INTRODUCTION TO DATA ANALYSIS

Partha Padmanabhan

UCSC Silicon Valley Extension
YOUR UC IN SILICON VALLEY

Lecture 2

Today's Topics

- Download & Install R Studio
- Running R from USB Drive
- How to run R
  - *Two Modes of operating R*
- Components of R
- R Studio Environment
- R Packages
- Data Types in R
- Mean,Median & Mode
- Vectors
- Matrices
- Lists
- Factors
- Data Frames
- Getting Data into R
  - *Importing Data into R*
- Importing Data
  - *Writing Data to a file*
  - *Getting Data out*
- Flow Controls/Loops
- Functions
- R Help

Download & Install R Studio

https://rstudio.com/products/rstudio/download/

https://cran.r-project.org/mirrors.html - Console

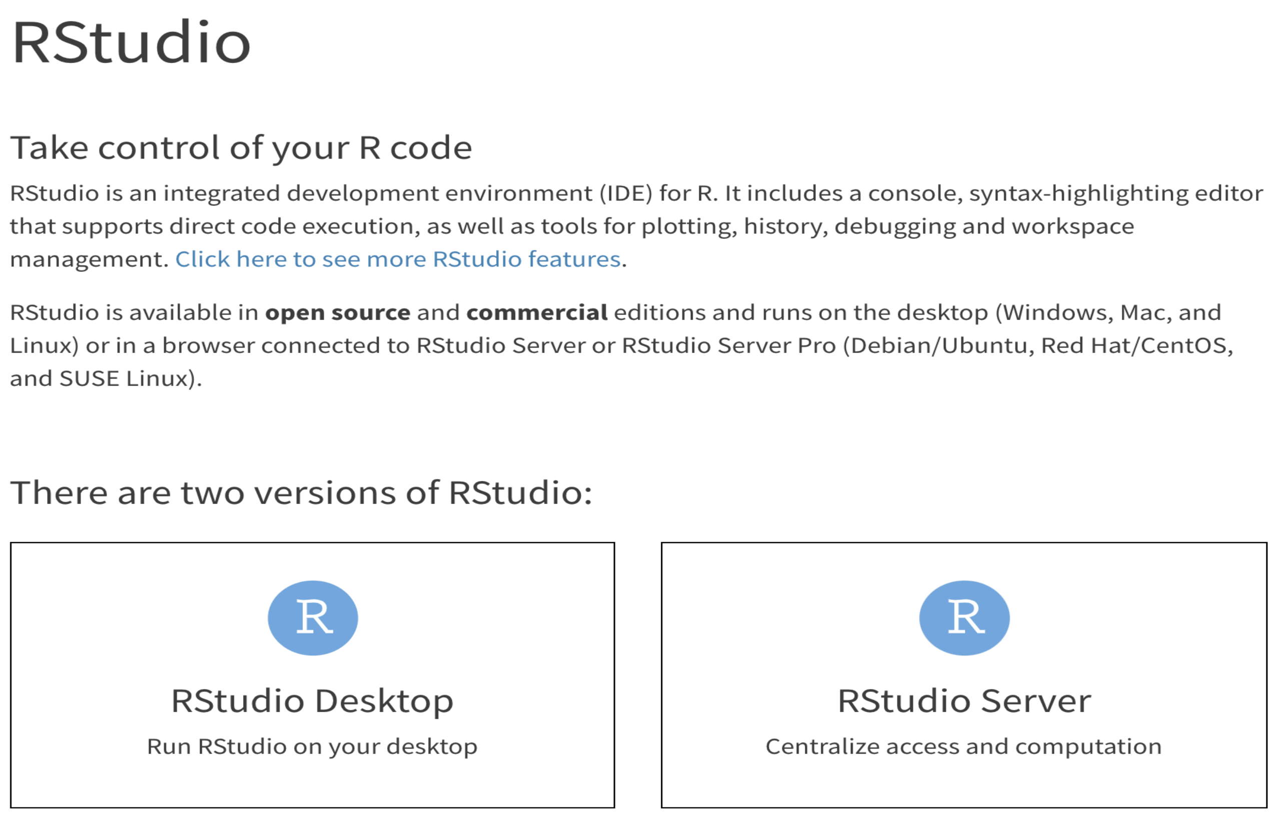
Running R from USB Drive

It is also possible to run R and RStudio from a USB stick instead of installing them.

For R related tutorials and/or resources see the following links:

– http://dss.princeton.edu/training
– http:////libguides.princeton.edu/dss
– https://www.rdocumentation.org/
– https://www.rstudio.com/resources/cheatsheets/

Download & Install R Studio



RStudio

Take control of your R code

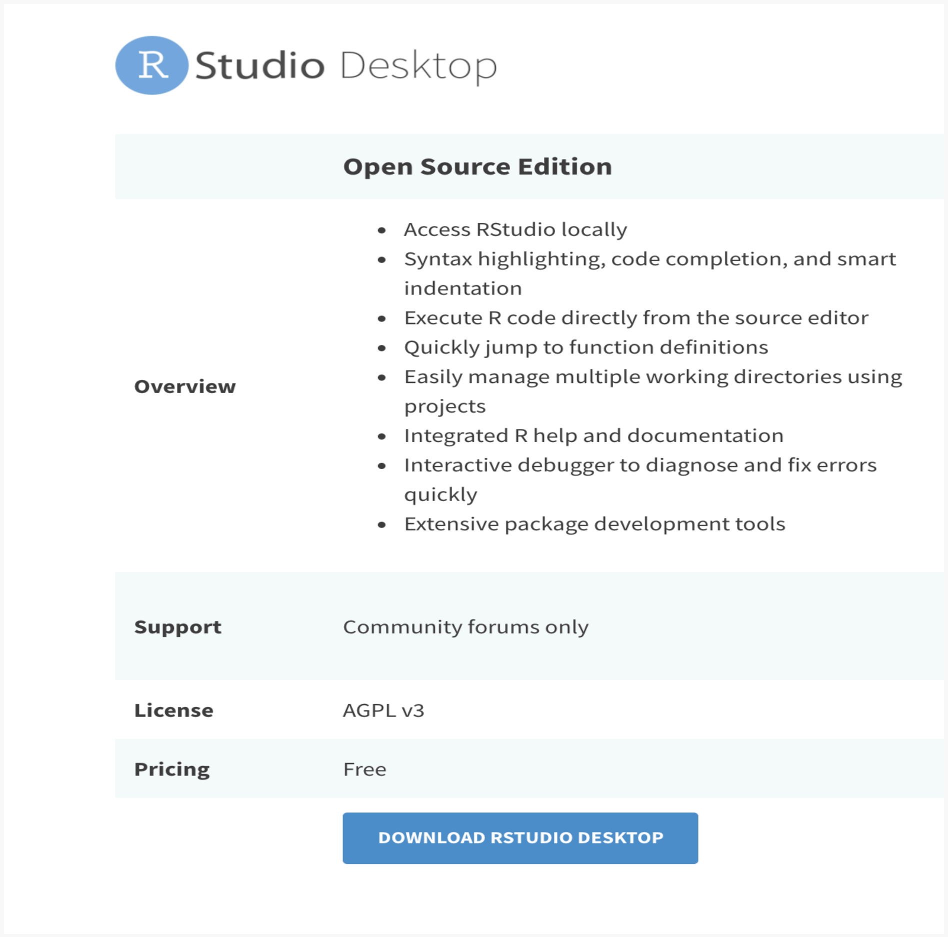
RStudio is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management. Click here to see more RStudio features.

RStudio is available in **open source** and **commercial** editions and runs on the desktop (Windows, Mac, and Linux) or in a browser connected to RStudio Server or RStudio Server Pro (Debian/Ubuntu, Red Hat/CentOS, and SUSE Linux).

There are two versions of RStudio:

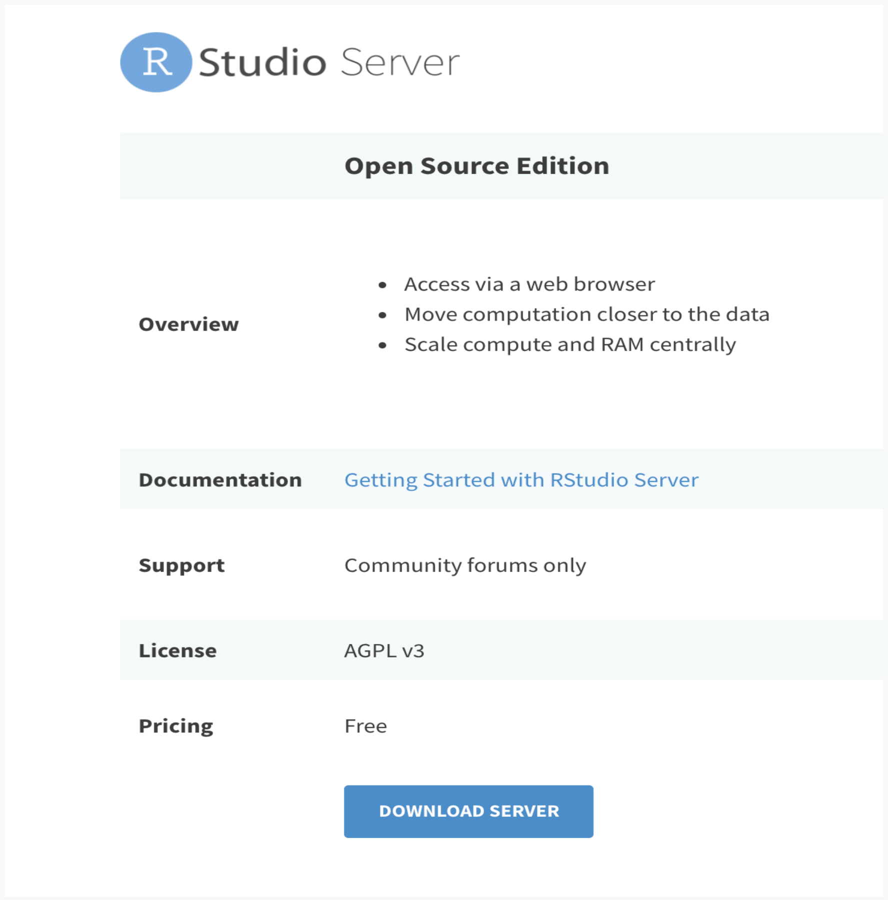
RStudio Desktop
Run RStudio on your desktop

RStudio Server
Centralize access and computation



R Studio Desktop

	Open Source Edition
Overview	• Access RStudio locally • Syntax highlighting, code completion, and smart indentation • Execute R code directly from the source editor • Quickly jump to function definitions • Easily manage multiple working directories using projects • Integrated R help and documentation • Interactive debugger to diagnose and fix errors quickly • Extensive package development tools
Support	Community forums only
License	AGPL v3
Pricing	Free



RStudio Server

Open Source Edition

Overview	• Access via a web browser • Move computation closer to the data • Scale compute and RAM centrally
Documentation	Getting Started with RStudio Server
Support	Community forums only
License	AGPL v3
Pricing	Free

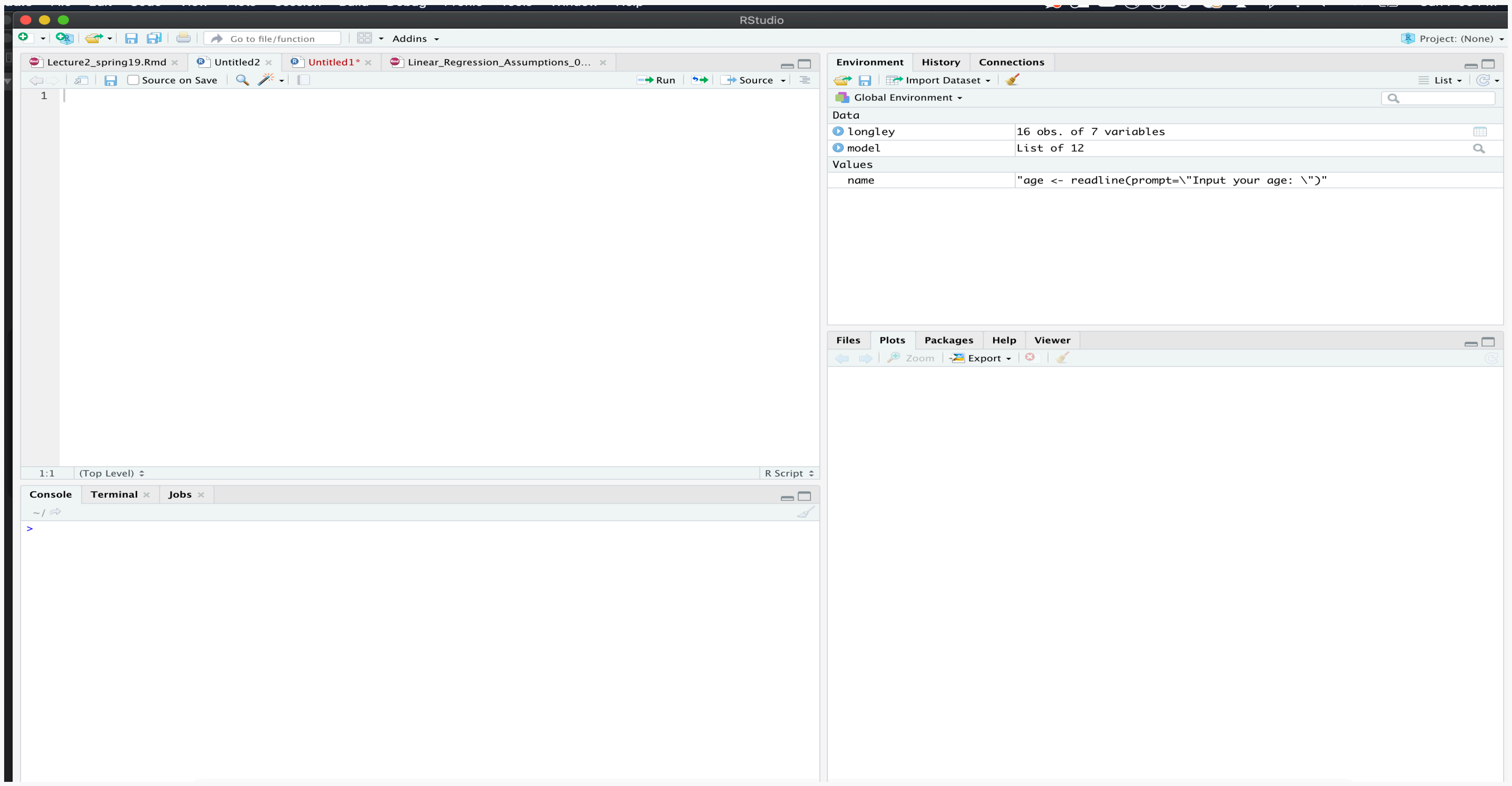
How to run ‘R’

R operates in two modes: *interactive* and *batch*. The one typically used is interactive mode. In this mode , you type in commands, R displays results.

On the other hand, batch mode does not require interaction with the user. It’s useful for production jobs, such as when a program must be run periodically, say once per day ,because you can automate the process.

Interactive Mode
The result is a greeting and the R prompt, which is the > sign.

R Studio Window



```
Data
longley    16 obs. of 7 variables
model      List of 12
Values
name       "age <- readline(prompt=\"Input your age: \")"
```

Console Window (Bottom Left)

All the “action” happen here

- You can type simple commands after the “>” prompt and R will execute them.

- The area shows the output of code you run. Also, you can directly write codes in console.

- Note that the current working directory is shown on the top left.

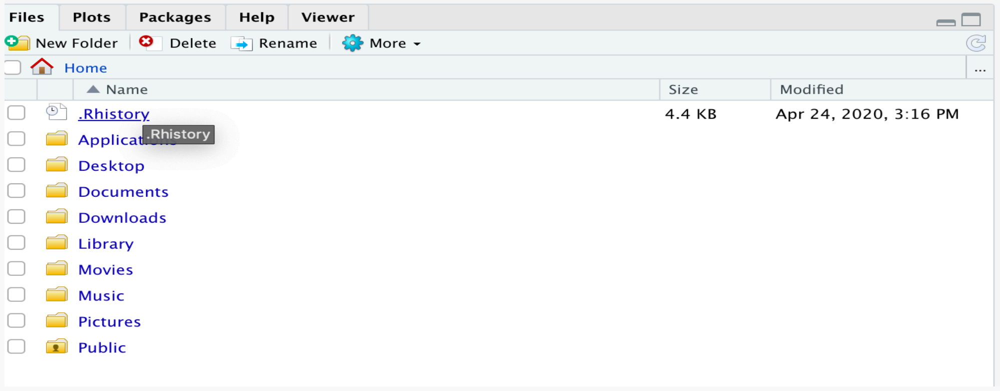
Environment/History Window

In this window you can see which data and values R has in memory.

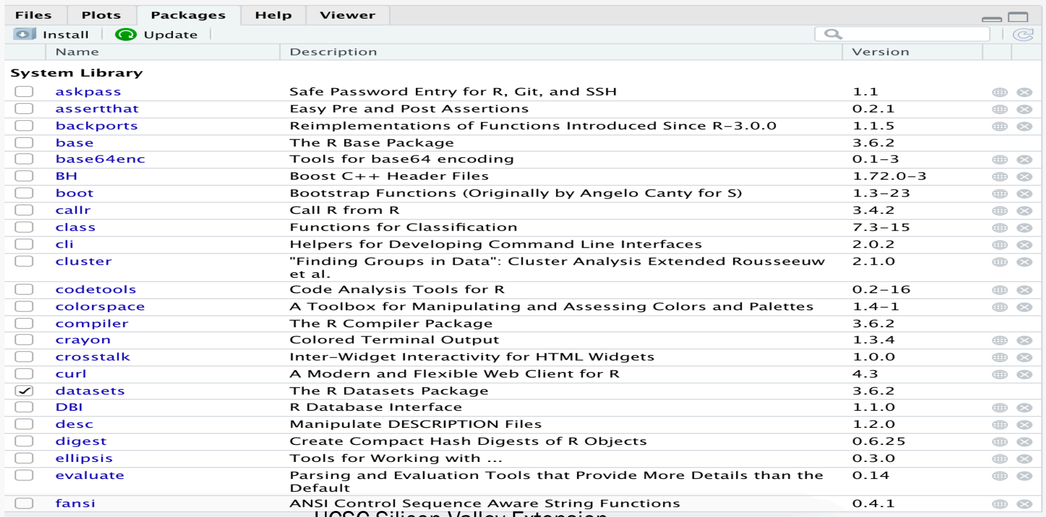
You can view and edit the values by clicking on them.

To check if data has been loaded correctly in R, always look at this area.

Files/Plot/Packages/Help/Viewer Window

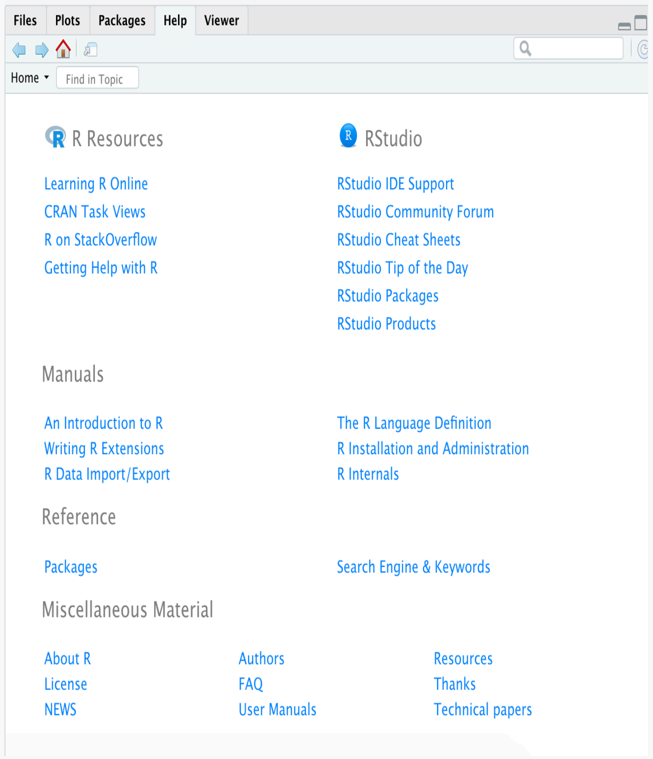


Name	Size	Modified
.Rhistory	4.4 KB	Apr 24, 2020, 3:16 PM
Applications		
Desktop		
Documents		
Downloads		
Library		
Movies		
Music		
Pictures		
Public		



	Name	Description	Version
	System Library		
☐	askpass	Safe Password Entry for R, Git, and SSH	1.1
☐	assertthat	Easy Pre and Post Assertions	0.2.1
☐	backports	Reimplementations of Functions Introduced Since R–3.0.0	1.1.5
☐	base	The R Base Package	3.6.2
☐	base64enc	Tools for base64 encoding	0.1–3
☐	BH	Boost C++ Header Files	1.72.0–3
☐	boot	Bootstrap Functions (Originally by Angelo Canty for S)	1.3–23
☐	callr	Call R from R	3.4.2
☐	class	Functions for Classification	7.3–15
☐	cli	Helpers for Developing Command Line Interfaces	2.0.2
☐	cluster	"Finding Groups in Data": Cluster Analysis Extended Rousseeuw et al.	2.1.0
☐	codetools	Code Analysis Tools for R	0.2–16
☐	colorspace	A Toolbox for Manipulating and Assessing Colors and Palettes	1.4–1
☐	compiler	The R Compiler Package	3.6.2
☐	crayon	Colored Terminal Output	1.3.4
☐	crosstalk	Inter–Widget Interactivity for HTML Widgets	1.0.0
☐	curl	A Modern and Flexible Web Client for R	4.3
✓	datasets	The R Datasets Package	3.6.2
☐	DBI	R Database Interface	1.1.0
☐	desc	Manipulate DESCRIPTION Files	1.2.0
☐	digest	Create Compact Hash Digests of R Objects	0.6.25
☐	ellipsis	Tools for Working with ...	0.3.0
☐	evaluate	Parsing and Evaluation Tools that Provide More Details than the Default	0.14
☐	fansi	ANSI Control Sequence Aware String Functions	0.4.1

You can open files, view plots, install and load packages or use the help function



R Resources

- Learning R Online
- CRAN Task Views
- R on StackOverflow
- Getting Help with R

RStudio

- RStudio IDE Support
- RStudio Community Forum
- RStudio Cheat Sheets
- RStudio Tip of the Day
- RStudio Packages
- RStudio Products

Manuals

- An Introduction to R
- Writing R Extensions
- R Data Import/Export
- The R Language Definition
- R Installation and Administration
- R Internals

Reference

- Packages
- Search Engine & Keywords

Miscellaneous Material

- About R
- License
- NEWS
- Authors
- FAQ
- User Manuals
- Resources
- Thanks
- Technical papers

How to Run ‘R’

Batch Mode
Sometimes it’s convenient to automate R sessions. For example, you may wish to run an R script that generates a graph without needing to bother with manually launching R and executing the script yourself.

```
pdf(“ex.pdf”)
plot(rnorm(100), type = ‘l’)
dev.off()
```

We could run this code automatically, without entering R’s interactive mode, by invoking R with an operating system shell command

```
R CMD BATCH ex.R
```

Working Directory

Working directory is the folder in which you are currently working.
- Before you start working, please set your working directory to where all your data and script files are or should be stored

- Show the working directory
– getwd()

- Type in the command to set the working directory
– setwd("C:/user/R/")

- R uses forward slashes instead of backward slashes in filenames (as shown in above)

- Object names in R can be any length consisting of letters, numbers, underscores "_" or the periods "."

- R is case sensitive, so make sure you write capitals where necessary

R Packages

R packages are a collection of R functions, complied code and sample data. They are stored under a directory called **"library"** in the R environment. By default, R installs a set of packages during installation. More packages are added later, when they are needed for some specific purpose.

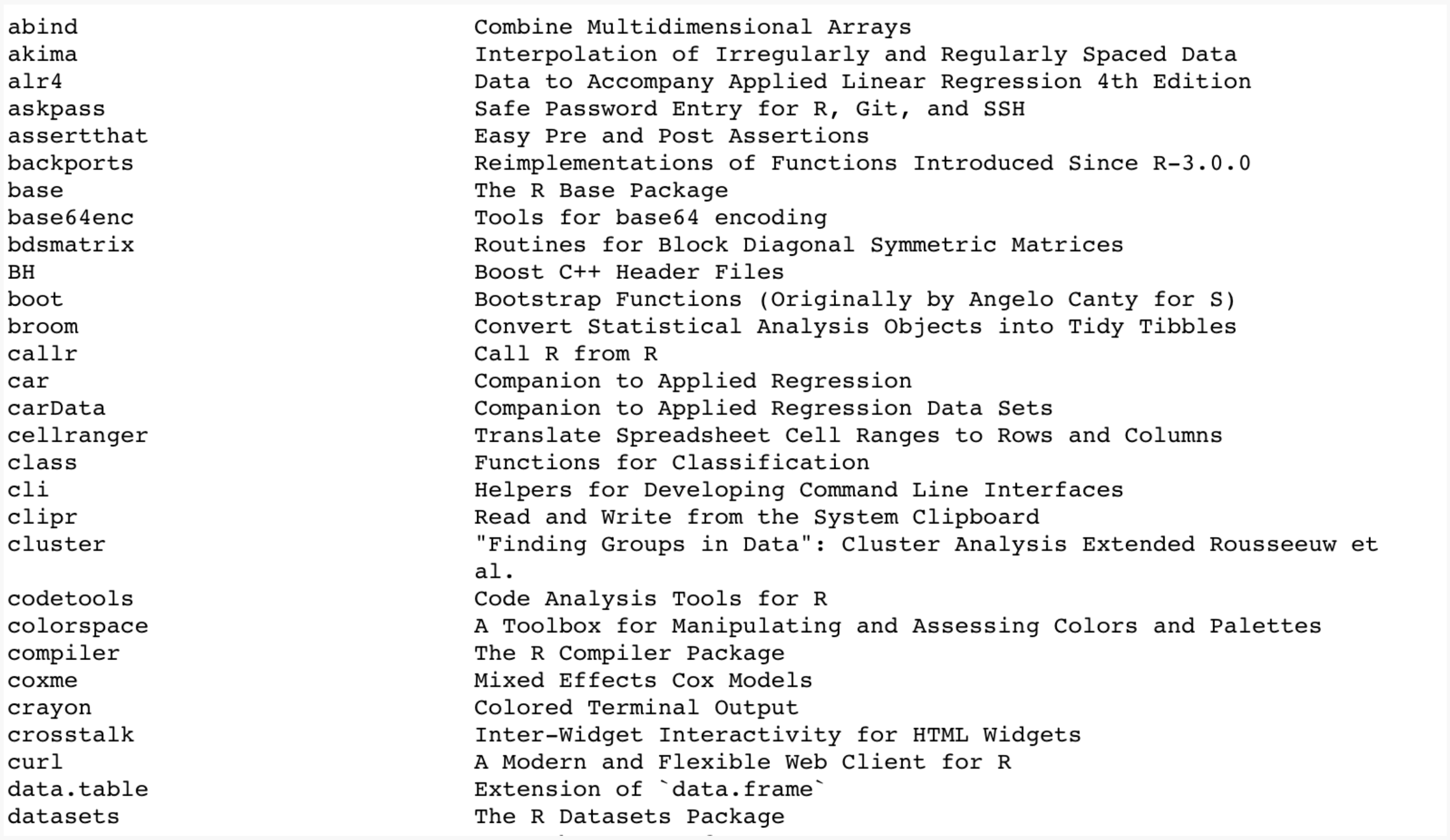
When we start the R console, only the default packages are available by default. Other packages which are already installed have to be loaded explicitly to be used by the R program that is going to use them.

Get library locations containing R packages

```
.libPaths()

        "/Library/Frameworks/R.framework/Versions/4.0/Resources/library”
```

> library()



```
abind                            Combine Multidimensional Arrays
akima                            Interpolation of Irregularly and Regularly Spaced Data
alr4                             Data to Accompany Applied Linear Regression 4th Edition
askpass                          Safe Password Entry for R, Git, and SSH
assertthat                       Easy Pre and Post Assertions
backports                        Reimplementations of Functions Introduced Since R-3.0.0
base                             The R Base Package
base64enc                        Tools for base64 encoding
bdsmatrix                        Routines for Block Diagonal Symmetric Matrices
BH                               Boost C++ Header Files
boot                             Bootstrap Functions (Originally by Angelo Canty for S)
broom                            Convert Statistical Analysis Objects into Tidy Tibbles
callr                            Call R from R
car                              Companion to Applied Regression
carData                          Companion to Applied Regression Data Sets
cellranger                       Translate Spreadsheet Cell Ranges to Rows and Columns
class                            Functions for Classification
cli                              Helpers for Developing Command Line Interfaces
clipr                            Read and Write from the System Clipboard
cluster                          "Finding Groups in Data": Cluster Analysis Extended Rousseeuw et
                                 al.
codetools                        Code Analysis Tools for R
colorspace                       A Toolbox for Manipulating and Assessing Colors and Palettes
compiler                         The R Compiler Package
coxme                            Mixed Effects Cox Models
crayon                           Colored Terminal Output
crosstalk                        Inter-Widget Interactivity for HTML Widgets
curl                             A Modern and Flexible Web Client for R
data.table                       Extension of `data.frame`
datasets                         The R Datasets Package
```

R Package Management

•R can do many statistical and data analysis. They are organized in so- called packages or libraries. With the standard installations, most common packages are installed.

• Before I get to those, let's see how you can tell what you've already got. First, let's look at the search path again.

```
> search()
 [1] ".GlobalEnv"        "package:vcd"        "package:grid"
 [4] "package:caret"      "package:ggplot2"    "package:gbm"
 [7] "package:parallel"   "package:splines"    "package:lattice"
[10] "package:survival"   "package:data.table" "package:Matrix"
[13] "package:corrplot"   "package:readr"      "package:dplyr"
[16] "tools:rstudio"      "package:stats"      "package:graphics"
[19] "package:grDevices"  "package:utils"      "package:datasets"
[22] "package:methods"    "Autoloads"          "package:base"
>
```

How to install R Packages

- The sheer power of R lies in its incredible packages. An R package is a collection of functions, data, and documentation that extends the capabilities of base R.

- There are many more (10000+) packages available on the R website. If you want to use particular package you should install the package. You can install package as follows:
– install.packages(“package name”)

If you want to know the packages currently installed on your computer you can issue
> installed.packages()

This produces a long output with each line containing a package, its version information, the packages it depends, and so on.

How to install R Packages

If you want more information on one of these packages, you can get its this way
>packageDescription(“caret”)
Another way to get information is to ask for help about the package > library(help = “caret”)

You can use the following command to update all your installed packages:
> update.package()
– To use package type library(“pn’) in the command window.

R Documentation
– http://www.rstudio.com/ide/docs/
– https://mran.microsoft.com/

Few things to know about R

Command prompt, If it is not there, then you are in the middle of entering lines (+ prompt)

- R is case sensitive! (x and X are different)
- R does not care about spaces
- Variables can have "." **var.names**
- # is for comments. Add anywhere to comment out till the end of that line

Some R command # this part is comment Block comments are possible in Rstudio Highlight the code and Command+Shift+C

Useful R packages

• R comes preloaded with {base}. As I mentioned already that there more than 10000 packages. I have listed some of the most powerful and commonly used packages.

Importing Data: R offers wide range of packages for importing data available in any format such as .txt, .csv. .json, .sql etc. Install and use *data.table, readr*, *RMySQL*, *sqldf*, *jsonlite*.

Data Visualization: R has in built plotting commands as well. They are good to create simple graphs. But, becomes complex when it comes to creating advanced graphics. Hence, you should install *ggplot2, lattice.*

Data Manipulation: R has a good collection of packages for data manipulation. These packages allows you to do basic and advanced computation quickly.

These packages are *tidyr*, *dplyr,plyr*, *reshape2, lubridate, stringr*
Instead you can install *tidyverse (ggplot2, tibble, tidyr, readr, purrr, dplyr)*

Modeling/Machine Learning: For modeling, *caret* package in R powerful enough to create most of the machine learning model. However, you can install packages algorithms wise such as *randomforest, rpart*, *gbm* etc.

Some other important packages
- Hmisc, e1071*, forecast, caret
- Knitr, shiny

*Misc Functions of the Department of Statistics,
Probability Theory Group (Formerly e1071)

Basic R Commands

Let’s start using RStudio

Basic R Commands

The Basics – R as a calculator >10+2
R will give 12

>10^2
R will give 100

This does not assign any variable.

What are we learning here?

R Programming as the mechanism for performing Data Analysis.

Can R programming be used for building applications?   Not Really

R is statistical computing language

Type 2+2
It gets you the answer
You can also assign it to a variable
Eg. myVal <-(gets) 2+2
myVal = 4

Using Combine
myVal >- c(5,10,15,20)
myVal = 5 10 15 20
Add another value to myVal
myVal + 10
Whats the answer (it adds 10 to each element)
15 20 25 30
This is called Matrix Arithmetic

Lets create another variable

```
my2ndVal <- c(2,4,6,8)
myVal + my2ndVal
7 14 21 28
```

How to find out individual value from the variable

```
myVal[1]
5
myVal[4]
20
```

Lets create a matrix

```
Matrix(1:30)
```

Lets create a matrix in columns

```
Matrix(1:30,ncol=6)
```

It created 7 columns

You can also create matrix in rows

```
Matrix(1:30,nrow=6)
```

Use the history box on the upper right to find all your previous commands

Pick any command and send it to Console

Assign the value to a matrix

matrix(1:30,ncol=6) ->(assign) mymatrix

How to retrieve the 4th column from mymatrix

mymatrix[,4]

First value is row and 2nd value is column, thats why I used a comma without specifying the row

You can also use the same command for retrieving the row

Mymatrix[3,]
It brings the 3rd row

mymatrix[3,4]. - gives what?
Not the rows and columns, but the 4th column in the 3rd row

Lets find the subset by dividing (modulo operator)

```
mymatrix[3,mymatrix[3,] %% 4 ==0]
```

Browser() function is used for debugging

Lets run a simple program

```
testrun <- 1:10

dosomething <- function (x) {
  print(x)
}

for (index in testrun){
  dosomething(index)
}
```

Look at the global environment value
Each value is stored
Browser() can be used to debug the problem step by step

Data Types in R

While R can handle many type of data, the three main varieties that we'll be using are numeric, character, and logical. In R, you can identify what type of object you're dealing with by using the *mode or typeof()* function. For example

```
>x <- 123
 > ls()
 [1] "x"
> x
[1] 123
>mode(x)
[1] "numeric"
>rm(x)
> rm(list=ls())
 > quit()
# creates the data object
# look at it your workspace
# see the vale in x
# see the type of the data
# erase it – remove(x) also works
# this completely clears the workspace # Quitting R
```

Assigning variables

In R, you can create a variable name using <- or = sign. Let us say I want to create a variable x equal to 5. I will write it as:

```
>x=5
>x <- 5
 >x<-x+10
 > 5-> x
```

Three things to note here.

- First, R is perfectly willing to let you be stupid and overwrite things you have in your workspace. There is no warning. If you assign something to an object name that already exists, the old object is gone!
- Second, the arrow assignment works from either direction. The equal sign does not! When using =, you must give the object name first followed by the value you wish to assign to it.

Assigning variables

Third, notice that when you do an assignment, nothing prints to the console. R creates the data object in your workspace and remains silent. If your intention is do assignment, thus creating a data object in your workspace, and you see the data spilling onto the console after pressing Enter, then chances are you've forgotten to give your new data object a name (and therefore have not created a new data object).

This is particularly painful when reading in a file or using a more complex "data-creating" function such as scan(). You can spend quite a long time typing in your data, press the Enter key, and see it all spill out onto the console. At that point, it's lost! You have to start over.

Be careful! When your intention is to create a data object in your workspace, make sure you assign it a name.

Data Types in R

R has various type of ‘data types’ which includes vector (numeric, integer etc.), matrices, data frames and list. Let us understand them one by one.

Vectors

It may be useful to have a variable to have only a single value, but usually we’ll want to store more than a single value in a variable. A vector is a collection of objects, all of same mode, that can be stored in a single variable and accessed through subscripts. Vector is a basic data structure in R. Vector contains element of the same type.

R - Mean, Median and Mode

Statistical analysis in R is performed by using many in-built functions. Most of these functions are part of the R base package. These functions take R vector as an input along with the arguments and give the result.
The functions we are discussing in this chapter are mean, median and mode.

Mean
It is calculated by taking the sum of the values and dividing with the number of values in a data series.

The function **mean()** is used to calculate this in R.

Syntax
The basic syntax for calculating mean in R is −
```
mean(x, trim = 0, na.rm = FALSE, ...)
```

X – input vector
Trim – used to drop some observations from both end of the sorted vector
na.rm is used to remove the missing values from the input vector.

Example

```
# Create a vector.  c = concatenate
x <- c(12,7,3,4.2,18,2,54,-21,8,-5)
# Find Mean.
result.mean <- mean(x)
print(result.mean)
```

Applying Trim Option
When trim parameter is supplied, the values in the vector get sorted and then the required numbers of observations are dropped from calculating the mean.
When trim = 0.3, 3 values from each end will be dropped from the calculations to find mean.
In this case the sorted vector is (−21, −5, 2, 3, 4.2, 7, 8, 12, 18, 54) and the values removed from the vector for calculating mean are (−21,−5,2) from left and (12,18,54) from right.

```
# Create a vector. x <- c(12,7,3,4.2,18,2,54,-21,8,-5)
# Find Mean.
result.mean <- mean(x,trim = 0.3)
print(result.mean)
```

Applying NA Option
If there are missing values, then the mean function returns NA.
To drop the missing values from the calculation use na.rm = TRUE. which means remove the NA values.

```
# Create a vector. X
 <- c(12,7,3,4.2,18,2,54,-21,8,-5,NA)
# Find mean.
result.mean <- mean(x)
 print(result.mean)
# Find mean dropping NA values.
result.mean <- mean(x,na.rm = TRUE)
print(result.mean)
```

Median
The middle most value in a data series is called the median. The **median()** function is used in R to calculate this value.

Syntax
The basic syntax for calculating median in R is −
median(x, na.rm = FALSE) Following is the description of the parameters used −
•**x** is the input vector.
•**na.rm** is used to remove the missing values from the input vector.

Example

```
# Create the vector. x <- c(12,7,3,4.2,18,2,54,-21,8,-5)
# Find the median.
median.result <- median(x)
print(median.result)
```

Mode

The mode is the value that has highest number of occurrences in a set of data. Unike mean and median, mode can have both numeric and character data.

R does not have a standard in-built function to calculate mode. So we create a user function to calculate mode of a data set in R. This function takes the vector as input and gives the mode value as output.

Example:

```
# Create the function.
getmode <- function(v) {
   uniqv <- unique(v)
   uniqv[which.max(tabulate(match(v, uniqv)))]
 }
# Create the vector with numbers.
v <- c(2,1,2,3,1,2,3,4,1,5,5,3,2,3)
# Calculate the mode using the user function.
result <- getmode(v)
print(result)
# Create the vector with characters.
charv <- c("o","it","the","it","it")
 # Calculate the mode using the user function.
result <- getmode(charv)
print(result)
```

predict() Function

Syntax

The basic syntax for predict() in linear regression is −

predict(object, newdata)

Following is the description of the parameters used −

- **object** is the formula which is already created using the lm() function. (lm used in **L**inear **M**odels)
- **newdata** is the vector containing the new value for predictor variable.

Predict the weight of new persons

```
# The predictor vector.
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
# The response vector.
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
# Apply the lm() function.
relation <- lm(y~x)
# ~ means Explained by
# Find weight of a person with height 170.
a <- data.frame(x = 170)
result <- predict(relation,a)
print(result)
```

Define a vector of the numbers we need the function c which is short for concatenate (paste together), and it’s used to put individual value together into vectors.

```
>B<-c(1,2,3,4,5)
 >B
```

You can find the number of elements in a vector using the length function

```
> length(B)
> [1] 5
```

To check the class of any object, use class(“Vector name”) function

```
 > class(B)
[1] "numeric"
```

Since, a vector must have elements of the same, this function will try and coerce elements to the same type, if they are different. Coercion is from lower to higher types from logical to double to character

```
>x <- c(1, 5.4, TRUE, "hello")
>x
[1] "1" "5.4" "TRUE" "hello" > typeof(x)
[1] "character”
```

Since, a vector must have elements of the same, this function will try and coerce elements to the same type, if they are different. Coercion is from lower to higher types from logical to double to character

```
>x <- c(1, 5.4, TRUE, "hello")
>x
[1] "1" "5.4" "TRUE" "hello"
> typeof(x)
[1] "character”
```

If we want to create a vector of consecutive numbers, the **:** operator is very helpful. More complex sequences can be created using the **seq**() function, like defining number of points in an interval, or the step size.

```
> x <- 1:7;
x
[1] 1 2 3 4 5 6 7
> y <- 2:-2;
>y
>[1] 2 1 0-1-2

>seq(1, 3, by=0.2)
># specify the step [1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0

> seq(1, 5, length.out=4)
# specify length of the vector [1]
1.000000 2.333333 3.666667 5.000000
```

The Vector function has three parameters:

1. VectorName
2. Values(Optional)
3. Value labels(Optional)

Accessing Elements in Vector
Elements of a vector can be accessed using vector indexing. The vector used for indexing can be logical, integer or character vector.

Using integer vector as index
Vector index in R starts from 1, unlike most programming languages where index start from 0. We can use a vector of integers as index to access specific elements. We can also use negative integers to return all elements except that those specified. But we cannot mix positive and negative integers while indexing and real numbers, if used, are truncated to integers.

```
>x<-seq(1,20, by =2)
>x
[1] 1 3 5 7 9 11 13 15 17 19
> x[2]
[1] 3


> x[c(2,5)]
>[1] 3 9
> x[-1]
# access all but 1st element
[1] 3 5 7 9 11 13 15 17  19
```

Vectorized Operations

In most programming languages, once you're dealing with vectors, you also have to start worrying about loops and other programming problems. No so in R! For example, suppose we want to use the conversion formula

C = 5/9*(F-32)

to convert our Fahrenheit temperatures into Celsius. We can simply put F as a single number and R will do the hard part:

Let us F = c(50.7, 52.8, 48.6, 49.9, 54.1)

```
>c = 5/9*(F-32)
>c
[1] 10.388889 11.555556 9.222222 9.944444 12.277778
```

There are various functions available which can be directly used, e.g.,
sum(B) mean(B), sd(B)

In fact, most similar operations in R are vectorized; that is they operate on entire vectors at once, without the need for loops or or other programming.

It’s possible to provide names for the elements of a vector. Suppose we were working with purchases in a number of states, and we needed to know the sales tax rate for a given state. We could create a named vector as follows:

```
>taxrate = c(AL = 4, CA = 8.75, TX = 0, NY = 4.25)
> taxrate
AL CA TX NY
4.00 8.75 0.00 4.25
```

To add names to a vector after a fact, you can use names function:

```
taxrate = c(4, 8.75, 0,4.25)
> names(taxrate)= c('AL', 'CA', 'TX', 'NY’)
> taxrate
AL CA TX NY 4.00 8.75 0.00 4.25
```

One of the most powerful tools in R is the ability to use logical expressions to extract or modify elements in the way that numeric subscripts are traditionally used. For example, suppose we want to find all of the observations in taxrate with a taxrate greater than 4.

```
>taxrate >4
AL CA TX NY

FALSE TRUE FALSE TRUE
```

The result is a logical vector of the same length as the vector we are asking about. If we use such a vector to extract values from the taxarate vector, it will give us all that corresponds to TRUE values, discarding the ones that corresponds to FALSE

```
> taxrate[taxrate >4]
CA NY
8.75 4.25
```

Another important use of logical variables is counting number of elements of a vector meet a particular condition. When a logical vector is passed to the sum function, TRUES count as one and FALSES count as 0, So we can count the number of logical expression by passing it to sum:

- ➢ sum(taxrate >4)
- ➢ [1] 2

This tells us two observations in the taxrate vectors had values greater than 4.
There are other functions also which can used.

```
> which(taxrate == max(taxrate))
CA
2
```

We can use which.max(taxrate)) also.

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
```

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
```

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
```

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
```

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”

```
➢ 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
```

Some of useful commands

rbind,cbind, cut(), table(), subset()

Transform to add columns

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

Lists

Another basic structure in R is a list.

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
```

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
```

Factors

R has a special data structure to store categorical variables. Categorical variables those which takes only discrete values such as 0,1,4, etc. In R, categorical variables are represented by factors. In the following example, gender is a factor variable having two unique levels. Factor or categorical variable are specially treated in data set.

Simplest form of the factor function :

```
> gender<-c(1,2,1,2,1,2,1,2)
Ø gender <-factor(gender)
Ø 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
```

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.

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.

```
> 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”
```

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
```

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
```