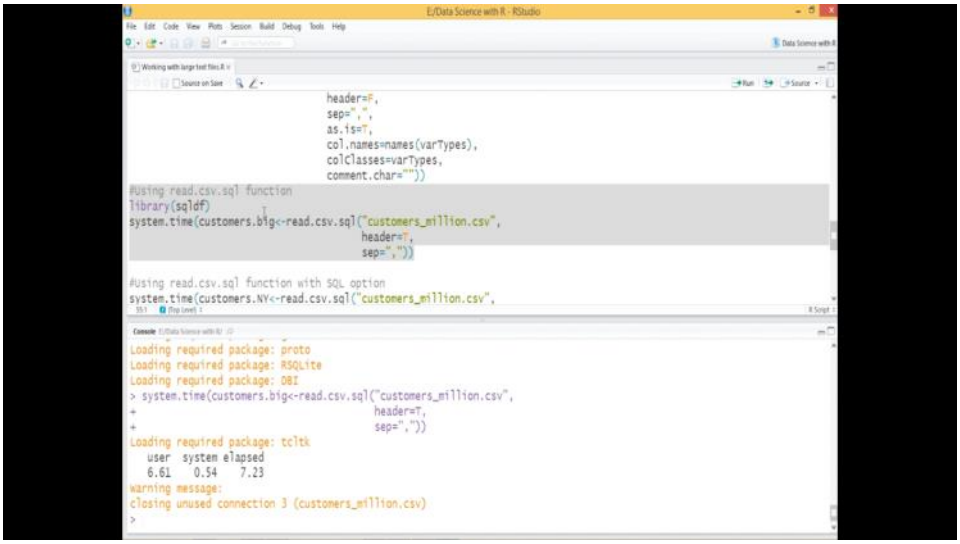
System.time(customer.big<-fread("customer_million.csv", header = T, sep = ","))



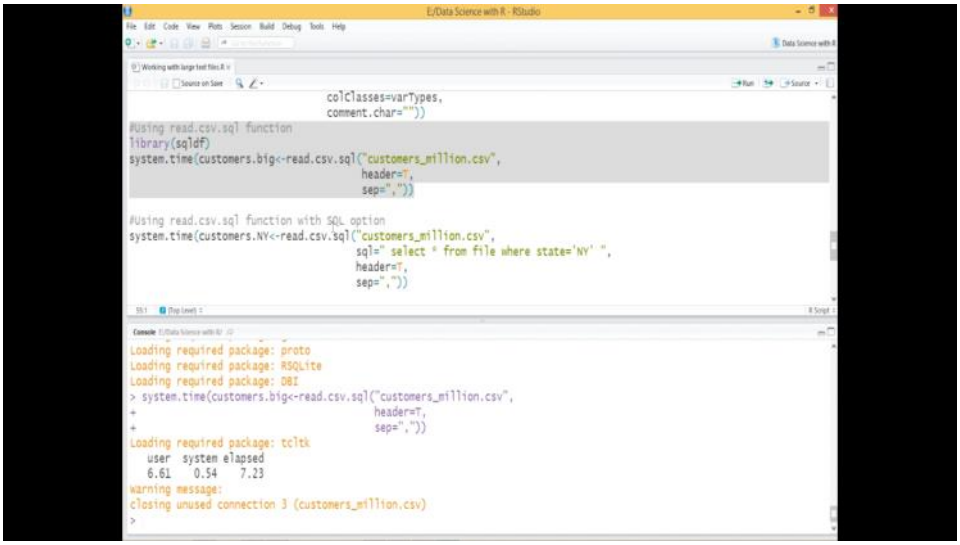
Customers.ffdf <- Read.csv.ffdf("customer_million.csv", header = F, verbose = T, first.rows=500000, next.rows = 500000, col.names=names(varTypes), colCLasses=varTypes)

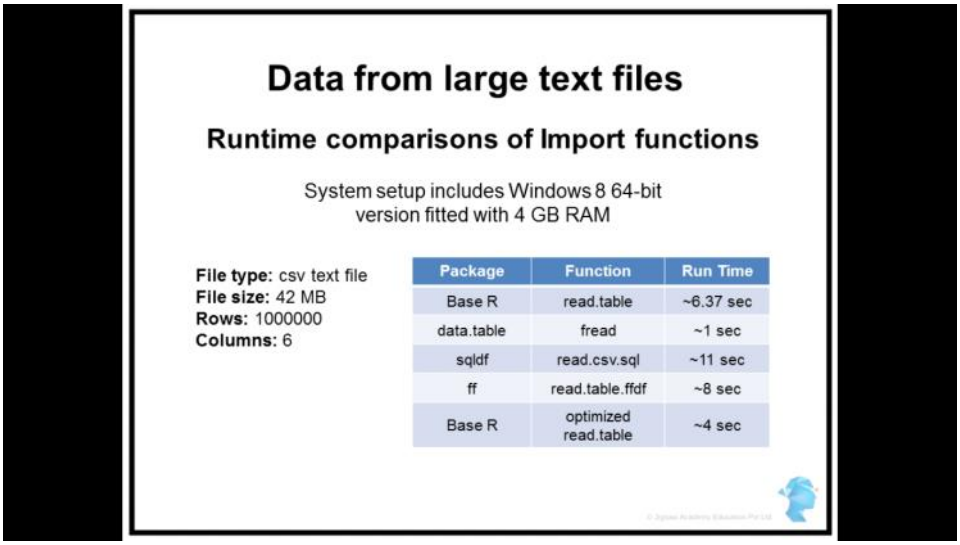


Using SQLDf package:

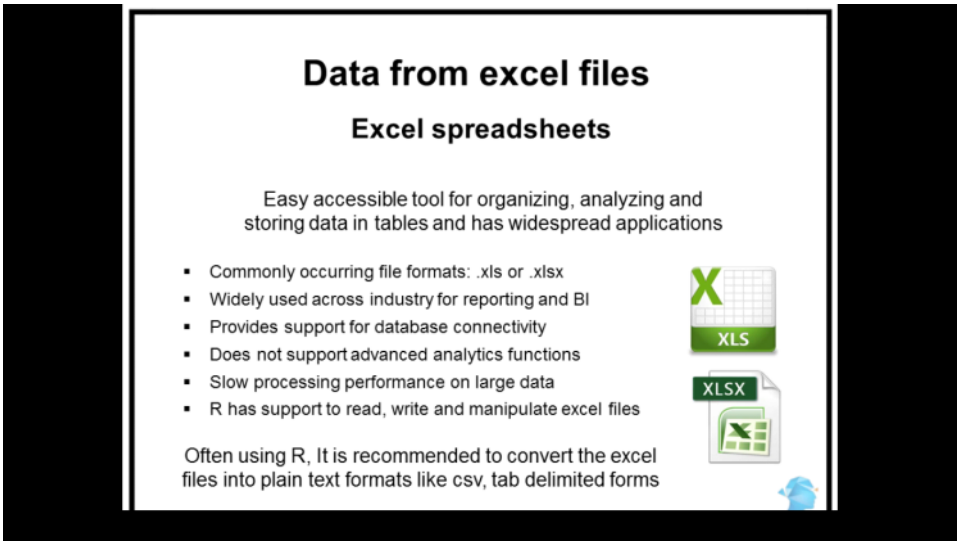


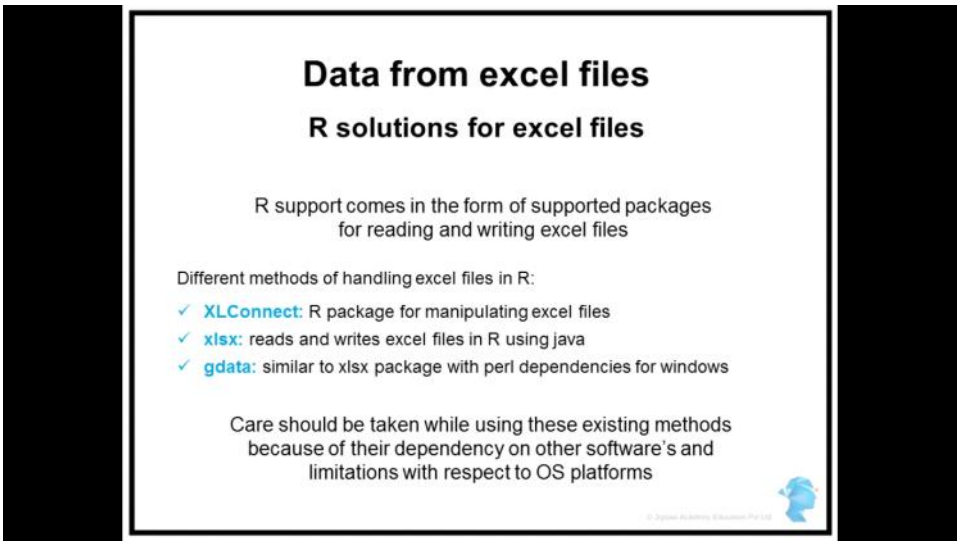
Using the SQL statement to filter data to load:

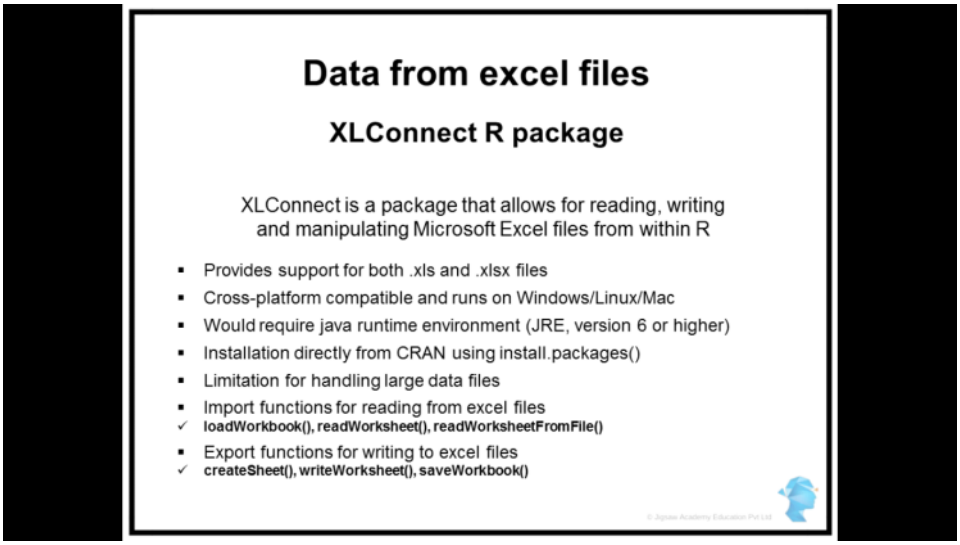




Working with Excel Workbooks:





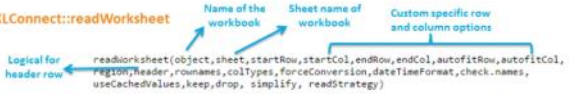


Data from excel files

XLConnect: Reading data

?XLConnect::loadworkbook

?XLConnect::readWorksheet

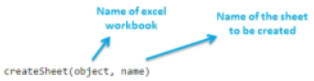


?XLConnect::readWorksheetFromFile

Data from excel files

XLConnect: Writing data

?XLConnect::createSheet



?XLConnect::writeWorksheet

?XLConnect::saveWorkbook

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()**

Data from excel files

xlsx: Reading data

?xlsx::read.xlsx

Name of file to read from

Number representing the sheet

Logical, if T converts output into data frame object

Logical for header row

```
read.xlsx(file, sheetIndex, sheetName=NULL, rowIndex=NULL,
  startRow=NULL, endRow=NULL, colIndex=NULL,
  as.data.frame=TRUE, header=TRUE, colClasses=NA,
  keepFormulas=FALSE, encoding="unknown", ...)
```

?xlsx::read.xlsx2

character string with the sheet name

Index of starting row

numeric vector indicating the cols to be extracted

```
read.xlsx2(file, sheetIndex, sheetName=NULL, startRow=1,
  colIndex=NULL, endRow=NULL, as.data.frame=TRUE, header=TRUE,
  colClasses="character", ...)
```

Data from excel files

xlsx: Writing data

?xlsx::write.xlsx

data.frame to be written

Name of file to write into

Name of the sheet in workbook

```
write.xlsx(x, file, sheetName="Sheet1",
  col.names=TRUE, row.names=TRUE, append=FALSE, showNA=TRUE)

write.xlsx2(x, file, sheetName="Sheet1",
  col.names=TRUE, row.names=TRUE, append=FALSE, ...)
```

- Workbook related functions

✓ **createWorkbook():** creates empty workbook object
✓ **loadWorkbook():** loads a workbook from file
✓ **addDataFrame():** adds a data.frame to sheet in workbook
✓ **saveWorkbook():** save an existing workbook to an excel file

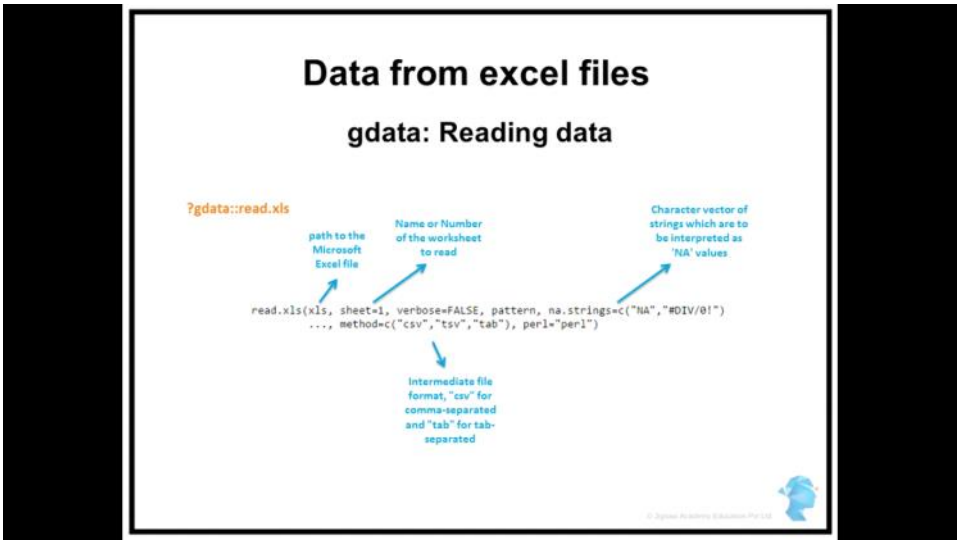
Data from excel files

gdata R package

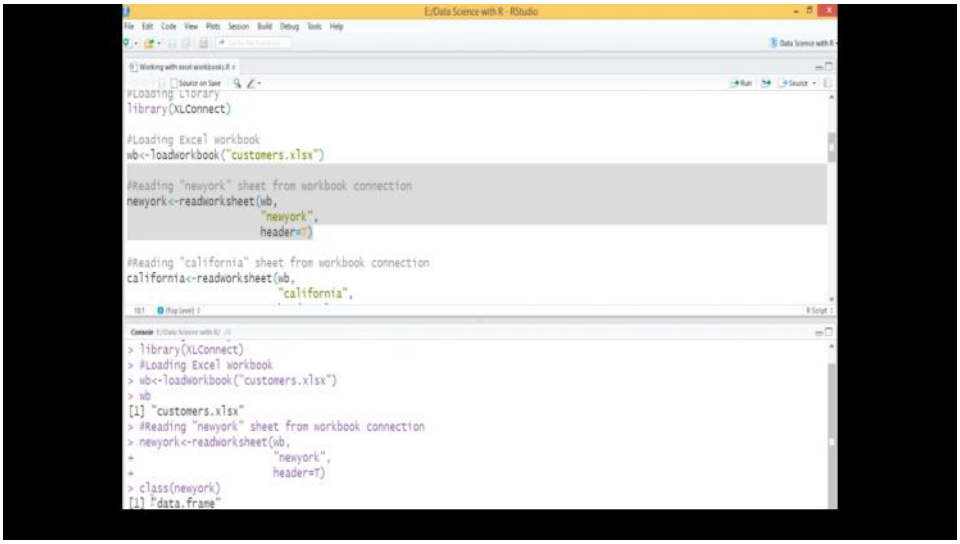
gdata is a general package that provides various R programming tools for data manipulations tasks

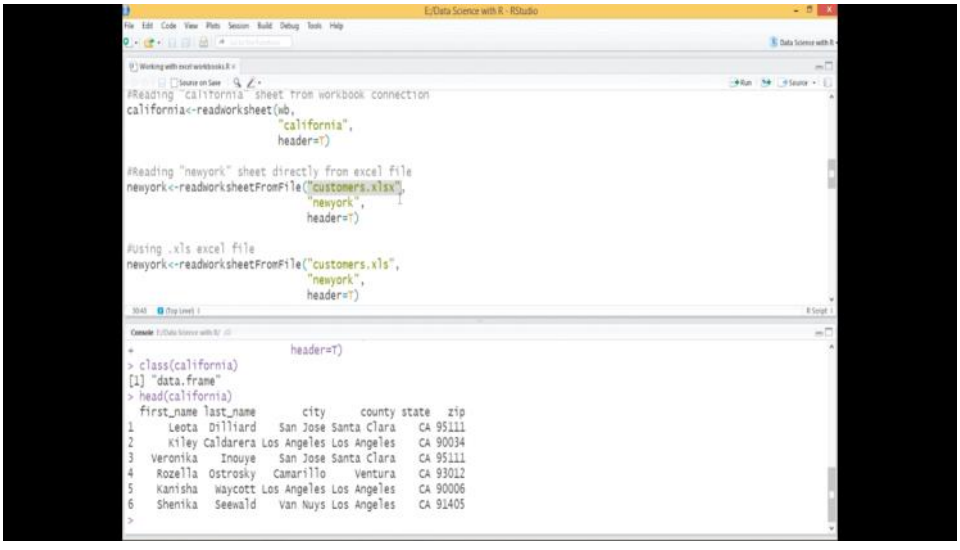
- Provides support for both .xls and .xlsx files
- Cross-platform compatible and runs on Windows/Linux/Mac
- Requires additional perl libraries on Windows systems
- Installation directly from CRAN using install.packages()
- Supports only import functions for excel files
- Import functions for reading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

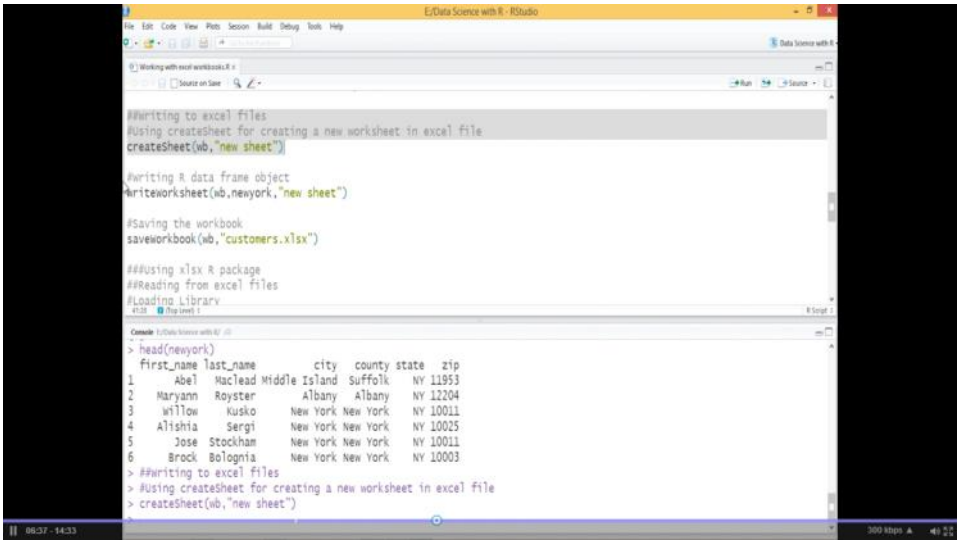
✓ **read.xls()**

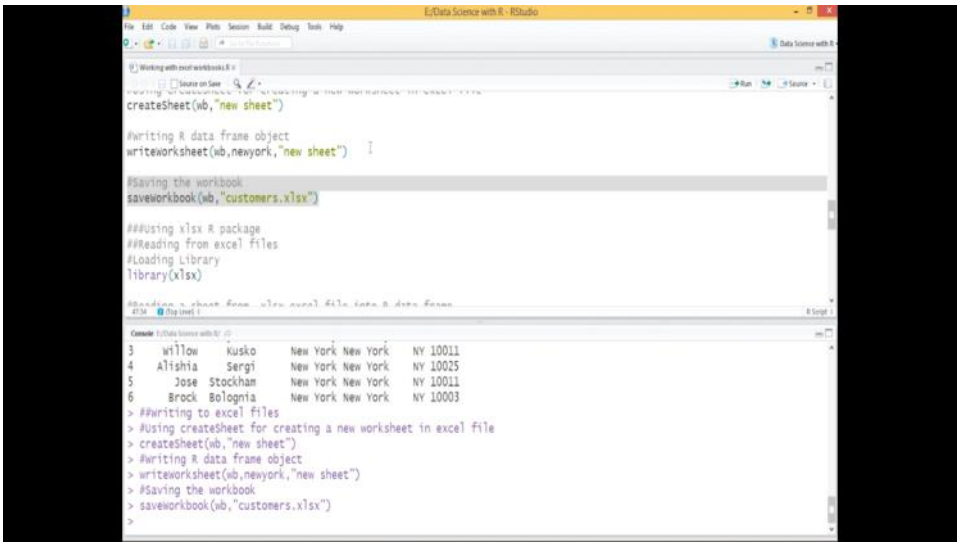


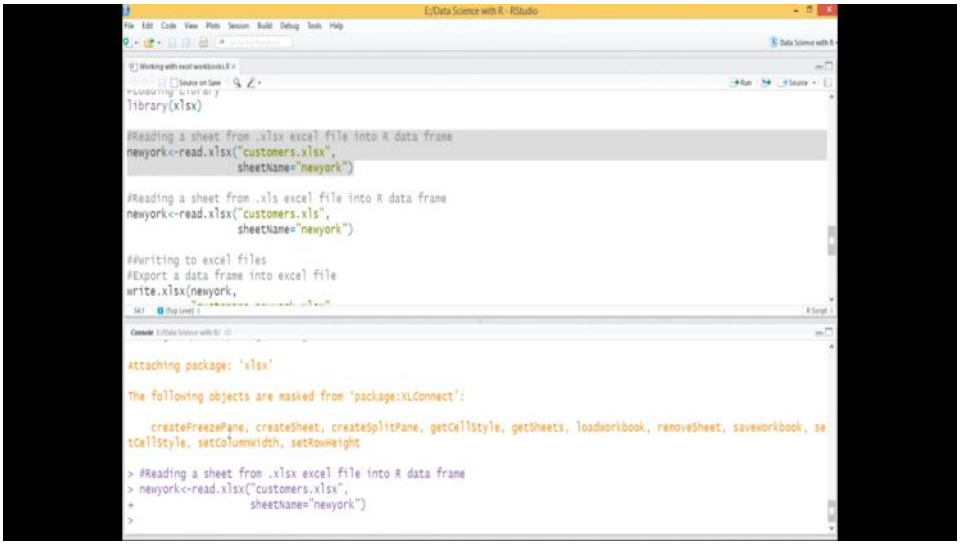
Working with Excel Workbooks Part 2:

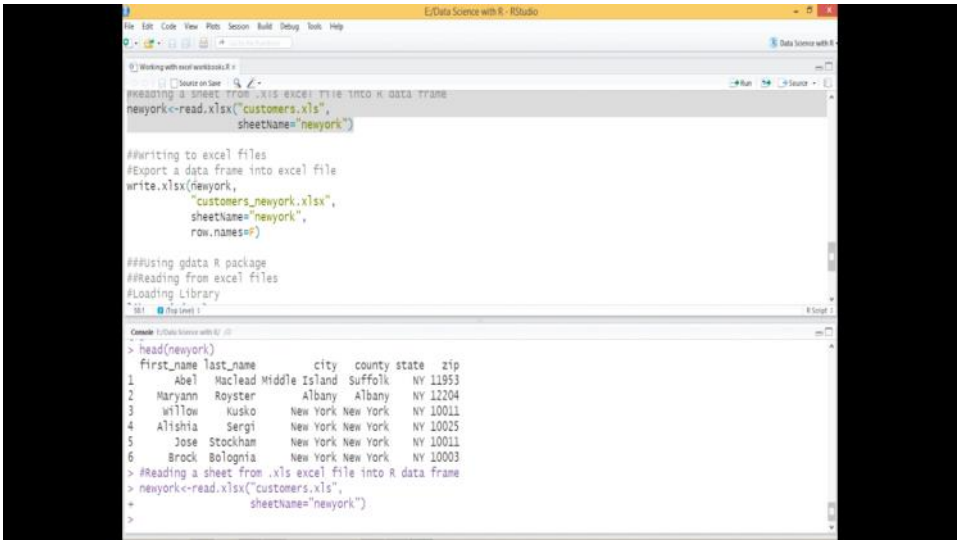




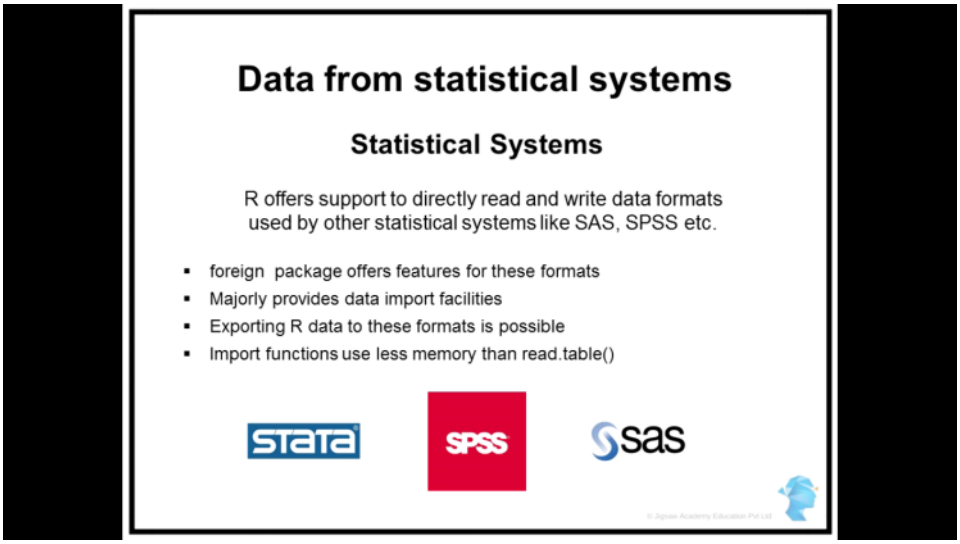

E:/Data Science with R - RStudio
#Writing to excel files
#Using createSheet for creating a new worksheet in excel file
createSheet(wb,"new sheet")
#writing R data frame object
writeworksheet(wb,newyork,"new sheet")
#Saving the workbook
saveworkbook(wb,"customers.xlsx")
##Using xlsx R package
##Reading from excel files
#Loading Library
> head(newyork)
first_name last_name city county state zip
1 Abel Maclead 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
> ##Writing to excel files
> #Using createSheet for creating a new worksheet in excel file
> createSheet(wb,"new sheet")

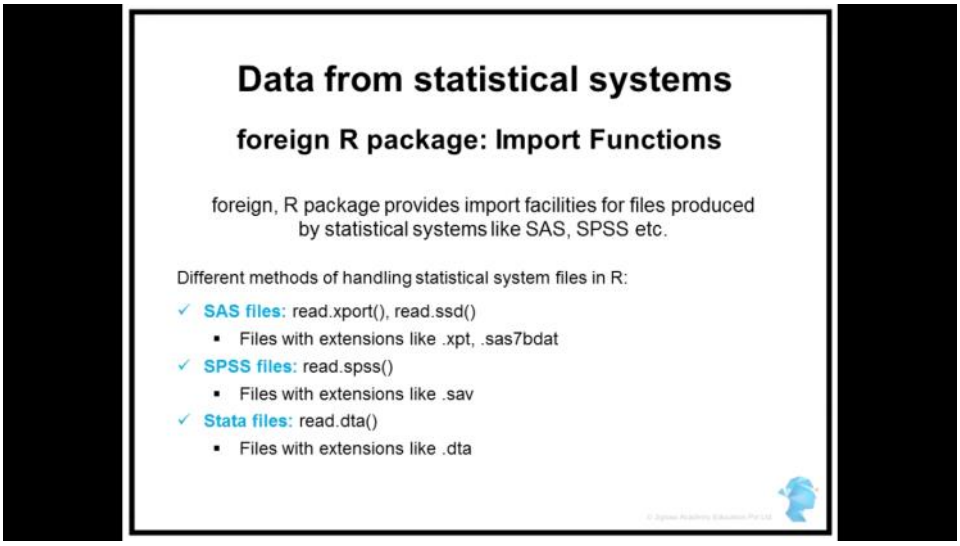

E:/Data Science with R - RStudio
createSheet(wb,"new sheet")
#writing R data frame object
writeworksheet(wb,newyork,"new sheet")
#Saving the workbook
saveworkbook(wb,"customers.xlsx")
##Using xlsx R package
##Reading from excel files
#Loading Library
library(xlsx)
3 Willow Kusko New York New York NY 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
> ##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")


E:/Data Science with R - RStudio
library(xlsx)
#Reading a sheet from .xlsx excel file into R data frame
newyork<-read.xlsx("customers.xlsx",
sheetName="newyork")
#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,
Attaching package: 'xlsx'
The following objects are masked from 'package:XLConnect':
createFreezePane, createSheet, createSplitPane, getCellStyle, getSheets, loadWorkbook, removeSheet, saveWorkbook, se
tCellStyle, setColumnWidth, setRowHeight
> #Reading a sheet from .xlsx excel file into R data frame
> newyork<-read.xlsx("customers.xlsx",
sheetName="newyork")



Working with Statistical System Formats:





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.ssd(libname, sectionnames,
   tmpXport=tempfile(), tmpProgLoc=tempfile(), sascmd="sas")
```

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.missings = to.data.frame)
```

Data from statistical systems

foreign: Reading Stata files

?foreign::read.dta

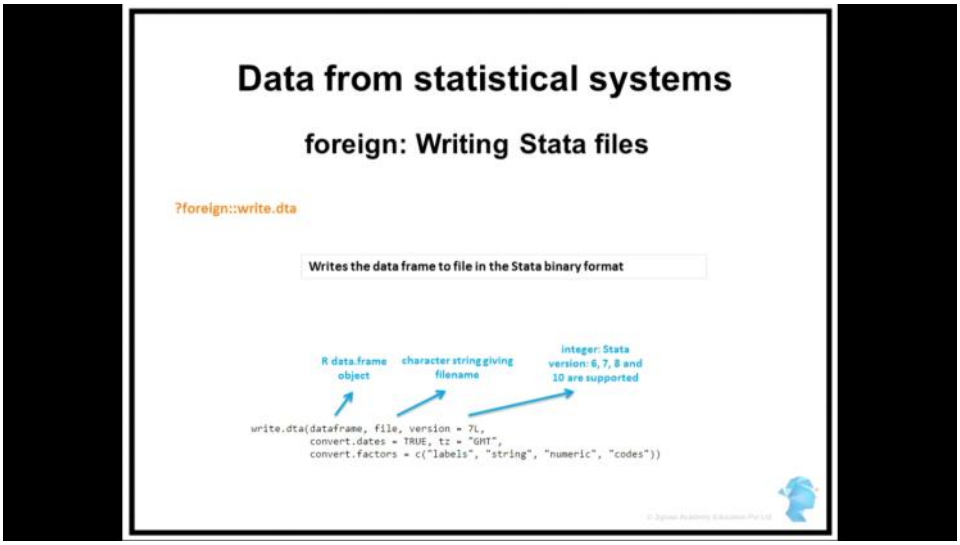
Reads a file in Stata version 5–12 binary format into a data frame

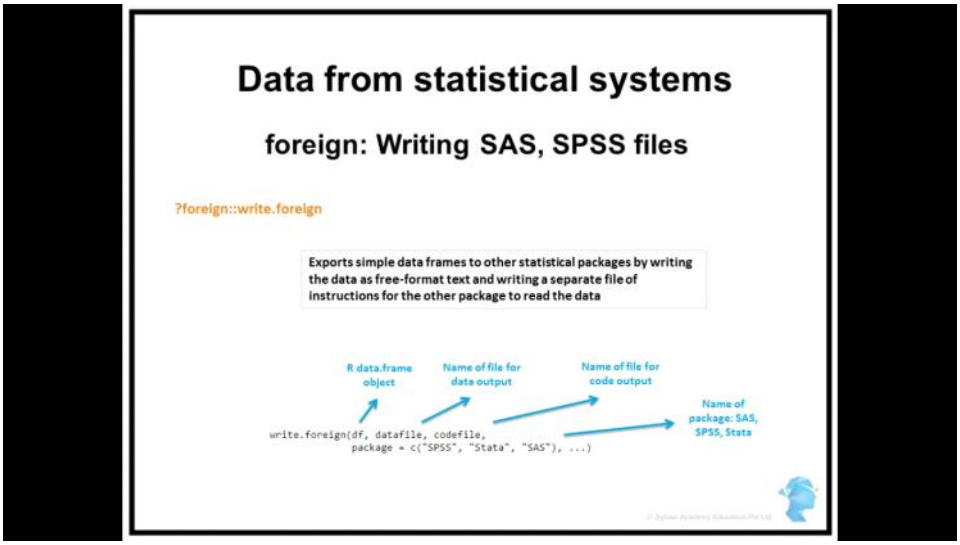
Name of the file to read

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)
```





Working with databases part 1

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

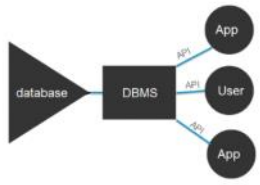
Data from databases

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

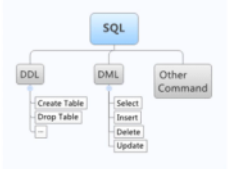
Data from databases

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

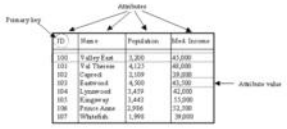


04:17 - 13:16 300 kbps

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

06:24 - 13:16 300 kbps

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

07:16 - 13:16 300 kbps

Data from databases

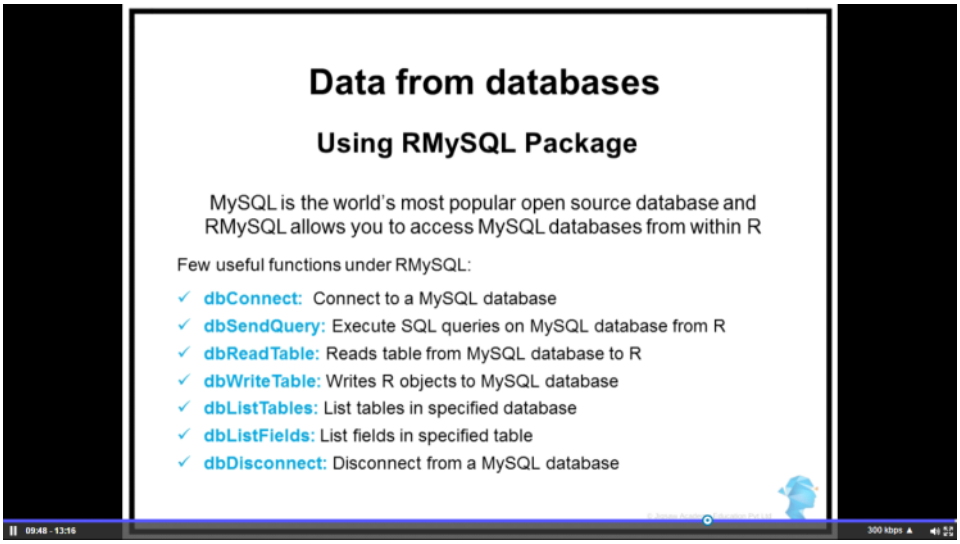
Using RSQLite Package

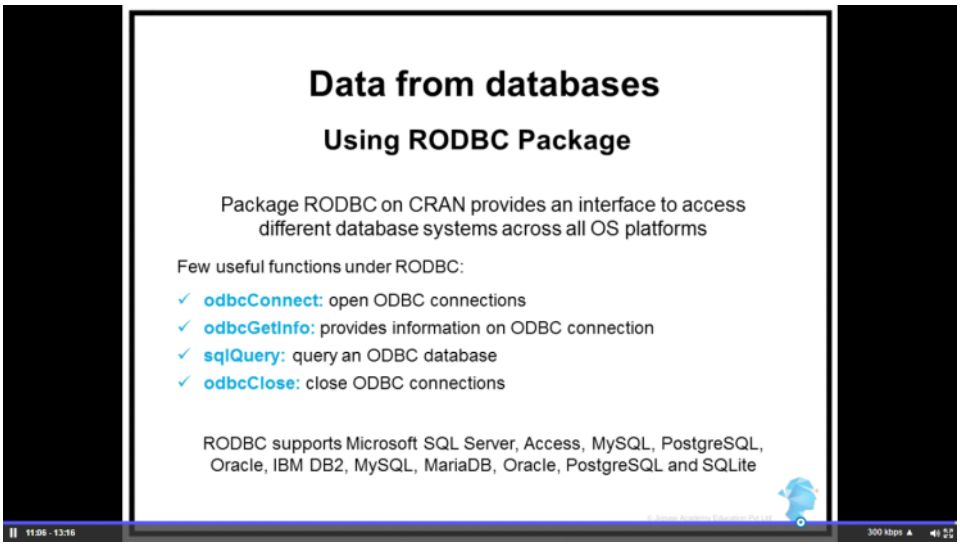
SQLite (http://www.sqlite.org/) is a lightweight, self-contained, zero-configuration, cross-platform, open source database engine

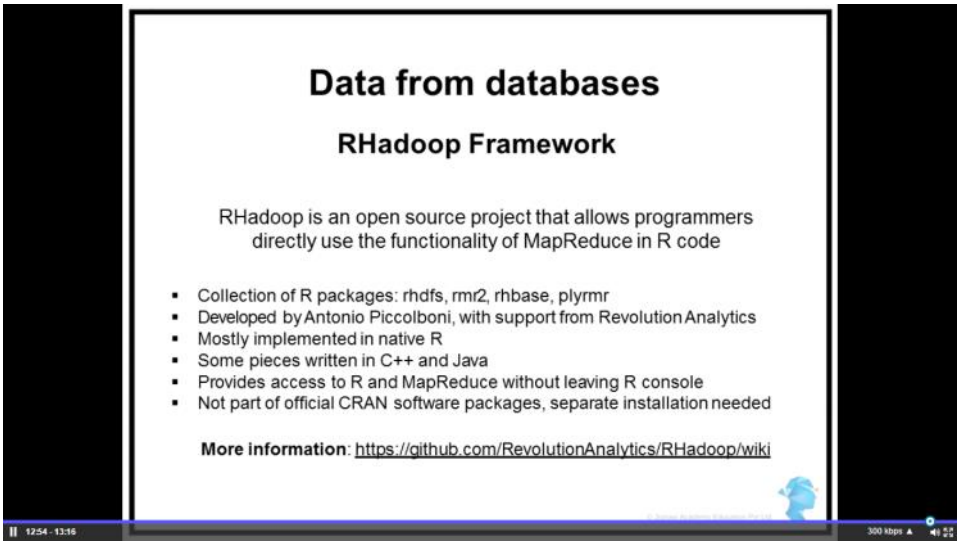
Few useful functions under RSQLite:

- ✓ **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

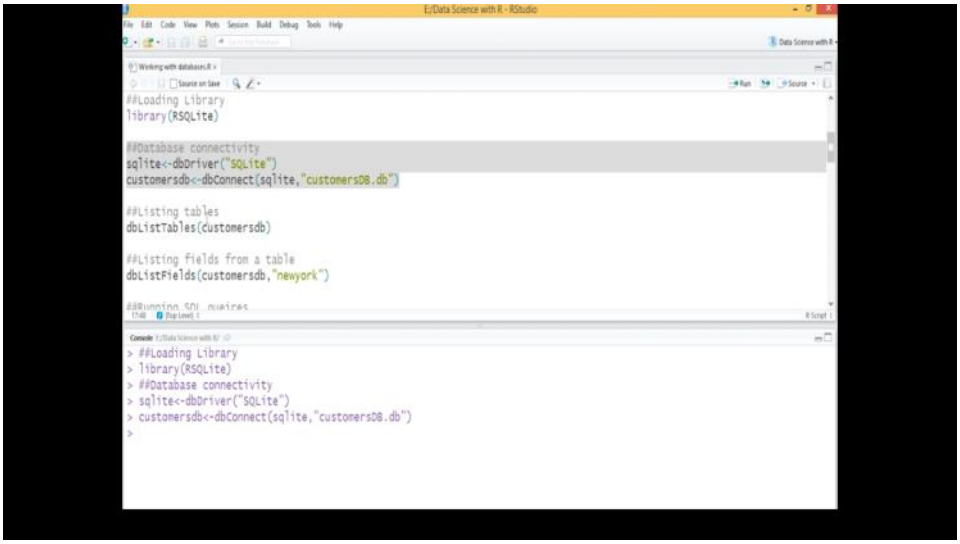
08:42 - 13:16 300 kbps

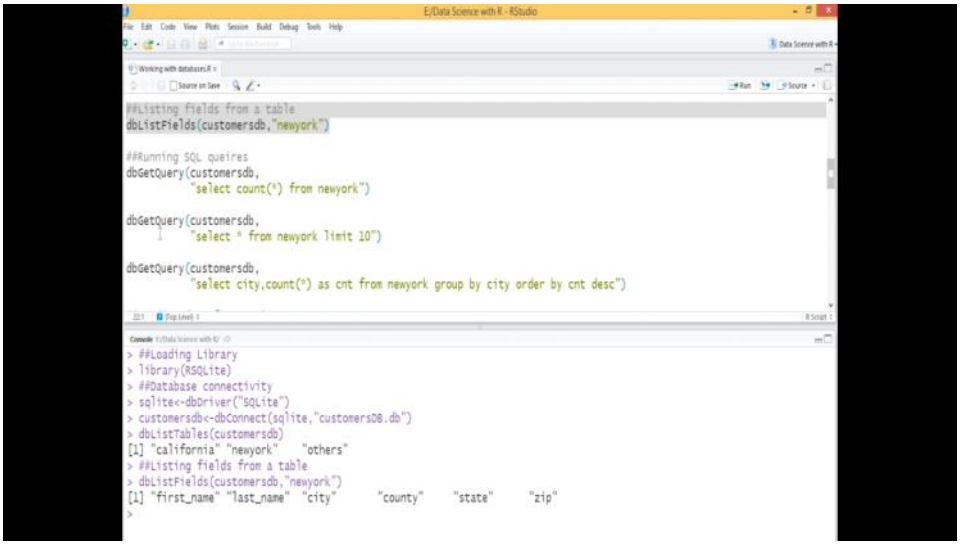


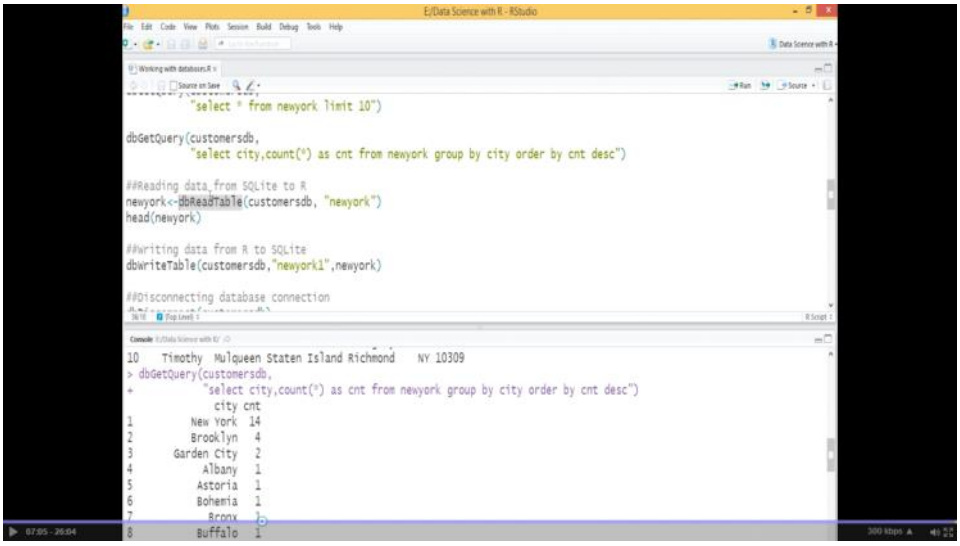


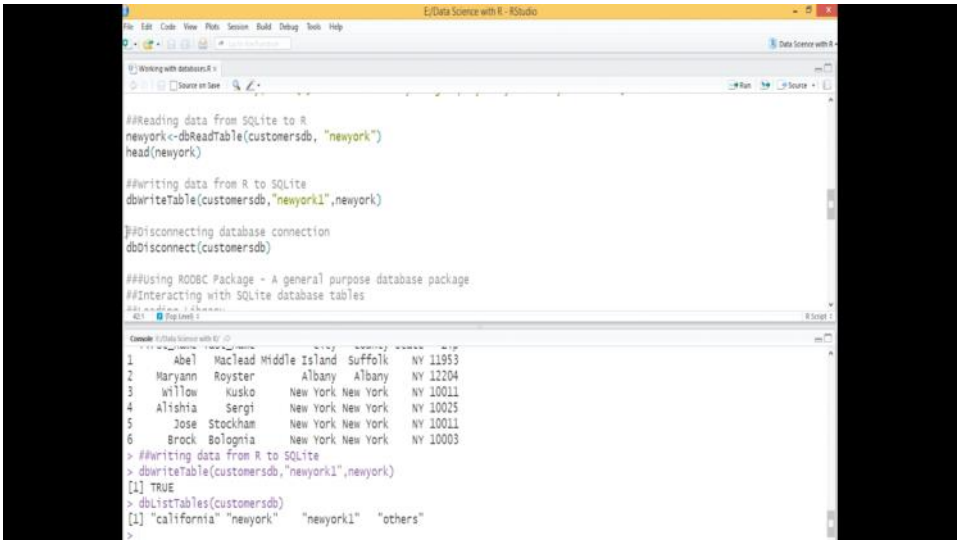


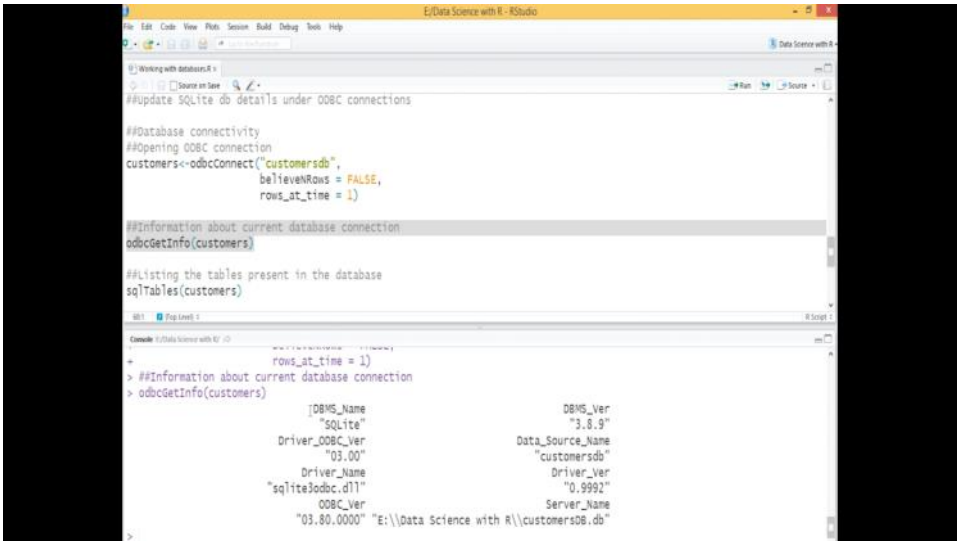
Working with databases Part 2:

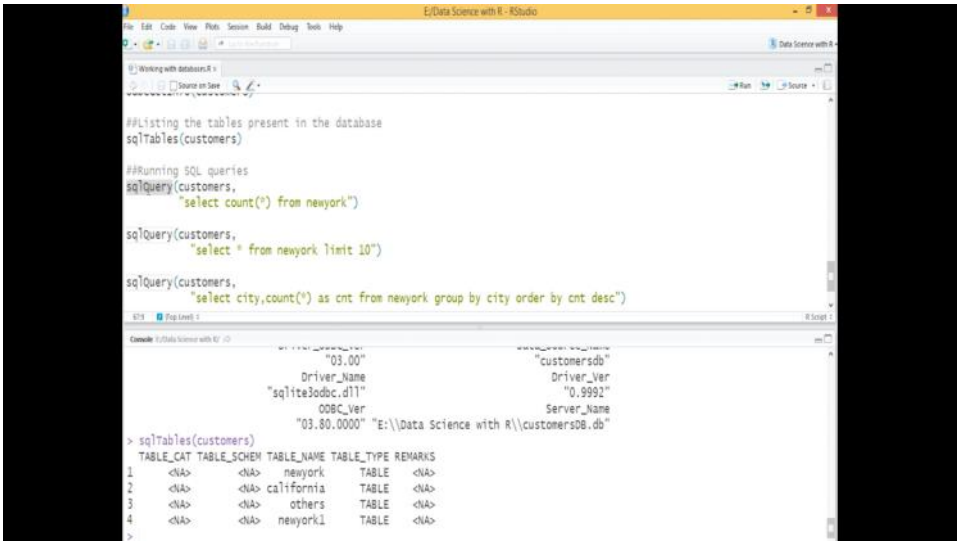

E:/Data Science with R - RStudio
##Loading Library
library(RSQLite)
##Database connectivity
sqlite<-dbDriver("SQLite")
customersdb<-dbConnect(sqlite,"customersDB.db")
##Listing tables
dbListTables(customersdb)
##Listing fields from a table
dbListFields(customersdb,"newyork")
> ##Loading Library
> library(RSQLite)
> ##Database connectivity
> sqlite<-dbDriver("SQLite")
> customersdb<-dbConnect(sqlite,"customersDB.db")


E:/Data Science with R - RStudio
##Listing fields from a table
dbListFields(customersdb,"newyork")
##Running SQL queires
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")
> ##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"

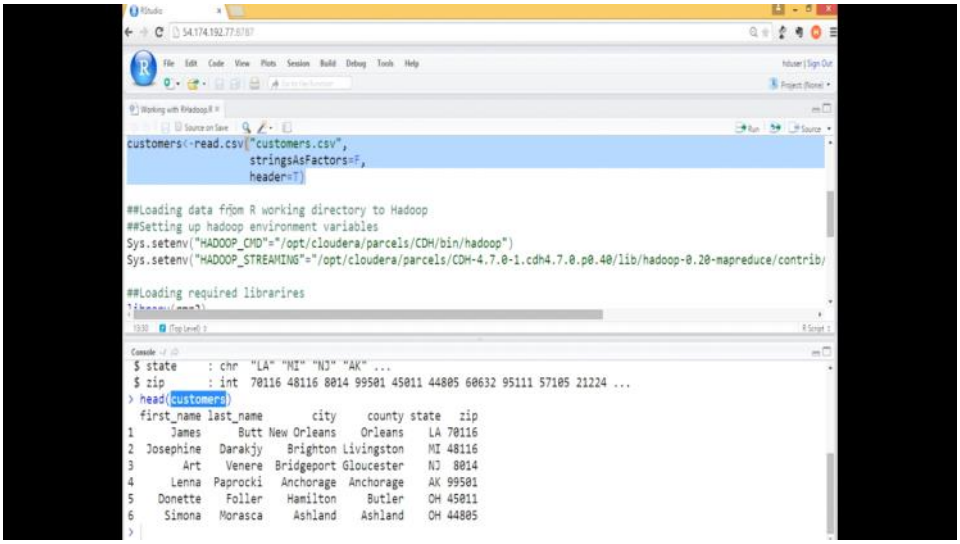

E:/Data Science with R - RStudio
"select * from newyork limit 10")
dbGetQuery(customersdb,
"select city,count(*) as cnt from newyork group by city order by cnt desc")
##Reading data from SQLite to R
newyork<-dbReadTable(customersdb, "newyork")
head(newyork)
##Writing data from R to SQLite
dbWriteTable(customersdb,"newyork1",newyork)
##Disconnecting database connection
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

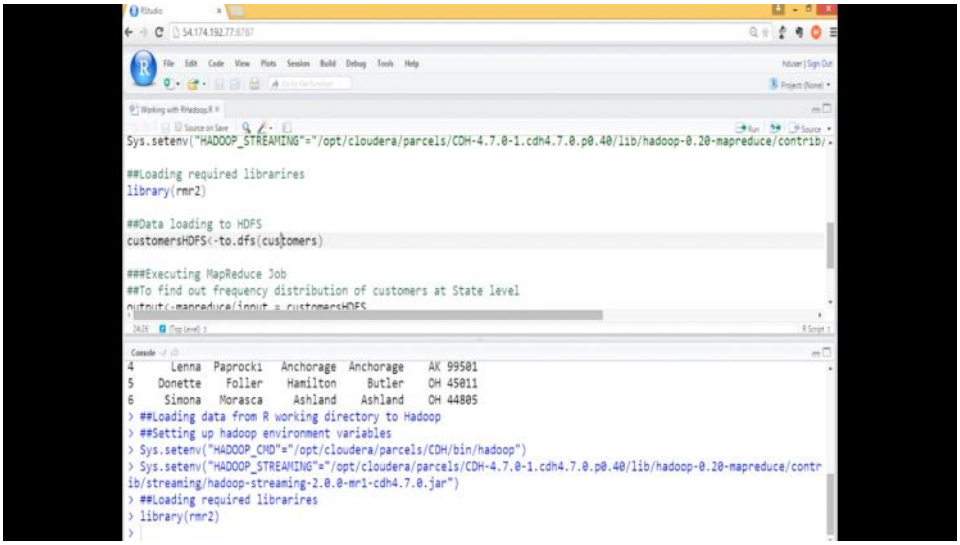




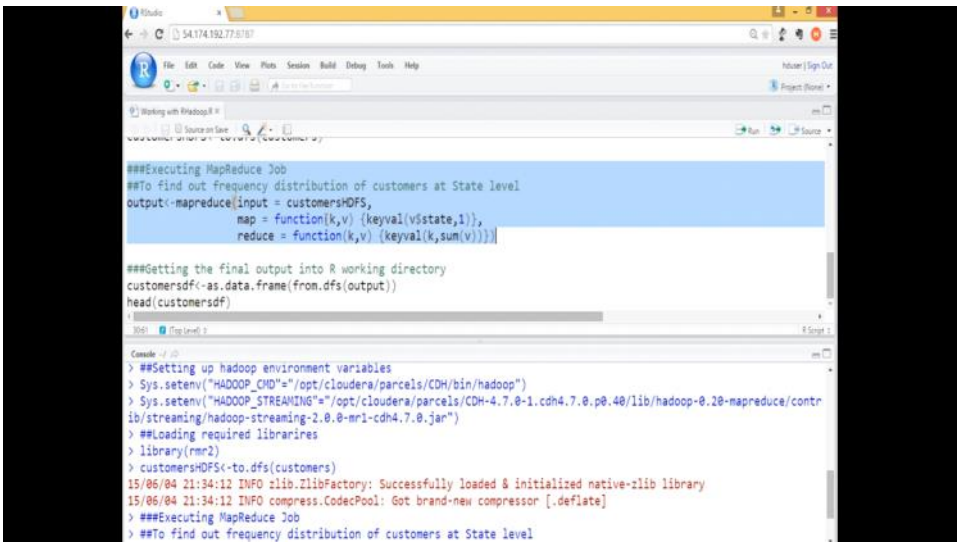


Loading the dataframe object into the hadoop cluster:

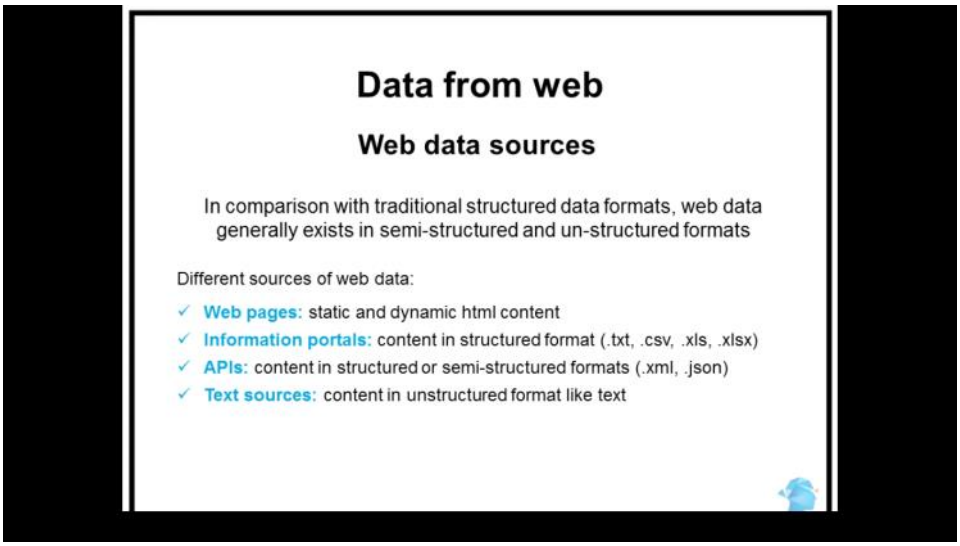




Map reduce job being run here:



Working with Web Data Part 1:

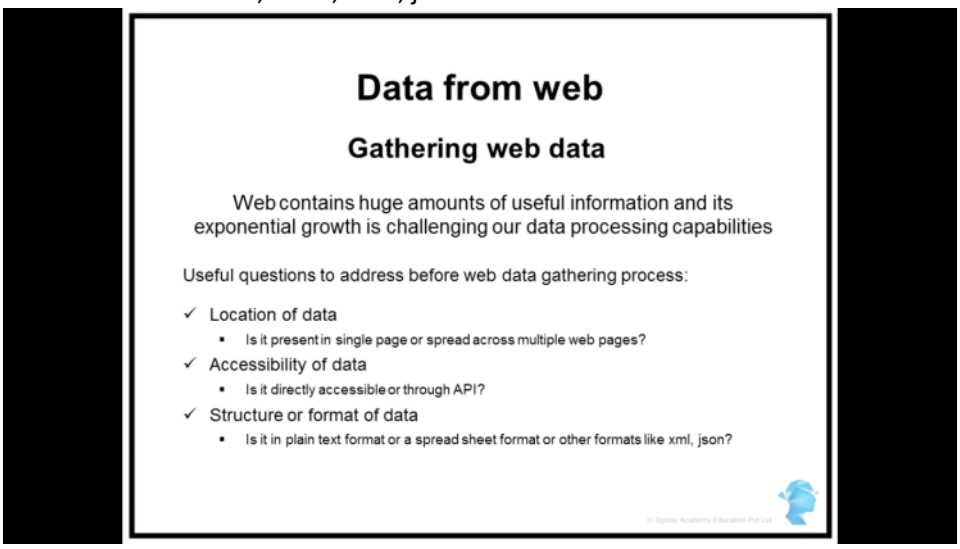


Semi-Structured and unstructured data.

Gathering the webdata:
Unstrucutred 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
- Acessing 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

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.

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)
```

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 support in R for xml data parsing
- ✓ **JSON**: JavaScript Object Notation
  - Common format for representing data
  - Web APIs provide data in json format
  - Package support in 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>
```

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

Data from web

XML package: parsing functions

?XML::xmlTreeParse

Name of file containing XML contents

Whether the file argument refers to a URL or a system file

Logical value that allows this function to be used for parsing HTML documents

```
xmlTreeParse(file, ignoreBlanks=TRUE, handlers=NULL, replaceEntities=FALSE,
             asText=FALSE, trim=TRUE, validate=FALSE, getDTD=TRUE,
             isURL=FALSE, asTree = FALSE, addAttributeNamespaces = FALSE,
             useInternalNodes = FALSE, isSchema = FALSE,
             fullNamespaceInfo = FALSE, encoding = character(),
             useDotNames = length(grep("^\\.", names(handlers))) > 0,
             xinclude = TRUE, addFinalizer = TRUE, error = xmlErrorCumulator(),
             isHTML = FALSE, options = integer(), parentFirst = FALSE)
```

?XML::xmlRoot

Object whose root/top-level XML node is to be returned

```
xmlRoot(x, skip = TRUE, ...)
```

?XML::xmlSApply

XMLNode on whose children the regular apply or sapply is to be performed

Function to apply to each child node

```
xmlSApply(X, FUN, ...)
```

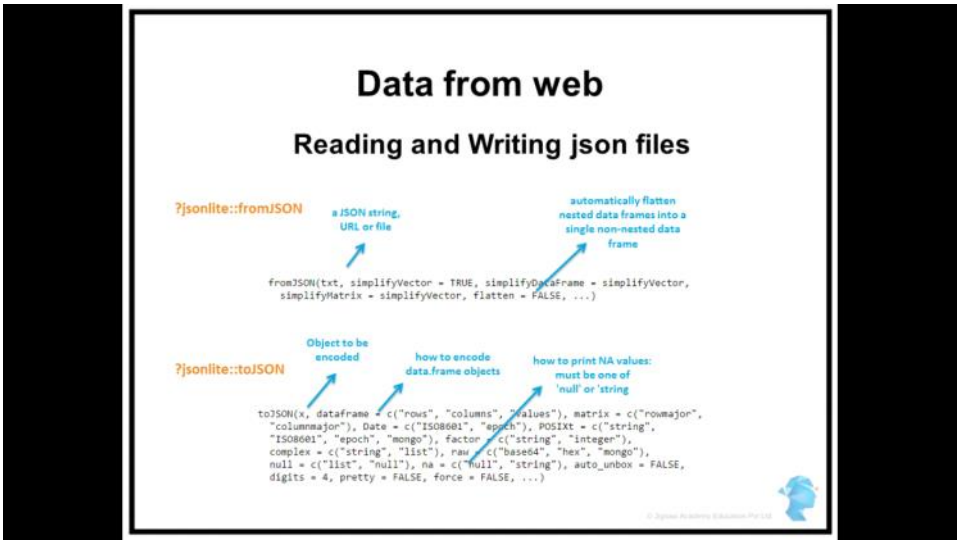
Data from web

Handling JSON data

R has 3 packages for working with json data:

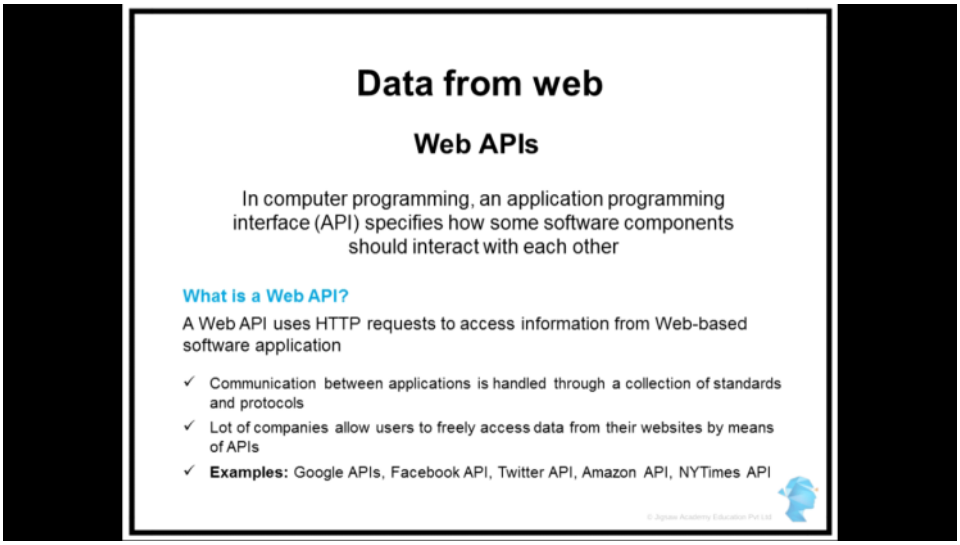
- ✓ **RJSONIO:** allows conversion to and from data in JSON format
- ✓ **rjson:** converts R object into JSON objects and vice-versa
- ✓ **jsonlite:** offers flexible, robust, high performance tools for working with JSON in R; and easier to convert json objects directly into R data.frames

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

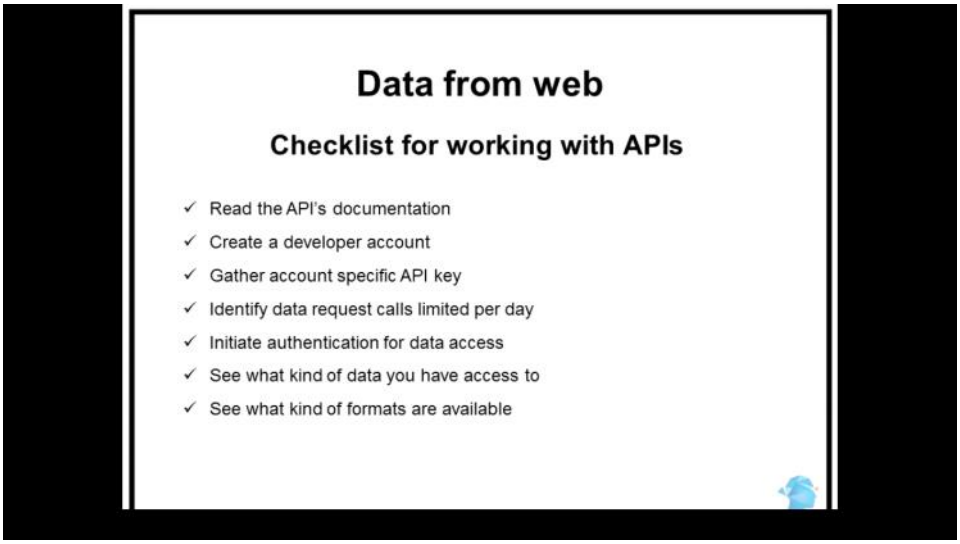


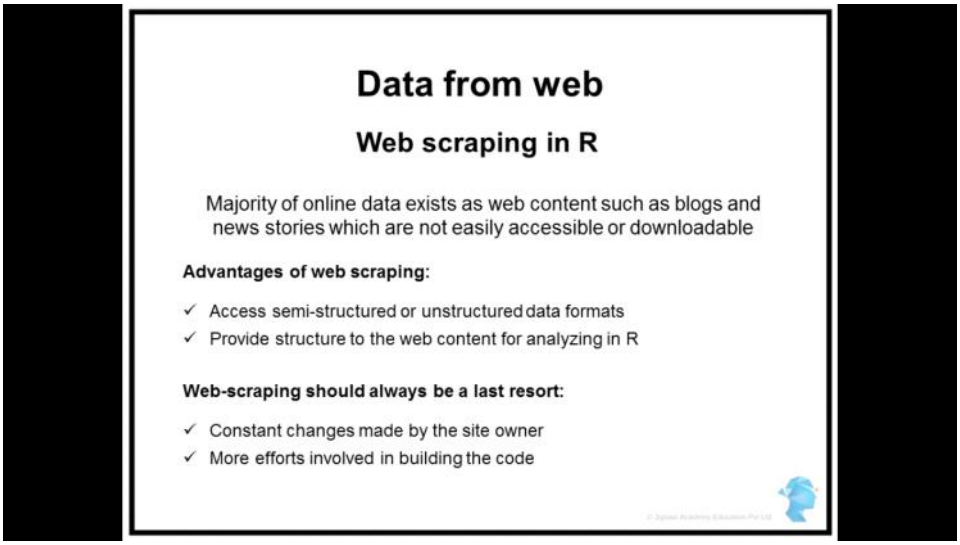
fromJSON(URL, flatten = False)

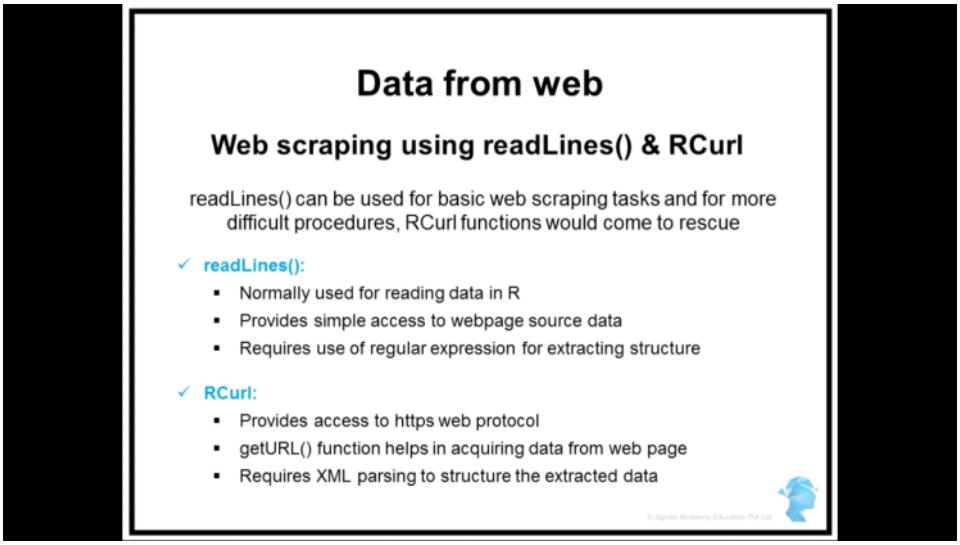
Using Web API:



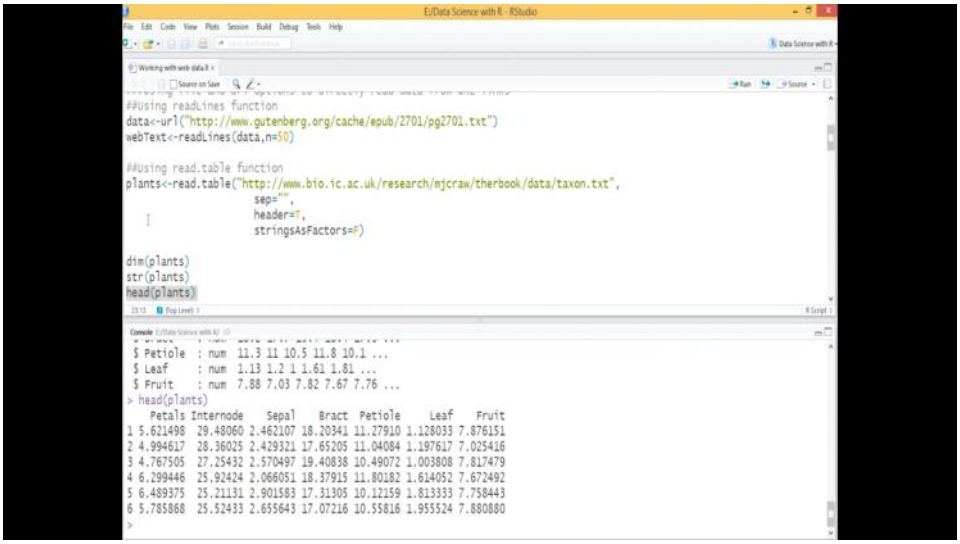
Checklist for working with APIs:

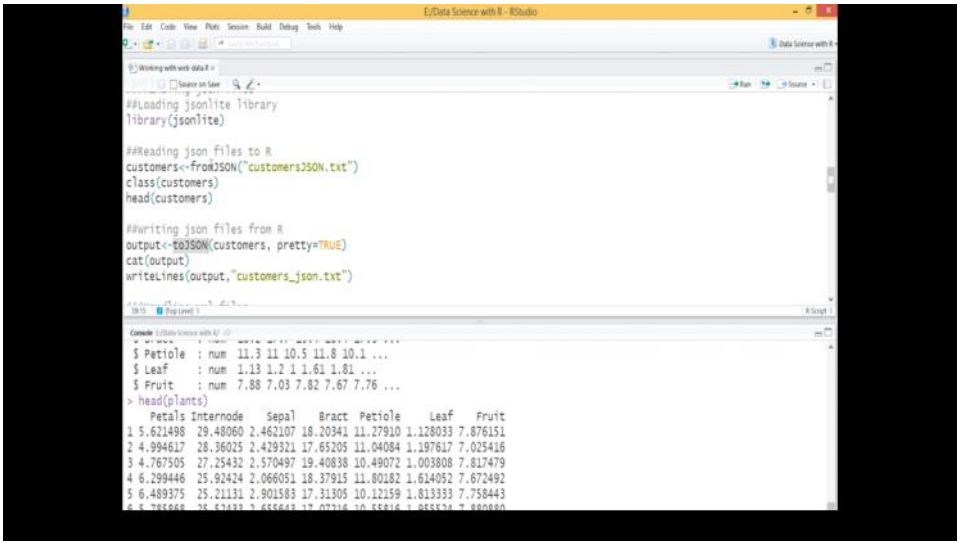


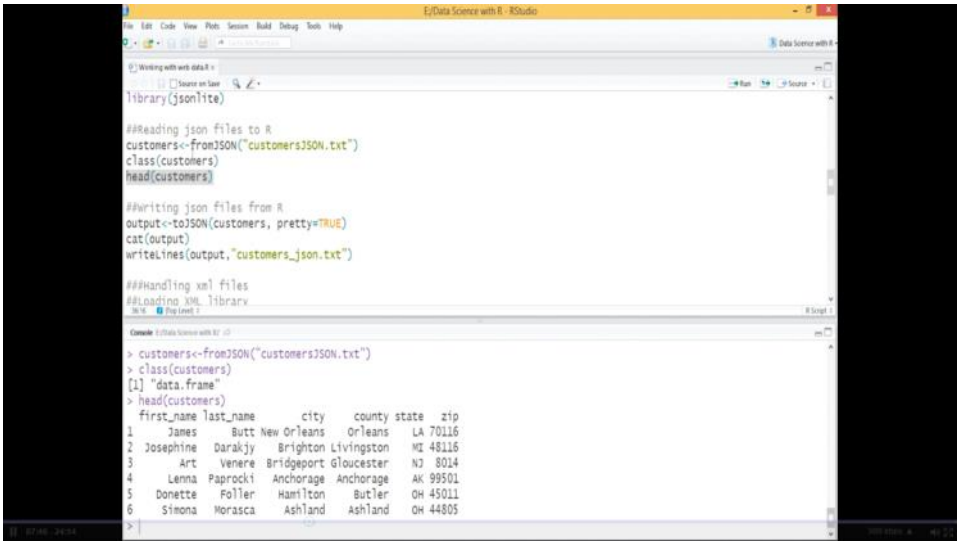


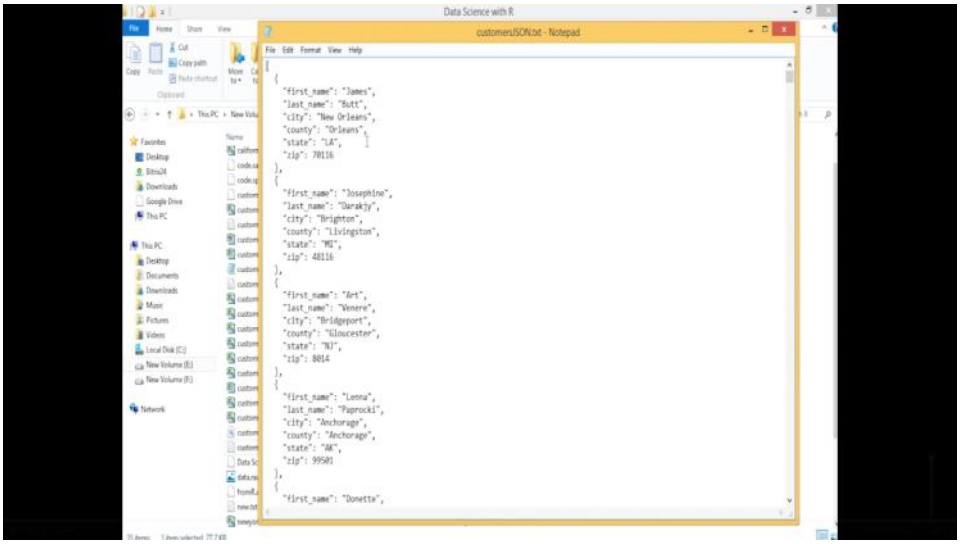


Working with Web Data Part 2:


##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/mjcraw/therbook/data/taxon.txt",
sep="",
header=T,
stringsAsFactors=F)
dim(plants)
str(plants)
head(plants)
$ Petiole : num 11.3 11 10.5 11.8 10.1 ...
$ Leaf : num 1.13 1.2 1 1.61 1.81 ...
$ Fruit : num 7.88 7.03 7.82 7.67 7.76 ...
> head(plants)
Petals 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 2.570497 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 2.901583 17.31305 10.12159 1.813333 7.758443
6 5.785868 25.52433 2.655643 17.07216 10.55816 1.955524 7.880880


##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")
$ Petiole : num 11.3 11 10.5 11.8 10.1 ...
$ Leaf : num 1.13 1.2 1 1.61 1.81 ...
$ Fruit : num 7.88 7.03 7.82 7.67 7.76 ...
> head(plants)
Petals 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 2.570497 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 2.901583 17.31305 10.12159 1.813333 7.758443


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
> 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



Loading and Reading XML files :

First we will need to load the XML library
>Library(XML)
>xmldata<-"sample.xml"

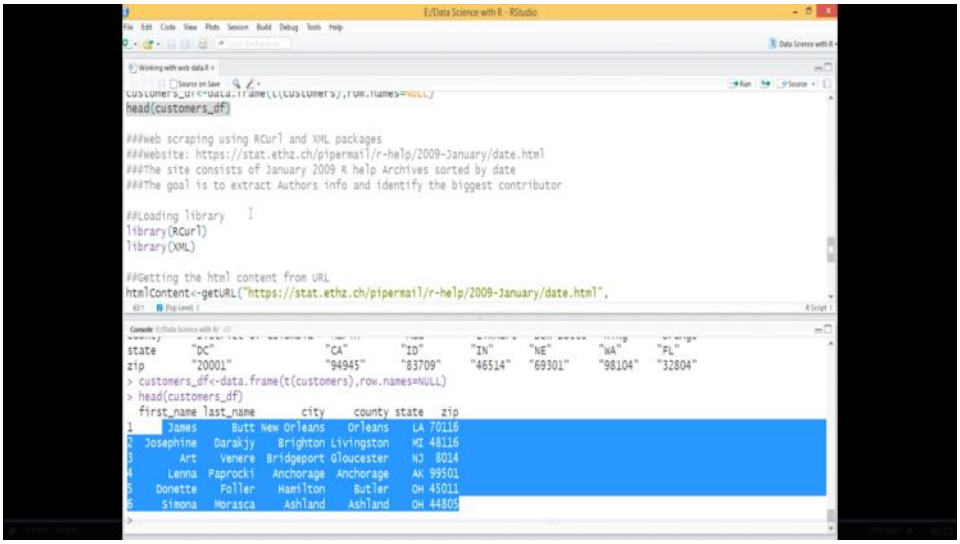
Then use the tree parser functionality to parse through the XML file that has been laoded
>xmlparse<- XMLTreeParse(xmldata)
>class(xmlparse)
After looking at the class, the class should be XMLDocument, we can look at the root node of the XML
>xmltop<-xmlroot(xmlparse)

Then to extract the values from the attributes of the nodes we will use XMLSApply
>cust <- XMLSApply(xmltop, function(X) XMLSApply(x, XMLValue))

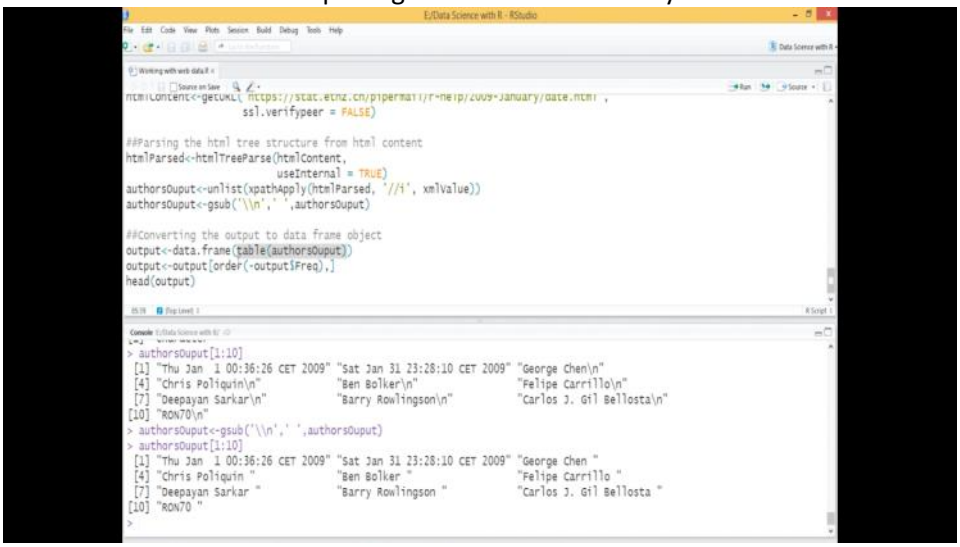
This will be a column by column output, but the end result we want to convert it into a data frame:

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



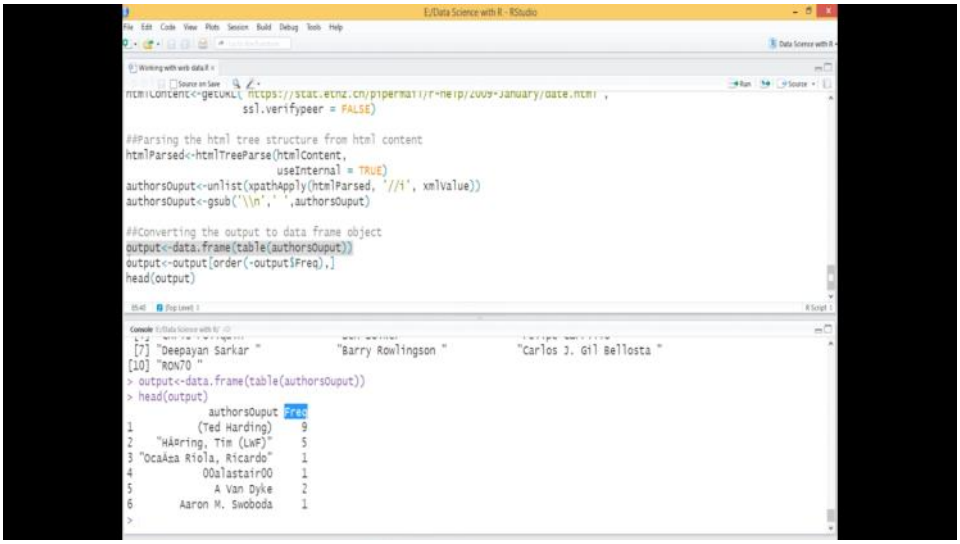
We can use Rcurl and XML packages for this functionality



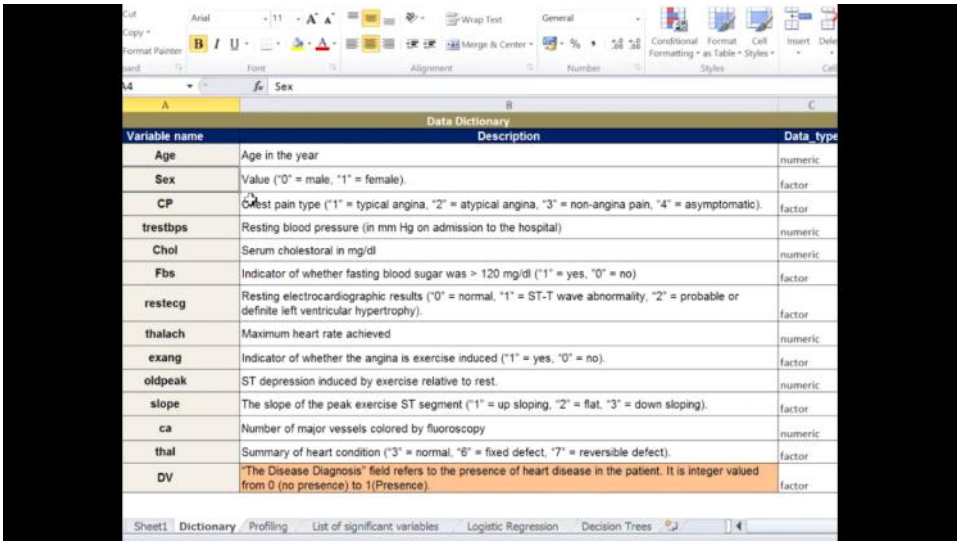
Use the get URL to load the HTML page to a content on R
>htmlcontent<-getURL("http://www.xyz.com/test.htm", ssl.verfypeer = FALSE)
Then use HTMLTreeParse to parse the HTML page content using the internal nodes that are already present
>htmParsed<-HTMLParse(htmlcontent, useInternal =TRUE)
The parsed output 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.

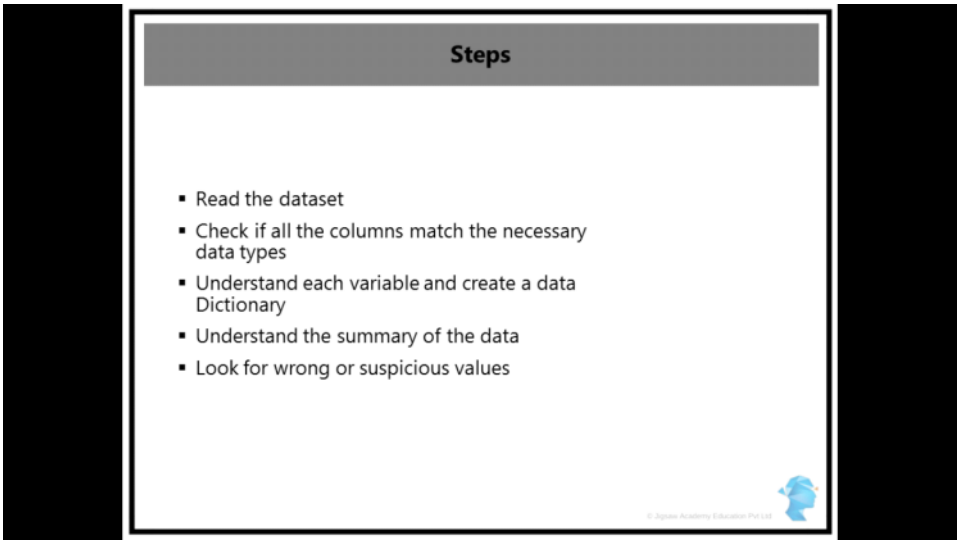


In order to put these in descending order we use the following:
Output<-output[order(-output$freq),]

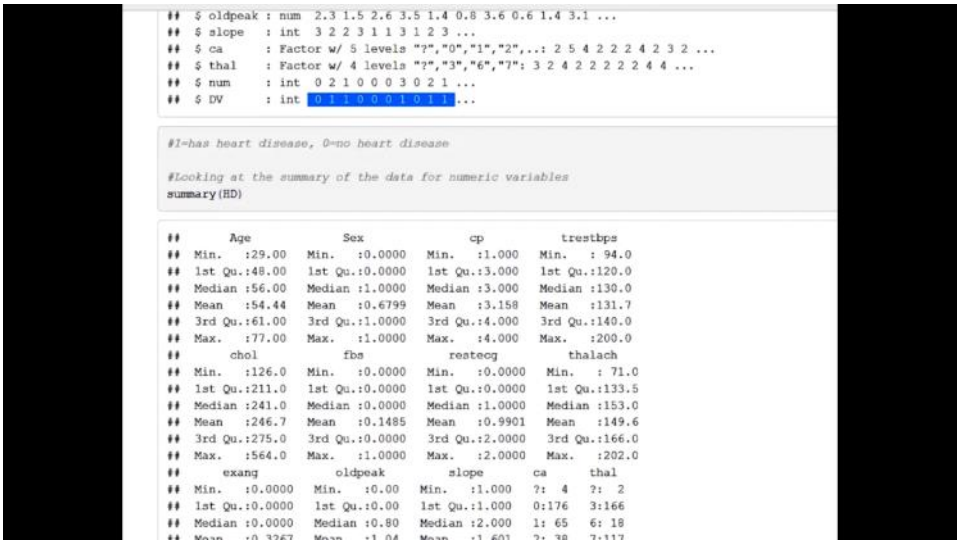


Data Dictionary

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 cholestoral 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



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



Run a summary to get the descriptive stats of these variables

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