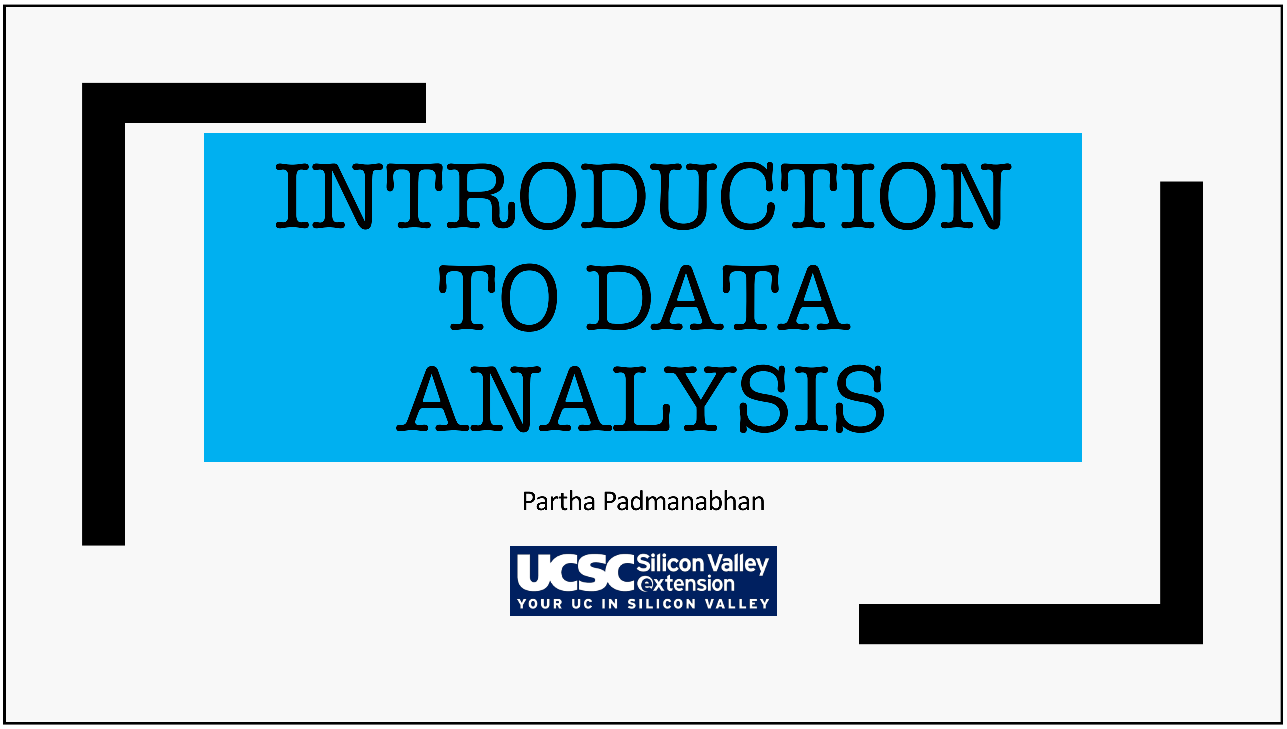
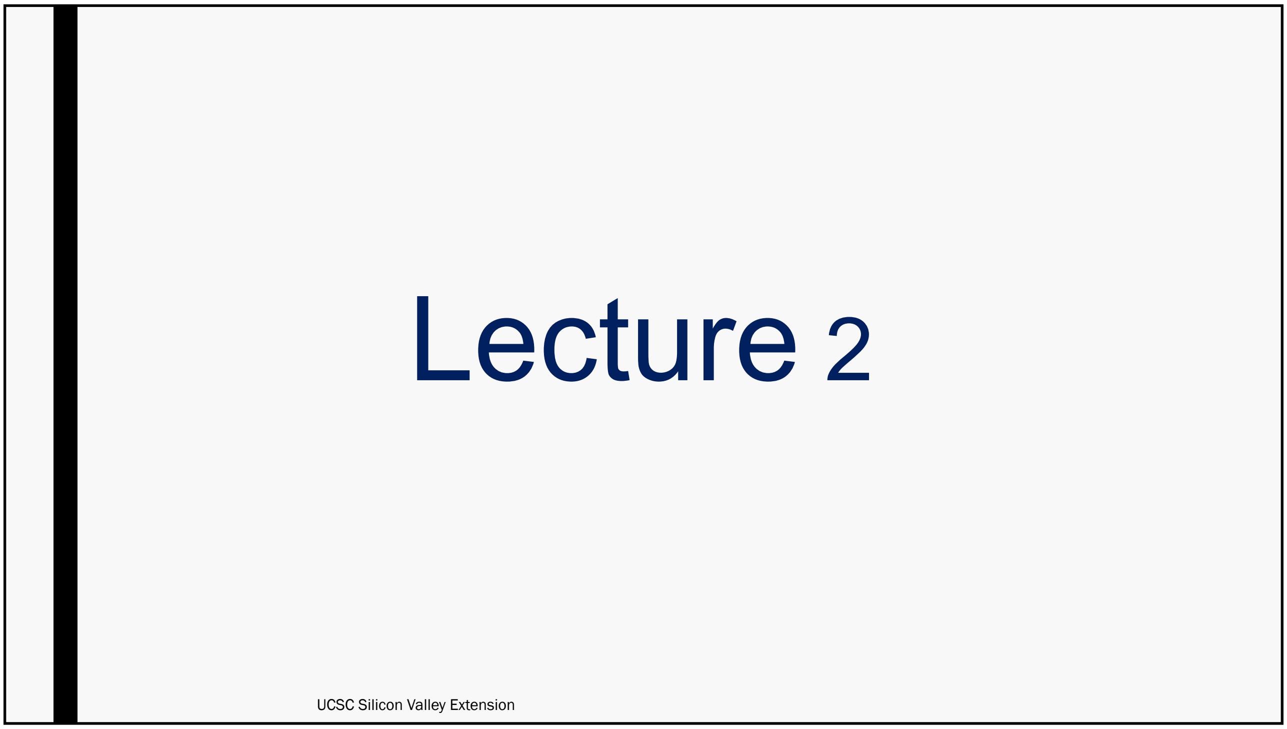


INTRODUCTION TO DATA ANALYSIS

Partha Padmanabhan

UCSC Silicon Valley Extension
YOUR UC IN SILICON VALLEY

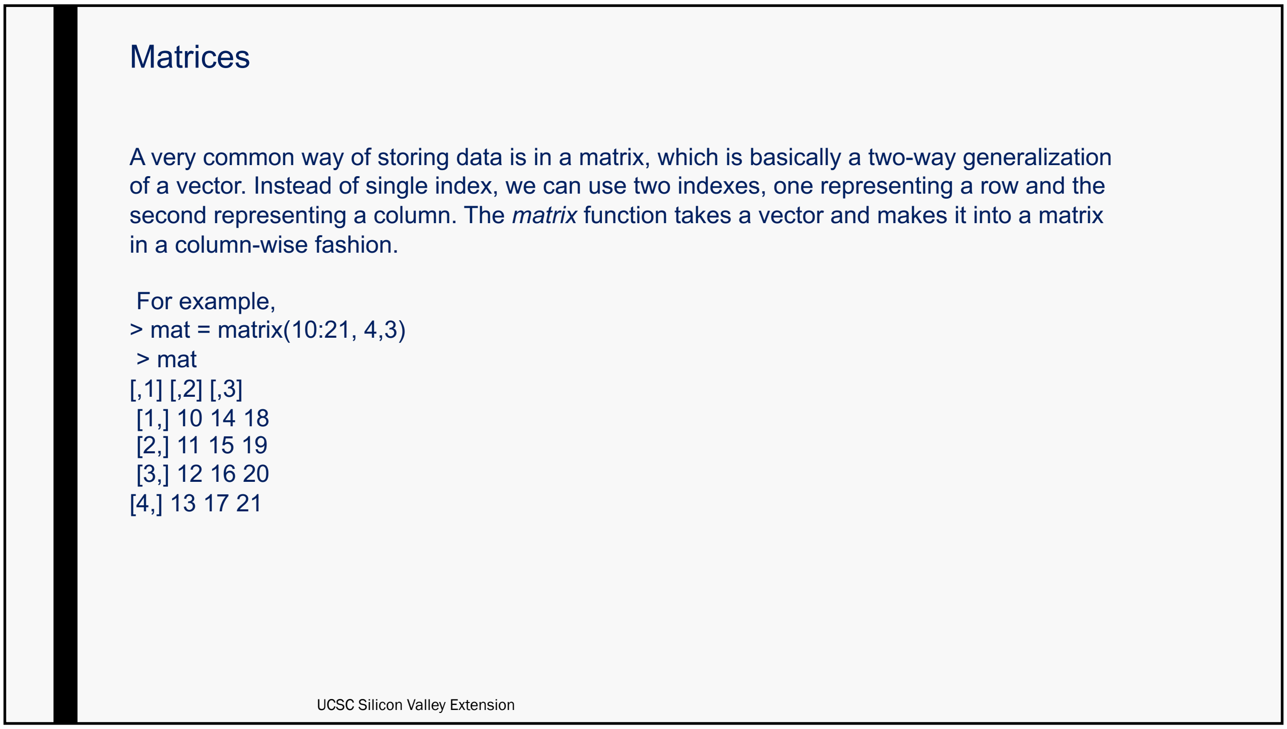
1



Lecture 2

UCSC Silicon Valley Extension

2



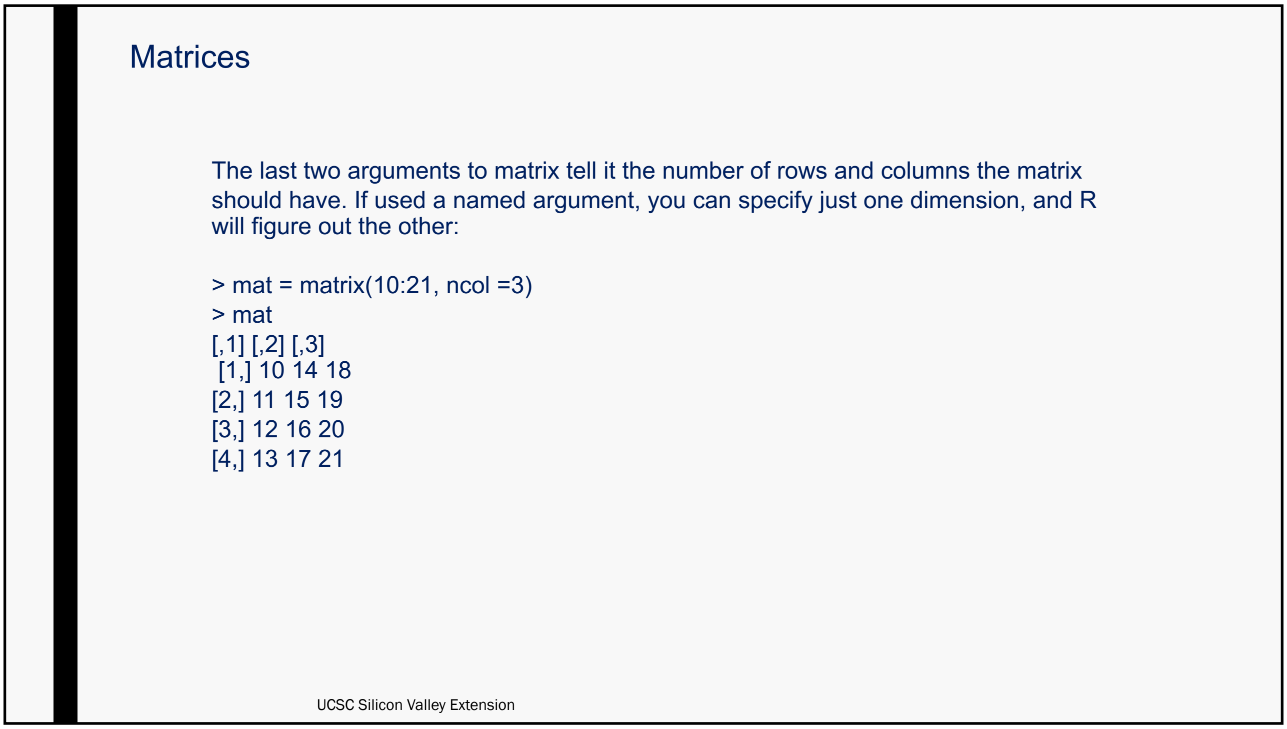
Matrices

A very common way of storing data is in a matrix, which is basically a two-way generalization of a vector. Instead of single index, we can use two indexes, one representing a row and the second representing a column. The *matrix* function takes a vector and makes it into a matrix in a column-wise fashion.

For example,

```
> mat = matrix(10:21, 4,3)
 > mat
[,1] [,2] [,3]
 [1,] 10 14 18
 [2,] 11 15 19
 [3,] 12 16 20
[4,] 13 17 21
```

3

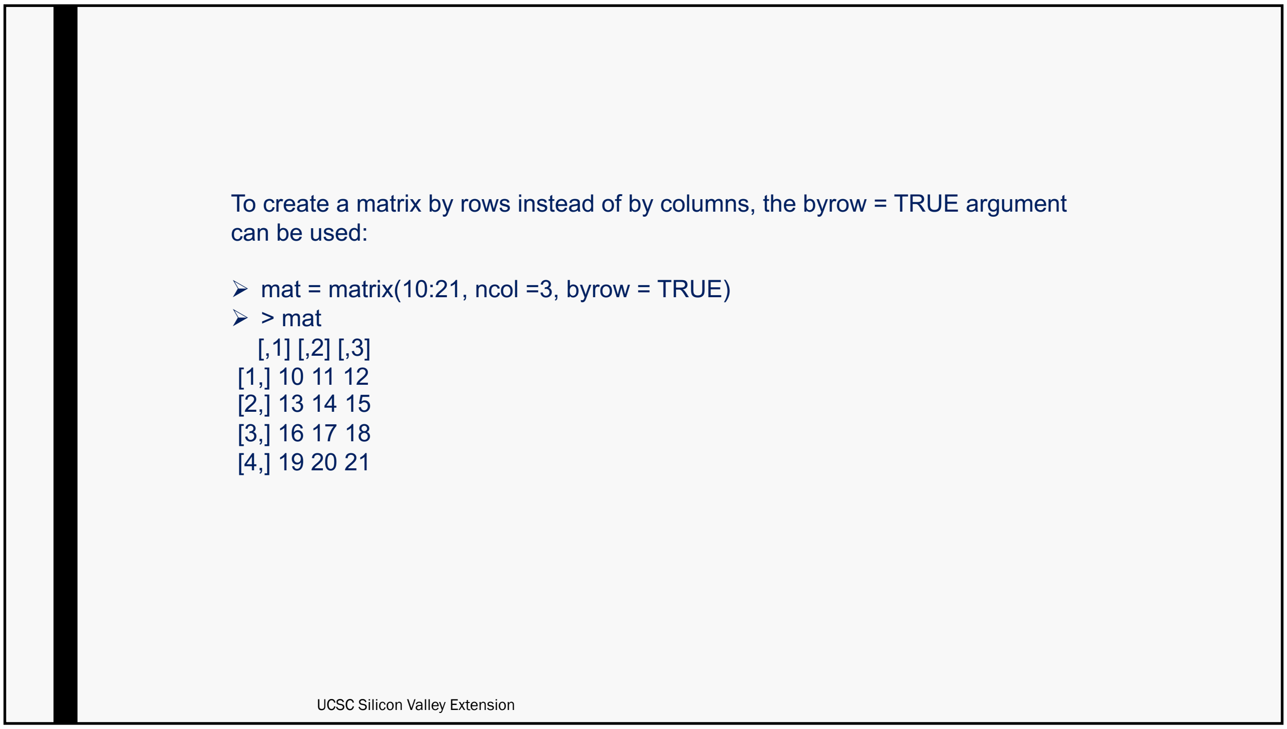


Matrices

The last two arguments to matrix tell it the number of rows and columns the matrix should have. If used a named argument, you can specify just one dimension, and R will figure out the other:

```
> mat = matrix(10:21, ncol =3)
> mat
[,1] [,2] [,3]
 [1,] 10 14 18
[2,] 11 15 19
[3,] 12 16 20
[4,] 13 17 21
```

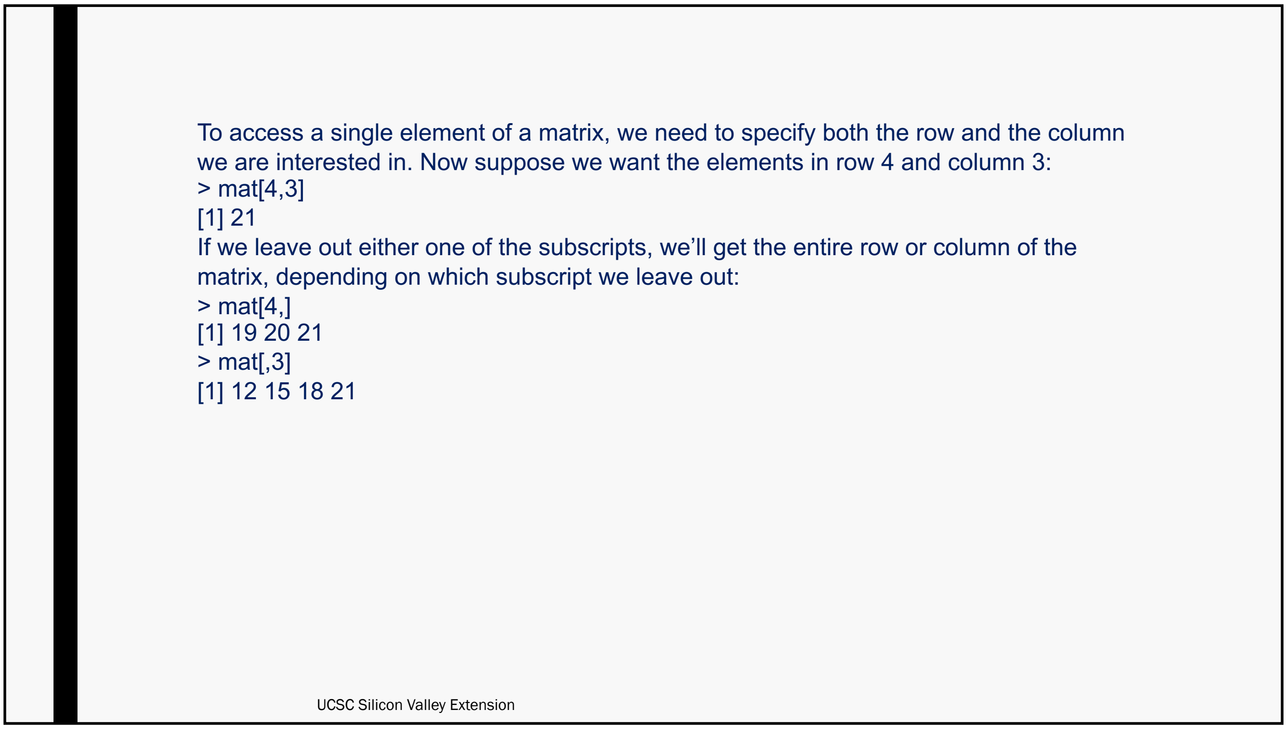
4



To create a matrix by rows instead of by columns, the byrow = TRUE argument can be used:

```
➢ mat = matrix(10:21, ncol =3, byrow = TRUE)
➢ > mat
    [,1] [,2] [,3]
 [1,] 10 11 12
 [2,] 13 14 15
 [3,] 16 17 18
 [4,] 19 20 21
```

5



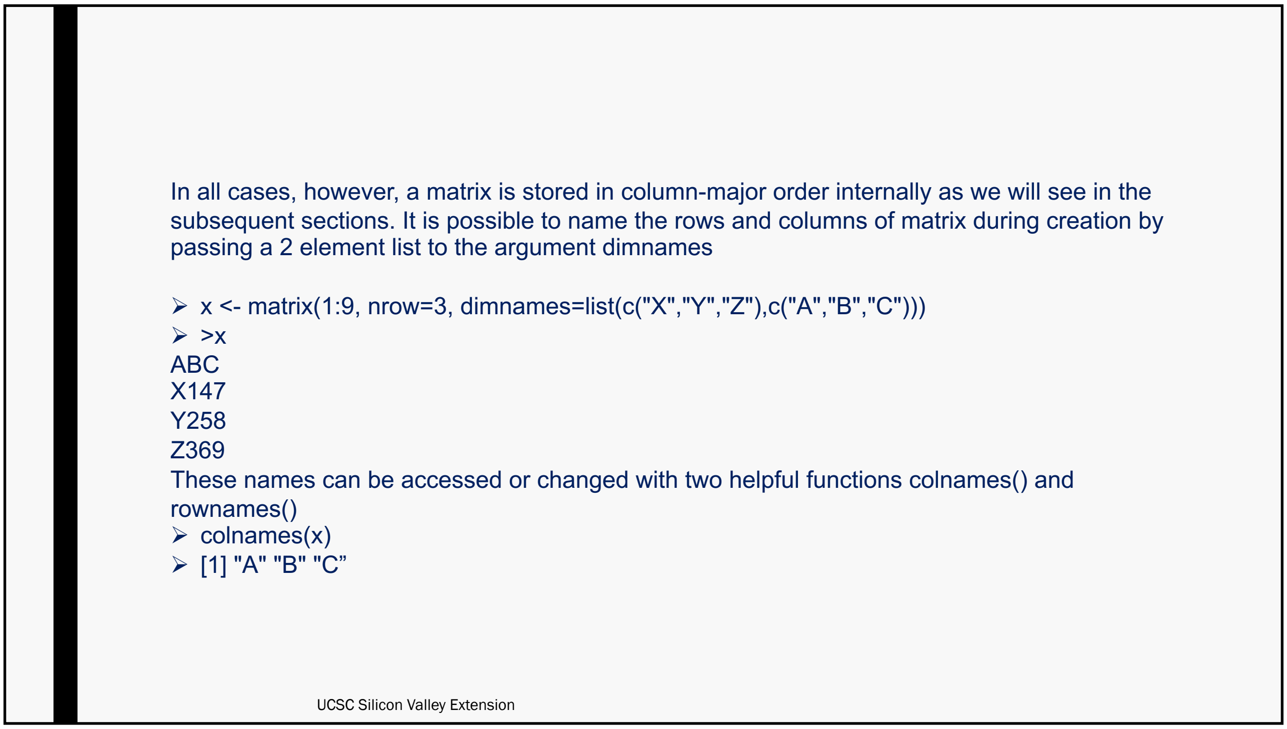
To access a single element of a matrix, we need to specify both the row and the column we are interested in. Now suppose we want the elements in row 4 and column 3:

```
> mat[4,3]
[1] 21
```

If we leave out either one of the subscripts, we’ll get the entire row or column of the matrix, depending on which subscript we leave out:

```
> mat[4,]
[1] 19 20 21
> mat[,3]
[1] 12 15 18 21
```

6



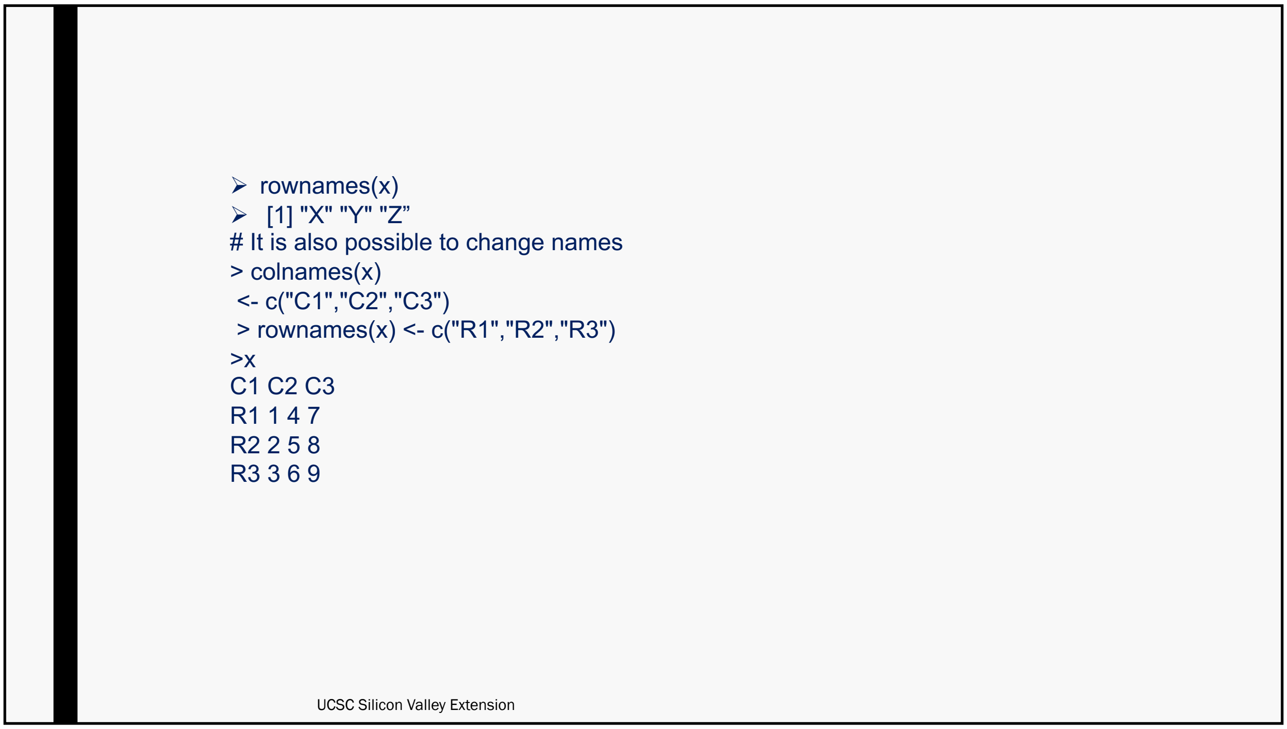
In all cases, however, a matrix is stored in column-major order internally as we will see in the subsequent sections. It is possible to name the rows and columns of matrix during creation by passing a 2 element list to the argument dimnames

```
➢ x <- matrix(1:9, nrow=3, dimnames=list(c("X","Y","Z"),c("A","B","C")))
➢ >x
ABC
X147
Y258
Z369
```

These names can be accessed or changed with two helpful functions colnames() and rownames()

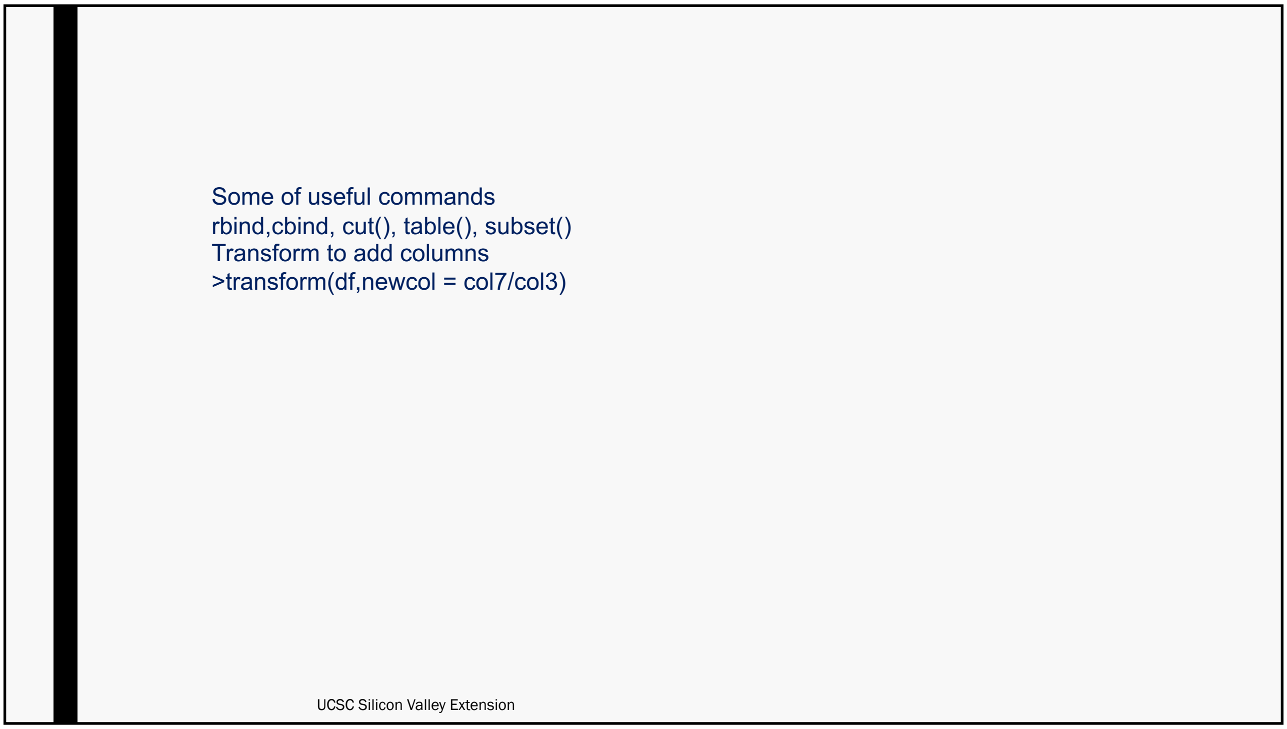
```
➢ colnames(x)
➢ [1] "A" "B" "C”
```

7



```
➢ rownames(x)
➢  [1] "X" "Y" "Z”
# It is also possible to change names
> colnames(x)
 <- c("C1","C2","C3")
 > rownames(x) <- c("R1","R2","R3")
>x
C1 C2 C3
R1 1 4 7
R2 2 5 8
R3 3 6 9
```

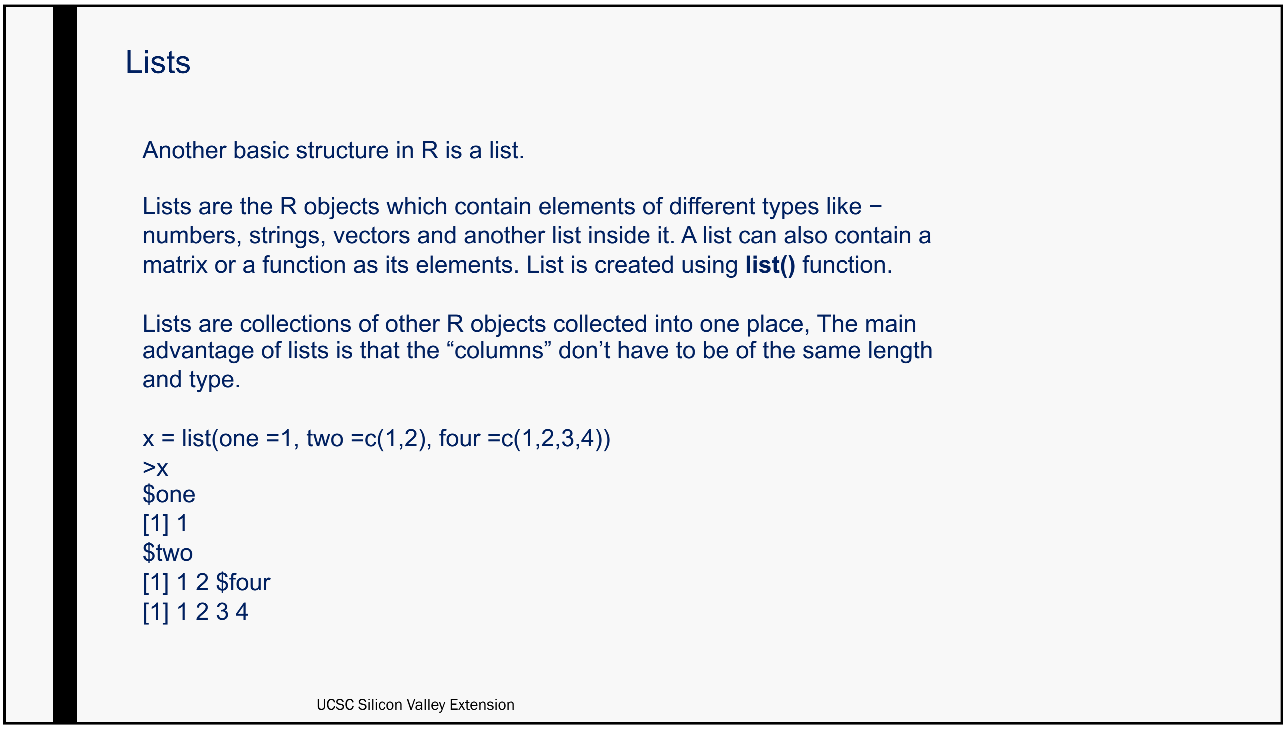
8



Some of useful commands
rbind,cbind, cut(), table(), subset()
Transform to add columns

```
>transform(df,newcol = col7/col3)
```

9



Lists

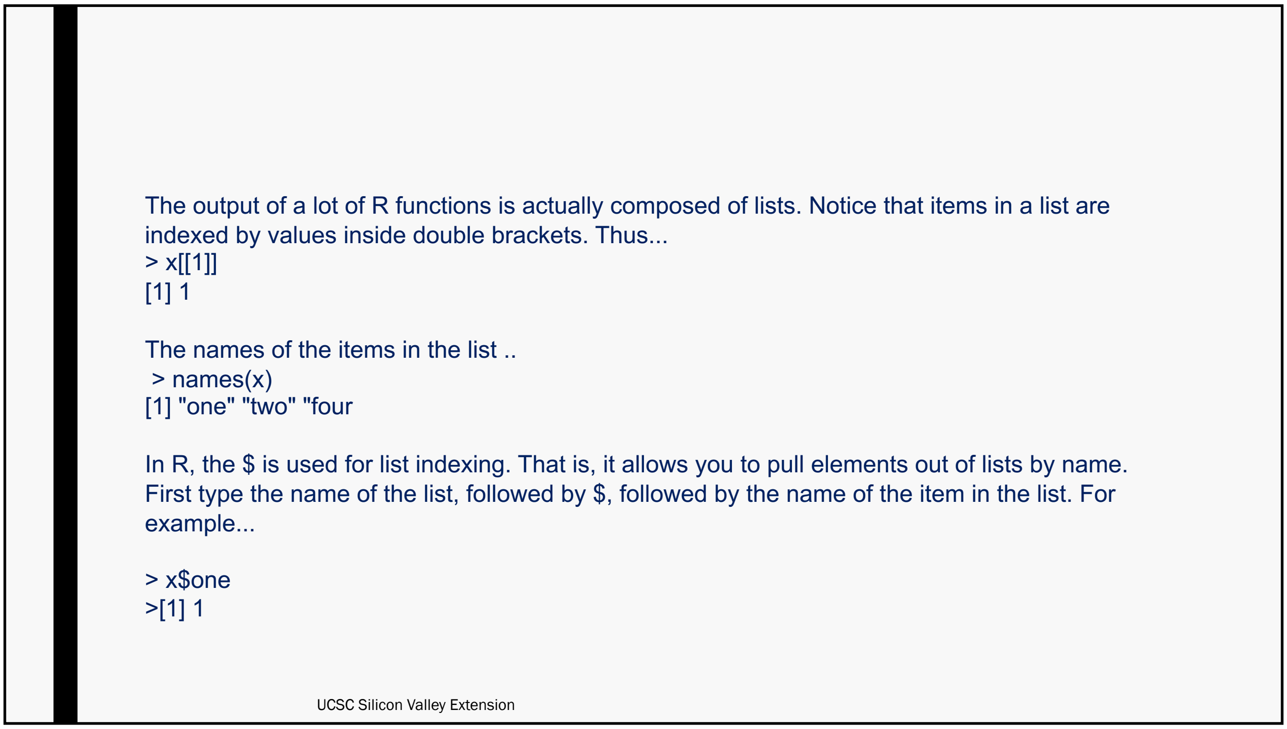
Another basic structure in R is a list.

Lists are the R objects which contain elements of different types like − numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using **list()** function.

Lists are collections of other R objects collected into one place, The main advantage of lists is that the “columns” don’t have to be of the same length and type.

```
x = list(one =1, two =c(1,2), four =c(1,2,3,4))
>x
$one
[1] 1
$two
[1] 1 2 $four
[1] 1 2 3 4
```

10



The output of a lot of R functions is actually composed of lists. Notice that items in a list are indexed by values inside double brackets. Thus...

```
> x[[1]]
[1] 1
```

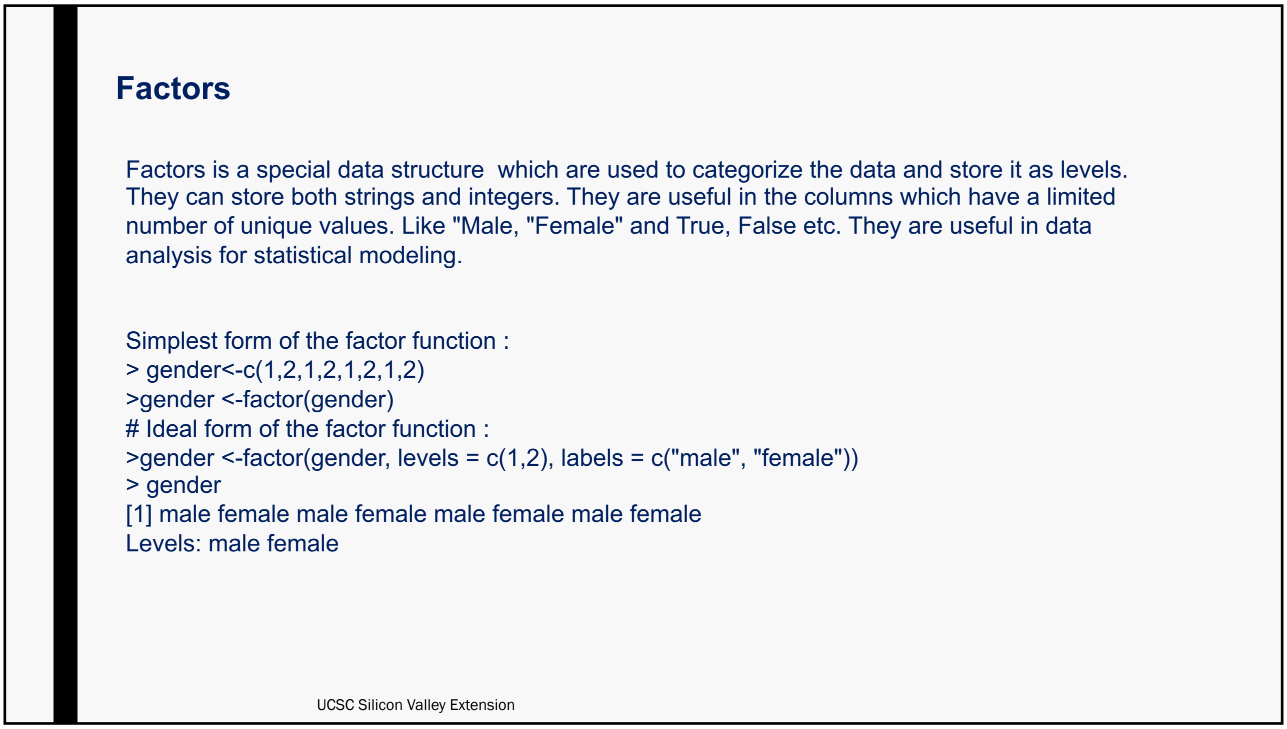
The names of the items in the list ..

```
 > names(x)
[1] "one" "two" "four
```

In R, the $ is used for list indexing. That is, it allows you to pull elements out of lists by name. First type the name of the list, followed by $, followed by the name of the item in the list. For example...

```
> x$one
>[1] 1
```

11



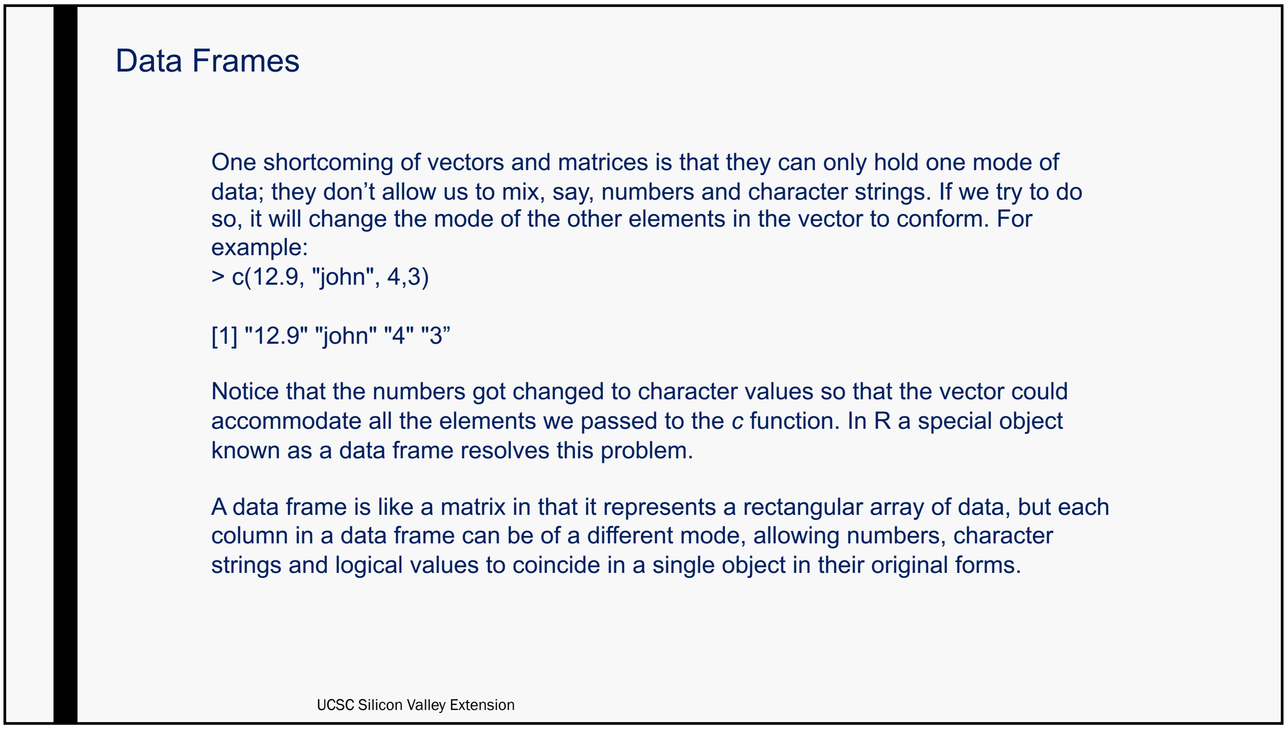
Factors

Factors is a special data structure which are used to categorize the data and store it as levels. They can store both strings and integers. They are useful in the columns which have a limited number of unique values. Like "Male, "Female" and True, False etc. They are useful in data analysis for statistical modeling.

Simplest form of the factor function :

```
> gender<-c(1,2,1,2,1,2,1,2)
>gender <-factor(gender)
# Ideal form of the factor function :
>gender <-factor(gender, levels = c(1,2), labels = c("male", "female"))
> gender
[1] male female male female male female male female
Levels: male female
```

12



Data Frames

One shortcoming of vectors and matrices is that they can only hold one mode of data; they don’t allow us to mix, say, numbers and character strings. If we try to do so, it will change the mode of the other elements in the vector to conform. For example:

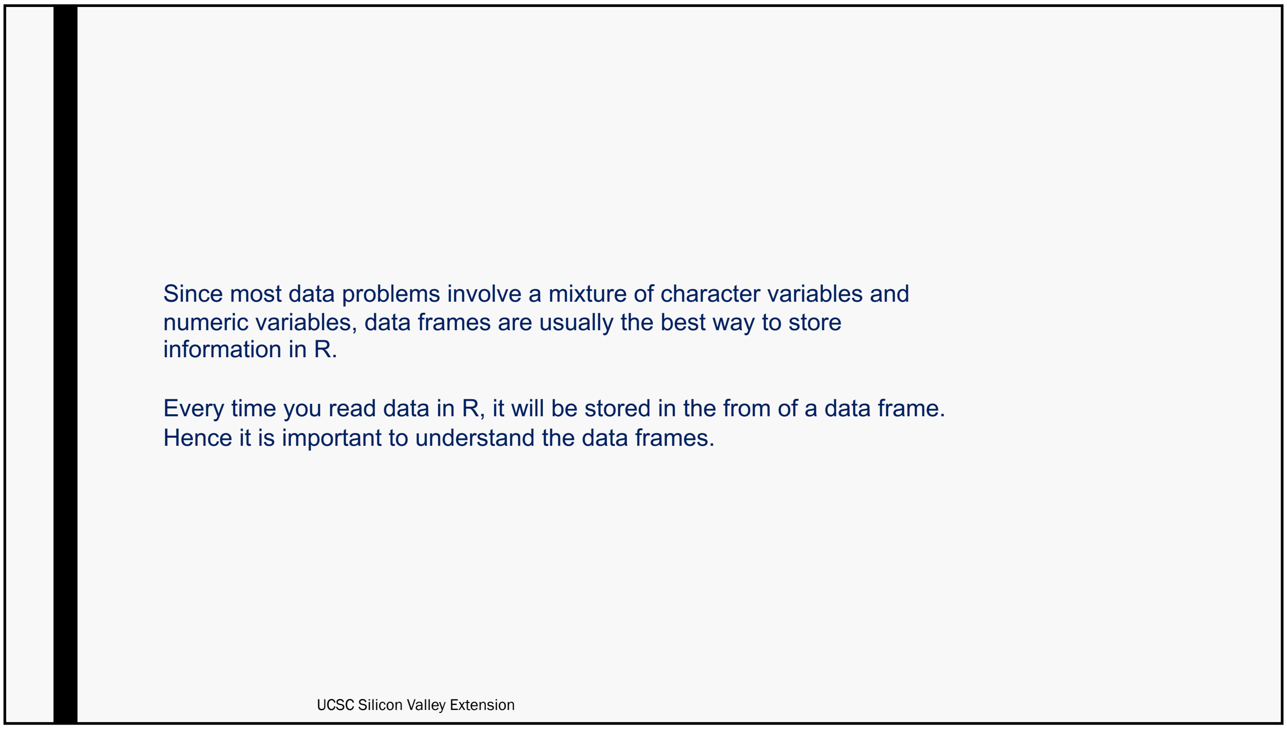
```
> c(12.9, "john", 4,3)

[1] "12.9" "john" "4" "3”
```

Notice that the numbers got changed to character values so that the vector could accommodate all the elements we passed to the *c* function. In R a special object known as a data frame resolves this problem.

A data frame is like a matrix in that it represents a rectangular array of data, but each column in a data frame can be of a different mode, allowing numbers, character strings and logical values to coincide in a single object in their original forms.

UCSC Silicon Valley Extension

13

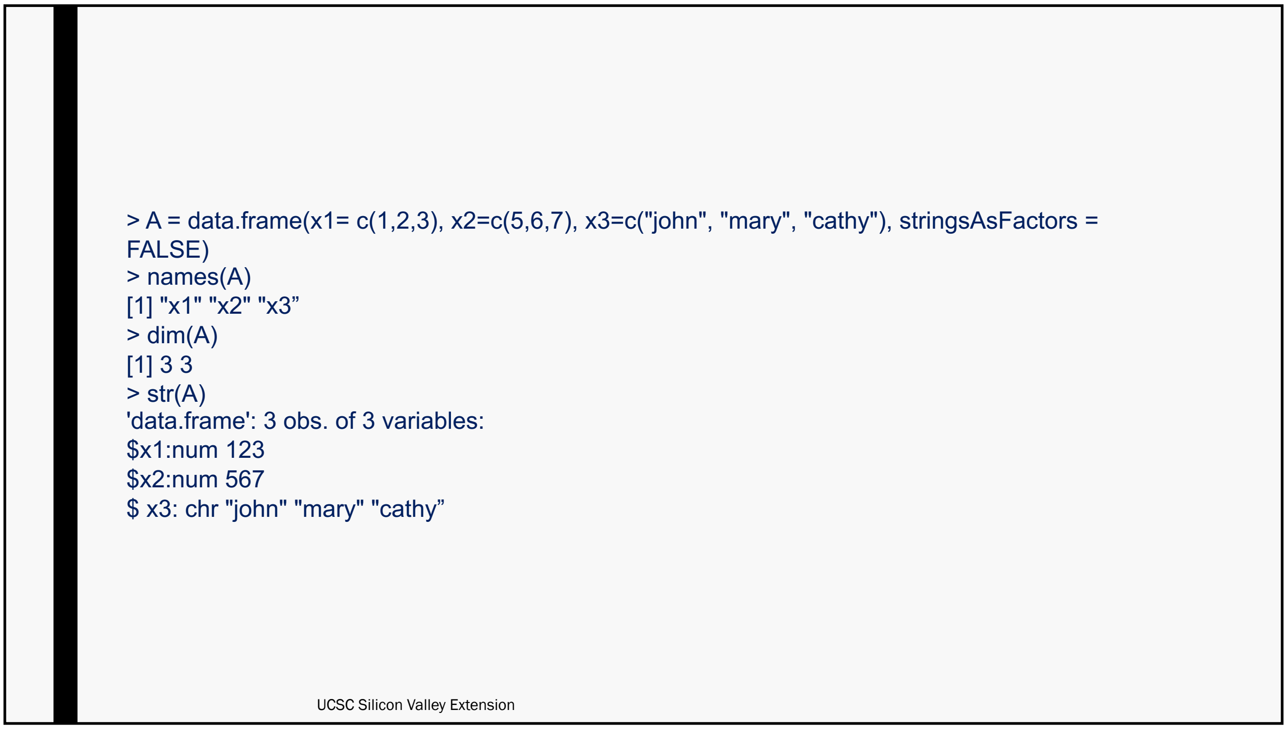


Since most data problems involve a mixture of character variables and numeric variables, data frames are usually the best way to store information in R.

Every time you read data in R, it will be stored in the from of a data frame. Hence it is important to understand the data frames.

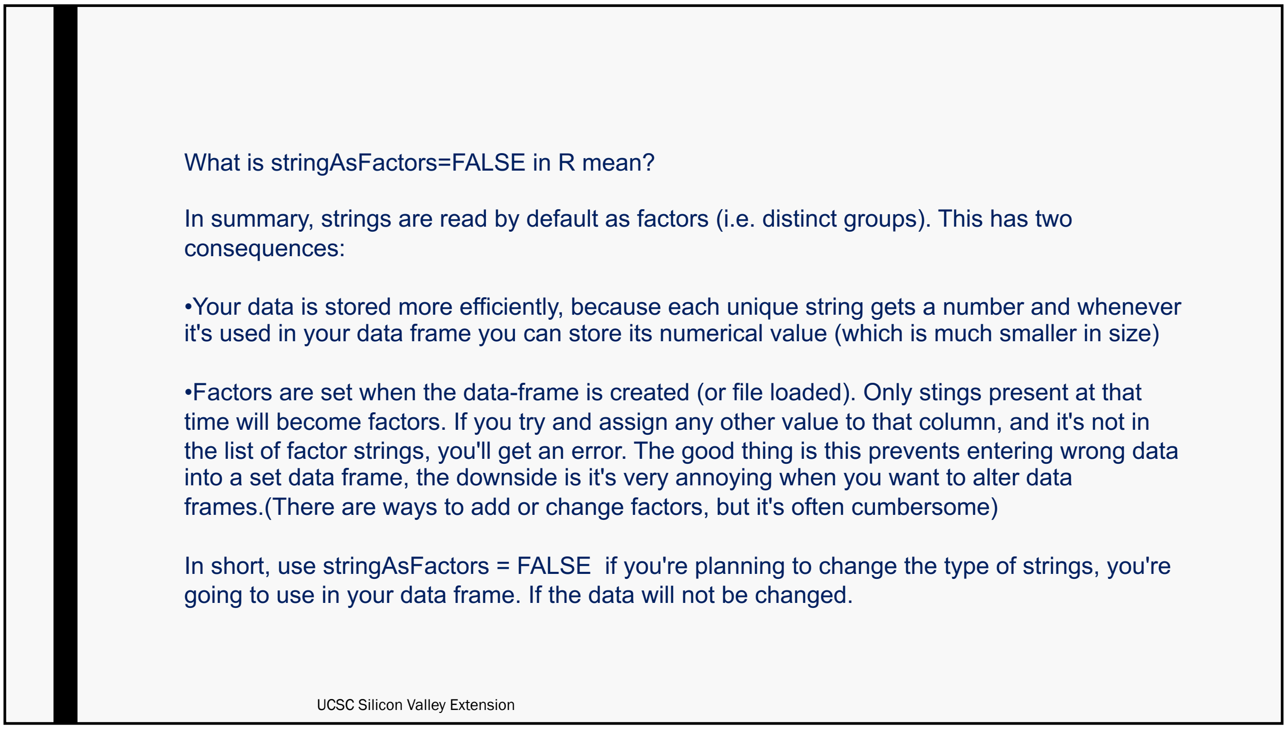
UCSC Silicon Valley Extension

14



```
> A = data.frame(x1= c(1,2,3), x2=c(5,6,7), x3=c("john", "mary", "cathy"), stringsAsFactors =
FALSE)
> names(A)
[1] "x1" "x2" "x3”
> dim(A)
[1] 3 3
> str(A)
'data.frame': 3 obs. of 3 variables:
$x1:num 123
$x2:num 567
$ x3: chr "john" "mary" "cathy”
```

15



What is stringAsFactors=FALSE in R mean?

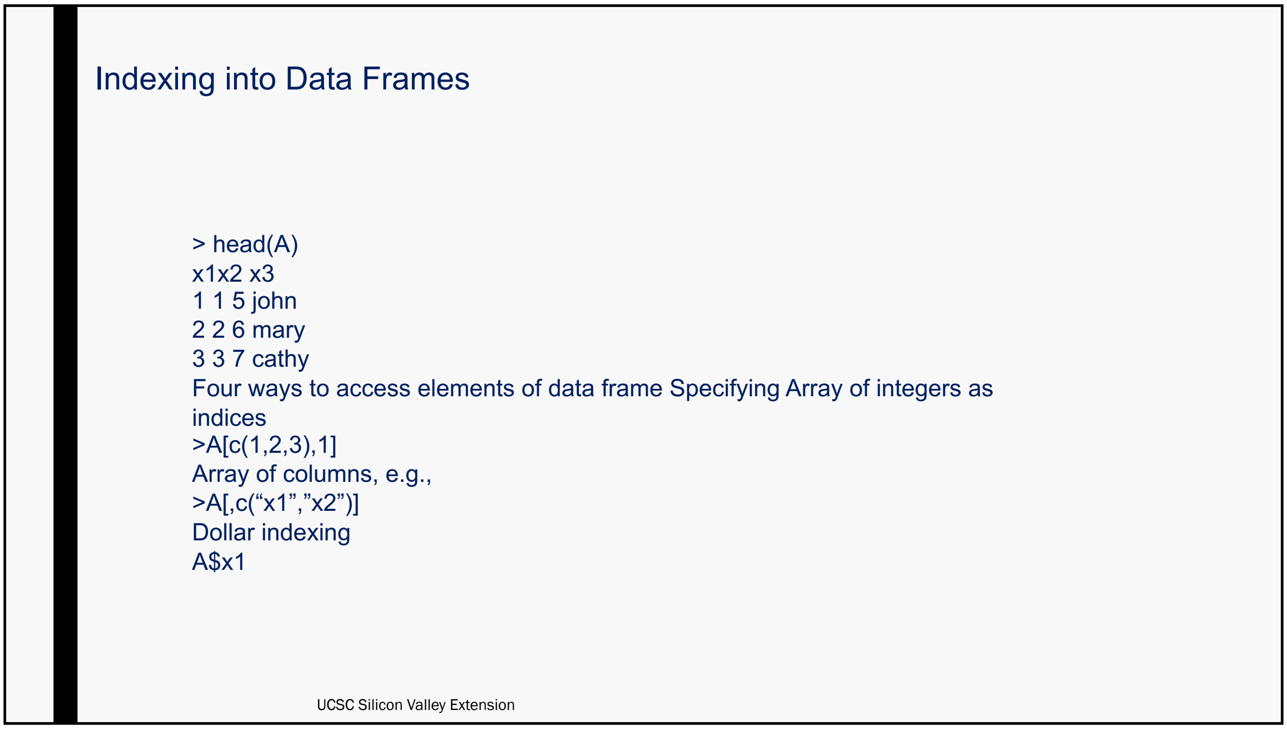
In summary, strings are read by default as factors (i.e. distinct groups). This has two consequences:

•Your data is stored more efficiently, because each unique string gets a number and whenever it's used in your data frame you can store its numerical value (which is much smaller in size)

•Factors are set when the data-frame is created (or file loaded). Only stings present at that time will become factors. If you try and assign any other value to that column, and it's not in the list of factor strings, you'll get an error. The good thing is this prevents entering wrong data into a set data frame, the downside is it's very annoying when you want to alter data frames.(There are ways to add or change factors, but it's often cumbersome)

In short, use stringAsFactors = FALSE if you're planning to change the type of strings, you're going to use in your data frame. If the data will not be changed.

16



Indexing into Data Frames

```
> head(A)
x1x2 x3
1 1 5 john
2 2 6 mary
3 3 7 cathy
```

Four ways to access elements of data frame Specifying Array of integers as indices

```
>A[c(1,2,3),1]
```

Array of columns, e.g.,

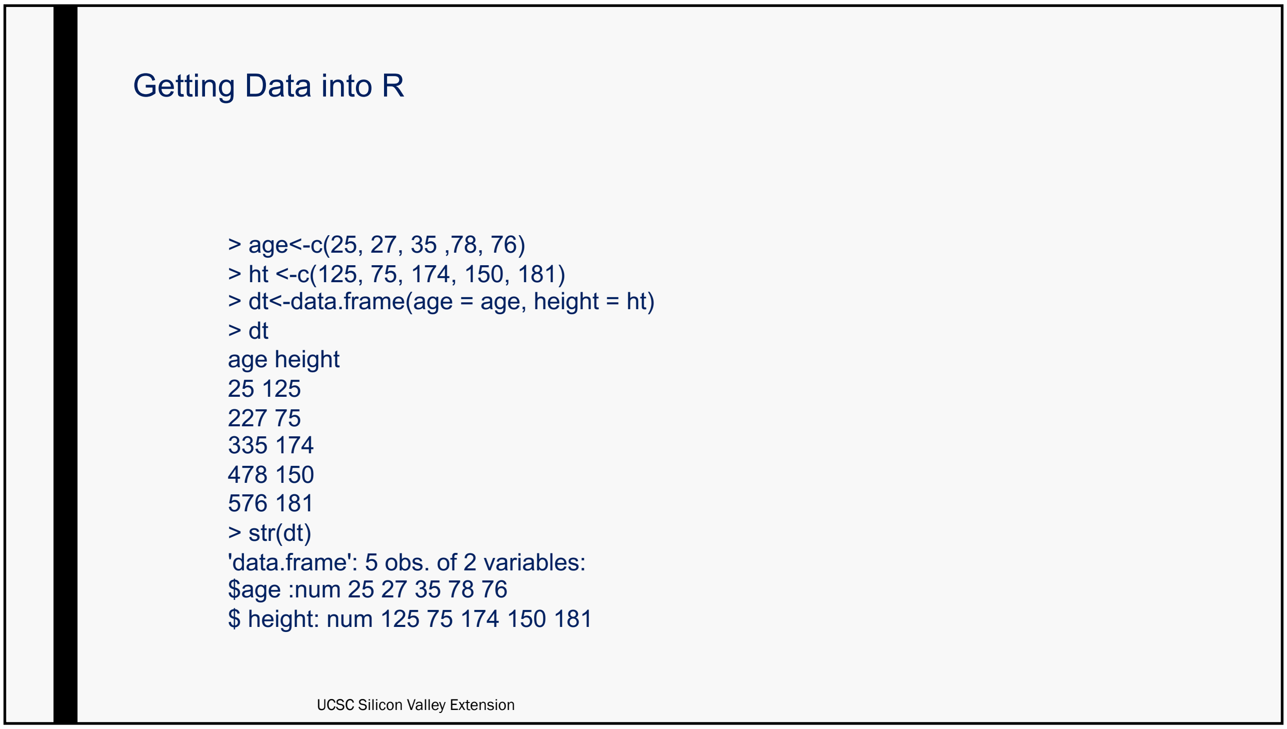
```
>A[,c(“x1”,”x2”)]
```

Dollar indexing

```
A$x1
```

UCSC Silicon Valley Extension

17



Getting Data into R

```
> age<-c(25, 27, 35 ,78, 76)
> ht <-c(125, 75, 174, 150, 181)
> dt<-data.frame(age = age, height = ht)
> dt
age height
25 125
227 75
335 174
478 150
576 181
> str(dt)
'data.frame': 5 obs. of 2 variables:
$age :num 25 27 35 78 76
$ height: num 125 75 174 150 181
```

UCSC Silicon Valley Extension

18

Assignment – Create Data Frame

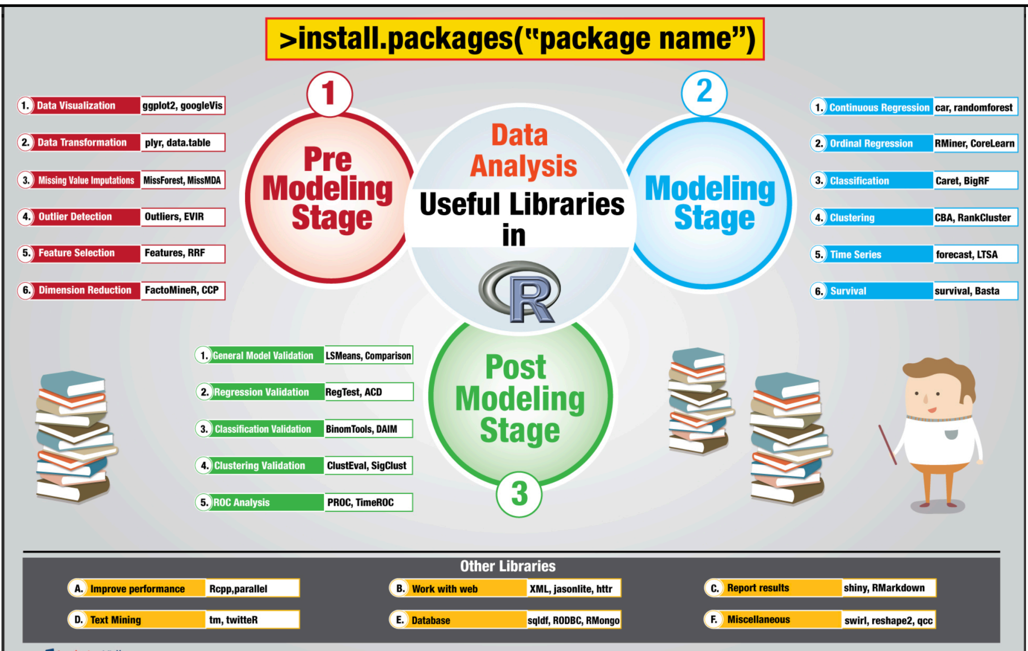
Create a Data Frame named emp.data

Column Names:

- emp_id
- emp_name
- salary
- start_date

1. Print the data frame
2. Get the structure of data frame
3. Print Summary of data frame
4. Extract specific Columns
5. Extract First two rows
6. Extract 3rd and 5th row with 2nd and 4th column
7. Add a new column “dept”
8. Create the second data frame
9. Bind the two data frames

19



>install.packages(“package name”)

1. Data Visualization	ggplot2, googleVis
2. Data Transformation	plyr, data.table
3. Missing Value Imputations	MissForest, MissMDA
4. Outlier Detection	Outliers, EVIR
5. Feature Selection	Features, RRF
6. Dimension Reduction	FactoMineR, CCP

1. Continuous Regression	car, randomforest
2. Ordinal Regression	RMiner, CoreLearn
3. Classification	Caret, BigRF
4. Clustering	CBA, RankCluster
5. Time Series	forecast, LTSA
6. Survival	survival, Basta

1. General Model Validation	LSMeans, Comparison
2. Regression Validation	RegTest, ACD
3. Classification Validation	BinomTools, DAIM
4. Clustering Validation	ClustEval, SigClust
5. ROC Analysis	PROC, TimeROC

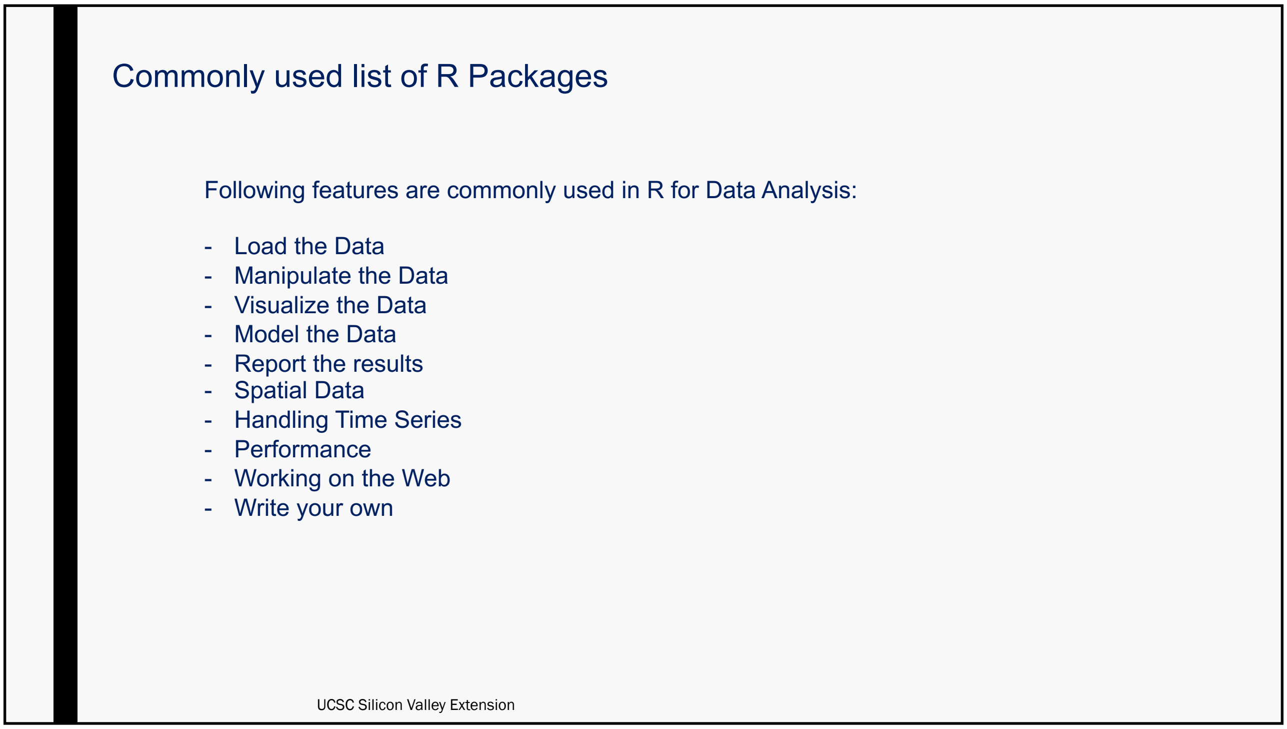
Other Libraries

A. Improve performance	Rcpp,parallel
B. Work with web	XML, jasonlite, httr
C. Report results	shiny, RMarkdown
D. Text Mining	tm, twitteR
E. Database	sqldf, RODBC, RMongo
F. Miscellaneous	swirl, reshape2, qcc

Analytics Vidhya
Learn Everything About Analytics

Source: analyticsvidhya.com

20



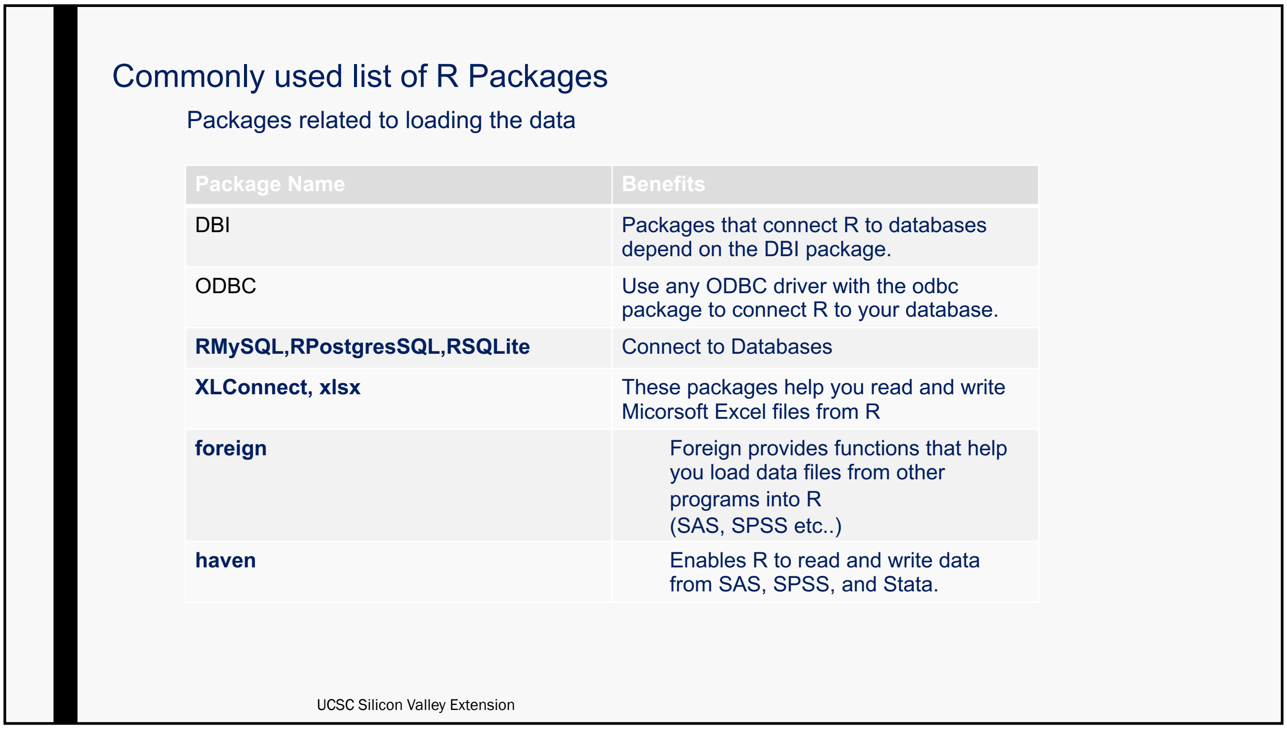
Commonly used list of R Packages

Following features are commonly used in R for Data Analysis:

- Load the Data
- Manipulate the Data
- Visualize the Data
- Model the Data
- Report the results
- Spatial Data
- Handling Time Series
- Performance
- Working on the Web
- Write your own

UCSC Silicon Valley Extension

21



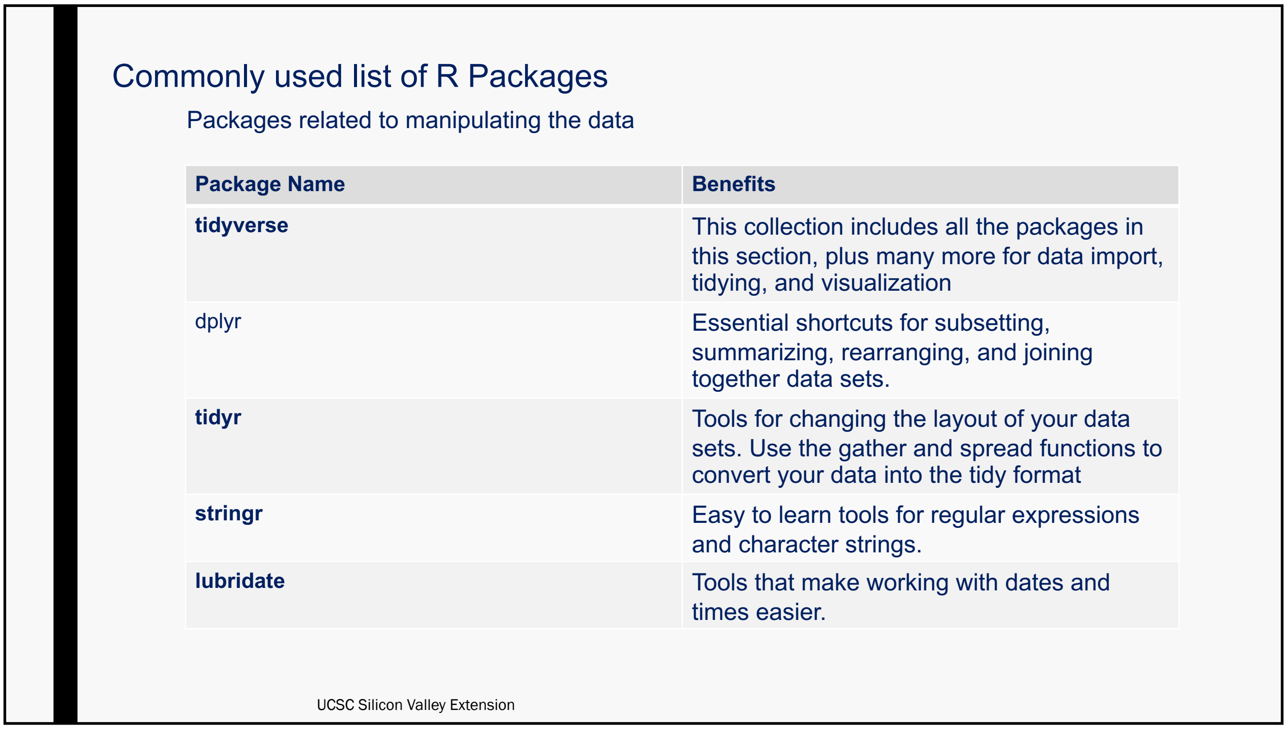
Commonly used list of R Packages

Packages related to loading the data

Package Name	Benefits
DBI	Packages that connect R to databases depend on the DBI package.
ODBC	Use any ODBC driver with the odbc package to connect R to your database.
RMySQL,RPostgresSQL,RSQLite	Connect to Databases
XLConnect, xlsx	These packages help you read and write Micorsoft Excel files from R
foreign	Foreign provides functions that help you load data files from other programs into R (SAS, SPSS etc..)
haven	Enables R to read and write data from SAS, SPSS, and Stata.

UCSC Silicon Valley Extension

22



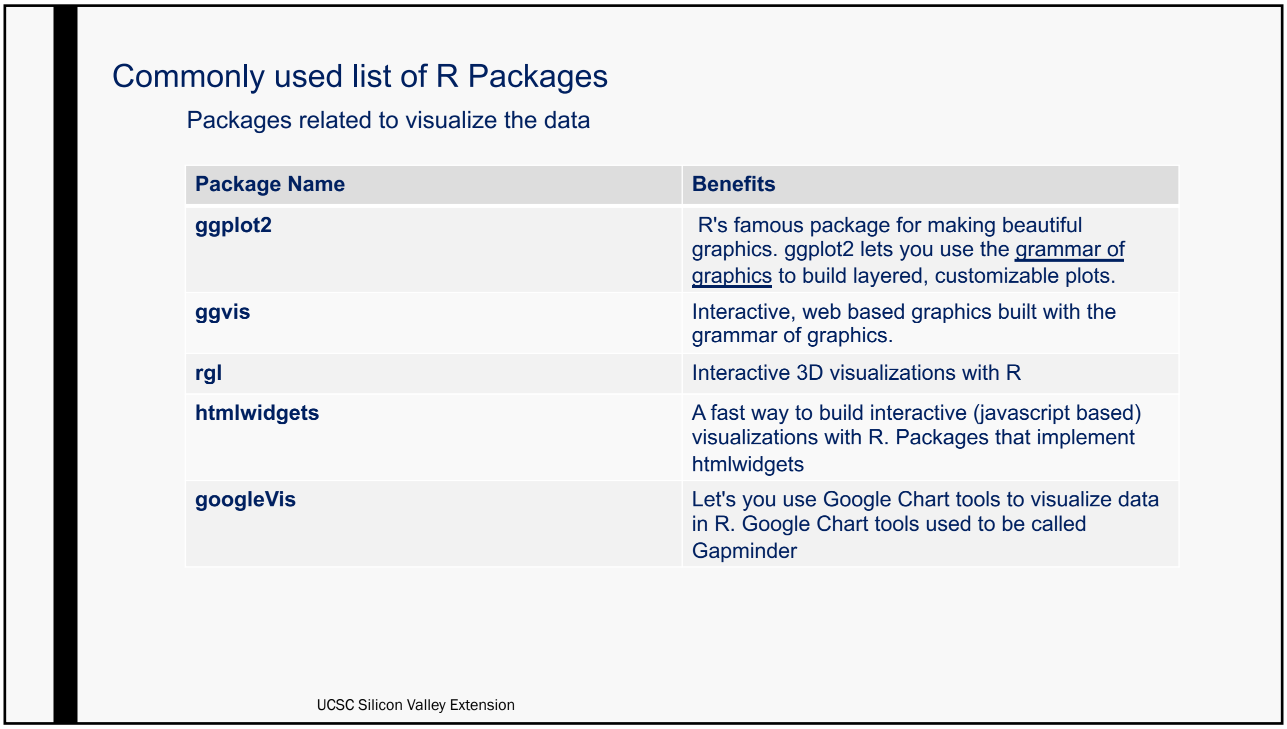
Commonly used list of R Packages

Packages related to manipulating the data

Package Name	Benefits
tidyverse	This collection includes all the packages in this section, plus many more for data import, tidying, and visualization
dplyr	Essential shortcuts for subsetting, summarizing, rearranging, and joining together data sets.
tidyr	Tools for changing the layout of your data sets. Use the gather and spread functions to convert your data into the tidy format
stringr	Easy to learn tools for regular expressions and character strings.
lubridate	Tools that make working with dates and times easier.

UCSC Silicon Valley Extension

23



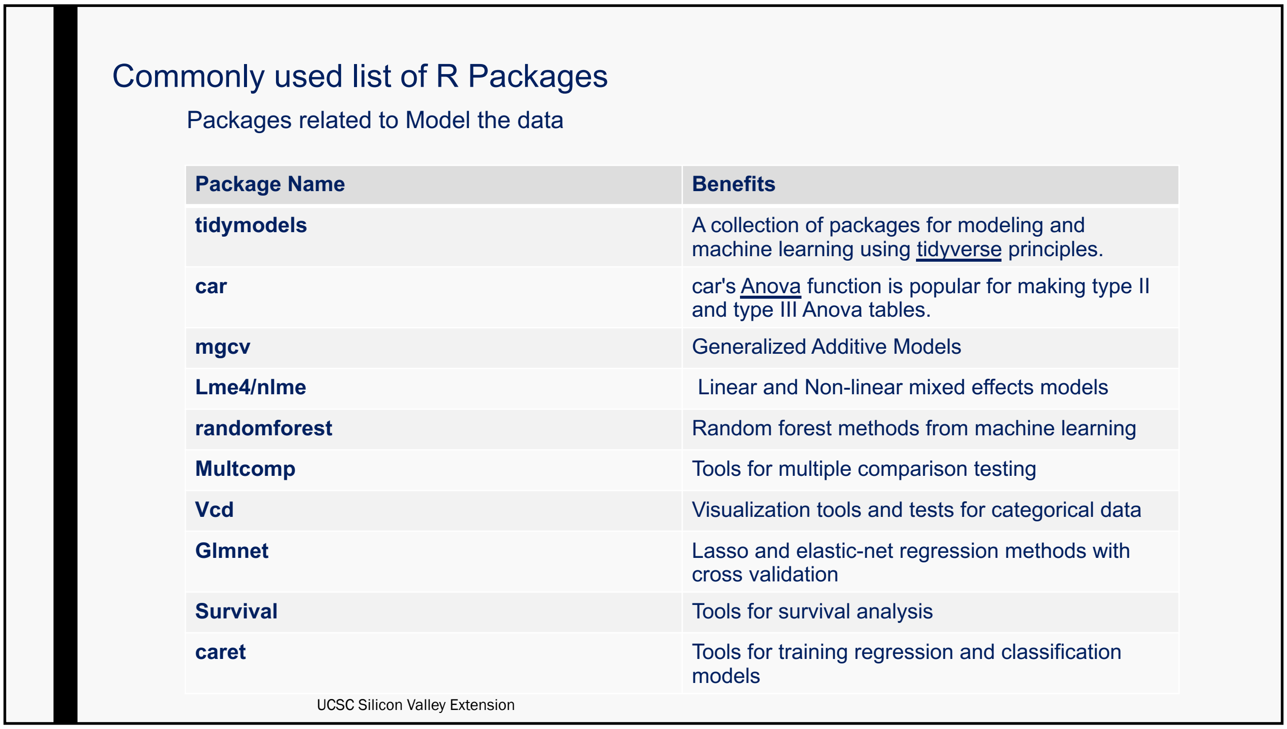
Commonly used list of R Packages

Packages related to visualize the data

Package Name	Benefits
ggplot2	R's famous package for making beautiful graphics. ggplot2 lets you use the grammar of graphics to build layered, customizable plots.
ggvis	Interactive, web based graphics built with the grammar of graphics.
rgl	Interactive 3D visualizations with R
htmlwidgets	A fast way to build interactive (javascript based) visualizations with R. Packages that implement htmlwidgets
googleVis	Let's you use Google Chart tools to visualize data in R. Google Chart tools used to be called Gapminder

UCSC Silicon Valley Extension

24



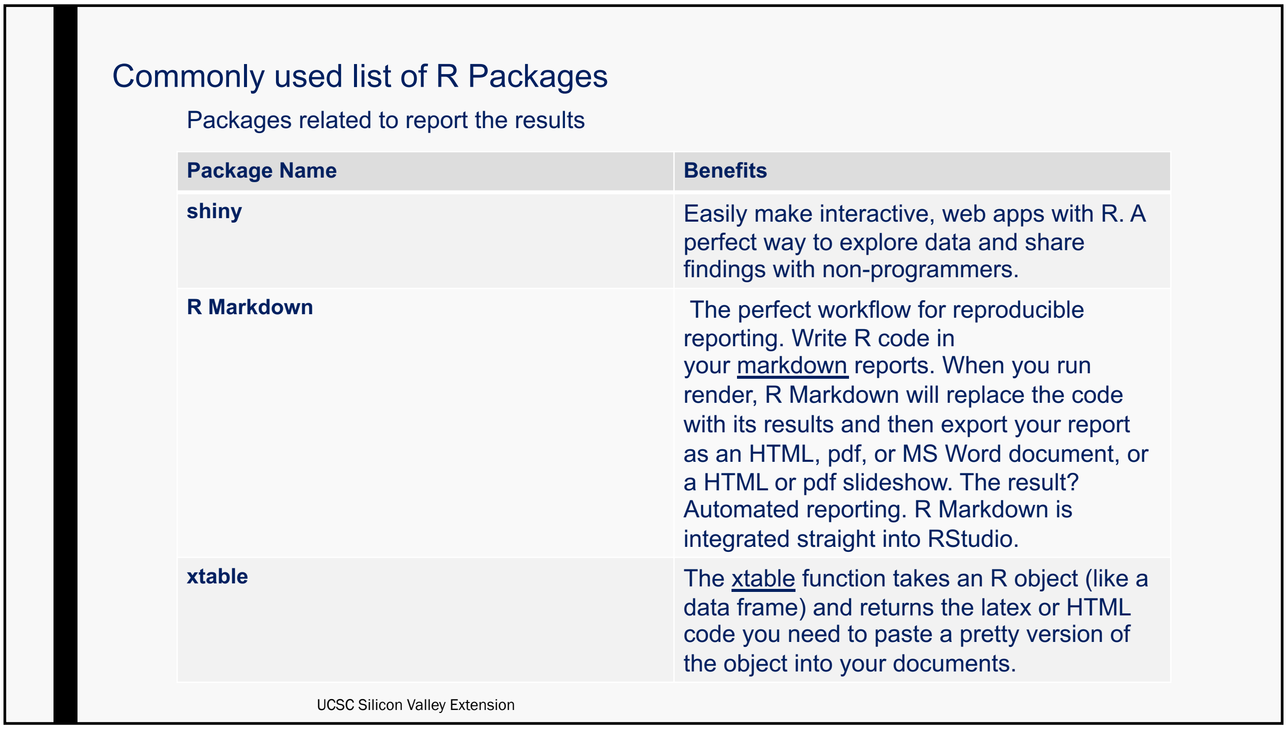
Commonly used list of R Packages

Packages related to Model the data

Package Name	Benefits
tidymodels	A collection of packages for modeling and machine learning using tidyverse principles.
car	car's Anova function is popular for making type II and type III Anova tables.
mgcv	Generalized Additive Models
Lme4/nlme	Linear and Non-linear mixed effects models
randomforest	Random forest methods from machine learning
Multcomp	Tools for multiple comparison testing
Vcd	Visualization tools and tests for categorical data
Glmnet	Lasso and elastic-net regression methods with cross validation
Survival	Tools for survival analysis
caret	Tools for training regression and classification models

UCSC Silicon Valley Extension

25



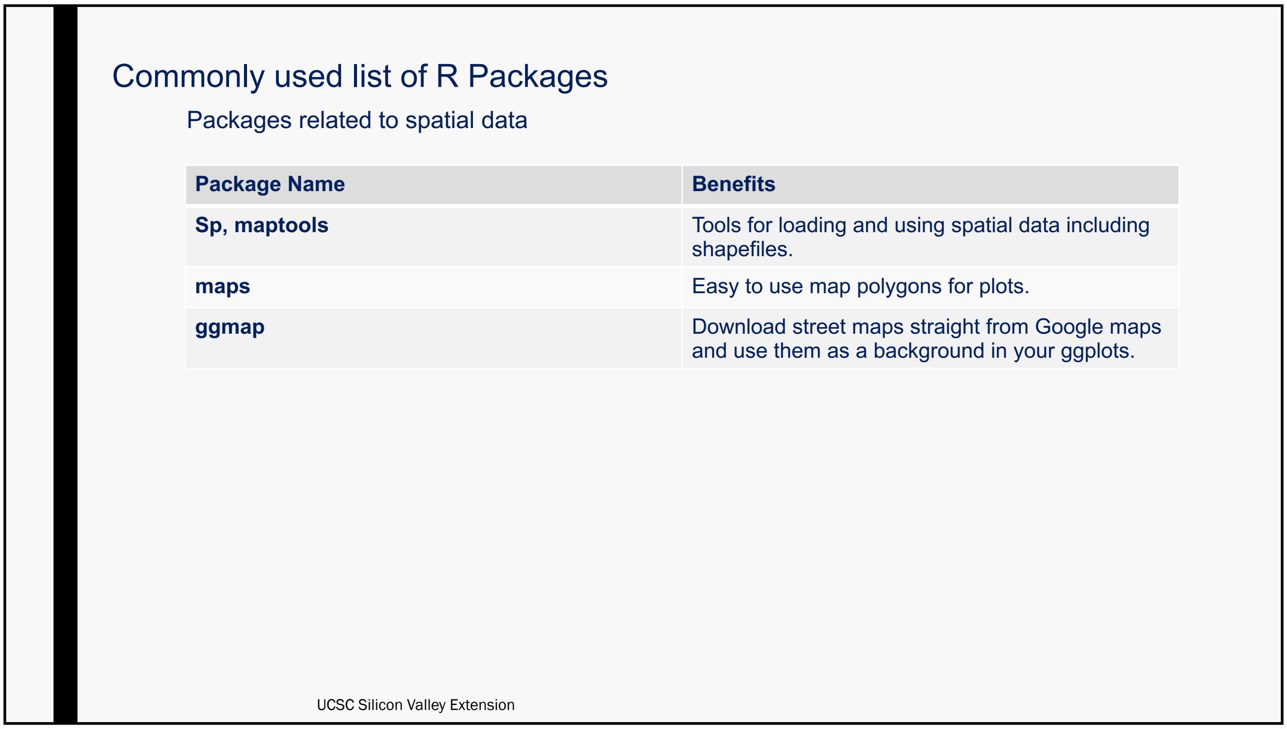
Commonly used list of R Packages

Packages related to report the results

Package Name	Benefits
shiny	Easily make interactive, web apps with R. A perfect way to explore data and share findings with non-programmers.
R Markdown	The perfect workflow for reproducible reporting. Write R code in your markdown reports. When you run render, R Markdown will replace the code with its results and then export your report as an HTML, pdf, or MS Word document, or a HTML or pdf slideshow. The result? Automated reporting. R Markdown is integrated straight into RStudio.
xtable	The xtable function takes an R object (like a data frame) and returns the latex or HTML code you need to paste a pretty version of the object into your documents.

UCSC Silicon Valley Extension

26

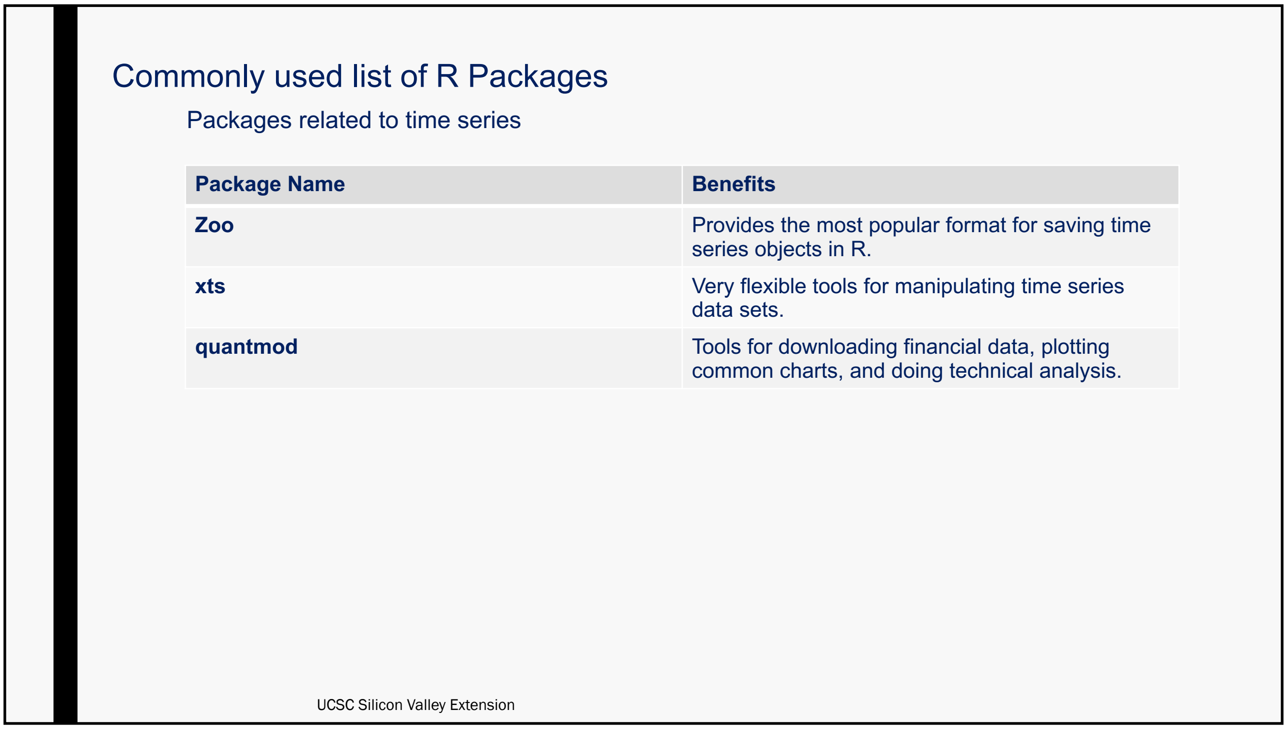


Commonly used list of R Packages

Packages related to spatial data

Package Name	Benefits
Sp, maptools	Tools for loading and using spatial data including shapefiles.
maps	Easy to use map polygons for plots.
ggmap	Download street maps straight from Google maps and use them as a background in your ggplots.

27

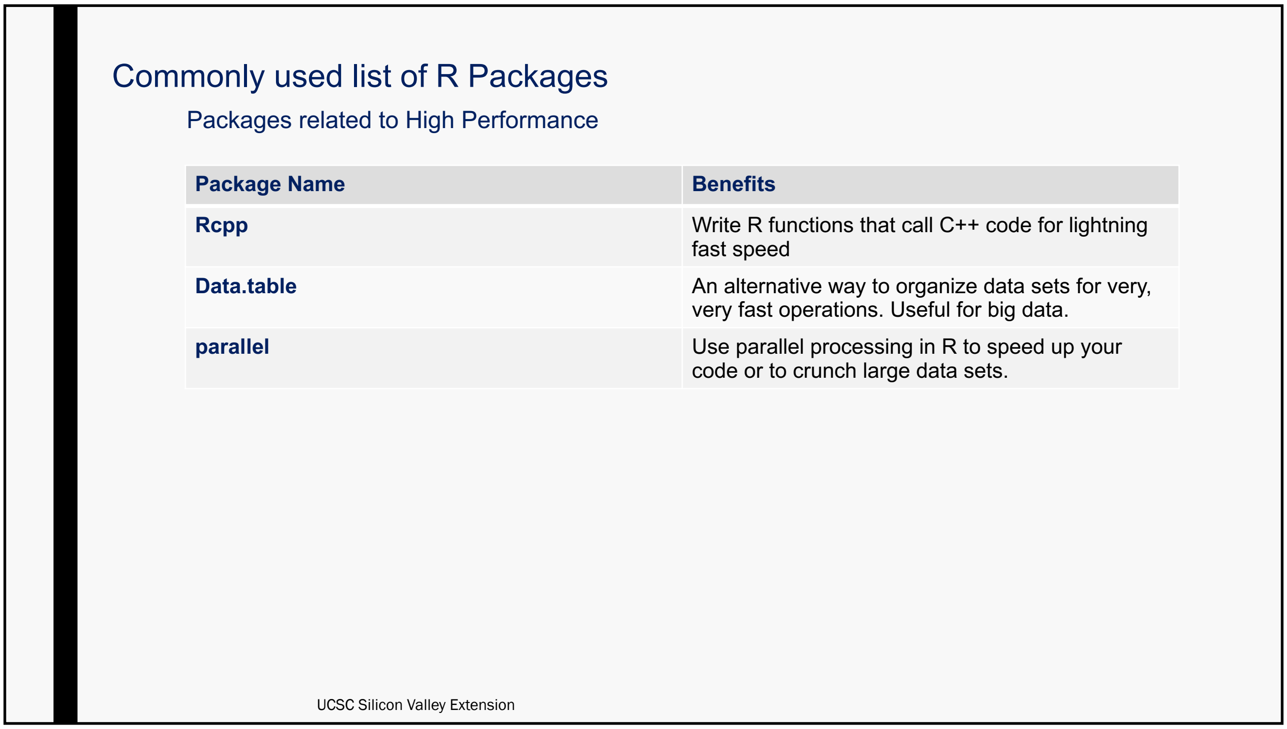


Commonly used list of R Packages

Packages related to time series

Package Name	Benefits
Zoo	Provides the most popular format for saving time series objects in R.
xts	Very flexible tools for manipulating time series data sets.
quantmod	Tools for downloading financial data, plotting common charts, and doing technical analysis.

28



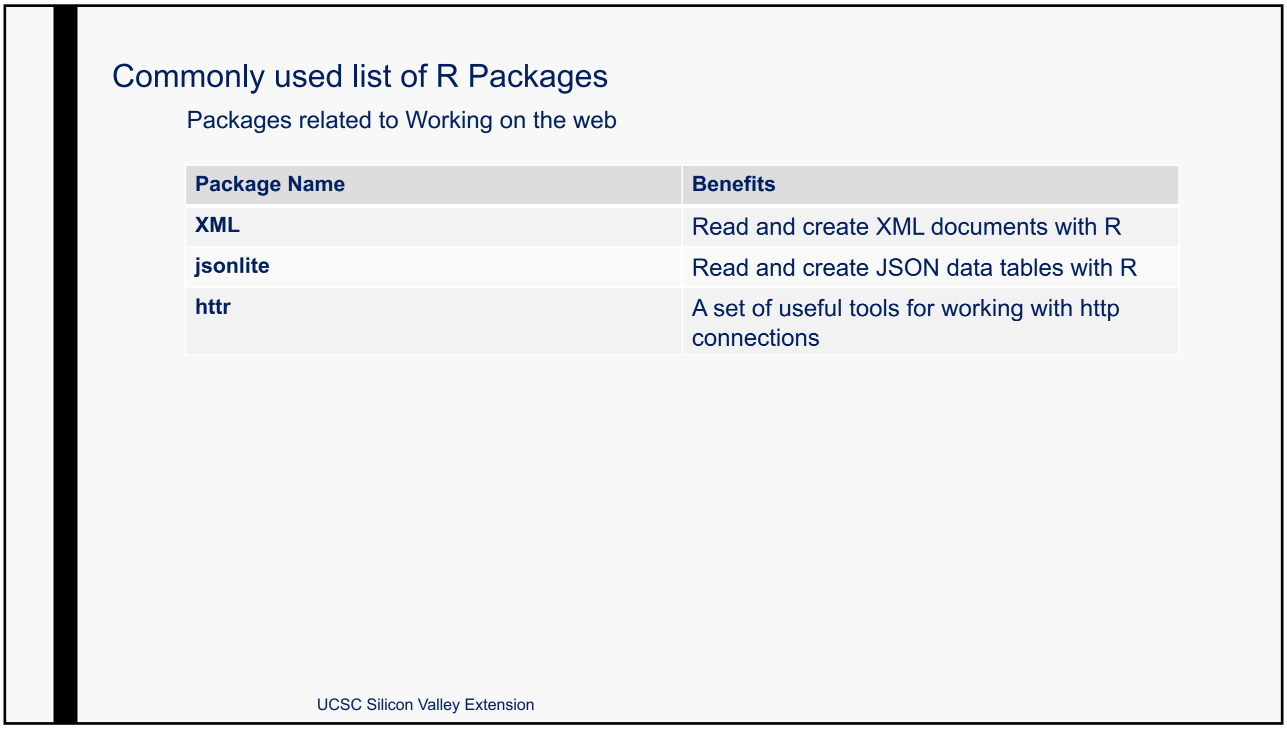
Commonly used list of R Packages

Packages related to High Performance

Package Name	Benefits
Rcpp	Write R functions that call C++ code for lightning fast speed
Data.table	An alternative way to organize data sets for very, very fast operations. Useful for big data.
parallel	Use parallel processing in R to speed up your code or to crunch large data sets.

UCSC Silicon Valley Extension

29



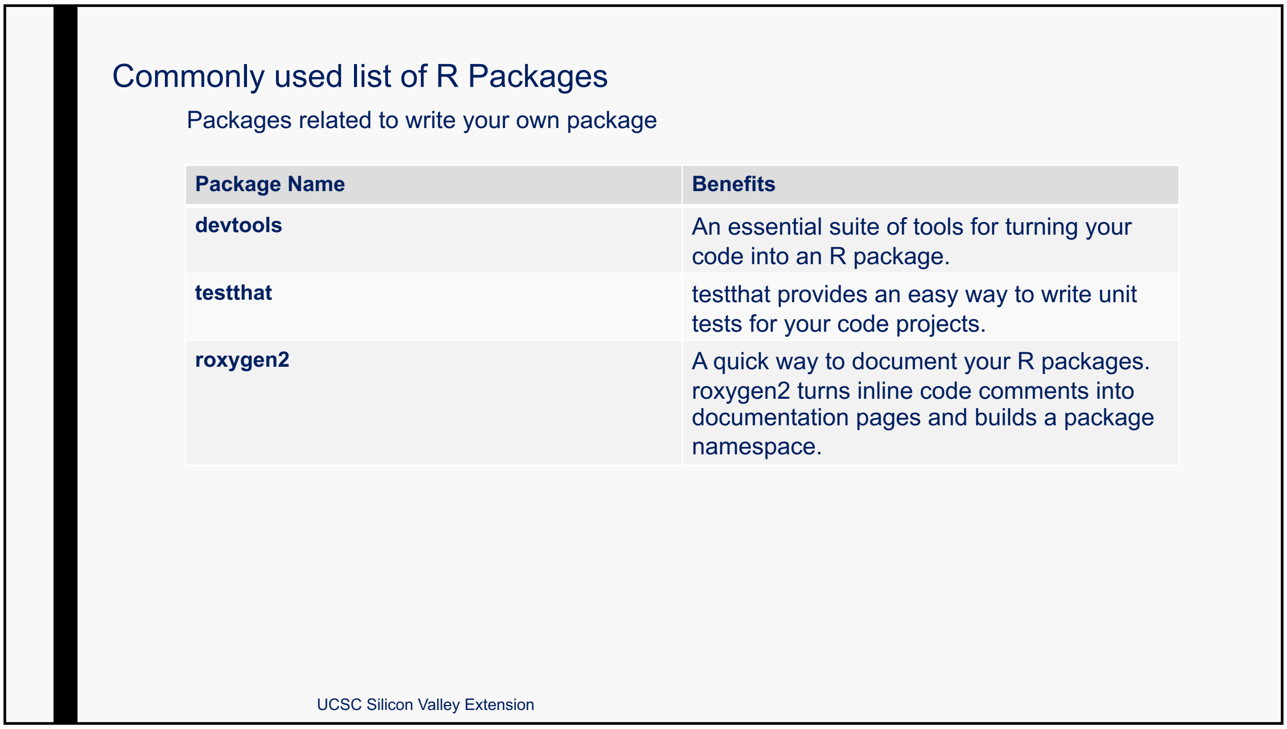
Commonly used list of R Packages

Packages related to Working on the web

Package Name	Benefits
XML	Read and create XML documents with R
jsonlite	Read and create JSON data tables with R
httr	A set of useful tools for working with http connections

UCSC Silicon Valley Extension

30



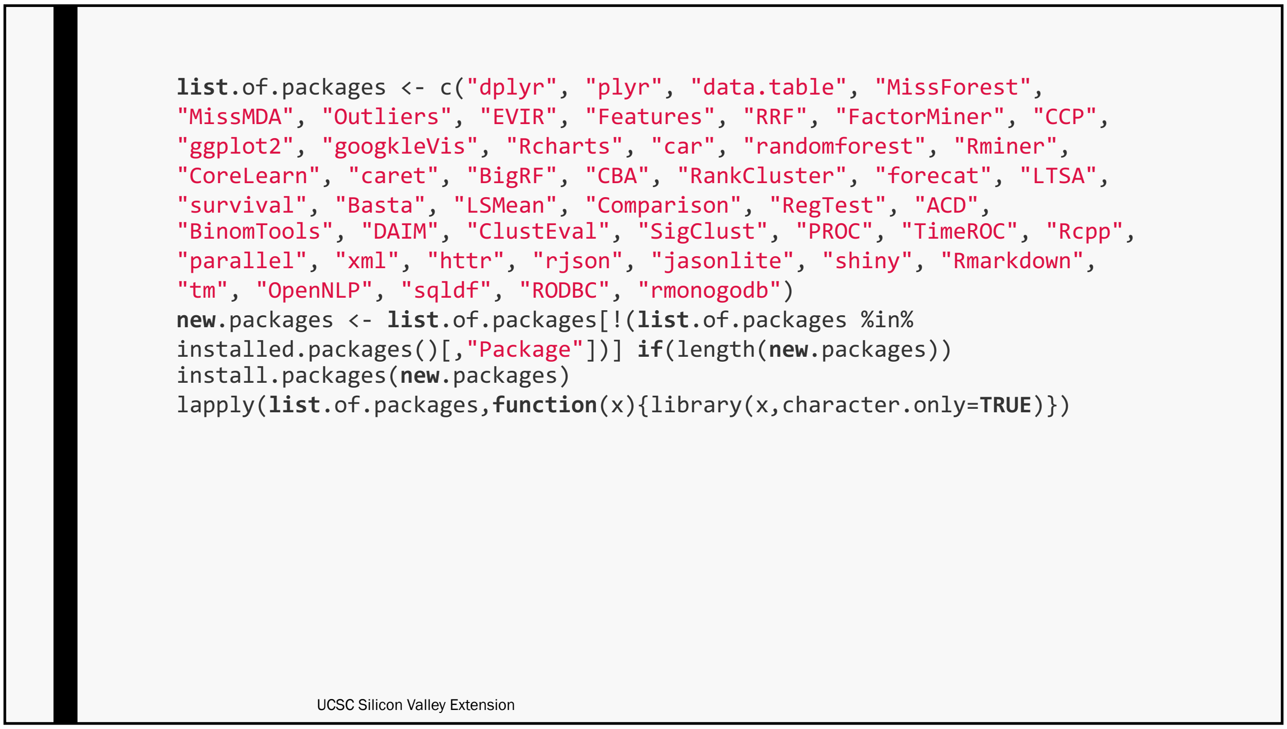
Commonly used list of R Packages

Packages related to write your own package

Package Name	Benefits
devtools	An essential suite of tools for turning your code into an R package.
testthat	testthat provides an easy way to write unit tests for your code projects.
roxygen2	A quick way to document your R packages. roxygen2 turns inline code comments into documentation pages and builds a package namespace.

UCSC Silicon Valley Extension

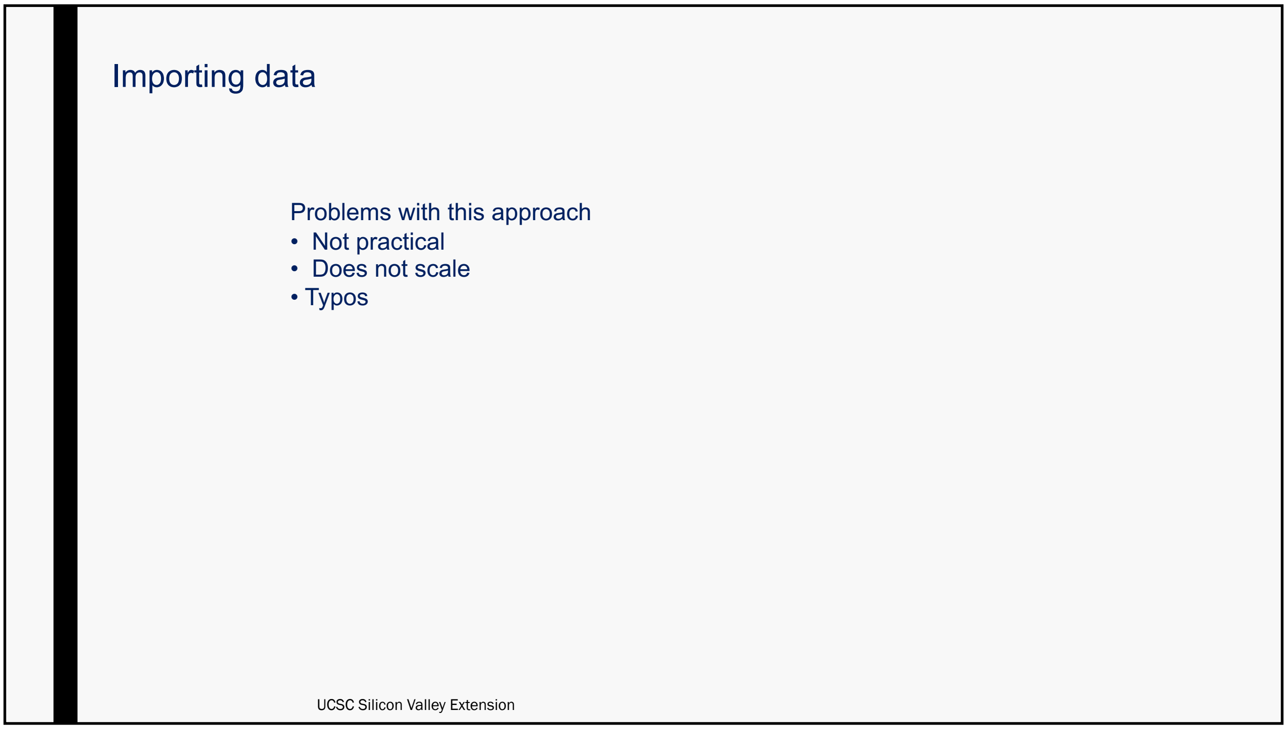
31



```
list.of.packages <- c("dplyr", "plyr", "data.table", "MissForest",
"MissMDA", "Outliers", "EVIR", "Features", "RRF", "FactorMiner", "CCP",
"ggplot2", "googkleVis", "Rcharts", "car", "randomforest", "Rminer",
"CoreLearn", "caret", "BigRF", "CBA", "RankCluster", "forecat", "LTSA",
"survival", "Basta", "LSMean", "Comparison", "RegTest", "ACD",
"BinomTools", "DAIM", "ClustEval", "SigClust", "PROC", "TimeROC", "Rcpp",
"parallel", "xml", "httr", "rjson", "jasonlite", "shiny", "Rmarkdown",
"tm", "OpenNLP", "sqldf", "RODBC", "rmonogodb")
new.packages <- list.of.packages[!(list.of.packages %in%
installed.packages()[,"Package"])] if(length(new.packages))
install.packages(new.packages)
lapply(list.of.packages,function(x){library(x,character.only=TRUE)})
```

UCSC Silicon Valley Extension

32



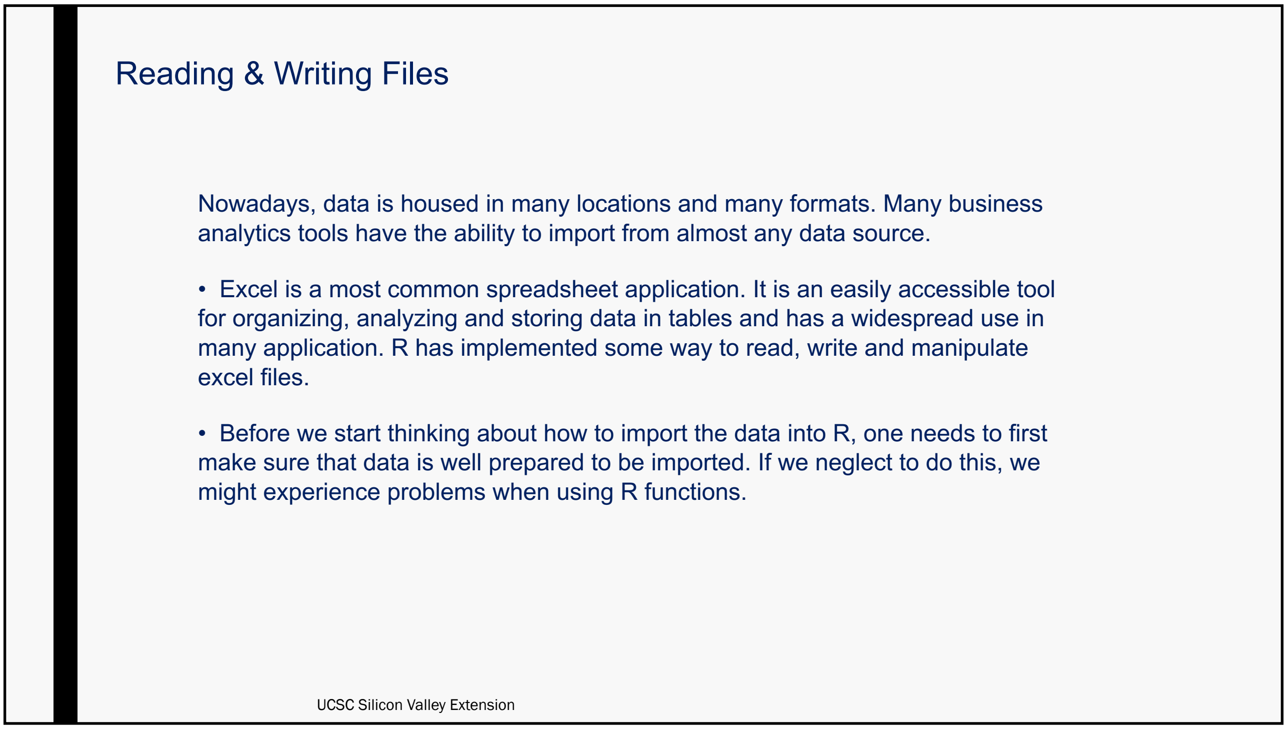
Importing data

Problems with this approach

- Not practical
- Does not scale
- Typos

UCSC Silicon Valley Extension

33



Reading & Writing Files

Nowadays, data is housed in many locations and many formats. Many business analytics tools have the ability to import from almost any data source.

- Excel is a most common spreadsheet application. It is an easily accessible tool for organizing, analyzing and storing data in tables and has a widespread use in many application. R has implemented some way to read, write and manipulate excel files.

- Before we start thinking about how to import the data into R, one needs to first make sure that data is well prepared to be imported. If we neglect to do this, we might experience problems when using R functions.

UCSC Silicon Valley Extension

34

Usually we will be using data already stored in a file that we need to read into R.

R can read data from a variety of file formats – for example, files created in excel, as text, csv, json, etc.

After reading the file in R it is stored in data frame. To read an entire data frame directly, the external file will normally have a special form

The first line of the file should have a name for each variable in the data frame known as header. Each additional line of the file has as its first item a row label and the values for each variable.

35

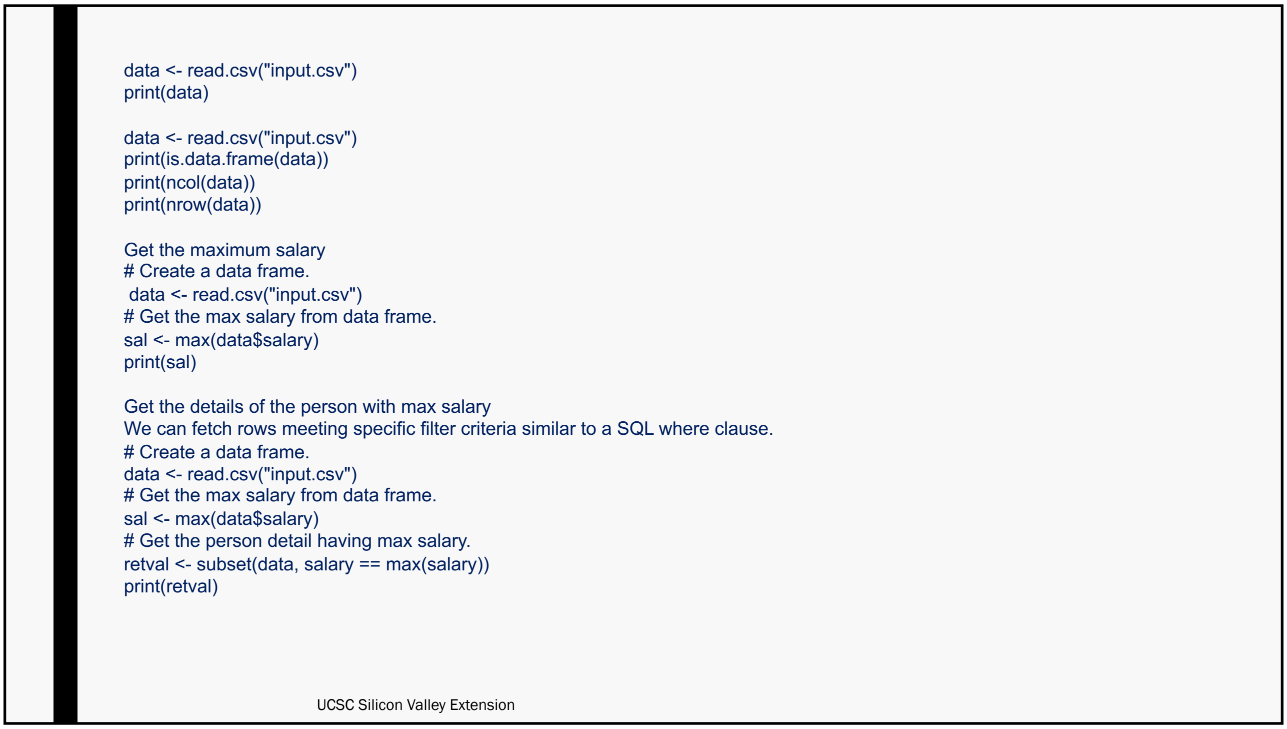
Reading Data in R

For reading (importing) data into R following are some functions

- **read.csv()** for reads comma delimited files, **read_csv2()** reads semicolon separated files, **read_tsv()** reads tab delimited files, and **read_delim()** reads in files with any delimiter.
- **readLines** for reading lines of a text file
- **read.xlsx** for reading .xlsx files
- **read.fwf** for reading fixed width format
- **load()** for reading in saved workspaces
- **read_log()** reads Apache style log files.

These functions all have similar syntax: once you've mastered one, you can use the others with ease.

36



```
data <- read.csv("input.csv")
print(data)

data <- read.csv("input.csv")
print(is.data.frame(data))
print(ncol(data))
print(nrow(data))
```

Get the maximum salary

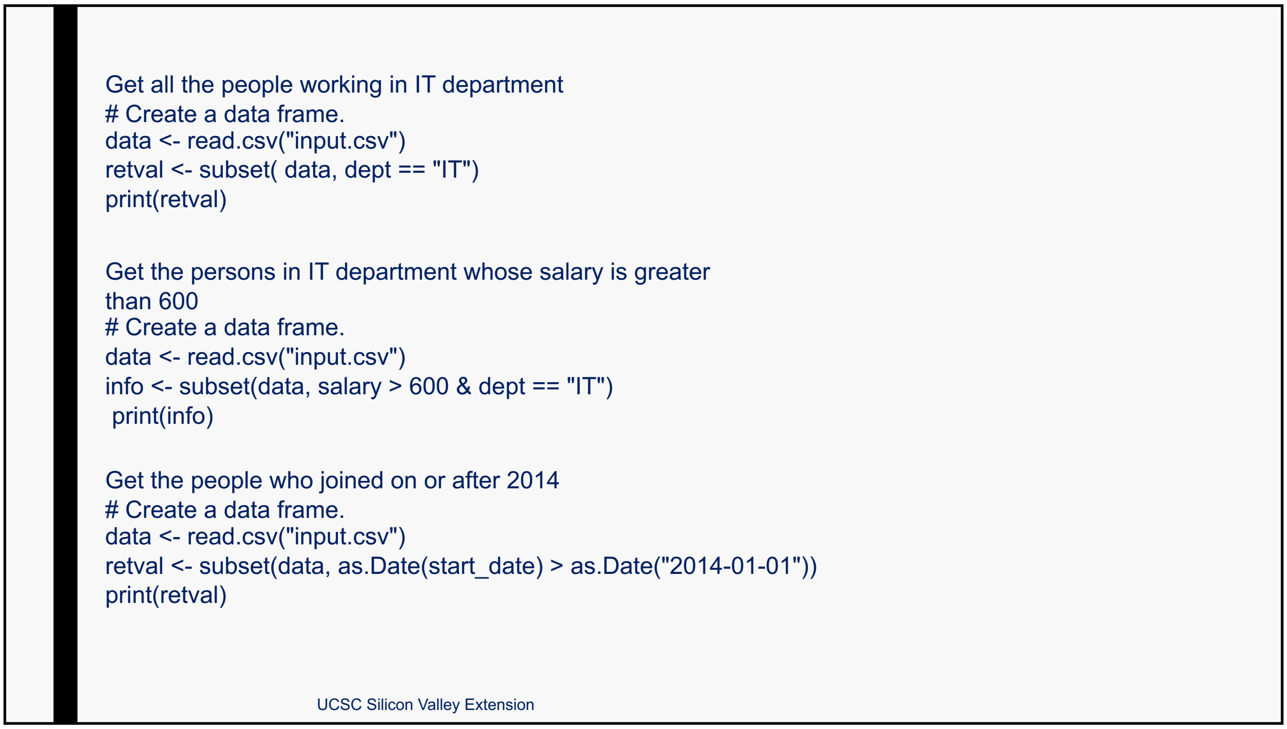
```
# Create a data frame.
 data <- read.csv("input.csv")
# Get the max salary from data frame.
sal <- max(data$salary)
print(sal)
```

Get the details of the person with max salary

We can fetch rows meeting specific filter criteria similar to a SQL where clause.

```
# Create a data frame.
data <- read.csv("input.csv")
# Get the max salary from data frame.
sal <- max(data$salary)
# Get the person detail having max salary.
retval <- subset(data, salary == max(salary))
print(retval)
```

37



Get all the people working in IT department

```
# Create a data frame.
data <- read.csv("input.csv")
retval <- subset( data, dept == "IT")
print(retval)
```

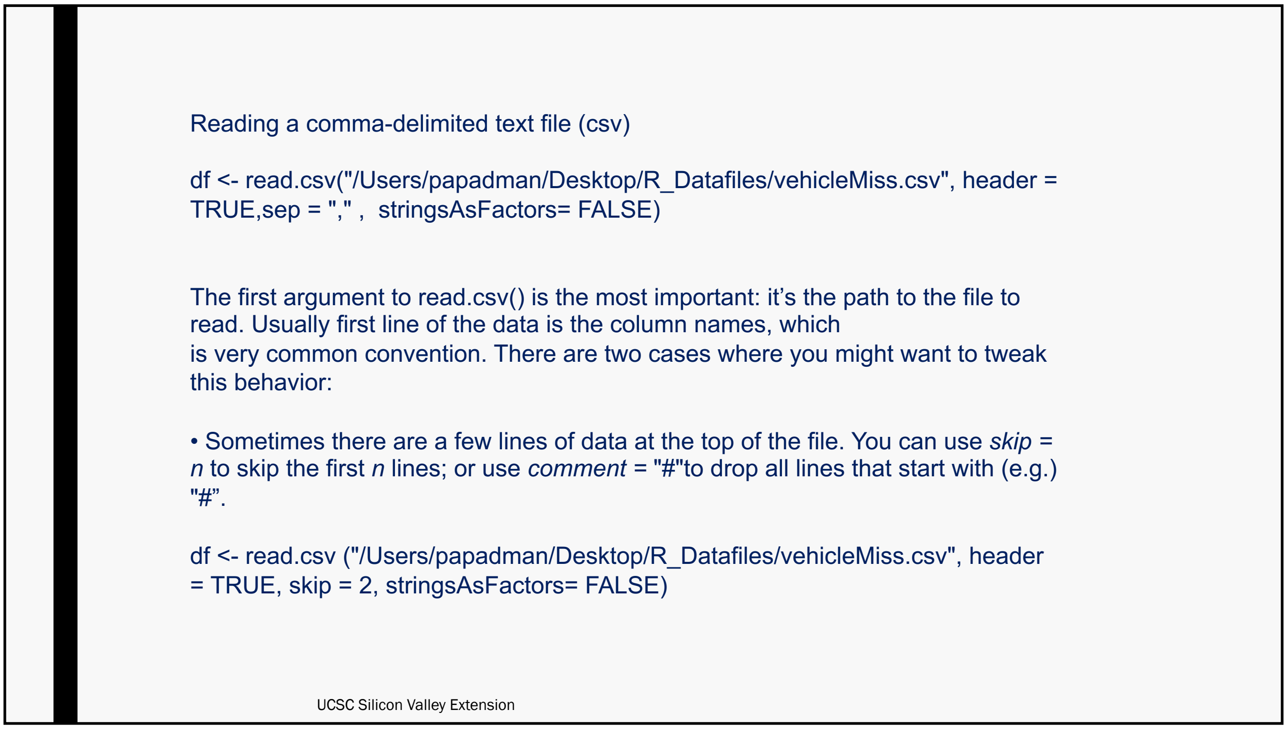
Get the persons in IT department whose salary is greater than 600

```
# Create a data frame.
data <- read.csv("input.csv")
info <- subset(data, salary > 600 & dept == "IT")
 print(info)
```

Get the people who joined on or after 2014

```
# Create a data frame.
data <- read.csv("input.csv")
retval <- subset(data, as.Date(start_date) > as.Date("2014-01-01"))
print(retval)
```

38



Reading a comma-delimited text file (csv)

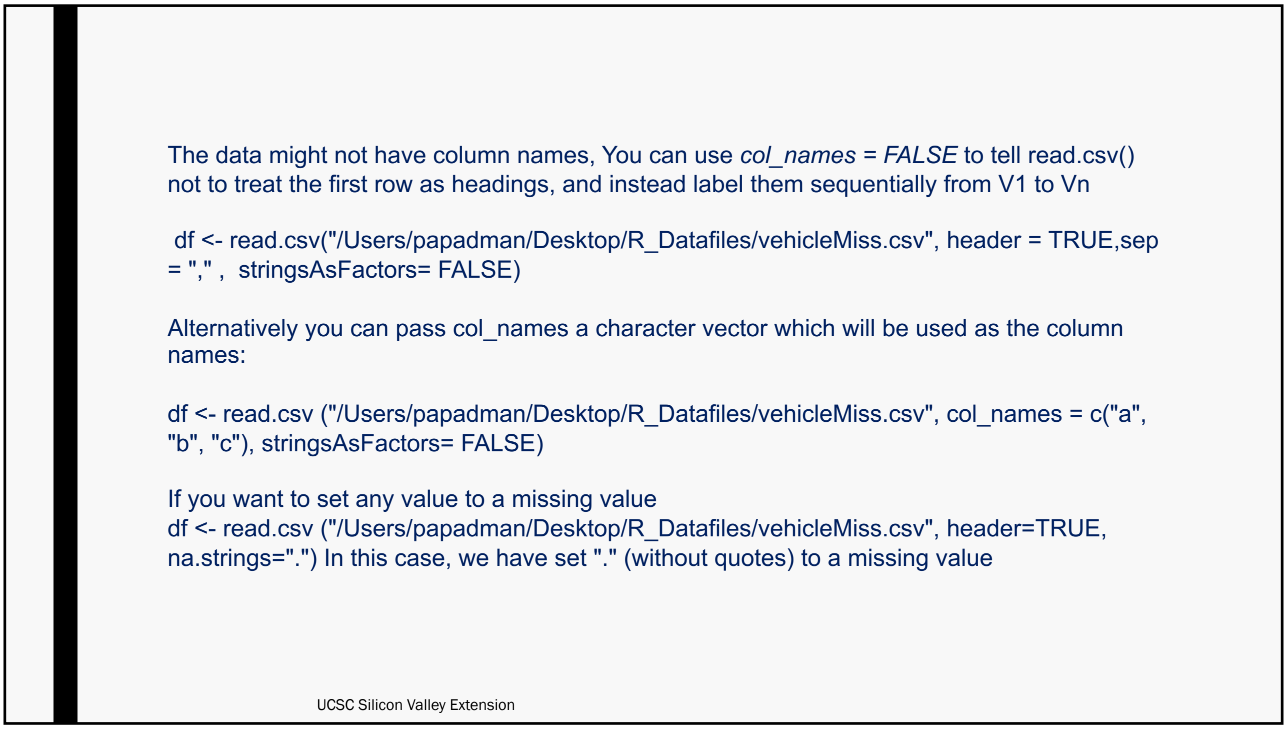
```
df <- read.csv("/Users/papadman/Desktop/R_Datafiles/vehicleMiss.csv", header =
TRUE,sep = "," ,  stringsAsFactors= FALSE)
```

The first argument to read.csv() is the most important: it's the path to the file to read. Usually first line of the data is the column names, which is very common convention. There are two cases where you might want to tweak this behavior:

- Sometimes there are a few lines of data at the top of the file. You can use *skip* = *n* to skip the first *n* lines; or use *comment* = "#"to drop all lines that start with (e.g.) "#".

```
df <- read.csv ("/Users/papadman/Desktop/R_Datafiles/vehicleMiss.csv", header
= TRUE, skip = 2, stringsAsFactors= FALSE)
```

39



The data might not have column names, You can use *col_names = FALSE* to tell read.csv() not to treat the first row as headings, and instead label them sequentially from V1 to Vn

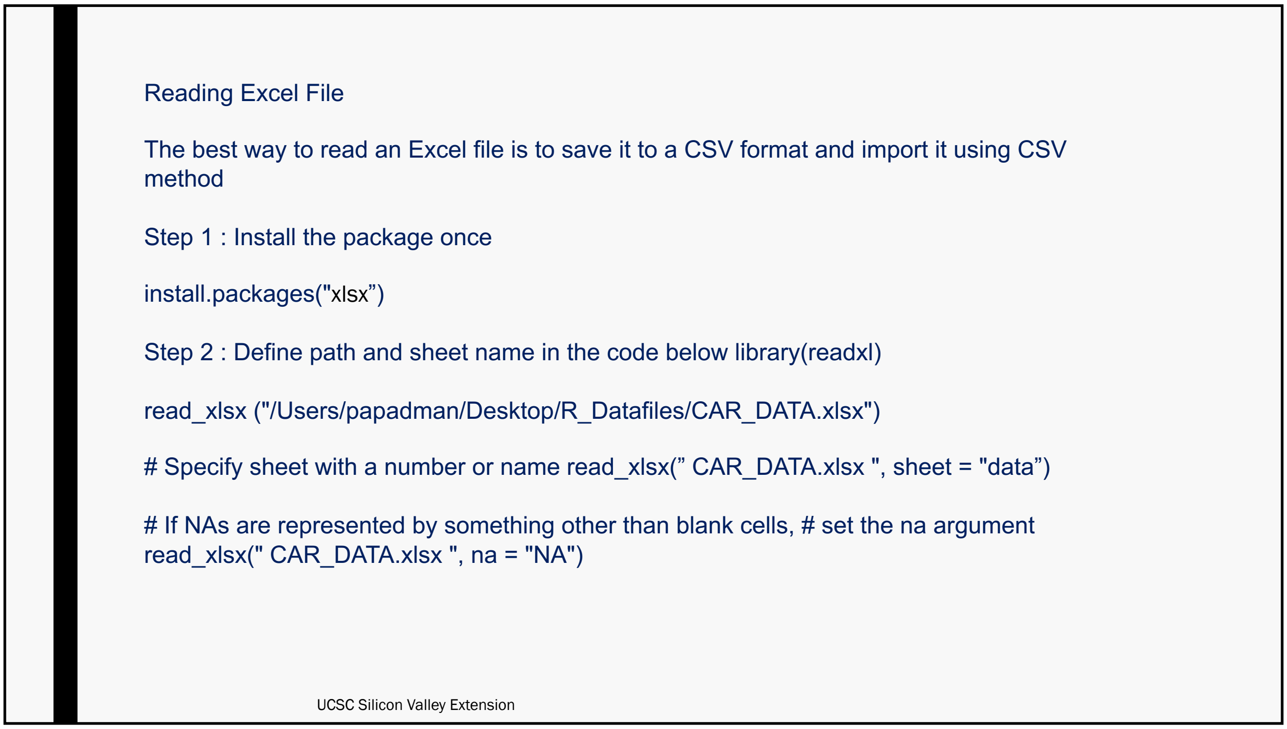
```
 df <- read.csv("/Users/papadman/Desktop/R_Datafiles/vehicleMiss.csv", header = TRUE,sep
= "," ,  stringsAsFactors= FALSE)
```

Alternatively you can pass col_names a character vector which will be used as the column names:

```
df <- read.csv ("/Users/papadman/Desktop/R_Datafiles/vehicleMiss.csv", col_names = c("a",
"b", "c"), stringsAsFactors= FALSE)
```

If you want to set any value to a missing value
df <- read.csv ("/Users/papadman/Desktop/R_Datafiles/vehicleMiss.csv", header=TRUE,
na.strings=".") In this case, we have set "." (without quotes) to a missing value

40



Reading Excel File

The best way to read an Excel file is to save it to a CSV format and import it using CSV method

Step 1 : Install the package once

```
install.packages("xlsx”)
```

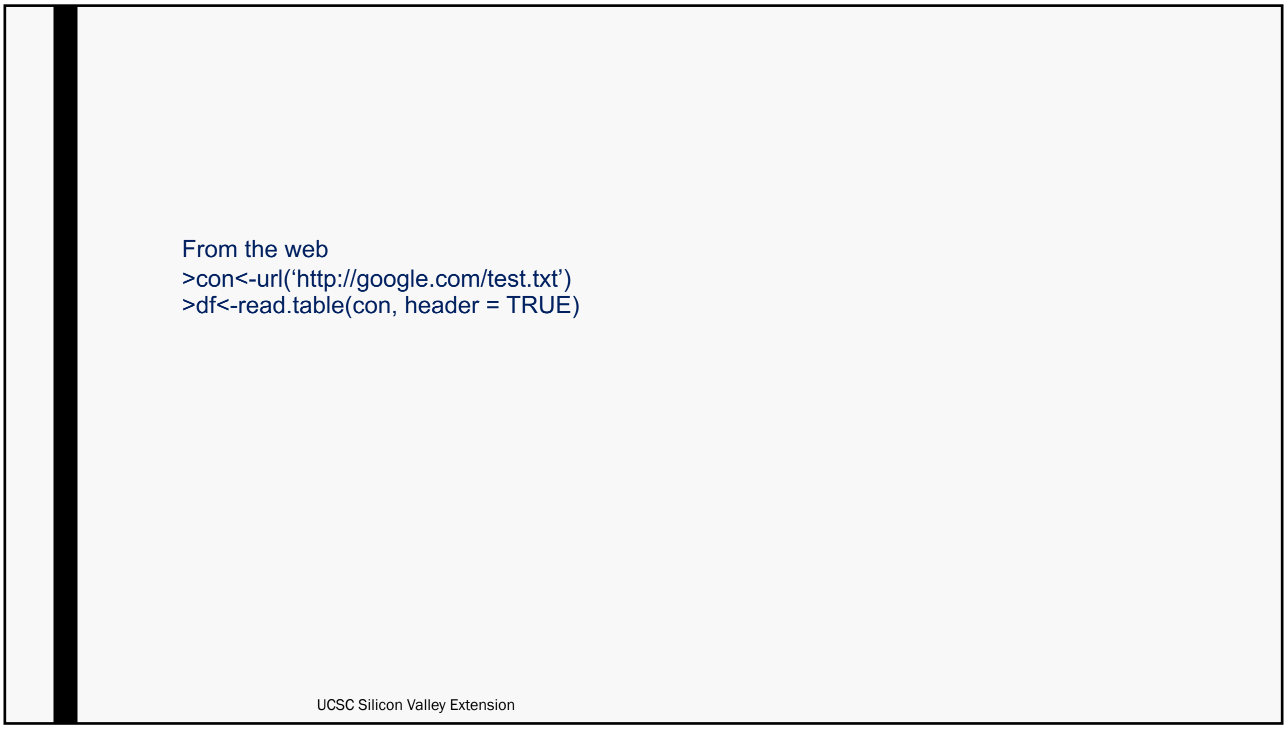
Step 2 : Define path and sheet name in the code below library(readxl)

```
read_xlsx ("/Users/papadman/Desktop/R_Datafiles/CAR_DATA.xlsx")
```

```
# Specify sheet with a number or name read_xlsx(” CAR_DATA.xlsx ", sheet = "data”)
```

```
# If NAs are represented by something other than blank cells, # set the na argument
read_xlsx(" CAR_DATA.xlsx ", na = "NA")
```

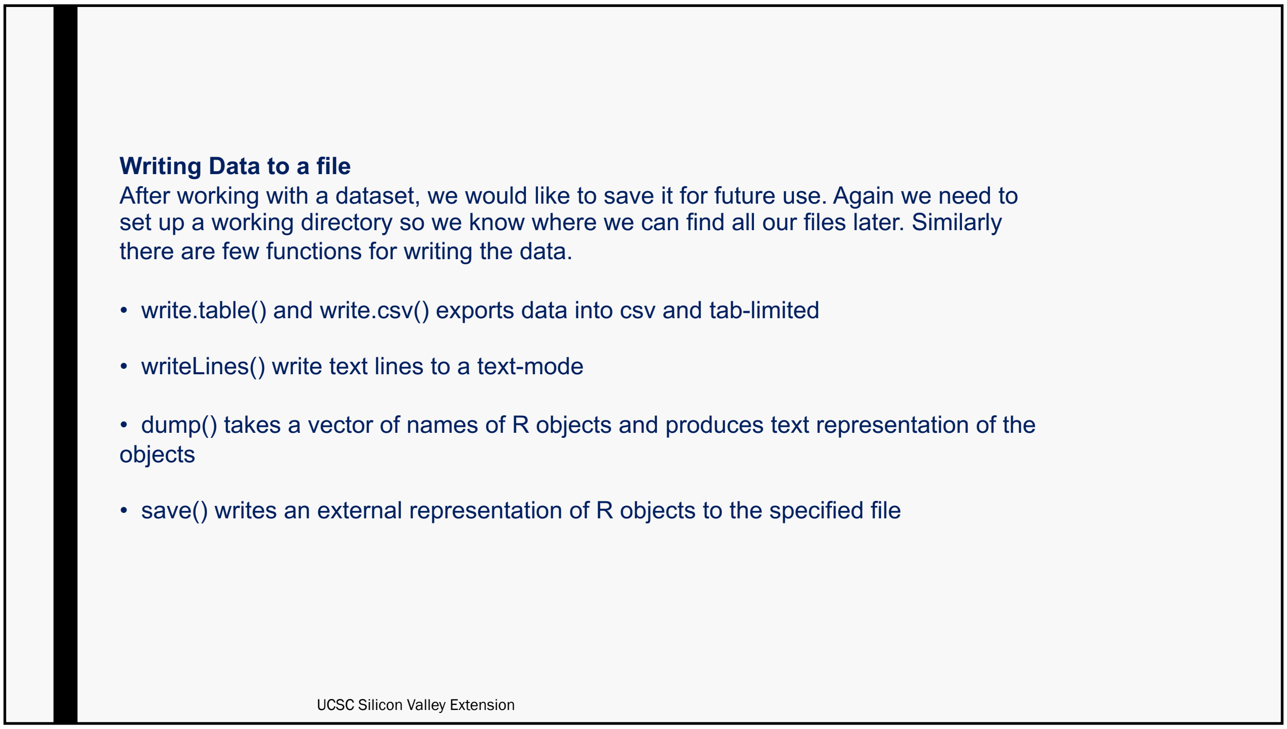
41



From the web

```
>con<-url(‘http://google.com/test.txt’)
>df<-read.table(con, header = TRUE)
```

42

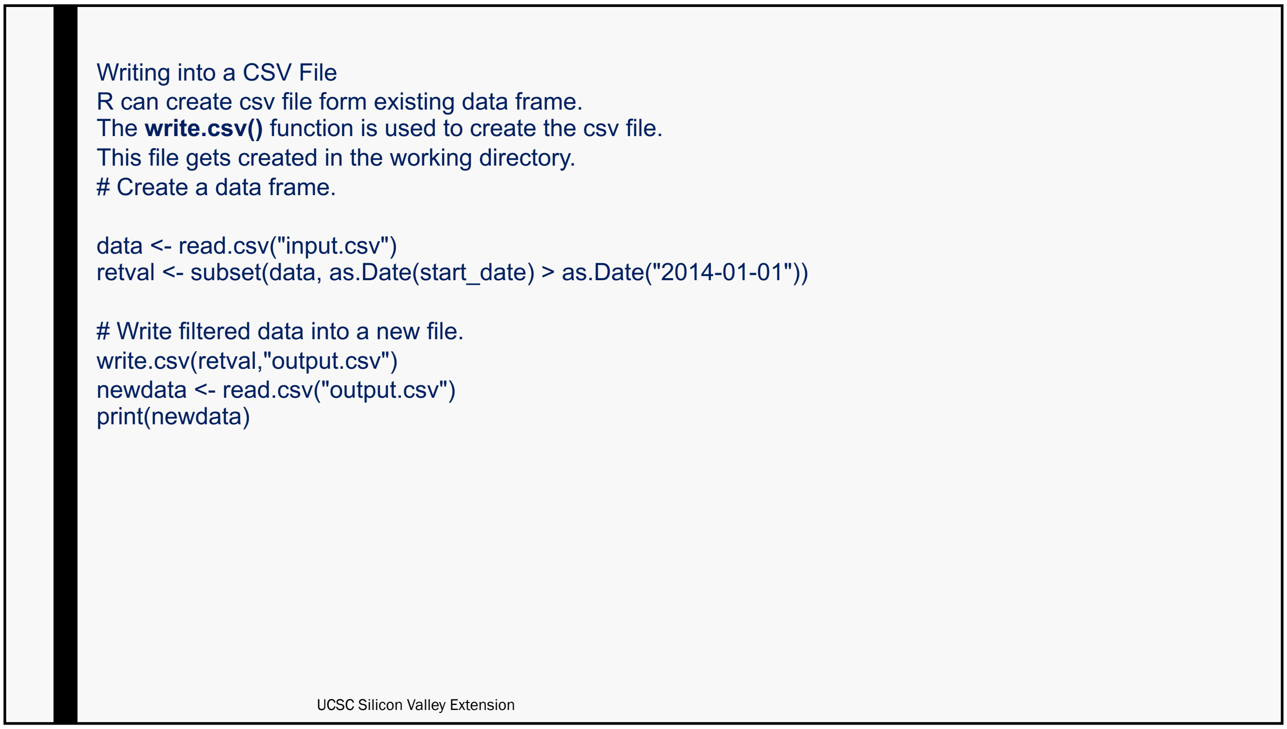


Writing Data to a file

After working with a dataset, we would like to save it for future use. Again we need to set up a working directory so we know where we can find all our files later. Similarly there are few functions for writing the data.

- write.table() and write.csv() exports data into csv and tab-limited
- writeLines() write text lines to a text-mode
- dump() takes a vector of names of R objects and produces text representation of the objects
- save() writes an external representation of R objects to the specified file

43



Writing into a CSV File

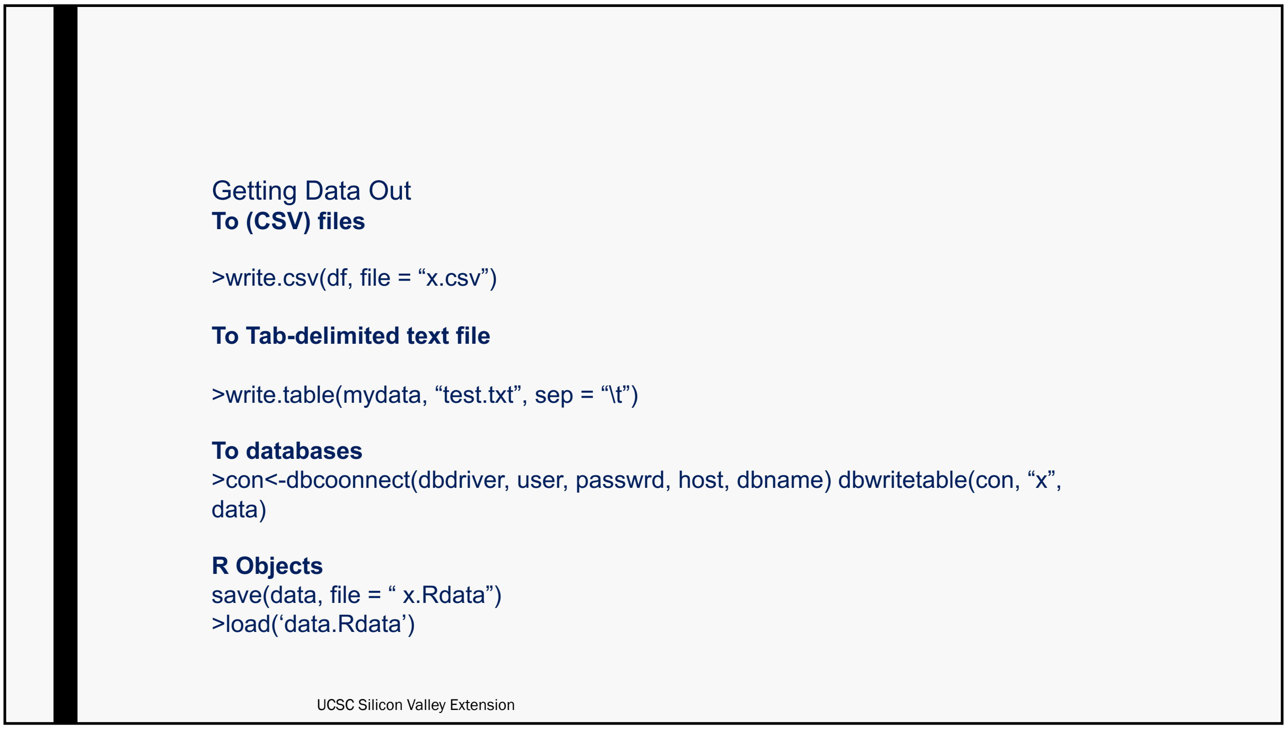
R can create csv file form existing data frame.

The **write.csv()** function is used to create the csv file.

This file gets created in the working directory.

```
# Create a data frame.

data <- read.csv("input.csv")
retval <- subset(data, as.Date(start_date) > as.Date("2014-01-01"))

# Write filtered data into a new file.
write.csv(retval,"output.csv")
newdata <- read.csv("output.csv")
print(newdata)
```

44



Getting Data Out

To (CSV) files

```
>write.csv(df, file = “x.csv”)
```

To Tab-delimited text file

```
>write.table(mydata, “test.txt”, sep = “\t”)
```

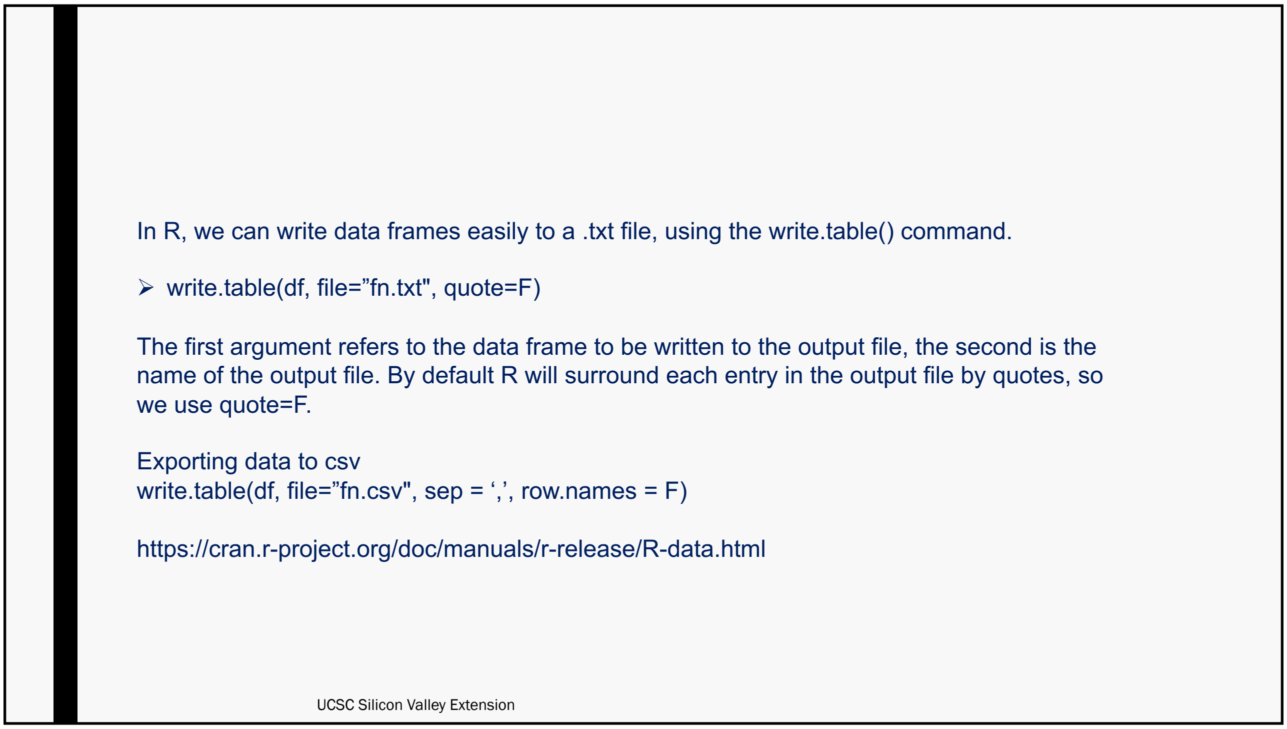
To databases

```
>con<-dbcoonnect(dbdriver, user, passwrd, host, dbname) dbwritetable(con, “x”, data)
```

R Objects

```
save(data, file = “ x.Rdata”)
>load(‘data.Rdata’)
```

45



In R, we can write data frames easily to a .txt file, using the write.table() command.

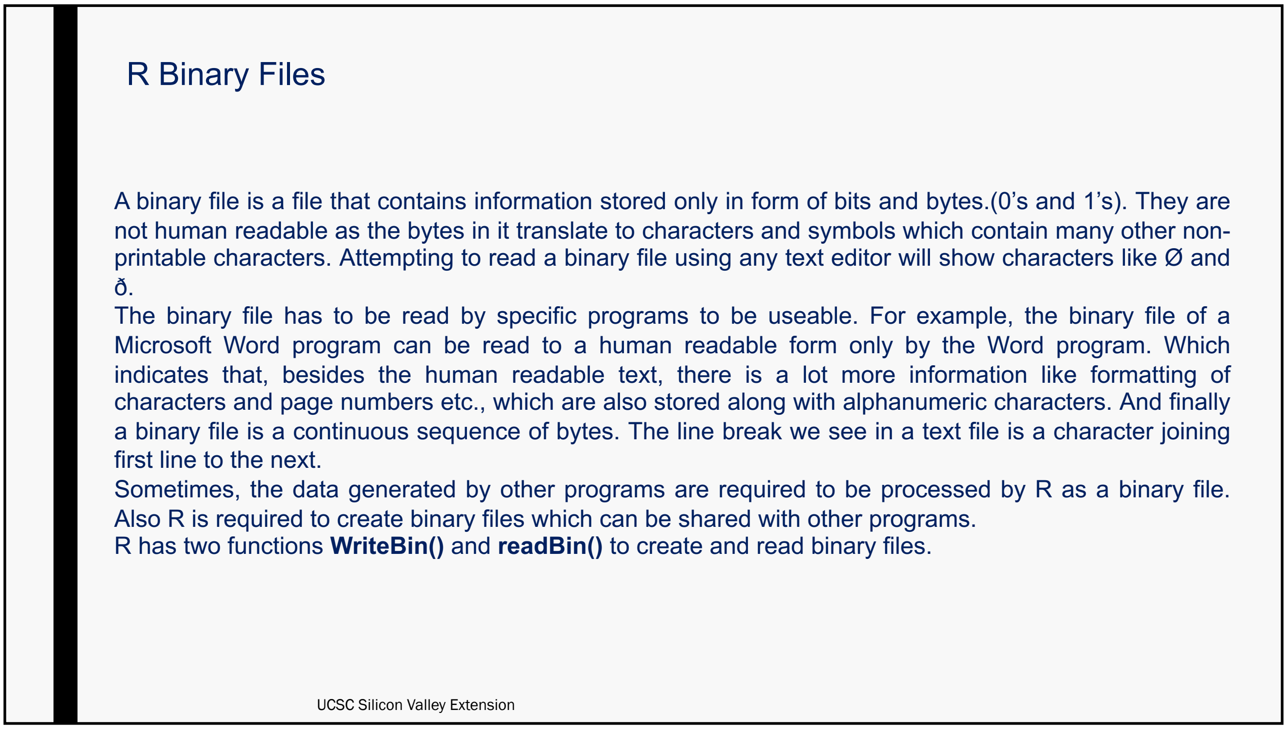
- ➢ write.table(df, file=”fn.txt", quote=F)

The first argument refers to the data frame to be written to the output file, the second is the name of the output file. By default R will surround each entry in the output file by quotes, so we use quote=F.

Exporting data to csv
write.table(df, file=”fn.csv", sep = ‘,’, row.names = F)

https://cran.r-project.org/doc/manuals/r-release/R-data.html

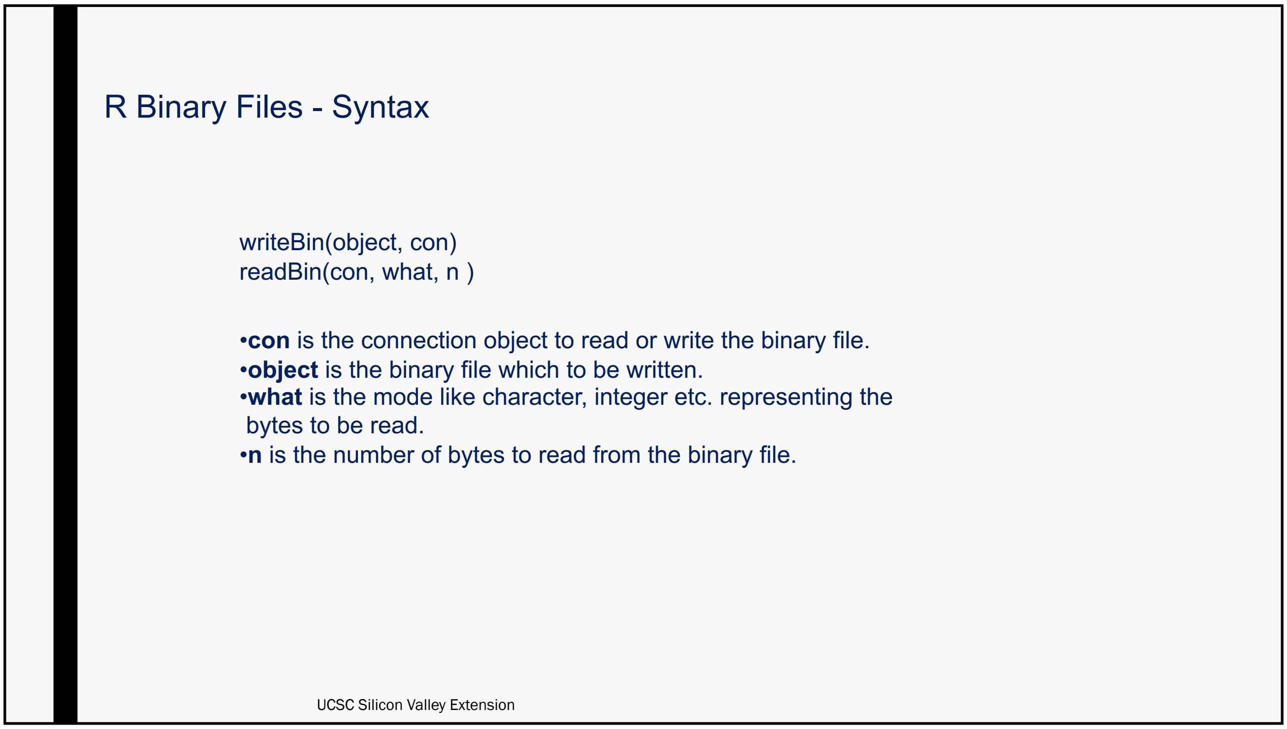
46



R Binary Files

A binary file is a file that contains information stored only in form of bits and bytes.(0’s and 1’s). They are not human readable as the bytes in it translate to characters and symbols which contain many other non-printable characters. Attempting to read a binary file using any text editor will show characters like Ø and ð.
The binary file has to be read by specific programs to be useable. For example, the binary file of a Microsoft Word program can be read to a human readable form only by the Word program. Which indicates that, besides the human readable text, there is a lot more information like formatting of characters and page numbers etc., which are also stored along with alphanumeric characters. And finally a binary file is a continuous sequence of bytes. The line break we see in a text file is a character joining first line to the next.
Sometimes, the data generated by other programs are required to be processed by R as a binary file. Also R is required to create binary files which can be shared with other programs.
R has two functions **WriteBin()** and **readBin()** to create and read binary files.

47

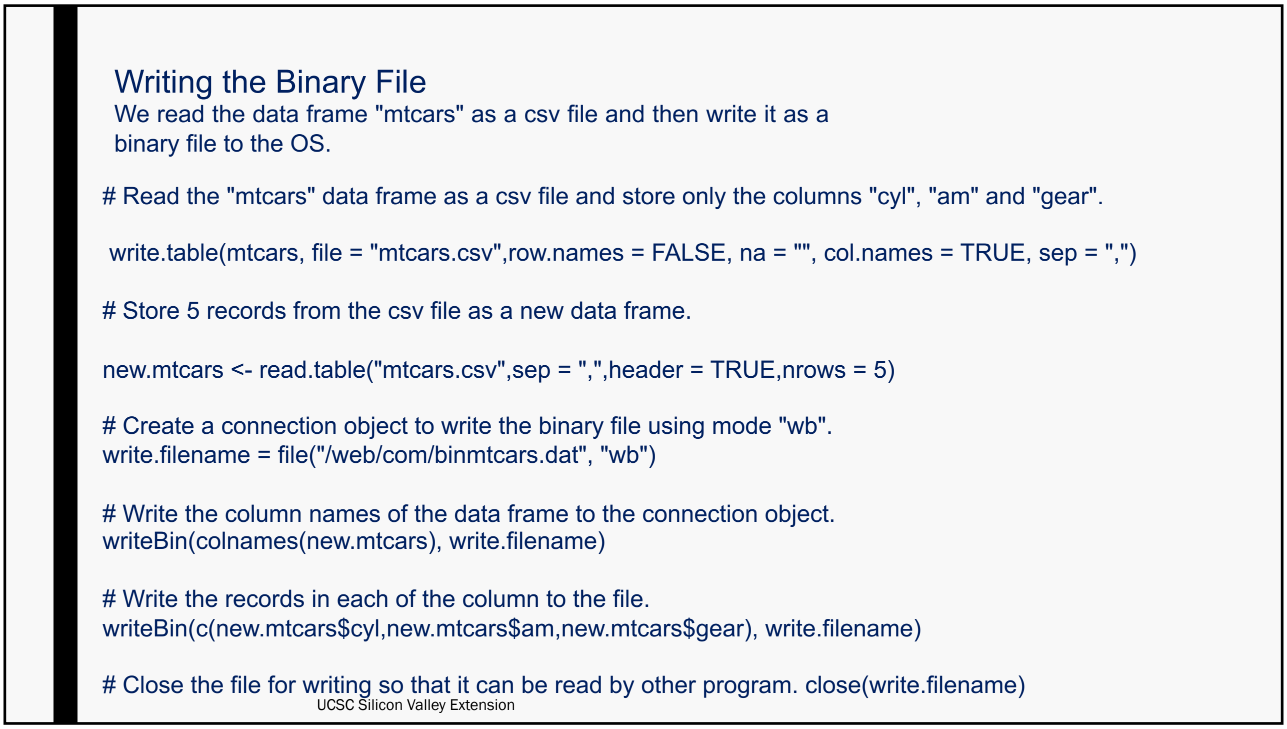


R Binary Files - Syntax

writeBin(object, con)
readBin(con, what, n)

•**con** is the connection object to read or write the binary file.
•**object** is the binary file which to be written.
•**what** is the mode like character, integer etc. representing the bytes to be read.
•**n** is the number of bytes to read from the binary file.

48

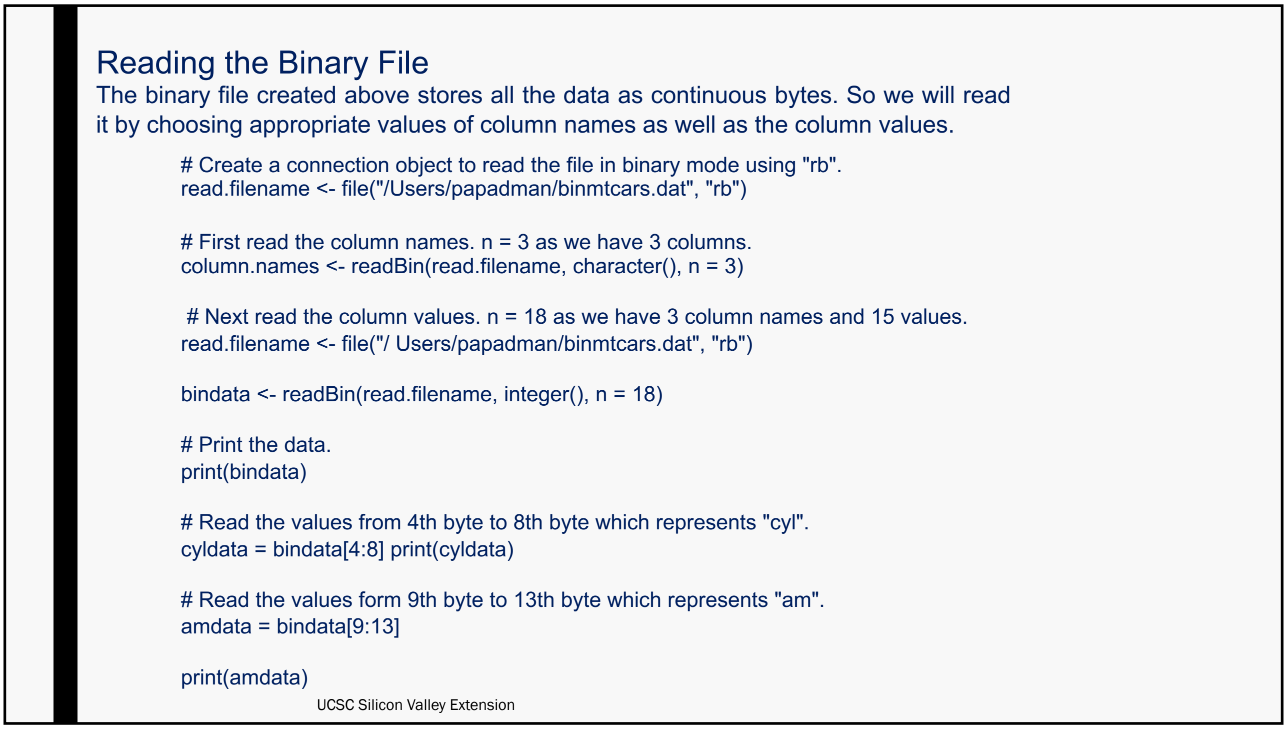


Writing the Binary File

We read the data frame "mtcars" as a csv file and then write it as a binary file to the OS.

```
# Read the "mtcars" data frame as a csv file and store only the columns "cyl", "am" and "gear".

 write.table(mtcars, file = "mtcars.csv",row.names = FALSE, na = "", col.names = TRUE, sep = ",")

# Store 5 records from the csv file as a new data frame.

new.mtcars <- read.table("mtcars.csv",sep = ",",header = TRUE,nrows = 5)

# Create a connection object to write the binary file using mode "wb".
write.filename = file("/web/com/binmtcars.dat", "wb")

# Write the column names of the data frame to the connection object.
writeBin(colnames(new.mtcars), write.filename)

# Write the records in each of the column to the file.
writeBin(c(new.mtcars$cyl,new.mtcars$am,new.mtcars$gear), write.filename)

# Close the file for writing so that it can be read by other program. close(write.filename)
```

49

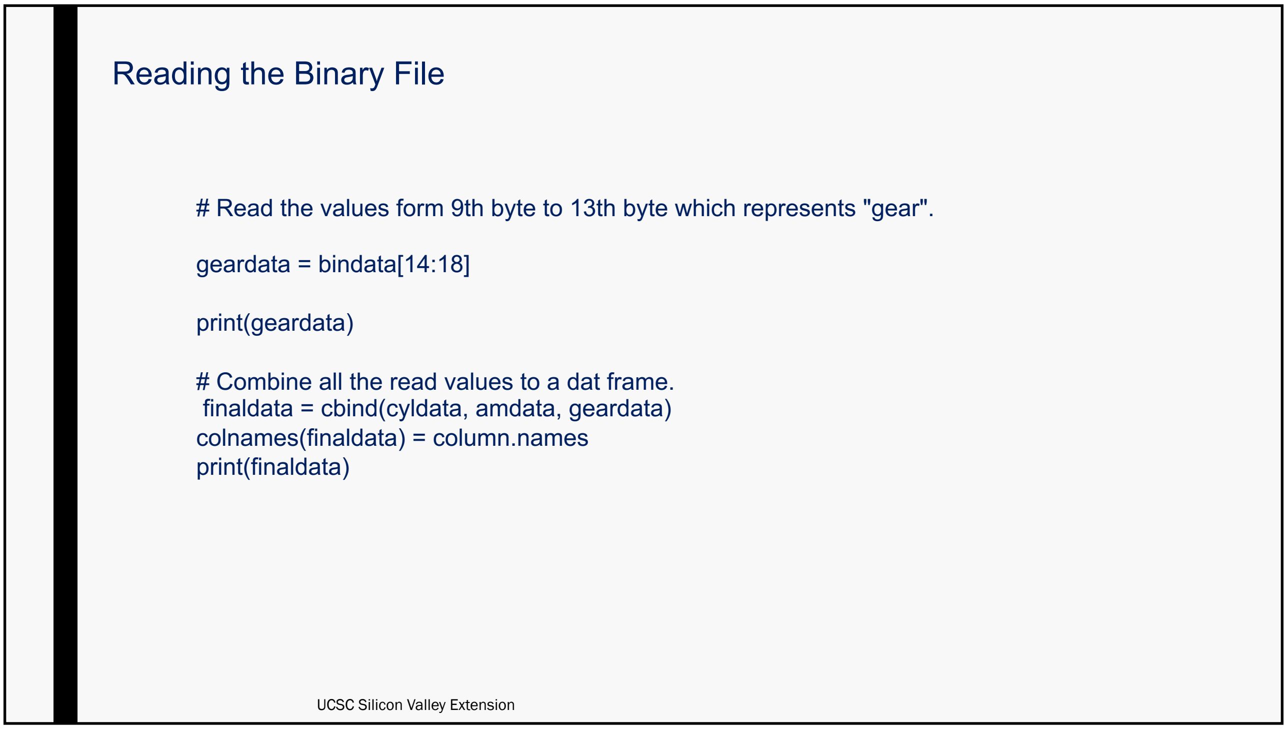


Reading the Binary File

The binary file created above stores all the data as continuous bytes. So we will read it by choosing appropriate values of column names as well as the column values.

```
# Create a connection object to read the file in binary mode using "rb".
read.filename <- file("/Users/papadman/binmtcars.dat", "rb")

# First read the column names. n = 3 as we have 3 columns.
column.names <- readBin(read.filename, character(), n = 3)

 # Next read the column values. n = 18 as we have 3 column names and 15 values.
read.filename <- file("/ Users/papadman/binmtcars.dat", "rb")

bindata <- readBin(read.filename, integer(), n = 18)

# Print the data.
print(bindata)

# Read the values from 4th byte to 8th byte which represents "cyl".
cyldata = bindata[4:8] print(cyldata)

# Read the values form 9th byte to 13th byte which represents "am".
amdata = bindata[9:13]

print(amdata)
```

50

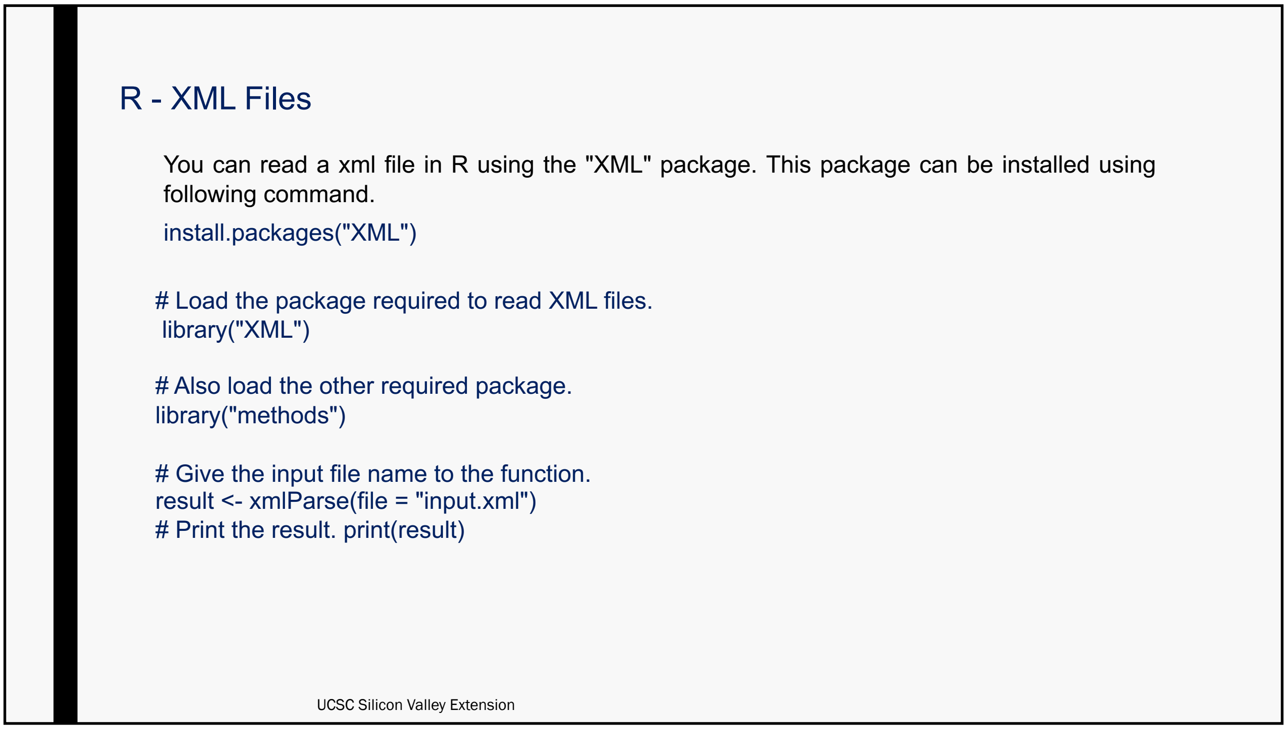


Reading the Binary File

```
# Read the values form 9th byte to 13th byte which represents "gear".

geardata = bindata[14:18]

print(geardata)

# Combine all the read values to a dat frame.
 finaldata = cbind(cyldata, amdata, geardata)
colnames(finaldata) = column.names
print(finaldata)
```

UCSC Silicon Valley Extension

51



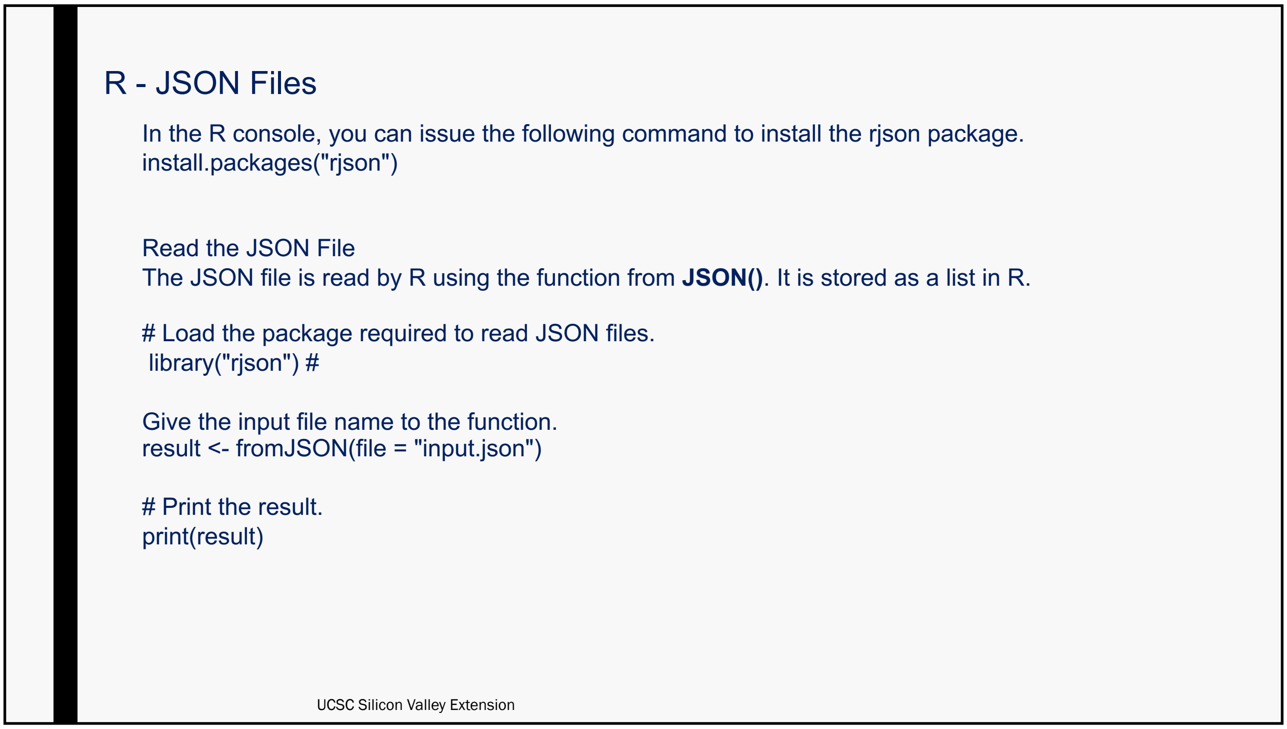
R - XML Files

You can read a xml file in R using the "XML" package. This package can be installed using following command.

```
install.packages("XML")

# Load the package required to read XML files.
 library("XML")

# Also load the other required package.
library("methods")

# Give the input file name to the function.
result <- xmlParse(file = "input.xml")
# Print the result. print(result)
```

UCSC Silicon Valley Extension

52



R - JSON Files

In the R console, you can issue the following command to install the rjson package.

```
install.packages("rjson")
```

Read the JSON File

The JSON file is read by R using the function from **JSON()**. It is stored as a list in R.

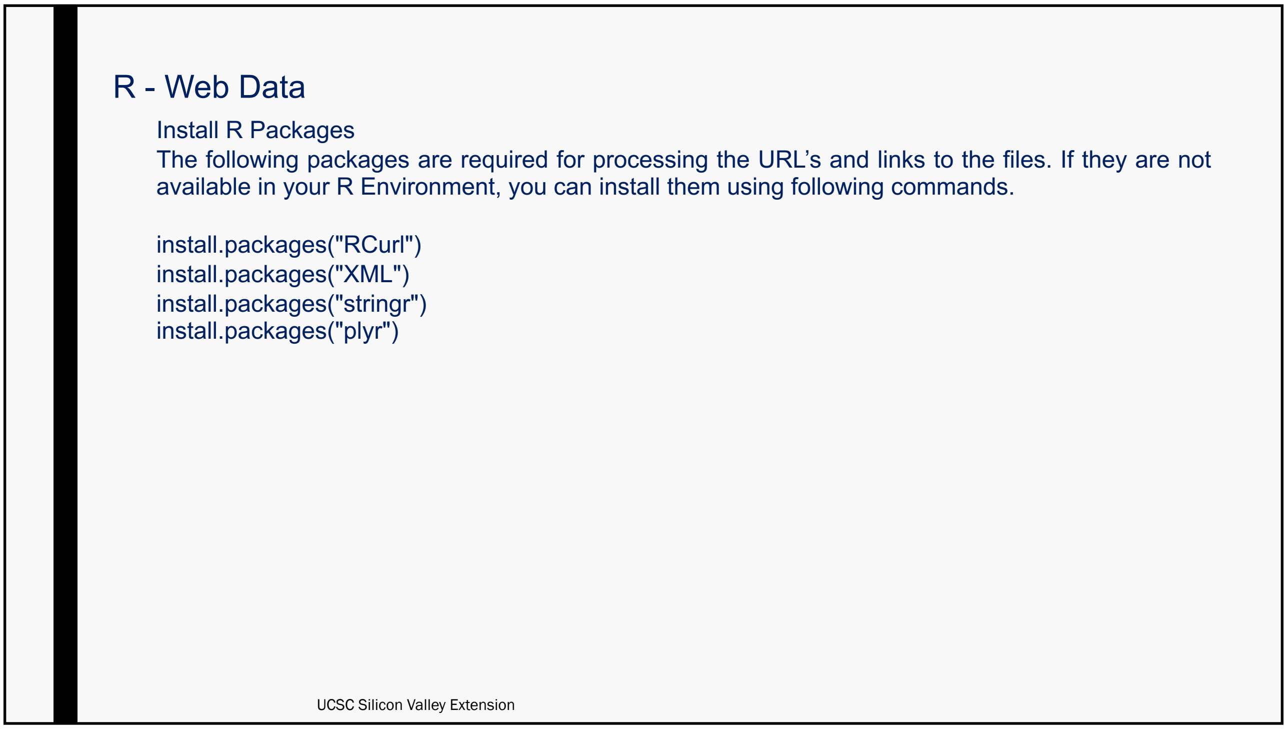
```
# Load the package required to read JSON files.
 library("rjson") #
```

Give the input file name to the function.

```
result <- fromJSON(file = "input.json")

# Print the result.
print(result)
```

UCSC Silicon Valley Extension

53



R - Web Data

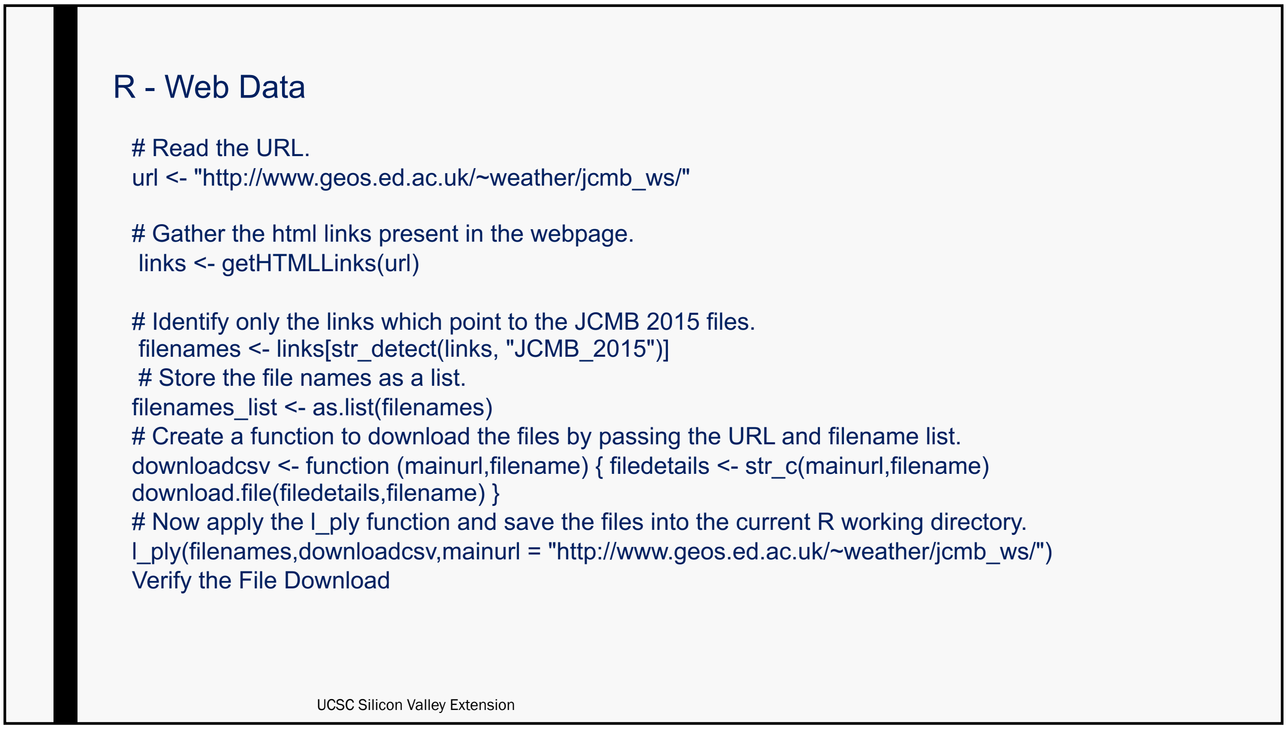
Install R Packages

The following packages are required for processing the URL’s and links to the files. If they are not available in your R Environment, you can install them using following commands.

```
install.packages("RCurl")
install.packages("XML")
install.packages("stringr")
install.packages("plyr")
```

UCSC Silicon Valley Extension

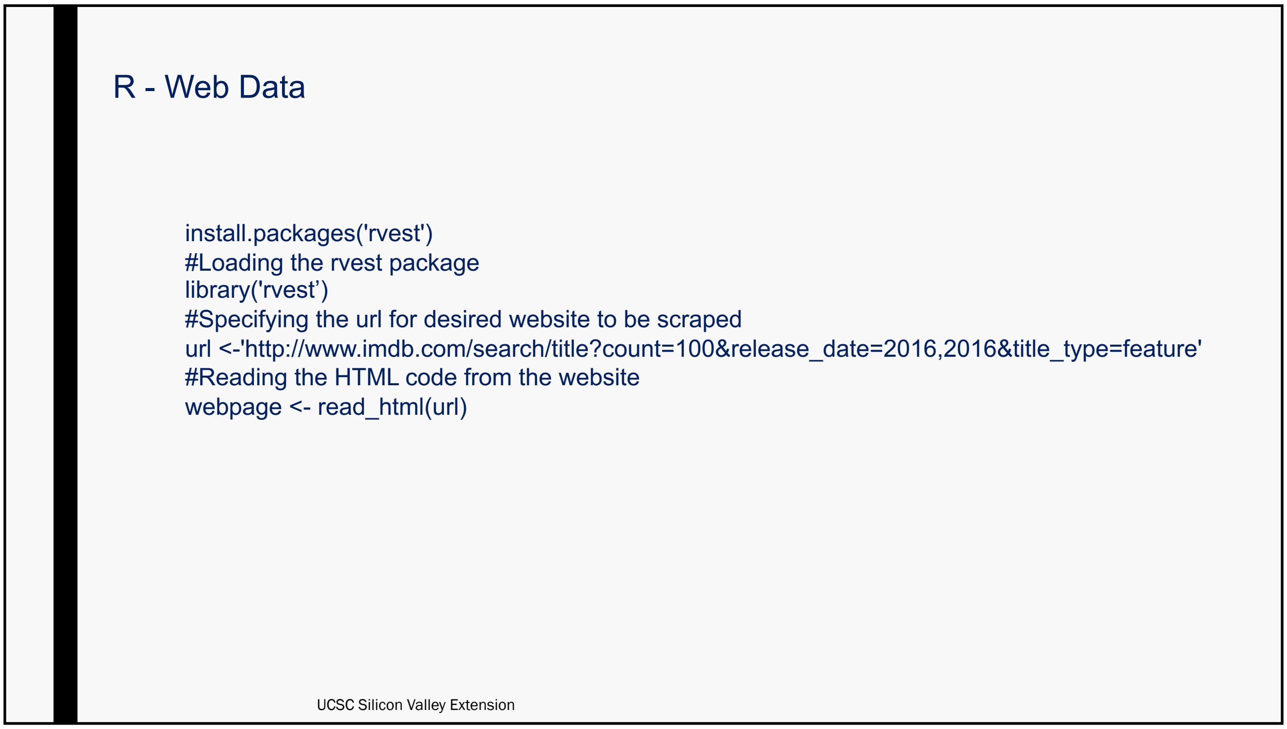
54



R - Web Data

```
# Read the URL.
url <- "http://www.geos.ed.ac.uk/~weather/jcmb_ws/"

# Gather the html links present in the webpage.
 links <- getHTMLLinks(url)

# Identify only the links which point to the JCMB 2015 files.
 filenames <- links[str_detect(links, "JCMB_2015")]
 # Store the file names as a list.
filenames_list <- as.list(filenames)
# Create a function to download the files by passing the URL and filename list.
downloadcsv <- function (mainurl,filename) { filedetails <- str_c(mainurl,filename)
download.file(filedetails,filename) }
# Now apply the l_ply function and save the files into the current R working directory.
l_ply(filenames,downloadcsv,mainurl = "http://www.geos.ed.ac.uk/~weather/jcmb_ws/")
Verify the File Download
```

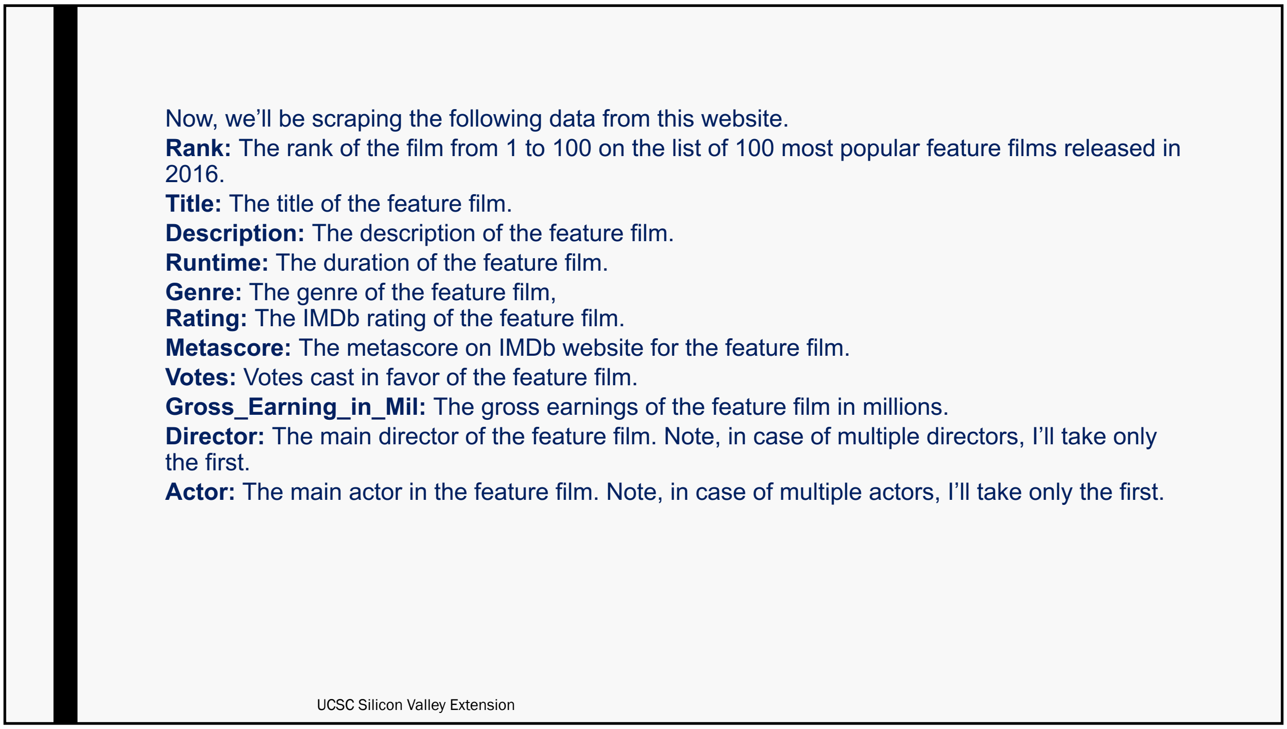
55



R - Web Data

```
install.packages('rvest')
#Loading the rvest package
library('rvest’)
#Specifying the url for desired website to be scraped
url <-'http://www.imdb.com/search/title?count=100&release_date=2016,2016&title_type=feature'
#Reading the HTML code from the website
webpage <- read_html(url)
```

56



Now, we’ll be scraping the following data from this website.

Rank: The rank of the film from 1 to 100 on the list of 100 most popular feature films released in 2016.

Title: The title of the feature film.

Description: The description of the feature film.

Runtime: The duration of the feature film.

Genre: The genre of the feature film,

Rating: The IMDb rating of the feature film.

Metascore: The metascore on IMDb website for the feature film.

Votes: Votes cast in favor of the feature film.

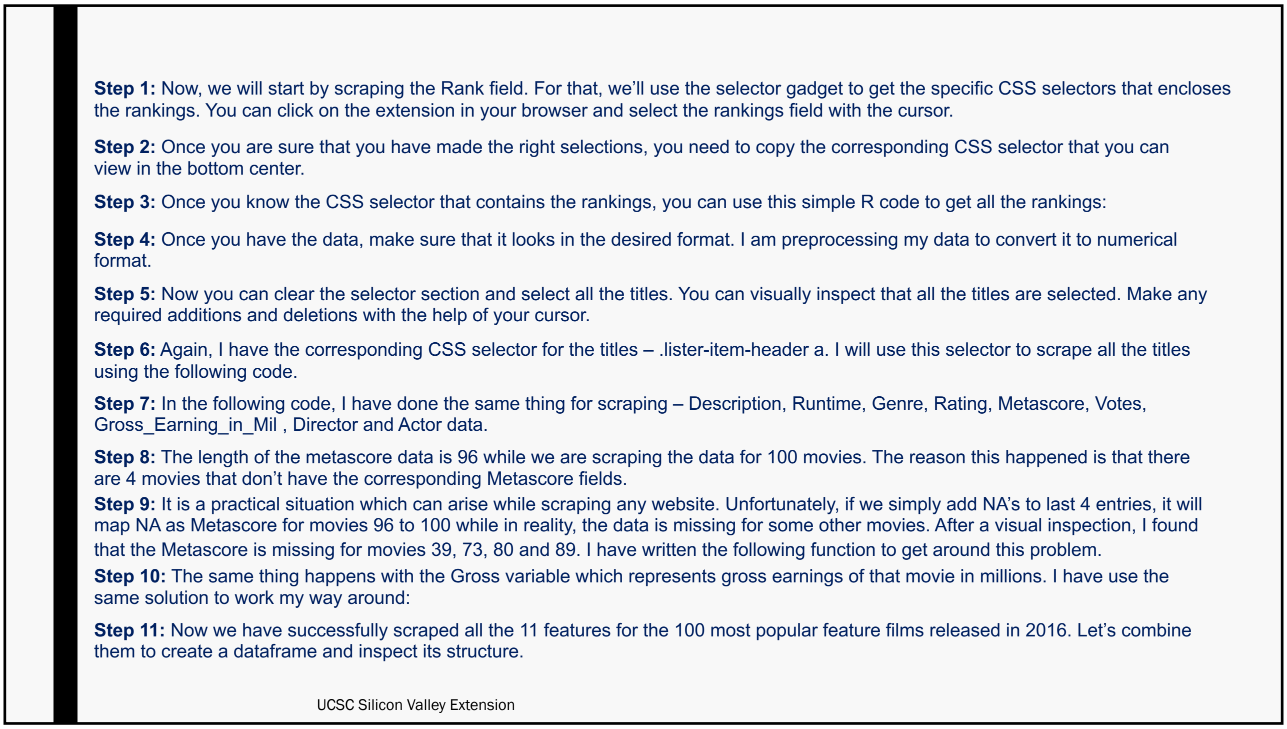
Gross_Earning_in_Mil: The gross earnings of the feature film in millions.

Director: The main director of the feature film. Note, in case of multiple directors, I’ll take only the first.

Actor: The main actor in the feature film. Note, in case of multiple actors, I’ll take only the first.

UCSC Silicon Valley Extension

57



Step 1: Now, we will start by scraping the Rank field. For that, we’ll use the selector gadget to get the specific CSS selectors that encloses the rankings. You can click on the extension in your browser and select the rankings field with the cursor.

Step 2: Once you are sure that you have made the right selections, you need to copy the corresponding CSS selector that you can view in the bottom center.

Step 3: Once you know the CSS selector that contains the rankings, you can use this simple R code to get all the rankings:

Step 4: Once you have the data, make sure that it looks in the desired format. I am preprocessing my data to convert it to numerical format.

Step 5: Now you can clear the selector section and select all the titles. You can visually inspect that all the titles are selected. Make any required additions and deletions with the help of your cursor.

Step 6: Again, I have the corresponding CSS selector for the titles – .lister-item-header a. I will use this selector to scrape all the titles using the following code.

Step 7: In the following code, I have done the same thing for scraping – Description, Runtime, Genre, Rating, Metascore, Votes, Gross_Earning_in_Mil , Director and Actor data.

Step 8: The length of the metascore data is 96 while we are scraping the data for 100 movies. The reason this happened is that there are 4 movies that don’t have the corresponding Metascore fields.

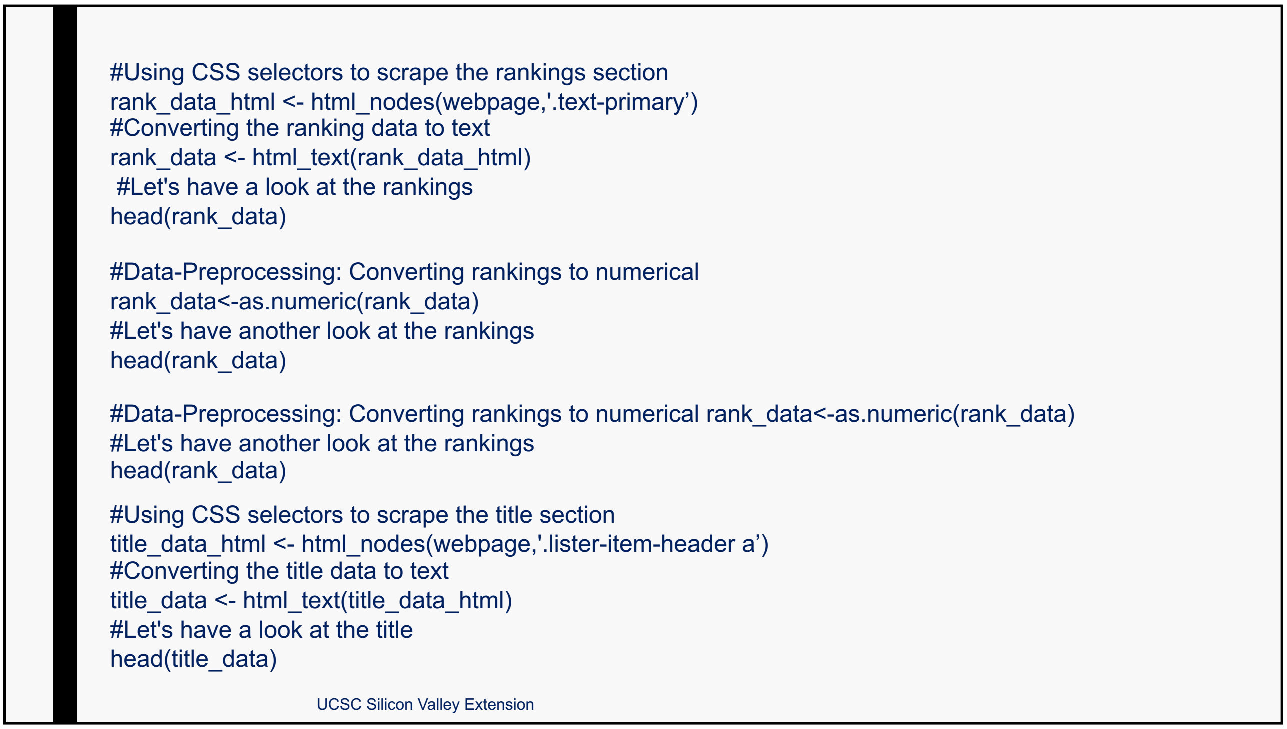
Step 9: It is a practical situation which can arise while scraping any website. Unfortunately, if we simply add NA’s to last 4 entries, it will map NA as Metascore for movies 96 to 100 while in reality, the data is missing for some other movies. After a visual inspection, I found that the Metascore is missing for movies 39, 73, 80 and 89. I have written the following function to get around this problem.

Step 10: The same thing happens with the Gross variable which represents gross earnings of that movie in millions. I have use the same solution to work my way around:

Step 11: Now we have successfully scraped all the 11 features for the 100 most popular feature films released in 2016. Let’s combine them to create a dataframe and inspect its structure.

UCSC Silicon Valley Extension

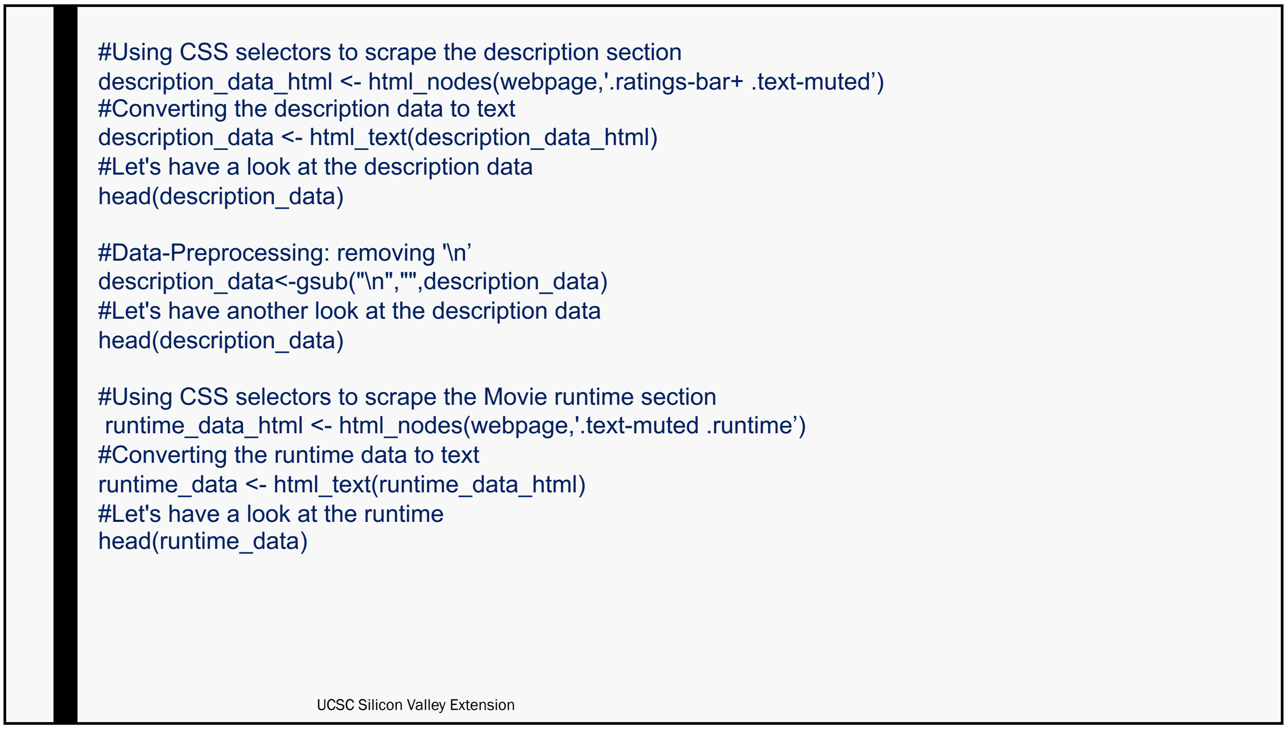
58



```
#Using CSS selectors to scrape the rankings section
rank_data_html <- html_nodes(webpage,'.text-primary’)
#Converting the ranking data to text
rank_data <- html_text(rank_data_html)
 #Let's have a look at the rankings
head(rank_data)

#Data-Preprocessing: Converting rankings to numerical
rank_data<-as.numeric(rank_data)
#Let's have another look at the rankings
head(rank_data)

#Data-Preprocessing: Converting rankings to numerical rank_data<-as.numeric(rank_data)
#Let's have another look at the rankings
head(rank_data)

#Using CSS selectors to scrape the title section
title_data_html <- html_nodes(webpage,'.lister-item-header a’)
#Converting the title data to text
title_data <- html_text(title_data_html)
#Let's have a look at the title
head(title_data)
```

UCSC Silicon Valley Extension

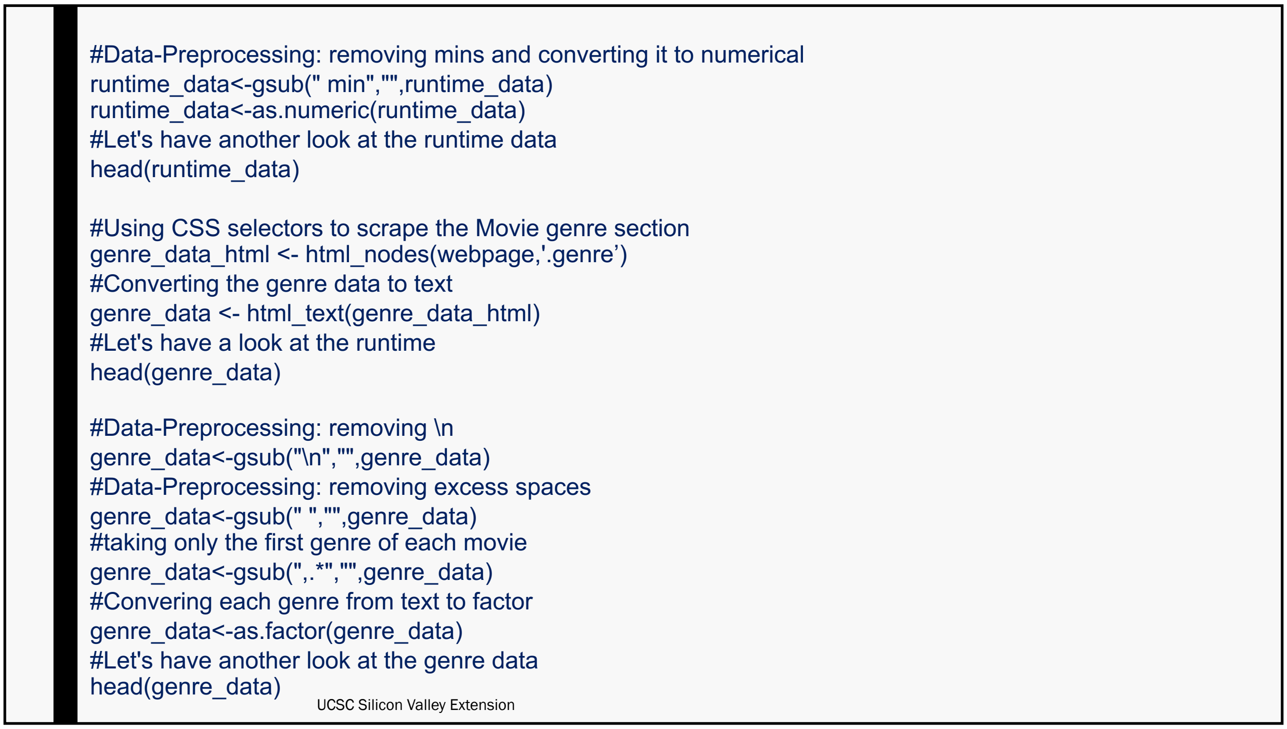
59



```
#Using CSS selectors to scrape the description section
description_data_html <- html_nodes(webpage,'.ratings-bar+ .text-muted’)
#Converting the description data to text
description_data <- html_text(description_data_html)
#Let's have a look at the description data
head(description_data)

#Data-Preprocessing: removing '\n’
description_data<-gsub("\n","",description_data)
#Let's have another look at the description data
head(description_data)

#Using CSS selectors to scrape the Movie runtime section
 runtime_data_html <- html_nodes(webpage,'.text-muted .runtime’)
#Converting the runtime data to text
runtime_data <- html_text(runtime_data_html)
#Let's have a look at the runtime
head(runtime_data)
```

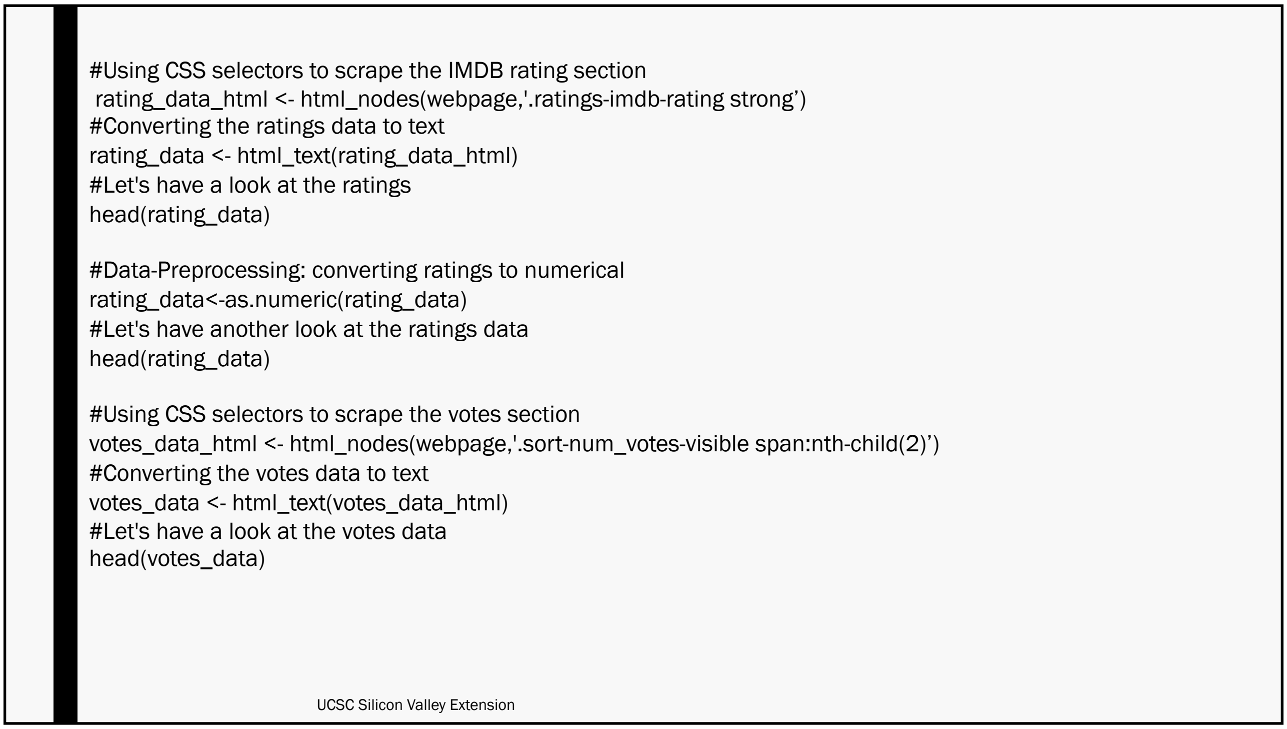
UCSC Silicon Valley Extension

60



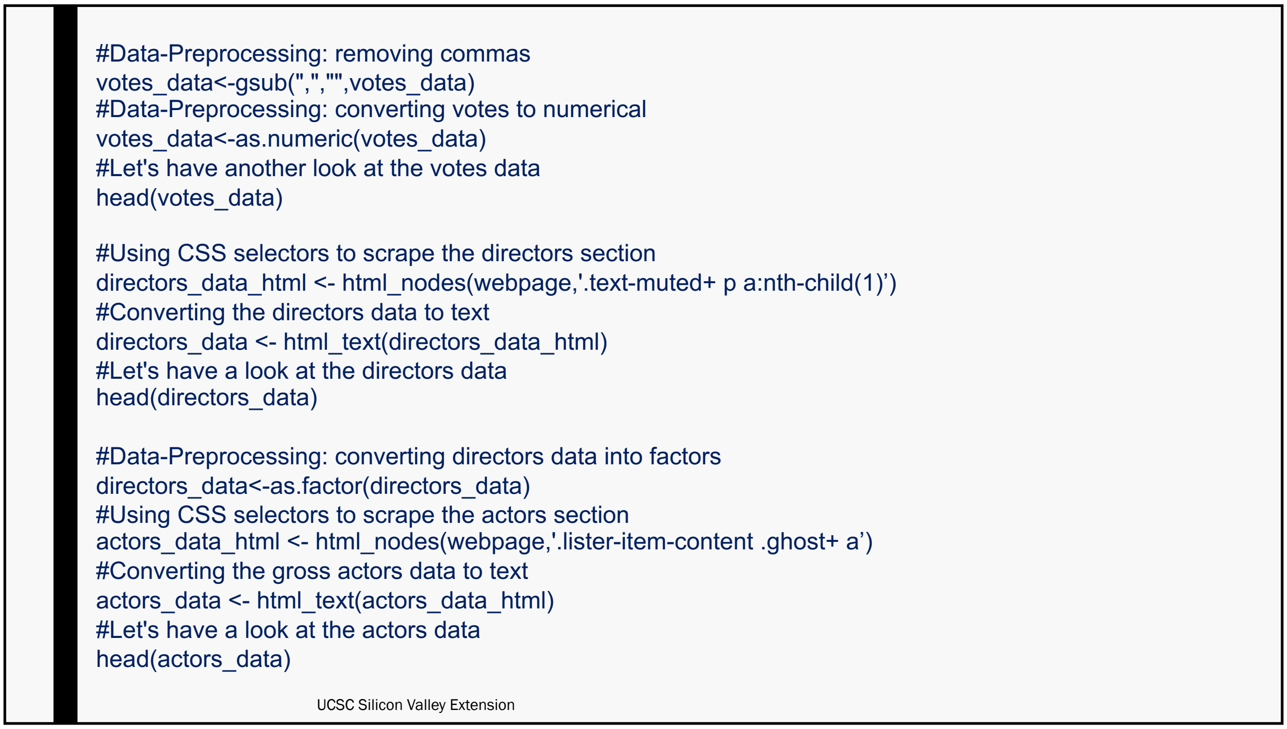
```
#Data-Preprocessing: removing mins and converting it to numerical
runtime_data<-gsub(" min","",runtime_data)
runtime_data<-as.numeric(runtime_data)
#Let's have another look at the runtime data
head(runtime_data)

#Using CSS selectors to scrape the Movie genre section
genre_data_html <- html_nodes(webpage,'.genre’)
#Converting the genre data to text
genre_data <- html_text(genre_data_html)
#Let's have a look at the runtime
head(genre_data)

#Data-Preprocessing: removing \n
genre_data<-gsub("\n","",genre_data)
#Data-Preprocessing: removing excess spaces
genre_data<-gsub(" ","",genre_data)
#taking only the first genre of each movie
genre_data<-gsub(",.*","",genre_data)
#Convering each genre from text to factor
genre_data<-as.factor(genre_data)
#Let's have another look at the genre data
head(genre_data)
```

61



```
#Using CSS selectors to scrape the IMDB rating section
 rating_data_html <- html_nodes(webpage,'.ratings-imdb-rating strong’)
#Converting the ratings data to text
rating_data <- html_text(rating_data_html)
#Let's have a look at the ratings
head(rating_data)

#Data-Preprocessing: converting ratings to numerical
rating_data<-as.numeric(rating_data)
#Let's have another look at the ratings data
head(rating_data)

#Using CSS selectors to scrape the votes section
votes_data_html <- html_nodes(webpage,'.sort-num_votes-visible span:nth-child(2)’)
#Converting the votes data to text
votes_data <- html_text(votes_data_html)
#Let's have a look at the votes data
head(votes_data)
```

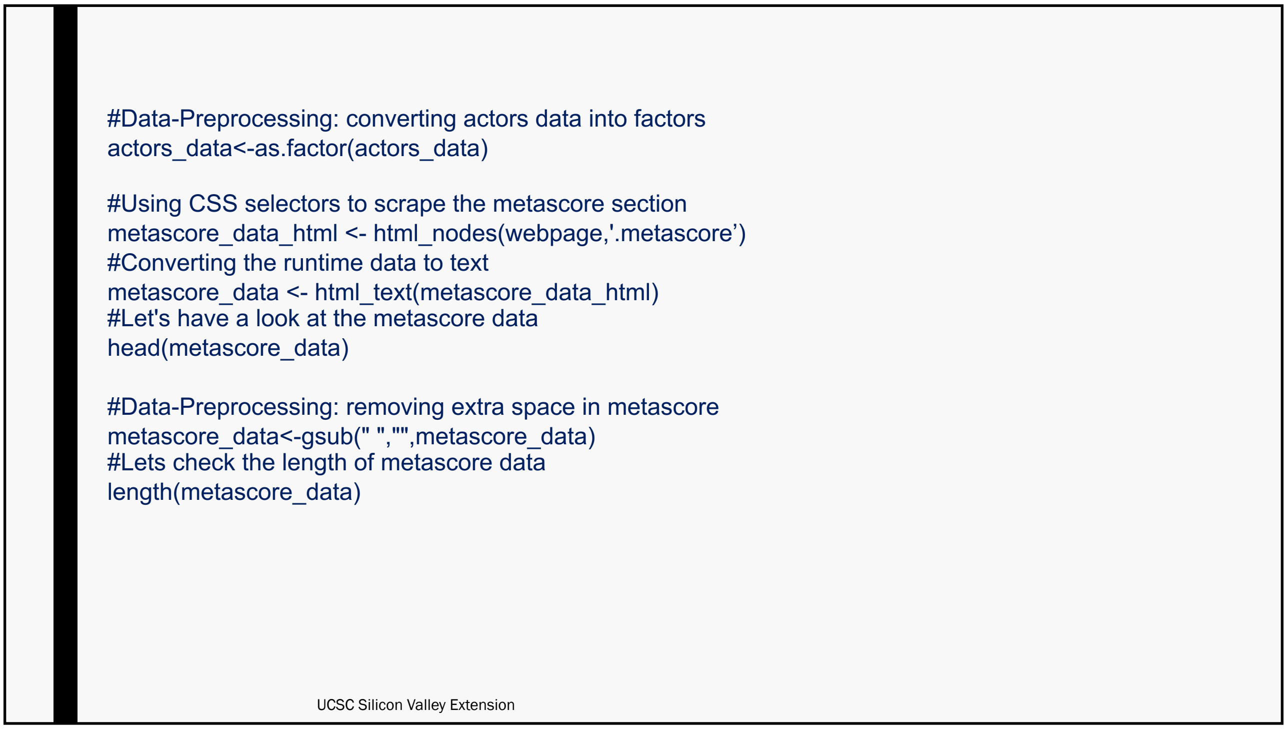
62



```
#Data-Preprocessing: removing commas
votes_data<-gsub(",","",votes_data)
#Data-Preprocessing: converting votes to numerical
votes_data<-as.numeric(votes_data)
#Let's have another look at the votes data
head(votes_data)

#Using CSS selectors to scrape the directors section
directors_data_html <- html_nodes(webpage,'.text-muted+ p a:nth-child(1)’)
#Converting the directors data to text
directors_data <- html_text(directors_data_html)
#Let's have a look at the directors data
head(directors_data)

#Data-Preprocessing: converting directors data into factors
directors_data<-as.factor(directors_data)
#Using CSS selectors to scrape the actors section
actors_data_html <- html_nodes(webpage,'.lister-item-content .ghost+ a’)
#Converting the gross actors data to text
actors_data <- html_text(actors_data_html)
#Let's have a look at the actors data
head(actors_data)
```

UCSC Silicon Valley Extension

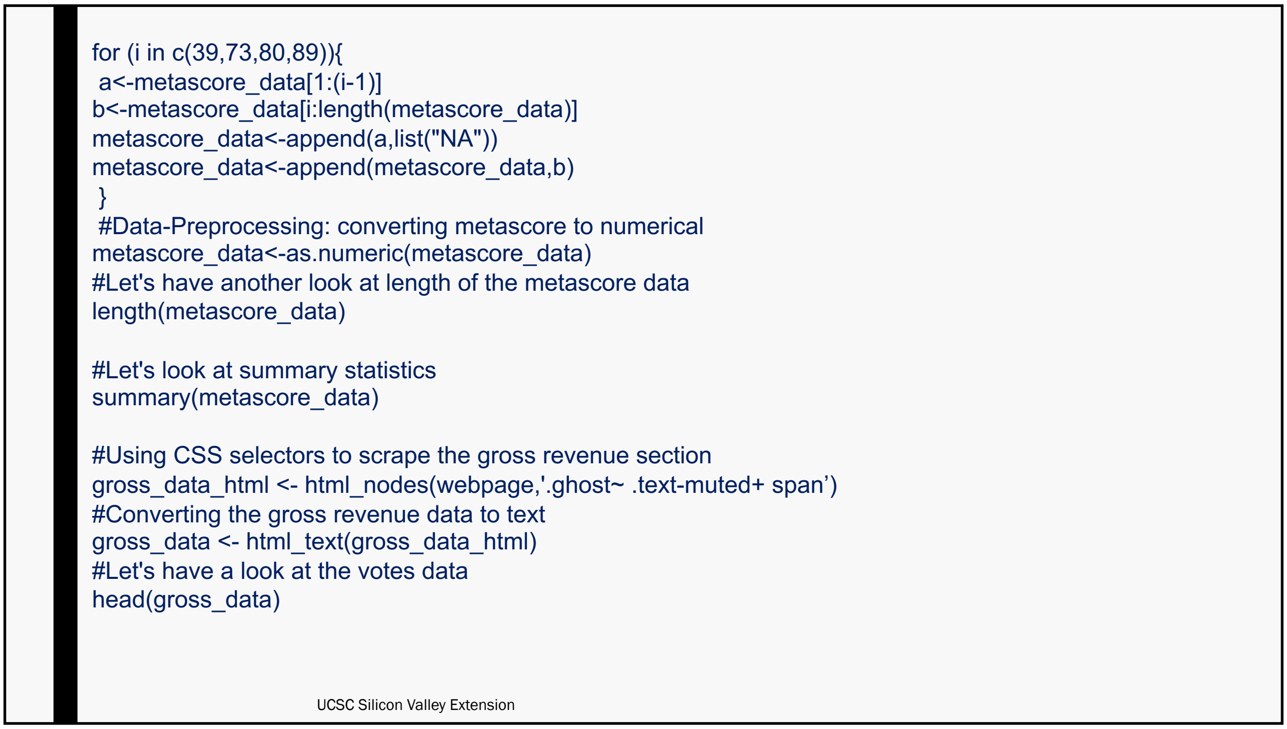
63



```
#Data-Preprocessing: converting actors data into factors
actors_data<-as.factor(actors_data)

#Using CSS selectors to scrape the metascore section
metascore_data_html <- html_nodes(webpage,'.metascore’)
#Converting the runtime data to text
metascore_data <- html_text(metascore_data_html)
#Let's have a look at the metascore data
head(metascore_data)

#Data-Preprocessing: removing extra space in metascore
metascore_data<-gsub(" ","",metascore_data)
#Lets check the length of metascore data
length(metascore_data)
```

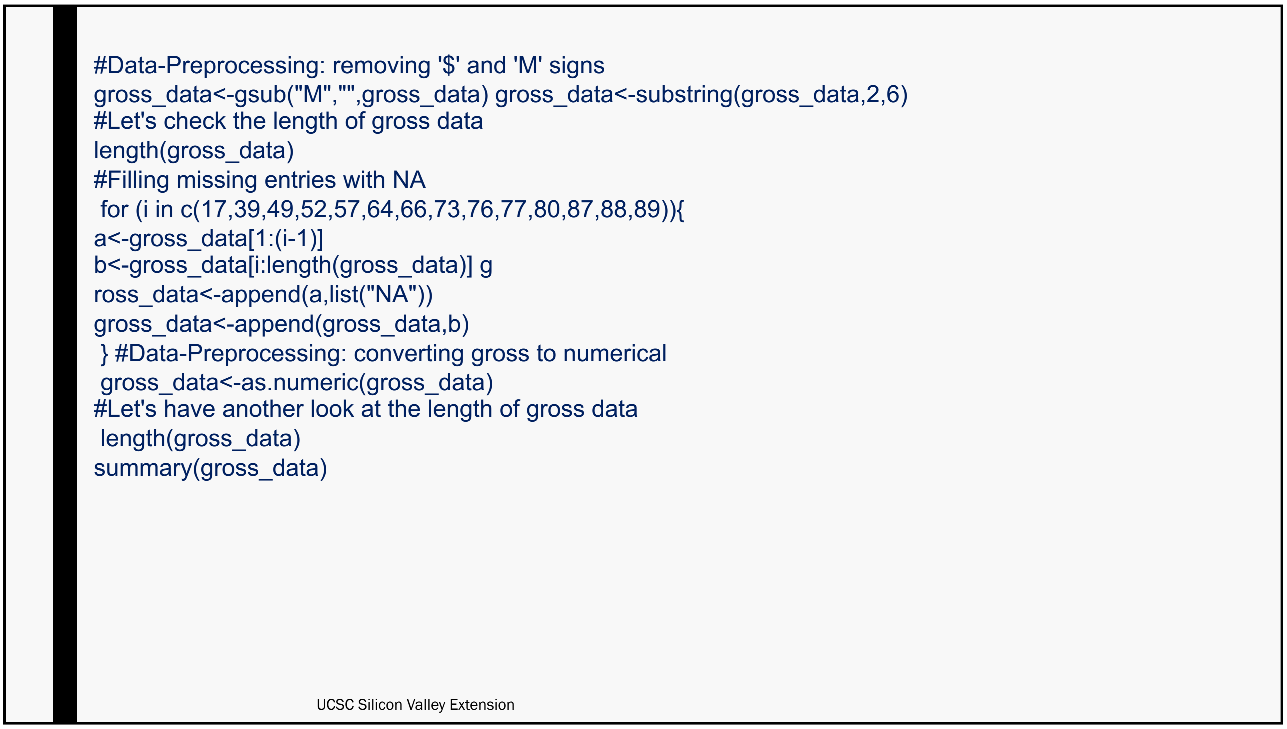
UCSC Silicon Valley Extension

64



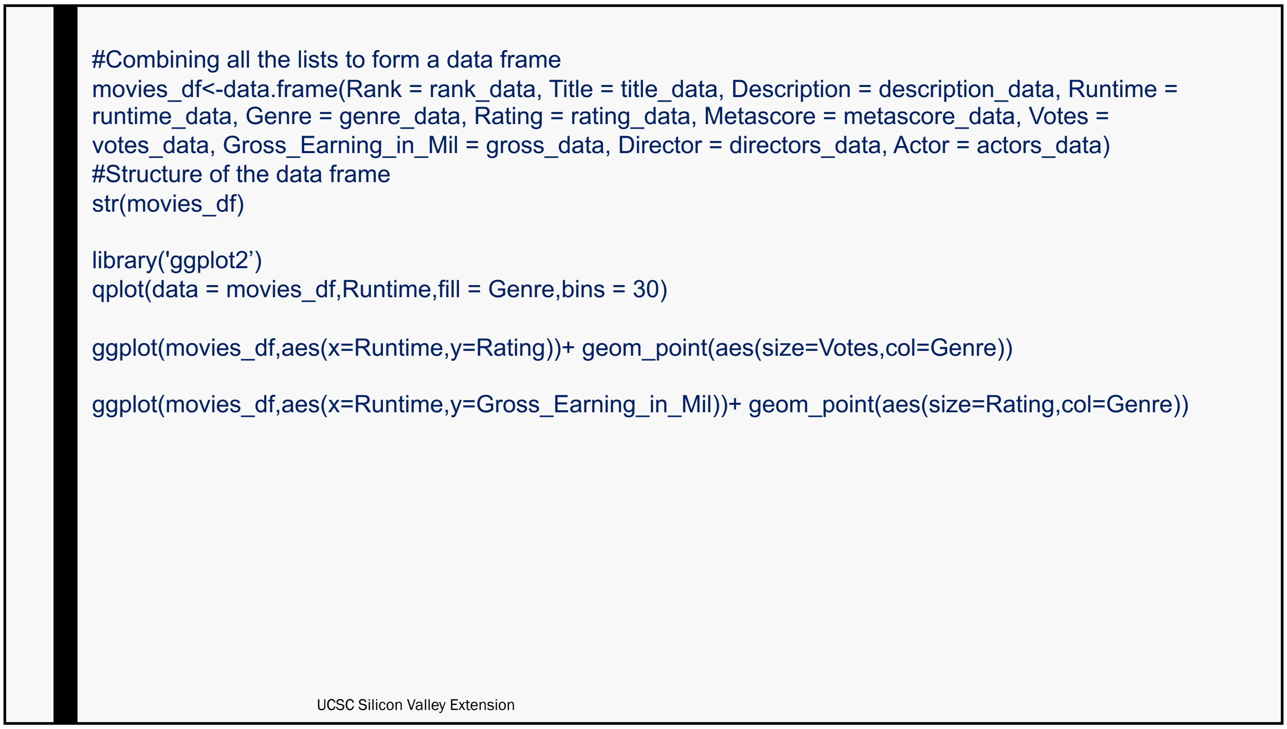
```
for (i in c(39,73,80,89)){
 a<-metascore_data[1:(i-1)]
b<-metascore_data[i:length(metascore_data)]
metascore_data<-append(a,list("NA"))
metascore_data<-append(metascore_data,b)
 }
 #Data-Preprocessing: converting metascore to numerical
metascore_data<-as.numeric(metascore_data)
#Let's have another look at length of the metascore data
length(metascore_data)

#Let's look at summary statistics
summary(metascore_data)

#Using CSS selectors to scrape the gross revenue section
gross_data_html <- html_nodes(webpage,'.ghost~ .text-muted+ span’)
#Converting the gross revenue data to text
gross_data <- html_text(gross_data_html)
#Let's have a look at the votes data
head(gross_data)
```

65



```
#Data-Preprocessing: removing '$' and 'M' signs
gross_data<-gsub("M","",gross_data) gross_data<-substring(gross_data,2,6)
#Let's check the length of gross data
length(gross_data)
#Filling missing entries with NA
 for (i in c(17,39,49,52,57,64,66,73,76,77,80,87,88,89)){
a<-gross_data[1:(i-1)]
b<-gross_data[i:length(gross_data)] g
ross_data<-append(a,list("NA"))
gross_data<-append(gross_data,b)
 } #Data-Preprocessing: converting gross to numerical
 gross_data<-as.numeric(gross_data)
#Let's have another look at the length of gross data
 length(gross_data)
summary(gross_data)
```

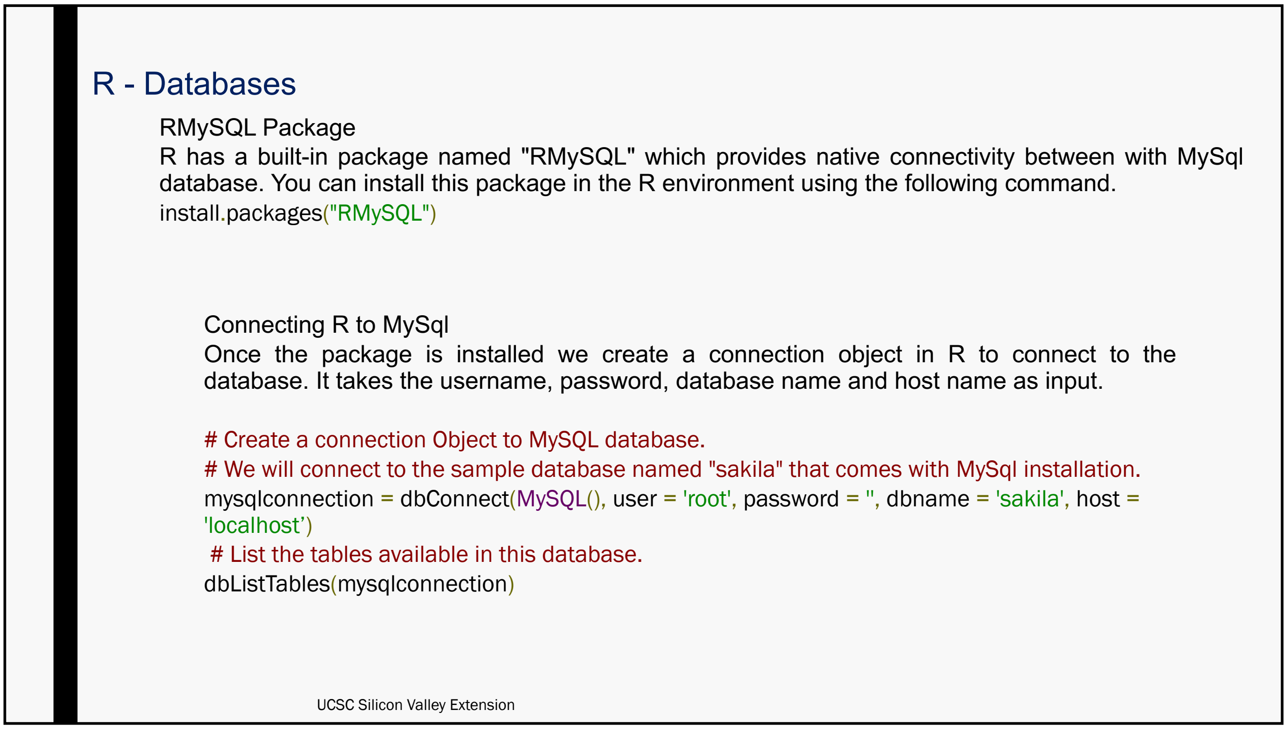
66



```
#Combining all the lists to form a data frame
movies_df<-data.frame(Rank = rank_data, Title = title_data, Description = description_data, Runtime =
runtime_data, Genre = genre_data, Rating = rating_data, Metascore = metascore_data, Votes =
votes_data, Gross_Earning_in_Mil = gross_data, Director = directors_data, Actor = actors_data)
#Structure of the data frame
str(movies_df)

library('ggplot2’)
qplot(data = movies_df,Runtime,fill = Genre,bins = 30)

ggplot(movies_df,aes(x=Runtime,y=Rating))+ geom_point(aes(size=Votes,col=Genre))

ggplot(movies_df,aes(x=Runtime,y=Gross_Earning_in_Mil))+ geom_point(aes(size=Rating,col=Genre))
```

UCSC Silicon Valley Extension

67



R - Databases

RMySQL Package

R has a built-in package named "RMySQL" which provides native connectivity between with MySql database. You can install this package in the R environment using the following command.

```
install.packages("RMySQL")
```

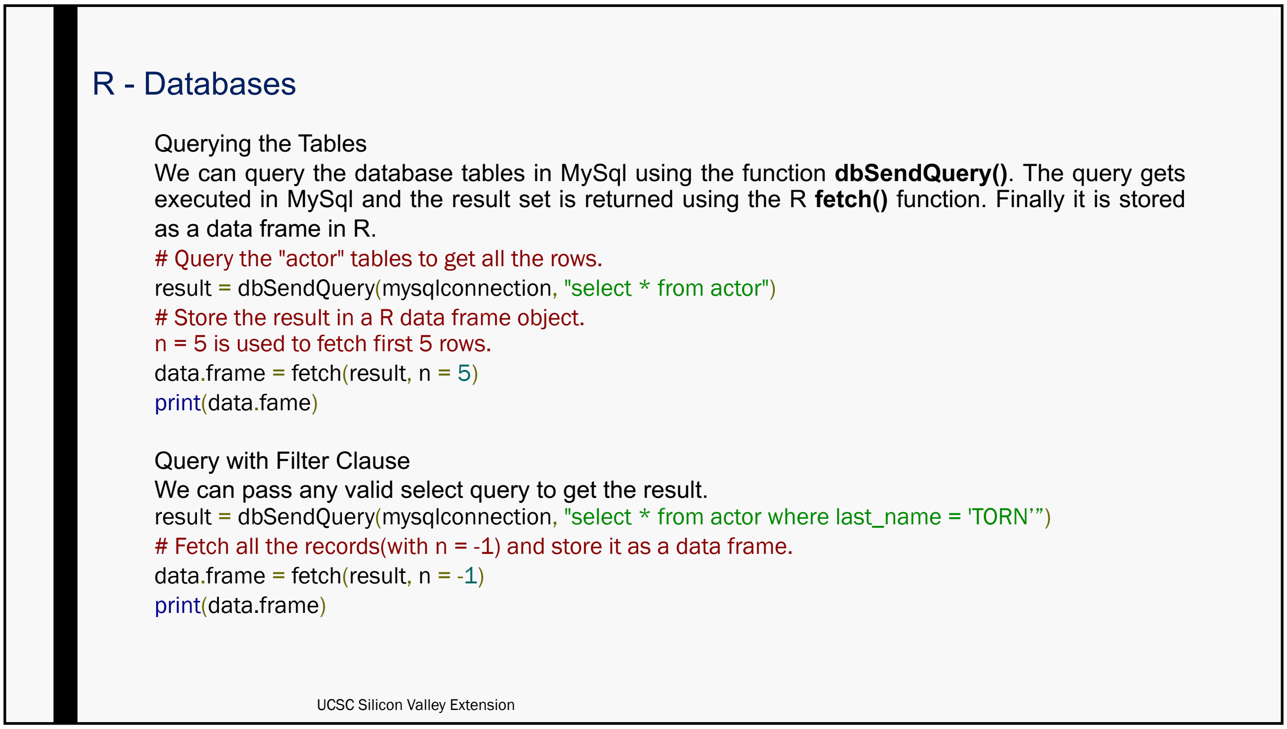
Connecting R to MySql

Once the package is installed we create a connection object in R to connect to the database. It takes the username, password, database name and host name as input.

```
# Create a connection Object to MySQL database.
# We will connect to the sample database named "sakila" that comes with MySql installation.
mysqlconnection = dbConnect(MySQL(), user = 'root', password = '', dbname = 'sakila', host =
'localhost’)
 # List the tables available in this database.
dbListTables(mysqlconnection)
```

UCSC Silicon Valley Extension

68



R - Databases

Querying the Tables

We can query the database tables in MySql using the function **dbSendQuery()**. The query gets executed in MySql and the result set is returned using the R **fetch()** function. Finally it is stored as a data frame in R.

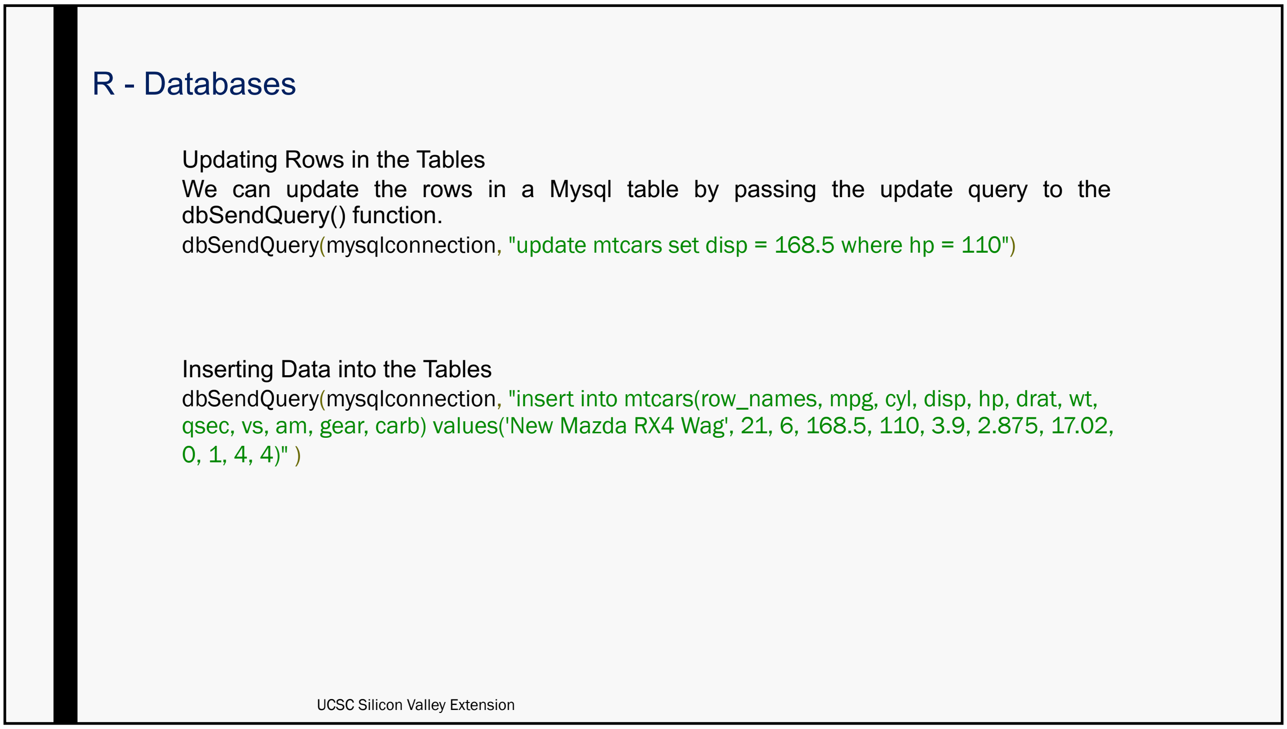
```
# Query the "actor" tables to get all the rows.
result = dbSendQuery(mysqlconnection, "select * from actor")
# Store the result in a R data frame object.
n = 5 is used to fetch first 5 rows.
data.frame = fetch(result, n = 5)
print(data.fame)
```

Query with Filter Clause

We can pass any valid select query to get the result.

```
result = dbSendQuery(mysqlconnection, "select * from actor where last_name = 'TORN’”)
# Fetch all the records(with n = -1) and store it as a data frame.
data.frame = fetch(result, n = -1)
print(data.frame)
```

69



R - Databases

Updating Rows in the Tables

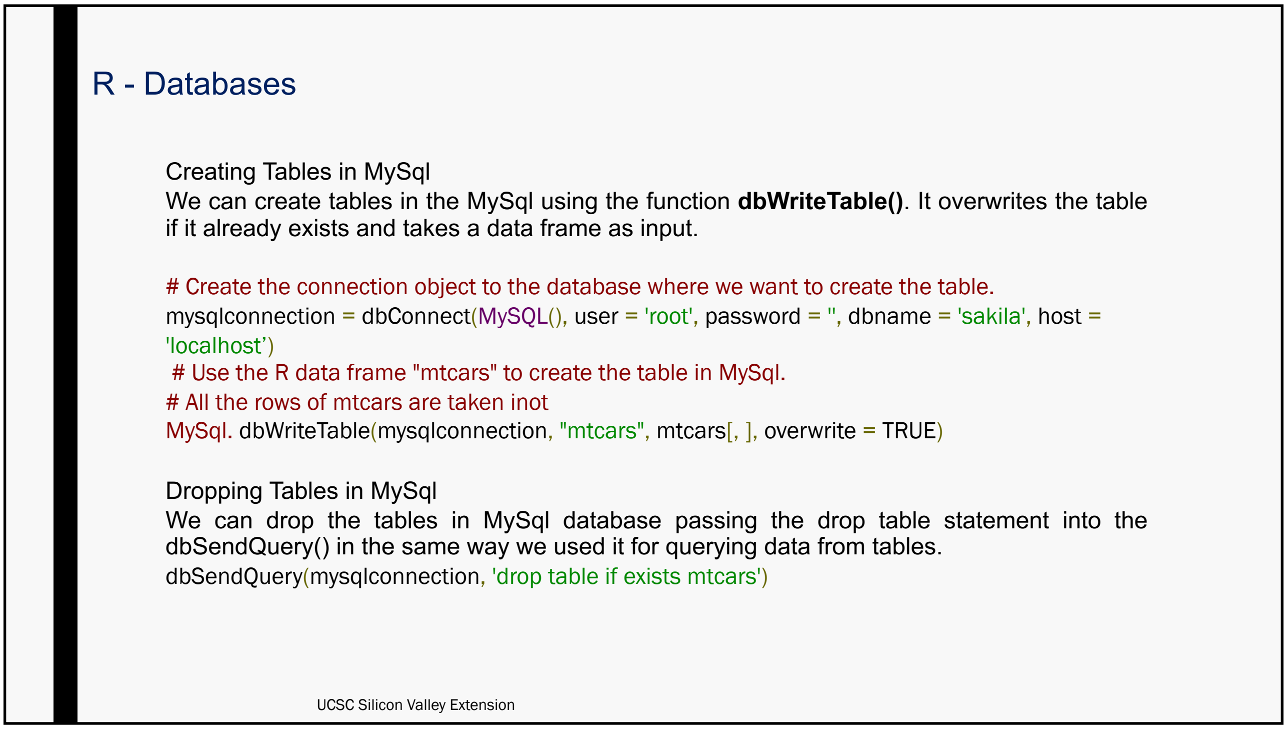
We can update the rows in a Mysql table by passing the update query to the dbSendQuery() function.

```
dbSendQuery(mysqlconnection, "update mtcars set disp = 168.5 where hp = 110")
```

Inserting Data into the Tables

```
dbSendQuery(mysqlconnection, "insert into mtcars(row_names, mpg, cyl, disp, hp, drat, wt,
qsec, vs, am, gear, carb) values('New Mazda RX4 Wag', 21, 6, 168.5, 110, 3.9, 2.875, 17.02,
0, 1, 4, 4)" )
```

70



R - Databases

Creating Tables in MySql
We can create tables in the MySql using the function **dbWriteTable()**. It overwrites the table if it already exists and takes a data frame as input.

```
# Create the connection object to the database where we want to create the table.
mysqlconnection = dbConnect(MySQL(), user = 'root', password = '', dbname = 'sakila', host =
'localhost’)
 # Use the R data frame "mtcars" to create the table in MySql.
# All the rows of mtcars are taken inot
MySql. dbWriteTable(mysqlconnection, "mtcars", mtcars[, ], overwrite = TRUE)
```

Dropping Tables in MySql
We can drop the tables in MySql database passing the drop table statement into the dbSendQuery() in the same way we used it for querying data from tables.

```
dbSendQuery(mysqlconnection, 'drop table if exists mtcars')
```

UCSC Silicon Valley Extension