

INTRODUCTION TO R

Statistical Tools

- The current analytics market is flooded with statistical software But most of the data miners use R



Why R?

Why R?

- R is an open source software that is free to download.
- R is the only analytics software to run on MAC.
- Data scientists can improve the software's code or write variations for specific tasks.
- R offers analytics capabilities ranging from Text Analytics, Predictive analytics, Time Series forecasting, Optimization.
- With Commercial version of R, Big Data Analytics made possible.
- R have number of built-in mechanisms for organizing data, running calculations on the information and creating graphical representations of data sets.

Data Analysis

- The volume of data we have available has grown tremendously over the past few years
- According to a recent study, the volume of data generated in 2013 alone was in the order of 4.4 zettabytes (1 ZB = 44 trillion GB)
- The science of data analysis has kept pace with this explosion of data
- With cheap and easy access to powerful computers, modern data analysis has shifted to a different paradigm
- Data analysis has become highly interactive with output from each stage serving as an input to the next stage

Why Use R for your Statistical Work?

- Open source implementation of the widely regarded S statistical language
- Comparable and often superior to commercial products. It has more than 3,700 user contributed packages catering to a huge variety of tasks across multiple domains of science and technology
- In addition to providing statistical operations, R is a general purpose programming language, so you can use it to automate analyses and create new functions that extend the existing language features
- It incorporates features found in object – oriented and functional programming languages
- The huge popularity of R has resulted in creation of a number of online forums where questions are answered everyday from experts all around the world

A bit of R History

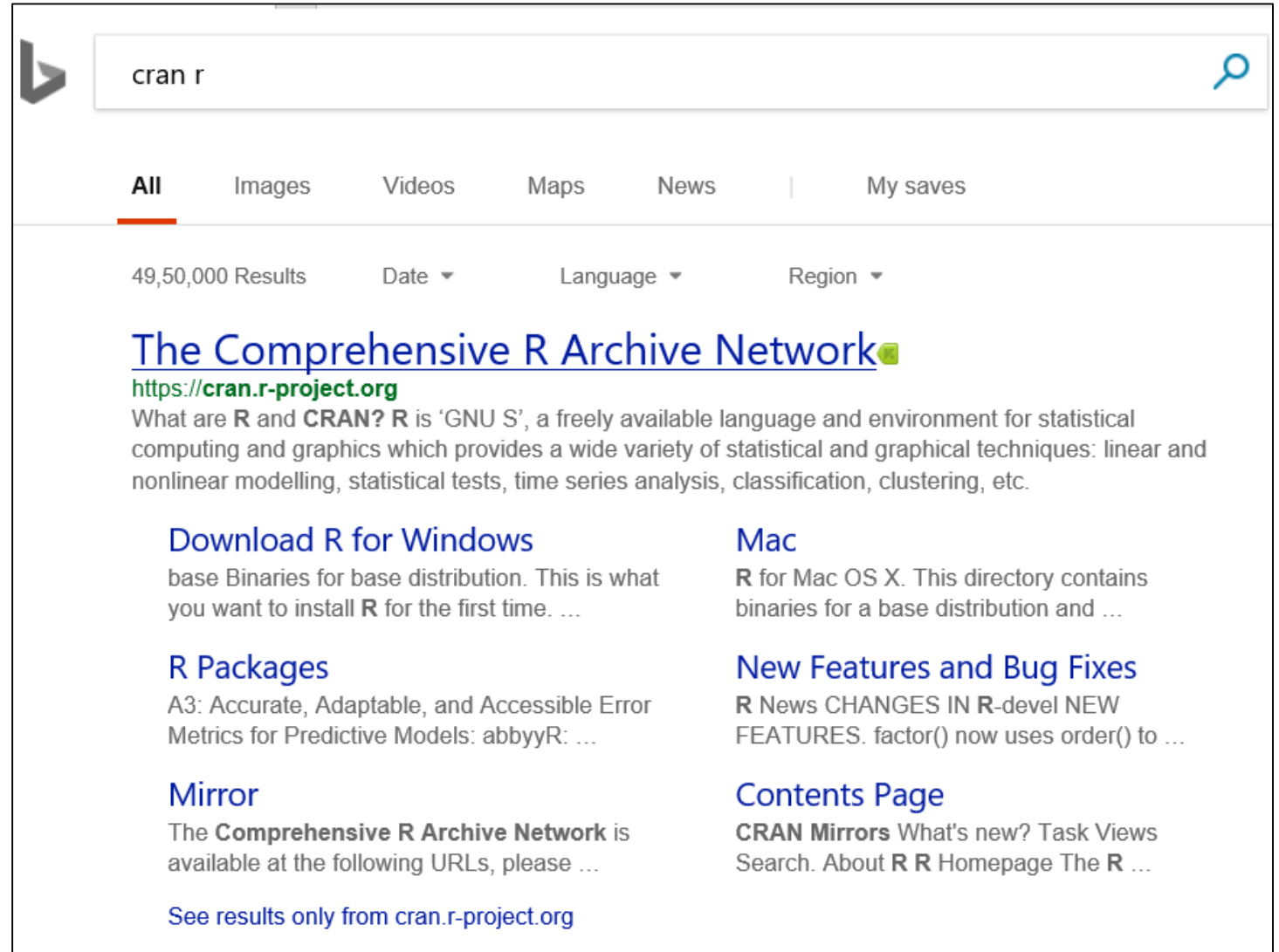
- R was first released in 1996 and has dramatically changed the landscape of research software
- It was first created by Ross Ihaka and Robert Gentleman at the University of Auckland, NZ
- It is considered by its developers to be an implementation of the S programming language
- This programming language was named **R**, based on the first letter of first name of the two R authors
- There are very few analyses which SAS or SPSS will do that R cannot, while R can do a wide range of things that others do not
- SAS and SPSS are very similar to each other, but R on the other hand can be quite confusing in the beginning



Downloading and Installing R

Download R

- R is 'GNU S', a freely available language and environment for statistical computing and graphics.
- It provides a wide variety of statistical and graphical techniques such as linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc.
- CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R.



The screenshot shows a search engine interface with the query 'cran r' in the search bar. Below the search bar, there are tabs for 'All', 'Images', 'Videos', 'Maps', 'News', and 'My saves'. The 'All' tab is selected. Below the tabs, it shows '49,500 Results' and filters for 'Date', 'Language', and 'Region'. The top result is 'The Comprehensive R Archive Network' with the URL 'https://cran.r-project.org'. Below the title, there is a brief description: 'What are R and CRAN? R is 'GNU S', a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc.' Below this description, there are several links: 'Download R for Windows', 'Mac', 'R Packages', 'New Features and Bug Fixes', 'Mirror', and 'Contents Page'. Each link has a short description. At the bottom, there is a link 'See results only from cran.r-project.org'.

cran r

All Images Videos Maps News | My saves

49,500 Results Date Language Region

[The Comprehensive R Archive Network](https://cran.r-project.org)

<https://cran.r-project.org>

What are **R** and **CRAN**? **R** is 'GNU S', a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc.

[Download R for Windows](#)
base Binaries for base distribution. This is what you want to install **R** for the first time. ...

[Mac](#)
R for Mac OS X. This directory contains binaries for a base distribution and ...

[R Packages](#)
A3: Accurate, Adaptable, and Accessible Error Metrics for Predictive Models: abbyyR: ...

[New Features and Bug Fixes](#)
R News **CHANGES IN R-devel** **NEW FEATURES**. factor() now uses order() to ...

[Mirror](#)
The **Comprehensive R Archive Network** is available at the following URLs, please ...

[Contents Page](#)
CRAN Mirrors What's new? Task Views Search. About **R** **R** Homepage The **R** ...

[See results only from cran.r-project.org](#)

Download R



CRAN

[Mirrors](#)

[What's new?](#)

[Task Views](#)

[Search](#)

About R

[R Homepage](#)

[The R Journal](#)

Software

[R Sources](#)

[R Binaries](#)

[Packages](#)

[Other](#)

Documentation

[Manuals](#)

[FAQs](#)

[Contributed](#)

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2017-11-30, Kite-Eating Tree) [R-3.4.3.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#).
- Contributed extension [packages](#)

Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

Download R



CRAN

[Mirrors](#)

[What's new?](#)

[Task Views](#)

[Search](#)

About R

[R Homepage](#)

[The R Journal](#)

Software

R for Windows

Subdirectories:

[base](#)

Binaries for base distribution. This is what you want to [install R for the first time](#).

[contrib](#)

Binaries of contributed CRAN packages (for R \geq 2.13.x; managed by Uwe Ligges). There is also information on [third party software](#) available for CRAN Windows services and corresponding environment and make variables.

[old contrib](#)

Binaries of contributed CRAN packages for outdated versions of R (for R $<$ 2.13.x; managed by Uwe Ligges).

[Rtools](#)


Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself.

Please do not submit binaries to CRAN. Package developers might want to contact Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the [R FAQ](#) and [R for Windows FAQ](#).

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

Download R



CRAN
[Mirrors](#)
[What's new?](#)
[Task Views](#)
[Search](#)

About R
[R Homepage](#)
[The R Journal](#)

Software
[R Sources](#)
[R Binaries](#)
[Packages](#)
[Other](#)

Documentation
[Manuals](#)
[FAQs](#)
[Contributed](#)

R-3.4.3 for Windows (32/64 bit)

[Download R 3.4.3 for Windows](#) (62 megabytes, 32/64 bit)

[Installation and other instructions](#)
[New features in this version](#)

If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the [md5sum](#) of the .exe to the [fingerprint](#) on the master server. You will need a version of md5sum for windows: both [graphical](#) and [command line versions](#) are available.

Frequently asked questions

- [Does R run under my version of Windows?](#)
- [How do I update packages in my previous version of R?](#)
- [Should I run 32-bit or 64-bit R?](#)

Please see the [R FAQ](#) for general information about R and the [R Windows FAQ](#) for Windows-specific information.

Other builds

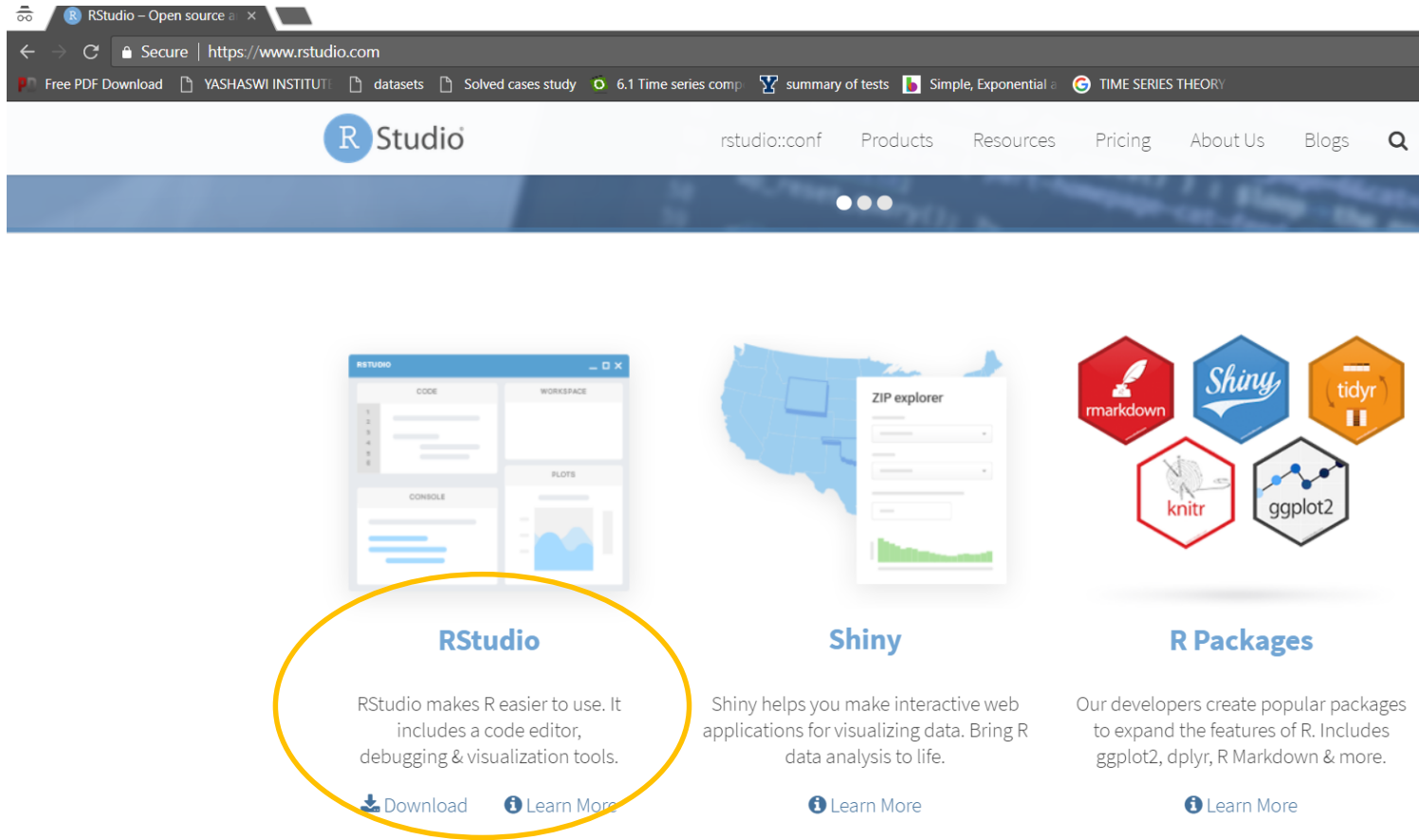
- Patches to this release are incorporated in the [r-patched snapshot build](#).
- A build of the development version (which will eventually become the next major release of R) is available in the [r-devel snapshot build](#).
- [Previous releases](#)

Note to webmasters: A stable link which will redirect to the current Windows binary release is [<CRAN MIRROR>/bin/windows/base/release.htm](https://cran.r-project.org/bin/windows/base/release.htm).

Last change: 2017-12-06

Download RStudio

- You can download and install R from: <https://www.rstudio.com/>
- Click on Download in Rstudio



Download RStudio

- Download Rstudio Desktop Open Source License
- Select installer as per your platform

	RStudio Desktop Open Source License	RStudio Desktop Commercial License	RStudio Server Open Source License
	FREE	\$995 per year	FREE
	DOWNLOAD Learn More	BUY Learn More	DOWNLOAD Learn More
Integrated Tools for R	●	●	●
Priority Support		●	
Access via Web Browser			●
Enterprise Security			
Project			

Installers for Supported Platforms

Installers

[RStudio 1.1.419 - Windows Vista/7/8/10](#)
[RStudio 1.1.419 - Mac OS X 10.6+ \(64-bit\)](#)
[RStudio 1.1.419 - Ubuntu 12.04-15.10/Debian 8 \(32-bit\)](#)
[RStudio 1.1.419 - Ubuntu 12.04-15.10/Debian 8 \(64-bit\)](#)
[RStudio 1.1.419 - Ubuntu 16.04+/Debian 9+ \(64-bit\)](#)
[RStudio 1.1.419 - Fedora 19+/RedHat 7+/openSUSE 13.1+ \(32-bit\)](#)
[RStudio 1.1.419 - Fedora 19+/RedHat 7+/openSUSE 13.1+ \(64-bit\)](#)

Zip/Tarballs

Zip/tar archives

[RStudio 1.1.419 - Windows Vista/7/8/10](#)
[RStudio 1.1.419 - Ubuntu 12.04-15.10/Debian 8 \(32-bit\)](#)
[RStudio 1.1.419 - Ubuntu 12.04-15.10/Debian 8 \(64-bit\)](#)
[RStudio 1.1.419 - Fedora 19+/RedHat 7+/openSUSE 13.1+ \(32-bit\)](#)
[RStudio 1.1.419 - Fedora 19+/RedHat 7+/openSUSE 13.1+ \(64-bit\)](#)

Source Code

[A tarball containing source code for RStudio v1.1.419 can be found here.](#)

Rstudio

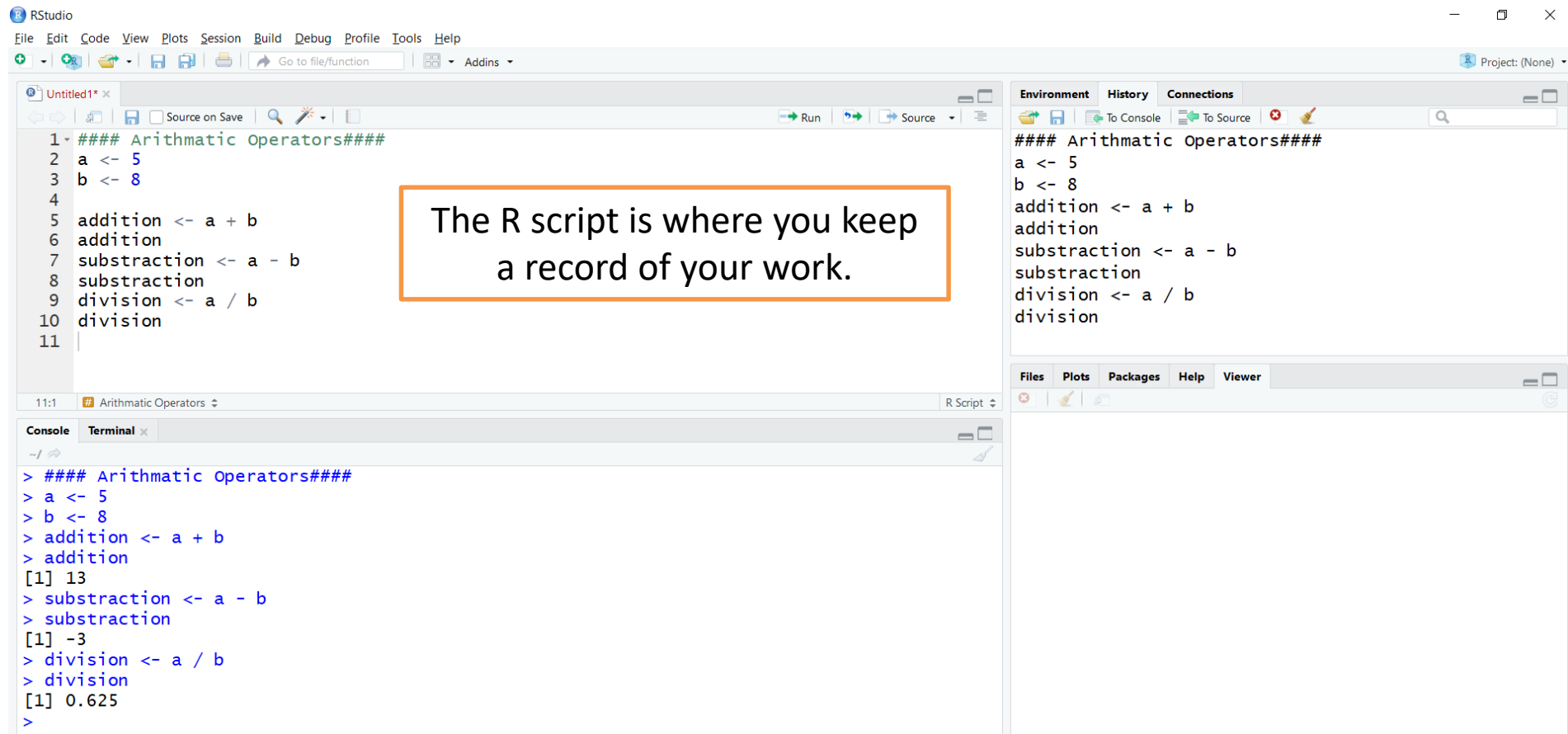
- The usual RStudio screen have four windows:

The Rscript
and Data

Environment,
History and
Connections

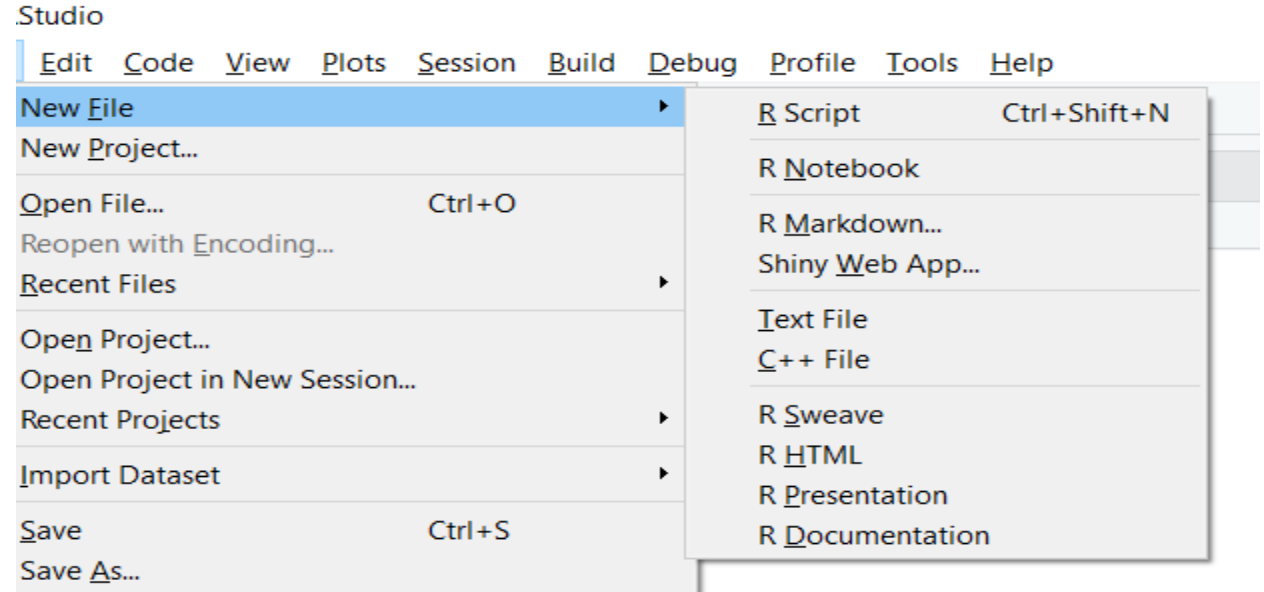
Console and
Terminal

Files, Plots,
Packages, Help
and Viewer



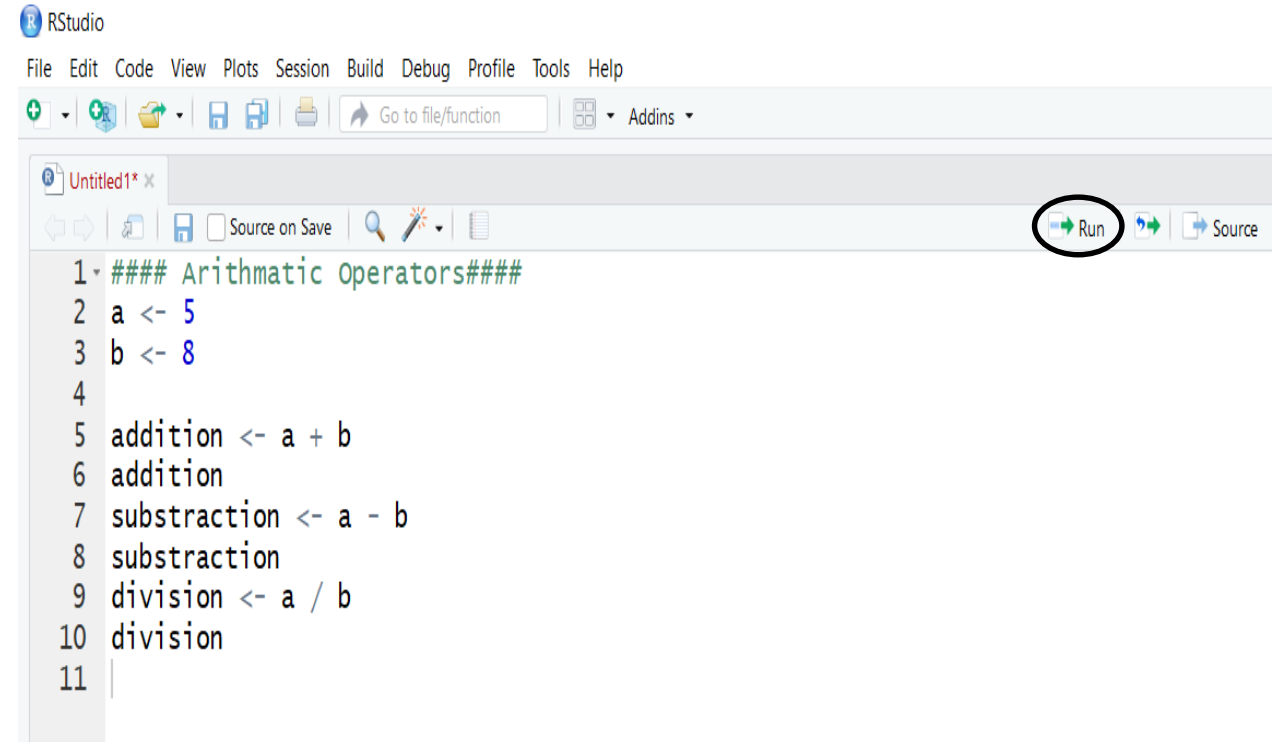
RStudio

- To create a new R script you can either go to File -> New -> R Script, or simply press Ctrl +Shift +N.
- Make sure to save the script.



RStudio

- **Here you can type R commands and run them.**
- Just leave the cursor anywhere on the line where the command is and press Ctrl+R or click on the 'Run' icon above.
- Output will appear in the console below.



RStudio

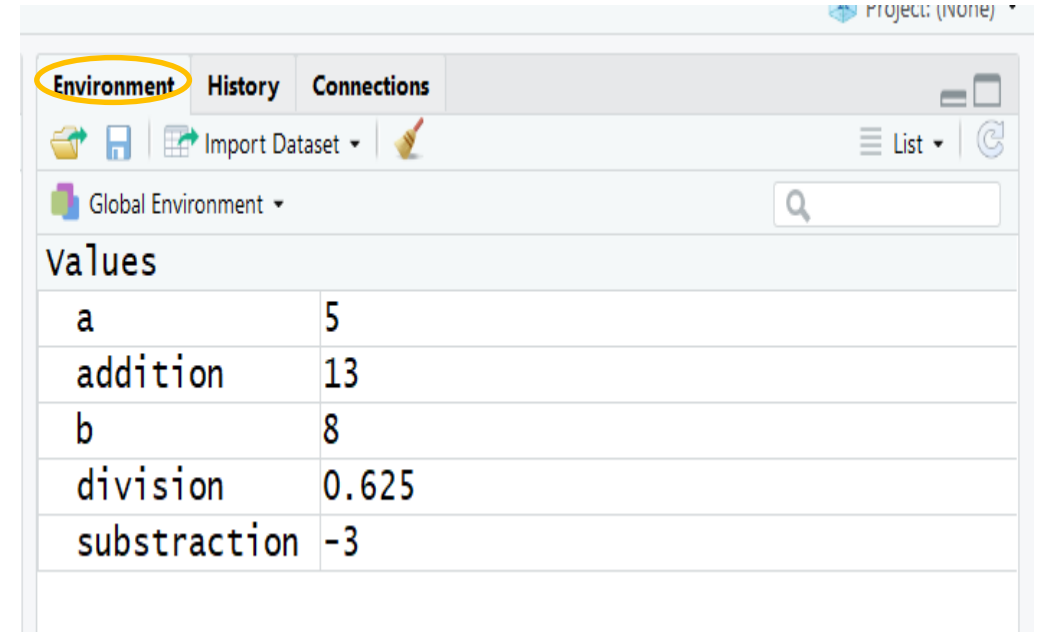
Environment Area :

1. View the Objects created as :

- ✓ Data frames
- ✓ Vectors
- ✓ Functions

2. Import Dataset which are in :

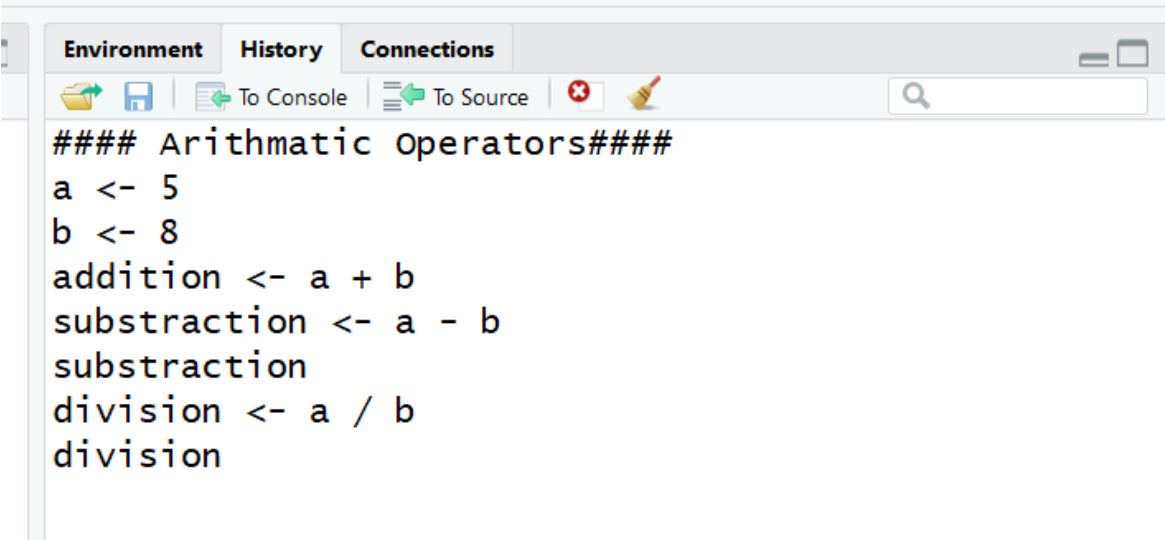
- ✓ CSV , Tab delimited or in web url.



RStudio

History Area :

- View the past commands
- Use the past commands either in console (To Console icon) or in source (To Source icon)
- To delete all commands use: CTRL + I



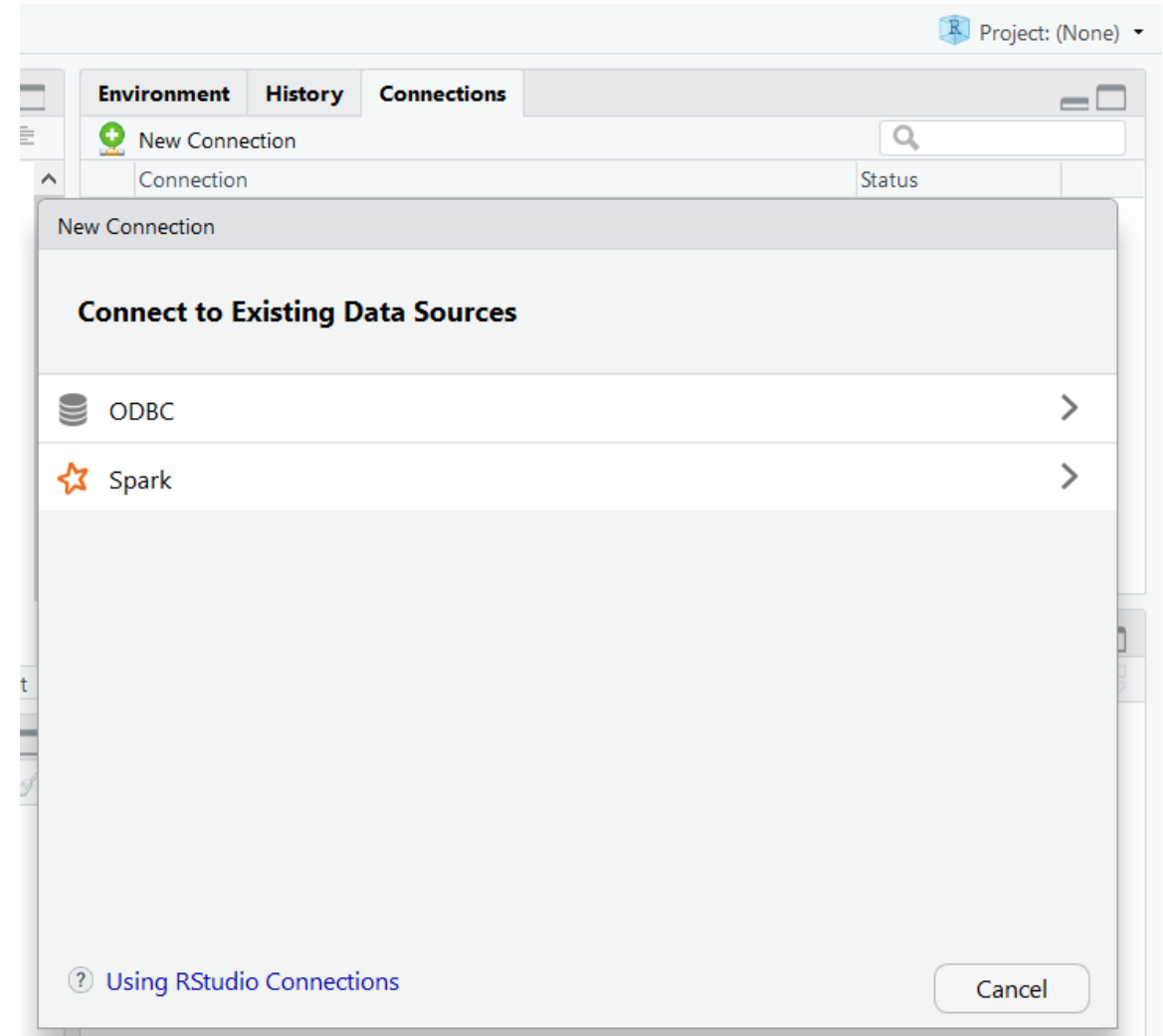
The screenshot shows the RStudio History pane. At the top, there are tabs for 'Environment', 'History', and 'Connections'. Below the tabs is a toolbar with icons for 'To Console', 'To Source', a red 'X' icon, and a broom icon. A search bar is located on the right side of the toolbar. The main area of the pane displays a list of commands that have been executed, starting with '#### Arithmetic Operators####'. The commands listed are: 'a <- 5', 'b <- 8', 'addition <- a + b', 'subtraction <- a - b', 'subtraction', 'division <- a / b', and 'division'.

```
#### Arithmetic Operators####  
a <- 5  
b <- 8  
addition <- a + b  
subtraction <- a - b  
subtraction  
division <- a / b  
division
```

RStudio

Connection Area :

- We use this to connect external server



RStudio

Console Area :

- We can type commands to
 - ✓ Call a library
 - ✓ Call a dataset
 - ✓ View the Dataset
 - ✓ Etc...

The screenshot shows the RStudio interface with the console area active. The console displays the R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch" startup message, including copyright information and usage instructions. The command `library(cluster)` has been entered at the prompt.

```
R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> library(cluster)
>
```

RStudio

File tab:

The files tab shows all the files and folders in your default environment.

Plot tab:

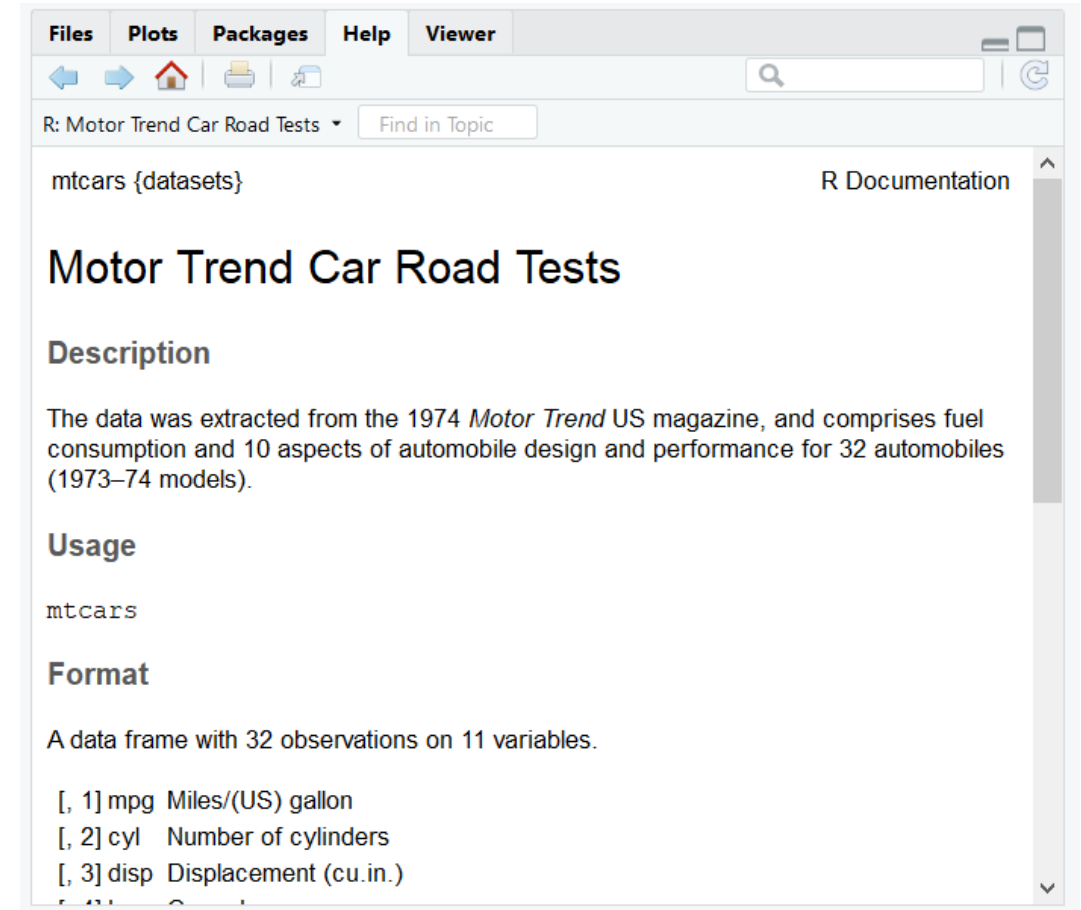
The plots tab will show all your graphs including 3d graphs.

Packages tab:

The packages tab will list a series of packages or add-ons needed to run certain processes.

Help tab:

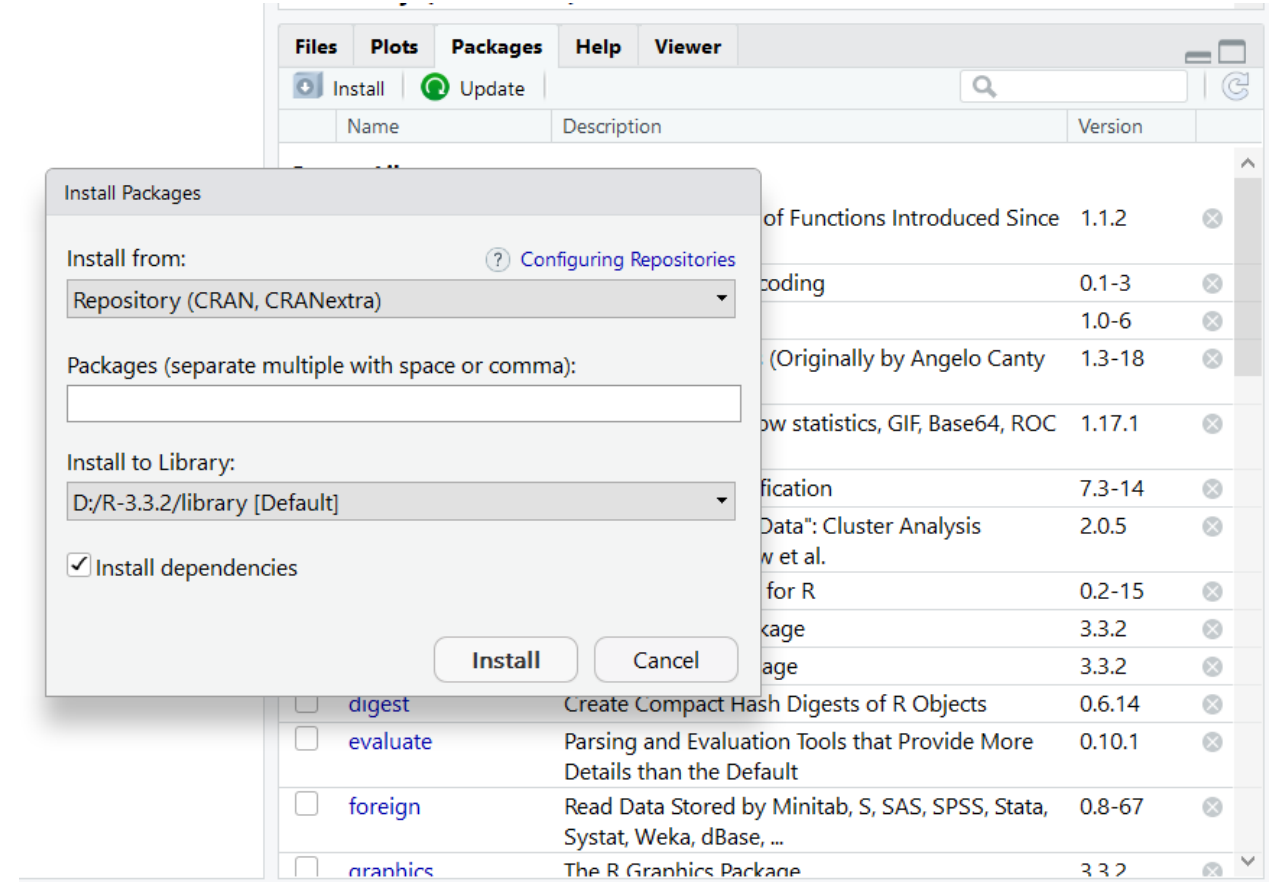
For additional info regarding functions and inbuilt datasets of R we can see the help tab or use `?functname` scripting window



RStudio

Packages tab:

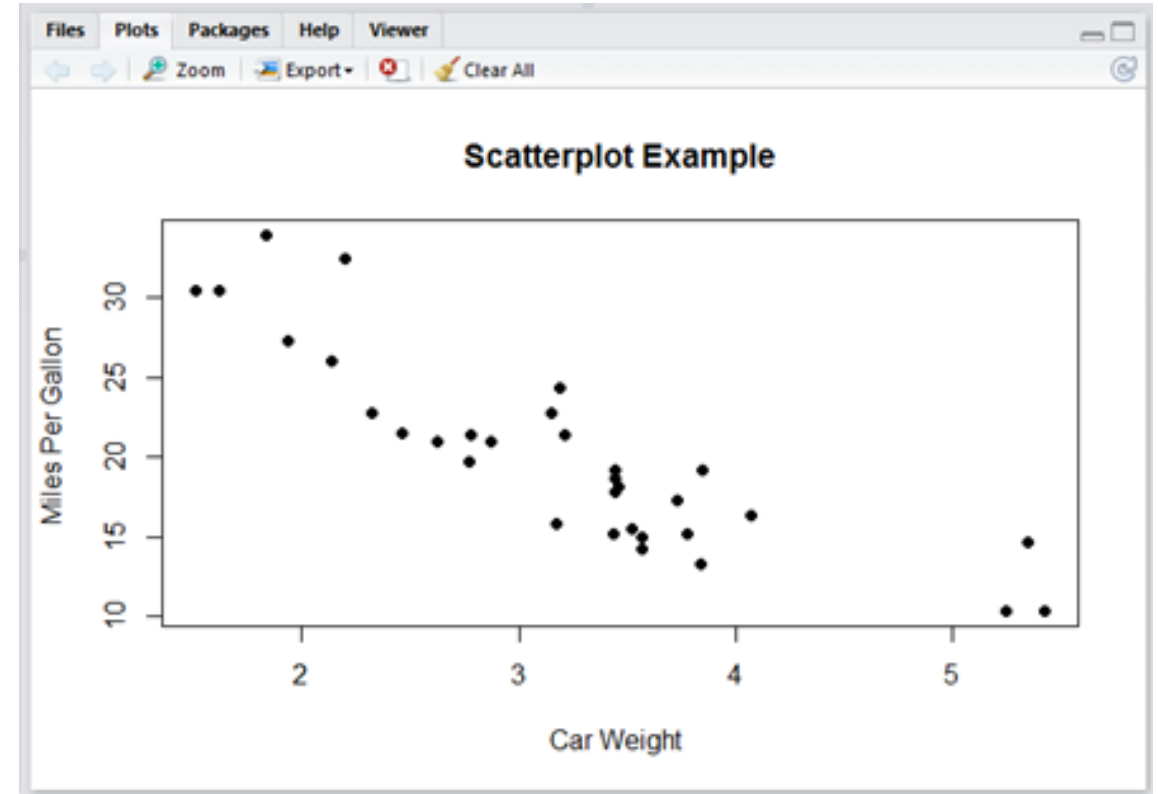
- ✓ Click on Install Packages tab and enter desired package name.
- ✓ Else, use console to install through command: `install.packages("packagename")`.
- ✓ The installed packages can be viewed in the list under install packages



RStudio

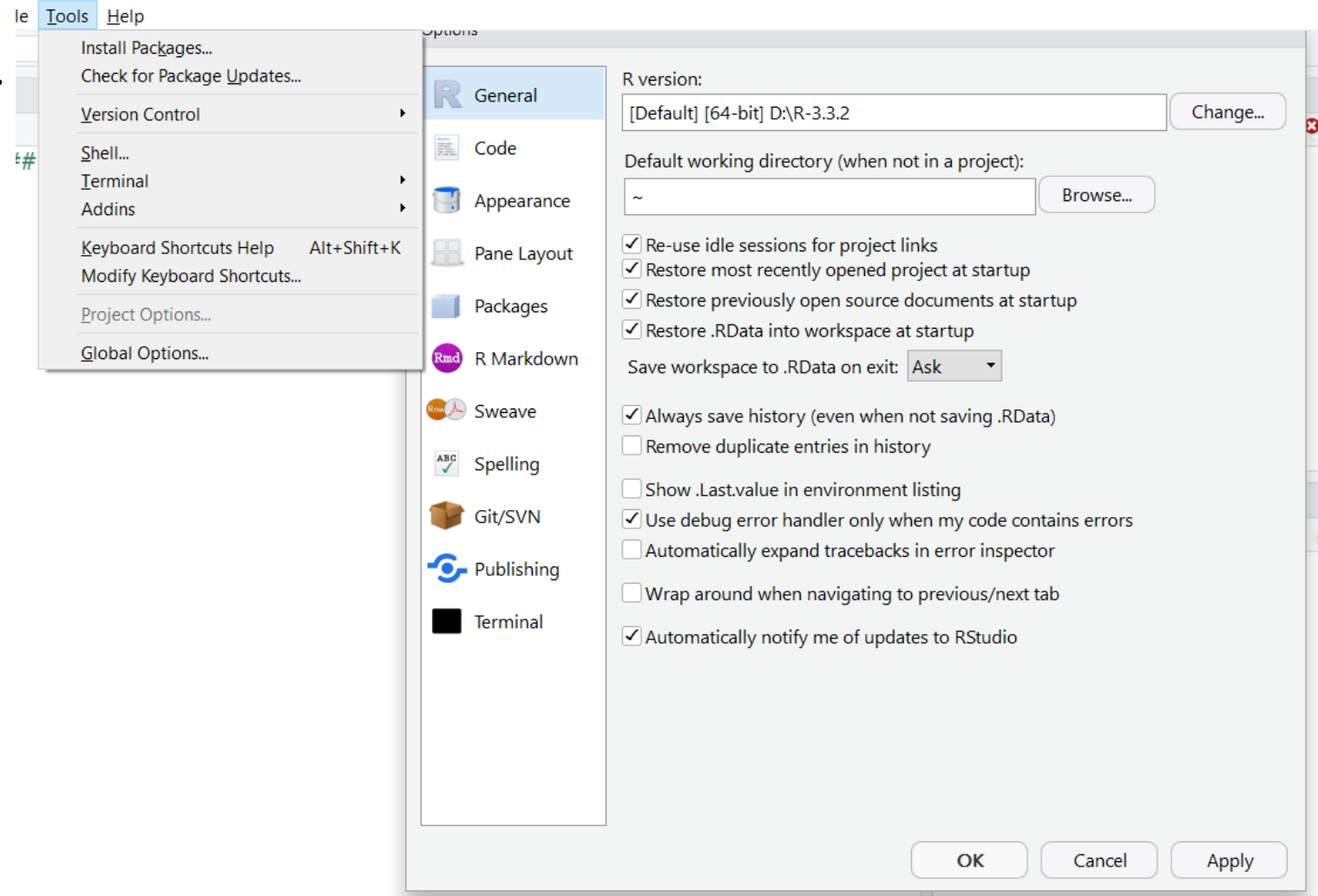
Plot Tab

- ✓ The plots tab will display the graphs.
- ✓ The plots will be created based on the script we have written.
- ✓ One can save graph as pdf or image by clicking on the Export Icon.



RStudio

- Using Tools -> Global Options :
- We can set default working directory.
- Change the look and feel of R Studio (Appearance Tab)
- Etc.....



Operators in R

- There are several number of operators in R programming
- List of operators in R:

Arithmetic Operators

- ✓ Addition
- ✓ Subtraction
- ✓ Multiplication
- ✓ Division
- ✓ Exponentiation
- ✓ Modulo

Relational Operators

- ✓ Greater than
- ✓ Greater than equal to
- ✓ Less than
- ✓ Less than equal to
- ✓ Equal to
- ✓ Not Equal to

Logical operators

- ✓ AND operator
- ✓ OR operator
- ✓ NOT operator
- ✓ Logical AND operator
- ✓ Logical OR operator

Assignment Operator

- There are two assignment operators i.e. <- or =.
- This operator means that “x gets a value”
- Create a scaler constant x with value 2, we type
- Difference between <- and = operator

<-	=
The <- operator can be used anywhere.	The = operator is only allowed at the top level.
For example, if we assign 2 -> a is perfectly valid.	For Example, if we assign 2 = a is confusing and raises an error.

```
#Assigning numeric value to
x #
x <- 2
x
## [1] 2

x = 2
x
## [1] 2

# Assigning character value
to x #

x <- "IMS"
x
## [1] "IMS"

x = "IMS"
x
## [1] "IMS"
```

Arithmetic Operators

- R can be used as a simple calculator. Consider the following arithmetic operators:

- Addition (+): It adds two numbers

```
# Arithmetic addition operator #
a <- 5
b <- 6
a + b

## [1] 11
```

- Subtraction(-): It Subtracts two numbers.

```
# Arithmetic subtraction operator #
a <- 5
b <- 6
a - b

## [1] -1
```

- Multiplication(*): It multiplies two numbers
- Example:

```
# Arithmetic multiplication operator #
a <- 5
b <- 6
a * b

## [1] 30
```

Arithmetic Operator

- Division(/): It divide first number with second
- Exponentiation(^): The first number raised to the exponent(power) of second number.
- Modulo(%): Gives the remainder of first number with second

```
# Arithmetic division operator #
a <- 5
b <- 6
a / b

## [1] 0.8333333
```

```
# Arithmetic exponentiation operator #
a <- 5
b <- 6
a^b

## [1] 15625
```

```
# Arithmetic modulo operator #
a <- 5
b <- 6
a%%b

## [1] 5
```

Relational Operators

- Greater than (>): It compares the two number whether the first number is greater than second.
- Greater than equal to (>=): It compares the two number whether the first number is greater than or equal to the second.
- Less than (<): It compares the two number whether the first number is less than second.

```
# Relational greater than Operator#
a <- 5
b <- 6
a > b

## [1] FALSE
```

```
# Relational greater than equal to Operator#
a <- 5
b <- 6
a >= b

## [1] FALSE
```

```
# Relational less than Operator#
a <- 5
b <- 6
a < b

## [1] TRUE
```

Relational Operator

- Less than equal to (<=): It compares the two number whether the first number is less than equal to second.
- Equal to(==): Check whether the first number is equal to the second
- Not Equal to(!=): Check whether the first number is not equal to the second

Relational less than equal to Operator#

a <- 5

b <- 6

a <= b

[1] TRUE

Relational equal to Operator#

a <- 5

b <- 6

a == b

[1] FALSE

Relational not equal to Operator#

a <- 5

b <- 6

a != b

[1] TRUE

Logical Operators

- It is applicable to logical, numeric or complex vectors only.
- Element Wise Logical AND or AND operator(&): It combines 1st number with the corresponding 2nd number and gives a output TRUE if both the numbers are TRUE.
- Element Wise Logical OR/OR operator(|): It combines 1st number with the corresponding 2nd number and gives a output TRUE if one of the numbers are TRUE.

```
# Check AND operator with z = 1 #
z <- 1
(z > 2) & (z > 5)
## [1] FALSE
```

```
# Check AND operator with z = 5 #
z <- 5
(z > 4) & (z < 7)
## [1] TRUE
```

```
# Check OR operator with z = 1 #
z <- 1
(z > 2) | (z > 5)
## [1] FALSE
```

```
# Check OR operator with z = 5 #
z <- 5
(z > 4) | (z < 7)
## [1] TRUE
```

Logical Operator

- Logical Not (!): It takes each number and gives the opposite logical value.
- Logical AND(&&): It takes both numbers and gives the TRUE only if both are TRUE.
- Logical OR(||): It takes first element of both the vectors and gives the TRUE if one of them is TRUE.

```
# Check NOT operator
with z = 1 #
z <- 1
!((z > 2) & (z > 5))
## [1] TRUE
```

```
# Check Logical AND
operator with z = 1
#
z <- 1
(z > 2) && (z > 5)
## [1] FALSE
```

```
# Check Logical AND
operator with z = 5
#
z <- 5
(z > 4) && (z < 7)
## [1] TRUE
```

```
# Check Logical OR
operator with z = 1
#
z <- 1
(z > 2) || (z > 5)
## [1] FALSE
```

```
# Check Logical OR
operator with z = 5
#
z <- 5
(z > 4) || (z < 7)
## [1] TRUE
```


Different Data Types

- Character - A character object is used to represent string values
- Numeric (Real / Decimal) - Decimal values are called numeric. even if we assign an integer to a variable a, it is saved as a numeric value.
- Integer - An integer is a whole number that can be positive, negative, or zero.
- L tells R to store this as an integer.

```
# character data types #
a <- "IMS"
#print class of a #
print(class(a))
## [1] "character"
```

```
# Numeric data types #
a <- 15.4
#print class of a #
print(class(a))
## [1] "numeric"
```

```
# Integer data types #
a <- 2L
#print class of a #
print(class(a))
## [1] "integer"
```

Different Data Types (Contd.)

- Logical - A **logical** value is created via comparison between variables.
- Complex - A **complex** number is defined as a pure imaginary value i .

```
# Logical data types #
a <- TRUE
# print class of a #
print(class(a))
## [1] "logical"
```

```
# complex data types #
a <- 4+5i
# print class of a #
print(class(a))
## [1] "complex"
```

as.function() and is.function()

as.function()

- as.function is a generic function which is used to convert objects to functions.
- This function is used to convert a type of the data which is useful to complete code.

is.function()

- is.function() checks whether its argument is a (primitive) function.

Syntax:

as.function(x,...)

Where, x = object to convert,
a list for the default method.

Syntax:

is.function(x)

Where x is an R object

```
# write a numeric vector #
x <- c(1,5,7,6,2,4,8)
x

## [1] 1 5 7 6 2 4 8

# check the type of the vector #
is.numeric(x)

## [1] TRUE

# Convert a numeric vector to factor #
as.factor(x)

## [1] 1 5 7 6 2 4 8
## Levels: 1 2 4 5 6 7 8
```

as.function() and is.function()

- Write a character vector of length 5
- is.character() check the type of the vector, the answer of this is TRUE which states that the vector is character.
- To convert a character vector to factor we use as.factor() which creates a levels.

```
# Write a character vector #  
y <- c("BA", "FM", "CFA", "CIMA", "DS")  
# check the type of the vector #  
is.character(y)  
## [1] TRUE  
  
# Convert a character vector to factor #  
as.factor(y)  
## [1] BA    FM    CFA   CIMA  DS  
## Levels: BA CFA CIMA DS FM
```

Data Structures

- Different types of data structure
 - ✓ Vector
 - ✓ Matrix
 - ✓ Data Frame
 - ✓ List

Vector

- A vector is a sequence of similar data elements i.e., integer, double, logical, complex, etc.
- A vector is an object that contains a set of values called its elements
- The concatenation function **c()** job is to combine multiple values into a single vector.
- All the values of workshop are numeric, so the vector's mode is **numeric**.
- If one value were alphabetic (character or string) then the mode would be coerced, or forced, to be **character**.
- Always put character in `" "`.
- R uses *NA (Not Available)* to represent missing values.

```
# Assigning a numeric vector to workshop #
workshop <- c(1,2,1,2,1,2,1,2)
workshop

## [1] 1 2 1 2 1 2 1 2

class(workshop)

## [1] "numeric"
```

```
# Assigning a character vector to gender including NA #
gender <- c("f","f","f",NA,"m","m","m","m")
gender

## [1] "f" "f" "f" NA "m" "m" "m" "m"

class(gender)

## [1] "character"
```

Vector Operations

- Continuous sequence of numbers can be generated using the colon(:) operator.
- A vector has only one dimension i.e. either a row or a column.
- In a mixed datatype vector, if character is present in a vector then it will take it as character vector.

```
#Numeric vector of continuous sequence #
```

```
x = c(1:5)
```

```
x
```

```
## [1] 1 2 3 4 5
```

```
#Numeric vector of continuous sequence along with additional numbers#
```

```
y=c(1:5,10,20)
```

```
y
```

```
## [1] 1 2 3 4 5 10 20
```

```
# class of mixed datatype vector #
```

```
a <- c(1,4,2,"a",3+5i,TRUE)
```

```
a
```

```
## [1] "1" "4" "2" "a" "3+5i" "TRUE"
```

```
class(a)
```

```
## [1] "character"
```

Vector Operations

- Vectors can be combined via the function `c()`.
- `rep()` function replicates vector again and again.
- In Example, `times = 2` means the sequence 1:3 will replicate twice.

```
# Combining Vectors #
x = c(1:5)
x
## [1] 1 2 3 4 5
z = c("aa", "bb", "cc", "dd", "ee")
z
## [1] "aa" "bb" "cc" "dd" "ee"
a <- c(x, z)
a
## [1] "1" "2" "3" "4" "5" "aa" "bb" "cc"
"dd" "ee"
```

```
# Vector replication #
rep(1:3, times = 2)
## [1] 1 2 3 1 2 3
```


Vector Arithmetic

- We can perform the normal arithmetic operations on the vectors
- If we multiply a vector x by 5 then we get a vector with each of its members multiplied by 5.
- If we subtract x from 10 then each element in x is subtracted from 10.
- If we add 15 to x then 15 is added to each element in x.

```
#Create x vector #  
x = c(1:5)  
# Multiply x by 5 #  
5*x  
## [1] 5 10 15 20 25  
# Subtract x from 10 #  
10-x  
## [1] 9 8 7 6 5  
# Add 15 to x #  
15+x  
## [1] 16 17 18 19 20
```

Vector Arithmetic

```
# Arithmetic Operations using vector #
a <- c(1,5,9)
b <- c(3,7,11)
# addition operator #
a + b

## [1] 4 12 20

# Subtraction operator #
a - b

## [1] -2 -2 -2

# Multiplication operator #
a * b

## [1] 3 35 99
```

```
# Division operator #
a/b

## [1] 0.3333333 0.7142857 0.8181818

# Exponentiation operator #
a^b

## [1] 1 78125 31381059609

# Modulo operator #
a%%b

## [1] 1 5 9
```

Relational Operators

```
# Relational Operations using vector #  
a <- c(1.5,3.8,6.7)  
b <- c(2.1,8.9,4.1)  
# Greater than Operator #  
a > b  
## [1] FALSE FALSE TRUE  
# Greater than equal to Operator #  
a >= b  
## [1] FALSE FALSE TRUE  
# Less than Operator #  
a < b  
## [1] TRUE TRUE FALSE
```

```
# Less than equal to Operator #  
a <= b  
## [1] TRUE TRUE FALSE  
# Equal to Operator #  
a == b  
## [1] FALSE FALSE FALSE  
# Not equal to Operator #  
a != b  
## [1] TRUE TRUE TRUE
```

Logical Operations

```
# Logical Operations using vector #  
a <- c(-1.5, 3.8, 6.7, TRUE)  
b <- c(2.1, 8.9, -4.1, FALSE)  
# AND Operator #  
a&b  
## [1] TRUE TRUE TRUE FALSE  
# OR operator #  
a|b  
## [1] TRUE TRUE TRUE TRUE  
# NOT operator #  
!a  
## [1] FALSE FALSE FALSE FALSE
```

```
# Logical AND Operator #  
a&&b  
## [1] TRUE  
# Logical OR Operator #  
a||b  
## [1] TRUE
```

Vector Index

- In order to access the values in the vectors we need to use the box brackets ([])

```
s=c("aa","bb","cc","dd","ee")
#Access third value in vector
s[3]
## [1] "cc"

#Remove third value in vector
s[-3]
## [1] "aa" "bb" "dd" "ee"

#Select a value out of the vector length #
s[10]
## [1] NA
```

```
#Select multiple values in the vector
s[c(2,3)]
## [1] "bb" "cc"

#Repeat vector values
s[c(2,3,3)]
## [1] "bb" "cc" "cc"

#Select a range of values in the vector
s[c(2:5)]
## [1] "bb" "cc" "dd" "ee"
```

Matrix

- A matrix is a two-dimensional data object consist of rows and columns.
- The values must be of the same mode, i.e. all numeric, all character, or all logical.

Syntax:

```
Matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
```

#Matrix#

```
mymatrix <- matrix(c(1,1,5,1,2,1,4,1,2,2,4,3,3,
NA,3,4), nrow =4, ncol = 4, byrow = TRUE)
```

mymatrix

##		[,1]	[,2]	[,3]	[,4]
##	[1,]	1	1	5	1
##	[2,]	2	1	4	1
##	[3,]	2	2	4	3
##	[4,]	3	NA	3	4

Matrix

- To access the elements of a matrix would require two index values, one for the rows and one for the columns
- The index values are represented as row and columns combination, with the first value as rows and second value as columns separated by comma(,)
- The elements of mth rows and nth columns can be accessed by specifying the values in this format M[r,c]. where M is the matrix object, r is the row number and c is the column number.
- For Example, access the element of 4th row and 2nd column.

```
# Element of 4th row and 2nd column is accessed  
#  
mymatrix[4,2]  
## [1] NA
```

Matrix

- In matrix, we can define the number of rows and columns using a concatenation function in matrix function.
- Extract a particular number of rows and all column using `mymatrix[c(m1,m2,m3....),]`
- In a example, extract the 1st and 3rd row and all columns using matrix and concatenation function.
- To extract the entire m^{th} row of matrix use `mymatrix[m,]`.
- For example,extract 4th row and all columns.
- To extract the entire n^{th} column of matrix use `mymatrix[,]`.
- For example, extract 3rd row and all columns.

```
# Element of 1st and 3rd row and all columns are accessed #
mymatrix[c(1,3),]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    1    5    1
## [2,]    2    2    4    3
```

```
# Element of 4th row and all columns are accessed#
mymatrix[4,]
```

```
## [1]  3 NA  3  4
```

```
# Elements of all rows and 3rd column is accessed#
mymatrix[,3]
```

```
## [1] 5 4 4 3
```


Matrix

- The values returned are vector and all vector operations can be performed on them.
- Any object can be coerced to a matrix using the `as.matrix` function.
- To check if the object is a matrix use the `is.matrix` function which would return a TRUE value if the object is a matrix.

Syntax:
`is.matrix(mymatrix)`

Matrix

- We can use the `dim()` function to find out the dimensions of a matrix
- To add a column in a matrix use a `cbind()` function
- It would add a column as the last column of the matrix with the sequence 1 to 4 as its values
- While using `cbind` the length of the column to be added should be same as the number of rows in the matrix
- Similarly `rbind` can be used to add rows.
- It would add a row as the last row of the matrix with the sequence 1 to 5 as its values

Syntax:
`dim(mymatrix)`

Dimensions of matrix

```
dim(mymatrix)
```

```
## [1] 4 4
```

Adding columns in matrix

```
mymatrix1<- cbind(mymatrix, c(1:4))
```

```
mymatrix1
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    1    5    1    1
## [2,]    2    1    4    1    2
## [3,]    2    2    4    3    3
## [4,]    3   NA    3    4    4
```

Adding rows in matrix

```
mymatrix2<- rbind(mymatrix1, c(1:5))
```

```
mymatrix2
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    1    5    1    1
## [2,]    2    1    4    1    2
## [3,]    2    2    4    3    3
## [4,]    3   NA    3    4    4
## [5,]    1    2    3    4    5
```

Matrix

- The index values used in a matrix can also be negative values.
- The negative values would imply that the value corresponding to that index position will not be shown or deleted

```
# Element of 3rd row and 2nd column is deleted#
mymatrix[-3,-2]

##      [,1] [,2] [,3]
## [1,]    1    5    1
## [2,]    2    4    1
## [3,]    3    3    4
```

Matrix

- We can also give names to the rows and columns using the "dimnames" function
- Access elements of a matrix using rows and column names.

```
# Give names to the matrix #
dimnames(mymatrix) <- list(c("row1", "row2", "row3", "row4"), c("col1", "col2", "col3", "col4"))
mymatrix
```

```
##           col1 col2 col3 col4
## row1         1     1     5     1
## row2         2     1     4     1
## row3         2     2     4     3
## row4         3    NA     3     4
```

```
# Access element of rows using row and column names of matrix #
mymatrix["row2", "col4"]

## [1] 1
```

Matrix

- The matrix can be transposed using the function "t". Transpose converts rows to columns and vice versa

Transpose of matrix

t(mymatrix)

##		row1	row2	row3	row4
##	col1	1	2	2	3
##	col2	1	1	2	NA
##	col3	5	4	4	3
##	col4	1	1	3	4

Data frame

- A data frame object in R has similar dimensional properties to a matrix, but may contain categorical as well as numeric data.
- A data frame is very much like a matrix, except it is designed for storing statistical or experimental data.
- Each row represents a unit, and each column a collection of measurements on the units. It is very similar to tables in excel.
- A data frame is basically a list of vectors of equal lengths.

Syntax:

```
dfName = data.frame(name1 = v1, name2 = v2, .....)
```

Where, v1 and v2 may be vectors or matrices, but length of the vector and number of rows of the matrix must be the same.

```
#Create a vectors of all types #
n=c(2,3,5)
s=c("aa","bb","cc")
b=c(TRUE,FALSE,TRUE)
#create a dataframe usig vectors #
df=data.frame(n,s,b)
df

##      n  s      b
## 1  2 aa  TRUE
## 2  3 bb FALSE
## 3  5 cc  TRUE
```

List

- A list is a special type of vector.
- It is used to store combinations of any other objects, even other lists
- A list can contain many lists nested inside.
- Elements in the list are indexed by double brackets[[]].
- Single bracket[] will still return another list.
- Although you can use a list to store any R object, they are most often used to store data in various forms, results from functions such as regression equation parameters, and sets of arguments to control functions.

Syntax:

```
mylist <- list(workshop, rp_seq, mymatrix)
```

Lists

- The objects in a list are also accessed by using the indices as explained earlier

```
# Create a list of numeric types #
my_list <- list(1,4,8,9)
my_list

## [[1]]
## [1] 1
##
## [[2]]
## [1] 4
##
## [[3]]
## [1] 8
##
## [[4]]
## [1] 9
```

```
# create a list with different data types #
x = list(1, "a", TRUE, c(1:8))
x

## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 1 2 3 4 5 6 7 8
```


list on iris dataset

- Use inbuilt iris dataset
- List is used to show the subset of the data

```
# Create a List on iris data #
list(iris[1:5,1:2],iris[11:17,3:4],iris[30:36,1:4])

## [[1]]
##      Sepal.Length Sepal.Width
## 1           5.1         3.5
## 2           4.9         3.0
## 3           4.7         3.2
## 4           4.6         3.1
## 5           5.0         3.6
##
## [[2]]
##      Petal.Length Petal.Width
## 11           1.5         0.2
## 12           1.6         0.2
## 13           1.4         0.1
## 14           1.1         0.1
## 15           1.2         0.2
## 16           1.5         0.4
## 17           1.3         0.4
##
## [[3]]
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## 30           4.7         3.2         1.6         0.2
## 31           4.8         3.1         1.6         0.2
## 32           5.4         3.4         1.5         0.4
## 33           5.2         4.1         1.5         0.1
## 34           5.5         4.2         1.4         0.2
## 35           4.9         3.1         1.5         0.2
## 36           5.0         3.2         1.2         0.2
```

Working Directory

- getwd() – getwd() function is used to find out the current working directory
- setwd() – setwd() function is used to set the working directory, while pasting the path use / instead of \.

```
# Used to get a current working directory #  
getwd()
```

```
## [1] "C:/Users/User02/Desktop"
```

```
# Used to set a working directory #  
setwd("C:/Users/User02/Desktop")
```

Read and Write file

- `read.csv("path_name")` – This function is used to read file.
- `write.csv(file name, file = "(new file name).csv")` – This function is used to extract the csv file.

```
# Used to read a .csv file #
urban_pop <- read.csv("urbanpop.csv")
```

```
# Used to extract a .csv file #
write.csv(urban_pop, file = "Mydata.csv")
```



Urbanpop.csv

- **Note: Open urbanpop.csv file attached here and save it in .csv(comma delimited) format. In coming slides we have used urban_pop to explain some useful functions**

Useful Functions in R

- "length" function is used to determine the size of the vector
- "paste" function is used to concatenate vectors
- "mode" is used to get or set the type or storage mode of an object.

```
# Gives the length of the vector #
y=c("aa","bb","cc","dd","ee")
length(y)

## [1] 5
```

```
# used to concatenate vectors after converting to character#
fname="Joe"
lname="Smith"
paste(fname, lname)

## [1] "Joe Smith"
```

```
#Mode#
#Get or set the type or storage mode of an object#
x <- c(1,5,4,7,2,1,5,3,1)
y <- mode(x)
y

## [1] "numeric"
```

Useful Functions in R

- The **which()** function will return the position of the elements(i.e., row number/column number/array index) in a logical vector which are TRUE.
- There are some inbuilt characters in R like letters – Print small alphabets
- LETTERS – Print capital alphabets

#Returns the position of the element #
letters

```
## [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
      "k" "l" "m" "n" "o" "p" "q"
## [18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
which(letters=="s")
```

```
## [1] 19
```

```
z <- c(6,5,-3,7)
```

```
which(z*z > 9)
```

```
## [1] 1 2 4
```

Useful Functions in R

- `order()` function is used to sort a data frame in R.
- If we use `order` function without giving vector it will show us position of the numbers.
- But, in 2nd code we are getting actual numbers.

```
# Order #
x1 <- c(1,5,4,7,2,1,5,3,1)
# This will give the position of the number #
order(x1, decreasing = FALSE)
## [1] 1 6 9 5 8 3 2 7 4

# This will give the actual number from the vector #
x1[order(x1, decreasing = FALSE)]
## [1] 1 1 1 2 3 4 5 5 7
```

Useful Functions in R

- Merge two data frames by common columns or row names.
- Here, we are merging 2 data frames using story id.

```
# merge #
# Make a data frame mapping story numbers to titles
story <- read.table(header=TRUE, text='
    storyid  title
    1        lions
    2        tigers
    3        bears')
# Make another data frame with the data and story numbers
data <- read.table(header=TRUE, text='
    subject storyid rating
    4        1      6.7
    4        2      4.5
    4        1      3.7
    3        2      3.3
    1        2      4.1
    2        1      5.2')
# Merge the two data frames
merge(story, data, "storyid")

##   storyid  title subject rating
## 1      1  lions      4      6.7
## 2      1  lions      4      3.7
## 3      1  lions      2      5.2
## 4      2 tigers      4      4.5
## 5      2 tigers      3      3.3
## 6      2 tigers      1      4.1
```

Useful functions in R

- “head()” is used to get a first 5 observations in a data

Shows first 5 observations of each column

head(urban_pop)

```
##          country  X1960      X1961      X1962      X1963      X1964
## 1  Afghanistan 769308 814923.049 858521.698 903913.86 951225.94
## 2    Albania 494443 511802.780 529438.851 547376.75 565571.75
## 3    Algeria 3293999 3515147.548 3739963.007 3973289.13 4220987.01
## 4 American Samoa      NA   13660.298   14165.797   14758.93   15396.42
## 5    Andorra      NA    8723.921    9700.346   10748.38   11865.86
## 6    Angola 521205 548265.046 579695.370 612086.70 645261.59
##          X1965      X1966
## 1 1000582.35 1058743.47
## 2  583982.89  602512.17
## 3 4488175.64 4649105.24
## 4   16044.82   16693.11
## 5   13052.75   14216.81
## 6  679109.12  717833.40
```


Useful functions in R

- “tail()” is used to get a last 5 observations in a data

Shows last 5 observations of each column

tail(urban_pop)

##		country	X1960	X1961	X1962	X1963
## 204		Vietnam	5107221	5329816.33	5558754.7	5795307.61
## 205	Virgin Islands (U.S.)		18080	19722.55	21488.8	23436.02
## 206		Yemen	475203	494623.59	524818.5	556070.43
## 207		Zambia	551011	601586.61	653480.6	708251.74
## 208		Zimbabwe	472675	504363.25	537348.4	572056.07
## 209		South Sudan	NA	287795.80	292120.9	296594.19
##		X1964	X1965	X1966		
## 204		6040676.77	6295975.61	6574579.81		
## 205		25572.38	27842.76	30528.18		
## 206		588186.39	621007.03	658653.50		
## 207		765969.16	826729.58	903398.07		
## 208		608445.60	646509.12	692932.86		
## 209		301196.05	305915.63	310762.87		

Useful functions in R

- “summary()” is used to find out the 5 point summary of the data

```
# Shows 5 point summary of data #
summary(urban_pop)
```

##	country	X1960	X1961
##	Afghanistan : 1	Min. : 3378	Min. : 1028
##	Albania : 1	1st Qu.: 88978	1st Qu.: 70644
##	Algeria : 1	Median : 580675	Median : 570159
##	American Samoa: 1	Mean : 4988124	Mean : 4991613
##	Andorra : 1	3rd Qu.: 3077228	3rd Qu.: 2807280
##	Angola : 1	Max. :126469700	Max. :129268133
##	(Other) :203	NA's :11	
##	X1962	X1963	X1964
##	Min. : 1090	Min. : 1154	Min. : 1218
##	1st Qu.: 74974	1st Qu.: 81870	1st Qu.: 84953
##	Median : 593968	Median : 619331	Median : 645262
##	Mean : 5141592	Mean : 5303711	Mean : 5468966
##	3rd Qu.: 2948396	3rd Qu.: 3148941	3rd Qu.: 3296444
##	Max. :131974143	Max. :134599886	Max. :137205240

Useful functions in R

- “View()” is used to see the loaded dataset

```
View(urban_pop)
```

- “str” is used to find out the structure of the data

```
# Shows the structure of the data #
```

```
str(urban_pop)
```

```
## 'data.frame':    209 obs. of  8 variables:
```

```
## $ country: Factor w/ 209 levels "Afghanistan",...: 1 2 3 4 5 6 7 8 9 10 ...
```

```
## $ X1960   : int  769308 494443 3293999 NA NA 521205 21699 15224096 957974 24  
996 ...
```

```
## $ X1961   : num  814923 511803 3515148 13660 8724 ...
```

```
## $ X1962   : num  858522 529439 3739963 14166 9700 ...
```

```
## $ X1963   : num  903914 547377 3973289 14759 10748 ...
```

```
## $ X1964   : num  951226 565572 4220987 15396 11866 ...
```

```
## $ X1965   : num  1000582 583983 4488176 16045 13053 ...
```

```
## $ X1966   : num  1058743 602512 4649105 16693 14217 ...
```

Control Structure

- These allow you to control the flow of execution of a script typically inside of a function.
 1. if statement
 2. if-else statement
 3. If-else-if statement
 4. for loop
 5. While loop

if Statement

- If statement: It holds a logical or numeric value and executes the codes if it is TRUE.
- If the logical value is FALSE then it comes out of the loop.
- **Example:** Write a program to know the popularity of the show, if the number of views are more than 15 then print the show is popular otherwise no message will be displayed?

Syntax of if statement:

```
if(condition)
{
    Statement
}
```

How if loop will work?

1. Assign 17 to the num_views
2. Now, check the condition of “if” loop “Is 17 greater than 15”
3. If the condition is TRUE, it will go to loop and print “This show is popular!”
4. If the condition is FALSE, it will directly come out of the loop.

```
# Assign 17 to num_views #
num_views <- 17
# Apply “if” loop to check the number of views
greater than 15 #
if(num_views > 15) {
    print("This show is popular!")
}
## [1] "This show is popular!"
```

If-else statement

- The if-else statement is the if statement followed by an optional statement, executed only when the condition is FALSE.
- In If else loop if the condition is TRUE then the if loop will be executed.
- If the condition is FALSE then the else part of the loop will be executed
- **Example:** Write a program to know the popularity of the show, if the number of views are more than 15 then it should print “This show is popular”, else it should display “This show is not popular” message?

```
If(condition)
{
    statement 1
} else {
    statement 2
}
```

How if-else loop will work?

1. Assign value 14 to the num_views variable.
2. Now, check the condition that the value of variable whether “14” greater than “15”?
3. Here, the condition is FALSE
4. Hence, it will not complete “if” loop and directly execute “else” part of the code
5. And print “This show is not popular!”

```
# Assign 14 to num_views #
num_views <- 14
# Apply ifelse loop to check the number of view
s greater than 15 #
if (num_views > 15) {
    print("This show is popular!")
}else{
    print("This show is not popular!")
}
## [1] "This show is not popular!"
```

If else if

- The else if statement executes different codes for more than two conditions.
- In if else if loop every time the condition will be checked.
- If the condition is TRUE then that loop will be executed otherwise it will check the condition in the next loop and process goes on.
- **Example:** Write a loop to know the popularity of the show, if the “number of views are more than 15” then it will print “This show is popular” otherwise it will check condition in “else if” loop?

```
if (condition1) {
    Statement 1
} else if (condition2) {
    Statement 2
} else if (condition3) {
    Statement 3
} else {
    Statement 4}
```

How if else if loop will work?

1. Assign 13 to num_views.
2. Now, check the condition of if loop, “is 13 greater than 15”?
3. Here, The condition is FALSE
4. Now it will check the condition in “else if” loop “is 13 is greater than 10 and less than 15”?
5. Here, The condition is TRUE
6. Hence it will print “This show is average!”

```
# Assign 13 to num_views #
num_views = 13
if (num_views > 15) {
    print("This show is popular!")
} else if (num_views <= 15 & num_views > 10)
{
    print("This show is average!")
} else {
    print("This show is not popular!")
}

## [1] "This show is average!"
```

for loop

- A looping vector is assigned a value which is iterated in a loop for certain number of times. For each iteration the statement is evaluated.
- Example: We need to create a loop to find out how many values in the vector (2,5,3,9,8,11,6,4,8,9,1,3,4,6,7,5,21,12,13,14,10) have reminder “0” when we divide the values by 2.

Syntax of for loop

```
for (val in sequence)
{
  statement
}
```

How “for loop” will be executed?

1. Create a numeric vector
2. Initiate a count with 0
3. In “for” loop every time it will take value from x and check the condition in “if”, loop
4. If the condition is TRUE
5. It will increase the count by 1 otherwise it will check for next number in a vector
6. Every time loop will be iterated till the last value of the vector and finally it will print the value of count.

```
# Create a vector #
x <- c(2,5,3,9,8,11,6, 4,8,9,
1,3,4,6,7,5,21,12,13,14,10)
# initiate count with 0 #
count <- 0
for (i in x)
{
  if(i %% 2 == 0)
    count = count+1
}
print(count)

## [1] 10
```


while loop

- The while continues executing a block of codes until a specific condition is met.
- Example:- We want to reduce the over speeded vehicle by displaying “Slow Down” message to driver. The safety speed is 30km/hr. So when the speed of vehicle is more than 30, it display warning message until the speed is below 30km/hr. After each warning message driver reduce speed by 5km/hr.

```
while (condition)
{
    statement
}
```

while loop

- **How while loop will work?**
 1. Assign 84 to speed variable
 2. Check the condition in “while” loop “is 84 greater than 30”?.
 3. If the condition is TRUE. It will print “Slow down!”
 4. Now, decrease the speed by 5
 5. Again check the condition in “while” loop
 6. If the condition is TRUE it will again print the same message.
 7. Once the condition in while loop is FALSE.
 8. It will stop printing the message and prints the speed.

```
# Assign the cut off speed to speed variable #
speed <- 84
# apply while loop to check the condition #
while (speed > 30 ) {

    print("Slow down!")
    speed = speed-5
}

## [1] "Slow down!"
## [1] "Slow down!"
## [1] "Slow down!"
## [1] "Slow down!"
## [1] "Slow down!"
## [1] "Slow down!"
## [1] "Slow down!"
## [1] "Slow down!"
## [1] "Slow down!"
## [1] "Slow down!"
## [1] "Slow down!"

print(speed)

## [1] 29
```

Apply function

- Apply Functions Over Array Margins returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.
- In apply() function an array or matrix can be applied.
- For example, We need to find out the sum of the matrix using apply function?

Syntax:

apply(X, MARGIN, FUN, ...)

Where,

x = An array, including a matrix.

MARGIN = A vector giving the subscripts which the function will be applied over. 1 if row and 2 if column.

FUN = The function to be applied

```
#Apply functions #
# Take a matrix discussed in previous topic #
mymatrix
```

```
##      col1 col2 col3 col4
## row1    1    1    5    1
## row2    2    1    4    1
## row3    2    2    4    3
## row4    3   NA    3    4
```

```
# Use apply function to take sum of column #
colSum <- apply(mymatrix,2,sum)
colSum
```

```
## col1 col2 col3 col4
##    8   NA   16    9
```

lapply()

- lapply() function is used over A List Or Vector
- lapply returns a list of the same length as X, each element of which is the result of applying FUN to the corresponding element of X.
- For example, We need to find out the mean of the list which contain different types of data?

Syntax:

`lapply(X, FUN, ...)`

Where, x = a list or a vector

FUN = The function to be applied to each element of x

```
# create a list with different data types #
mylist = list(x = 1:15, y = c(TRUE,FALSE,TRUE,T
RUE,FALSE))
# lapply functions to take mean #
lapply(mylist, mean)

## $x
## [1] 8
##
## $y
## [1] 0.6
```

supply()

- supply() function is similar to lapply, but it returns the output as a vector instead of list
- The difference between lapply and supply is that lapply gives output as a list and supply gives output as a vector.
- Example: We need to find out the mean of the list which contain different types of data also print output in vector form?

Syntax:
supply(X, FUN,...)
Where,
X = vector or a list
FUN = Function to be applied
on x

```
# create a list with different data types #
mylist = list(x = 1:15, y = c(TRUE,FALSE,TRUE,
TRUE,FALSE))

# supply function to take mean #
supply(mylist, mean)

##      x      y
## 8.0 0.6
```

Summary of apply Functions

Function Name	Objects the Function Works On	What the Function Sees as Elements	Result Type
apply()	Matrix	Rows or columns	Vector, matrix, array, or list
	Array	Rows, columns, or any dimension	Vector, matrix, array, or list
	Data frame	Rows or columns	Vector, matrix, array, or list
lapply()	Vector	Elements	List
	Data frame	Variables	List
	List	Elements	List
sapply()	Vector	Elements	Vector, matrix, or list
	Data frame	Variables	Vector, matrix, or list
	List	Elements	Vector, matrix, or list

Packages

- A package is a bundle of code, data, documentation and tests
- It is easy to share with others.
- Packages are located in a repository so that we can install it from there.
- The most popular repositories for R packages are CRAN, Bioconductor and Github
- To install a package in R
- To load a installed package

Syntax to install a package in R
`install.packages("package")`

Syntax to load a installed package
`library(package)`

```
install.packages("foreign")
warning in install.packages : dependency 'utils' is not available trying URL
'https://cran.rstudio.com/bin/windows/contrib/3.3/foreign_0.8-69.zip' Content type 'application/zip' length 305546
bytes (298 KB) downloaded 298 KB package 'foreign' successfully unpacked and MD5 sums checked The downloaded binary
packages are in C:\Users\User02\AppData\Local\Temp\RtmpOkoORR\downloaded_packages
library(foreign)
```

ggplot2()

- ggplot is enriched with customized features to make your visualization better and better.
- It becomes even more powerful when grouped with other packages like cowplot, gridExtra.
- At least plot these 3 graphs: Box Plot, Bar Plot, Histogram

```
# ggplot2 #
library(ggplot2)
library(gridExtra)
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##      combine

# Load a inbuilt dataset #
df <- ToothGrowth
# convert variable to factor #
df$dose <- as.factor(df$dose)
# Show the first five observations of the data #
head(df)

##      len  supp dose
## 1   4.2    VC  0.5
## 2  11.5    VC  0.5
## 3   7.3    VC  0.5
## 4   5.8    VC  0.5
## 5   6.4    VC  0.5
## 6  10.0    VC  0.5
```


Boxplot

- To create a boxplot use a ggplot() function

Syntax:

ggplot(data = NULL, mapping = aes(), ...,
environment = parent.frame())

Where,

data: Default dataset to use for plot

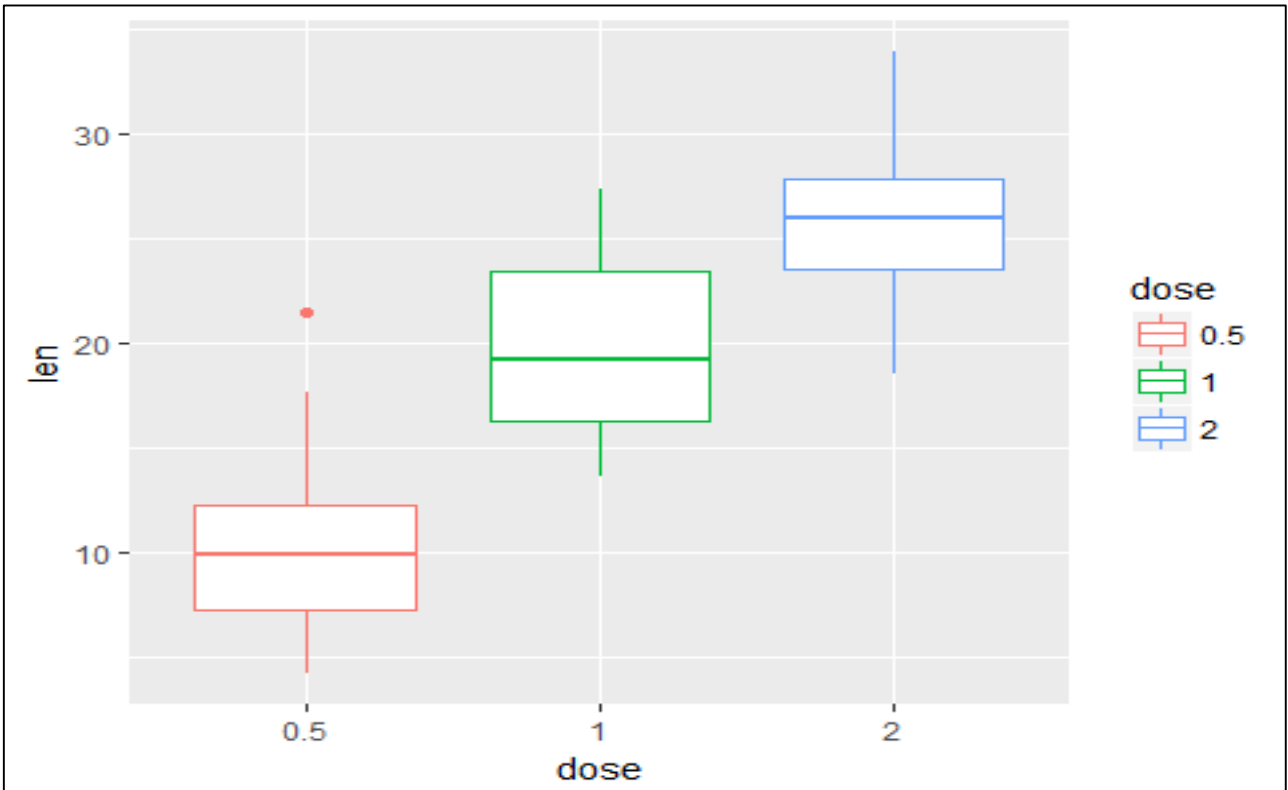
Mapping: Default list of aesthetic mappings to
use for plot.

... : Other arguments passed on to methods

environment: If an variable defined in the
aesthetic mapping is not found in the data,
ggplot will look for it in this environment.

It defaults in environment in which ggplot() is
called.

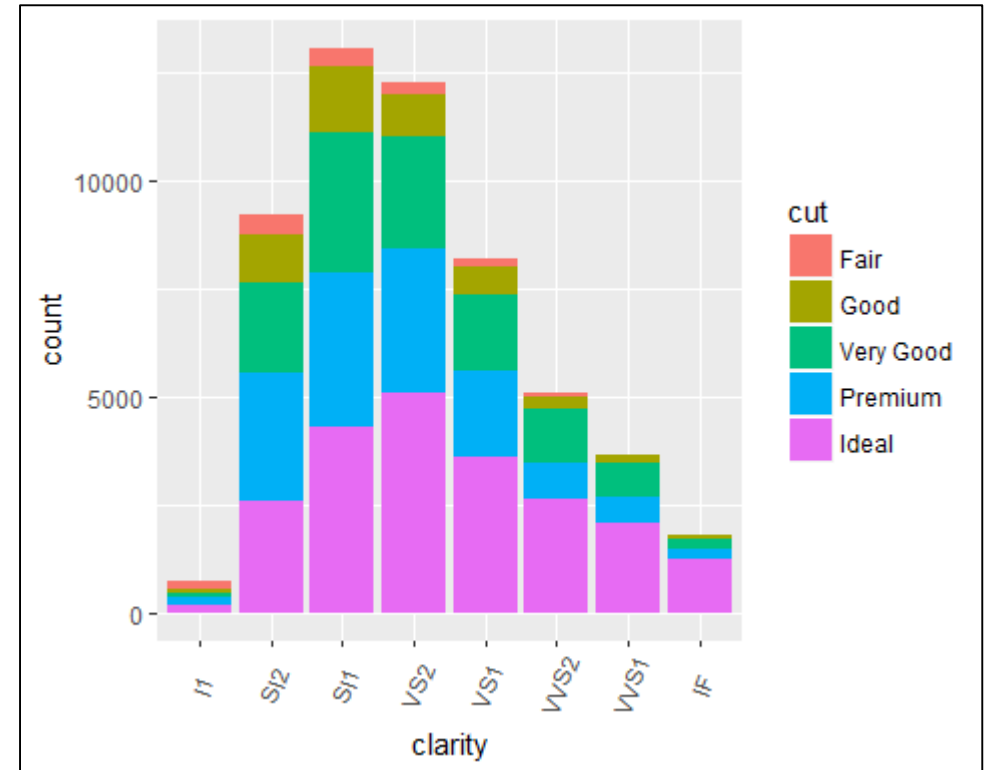
```
# plot a boxplot using ggplot function #
model<- ggplot(df, aes(x = dose, y = len, color
= dose)) + geom_boxplot()
model
```



Barplot

- The same ggplot function is used to draw a barplot

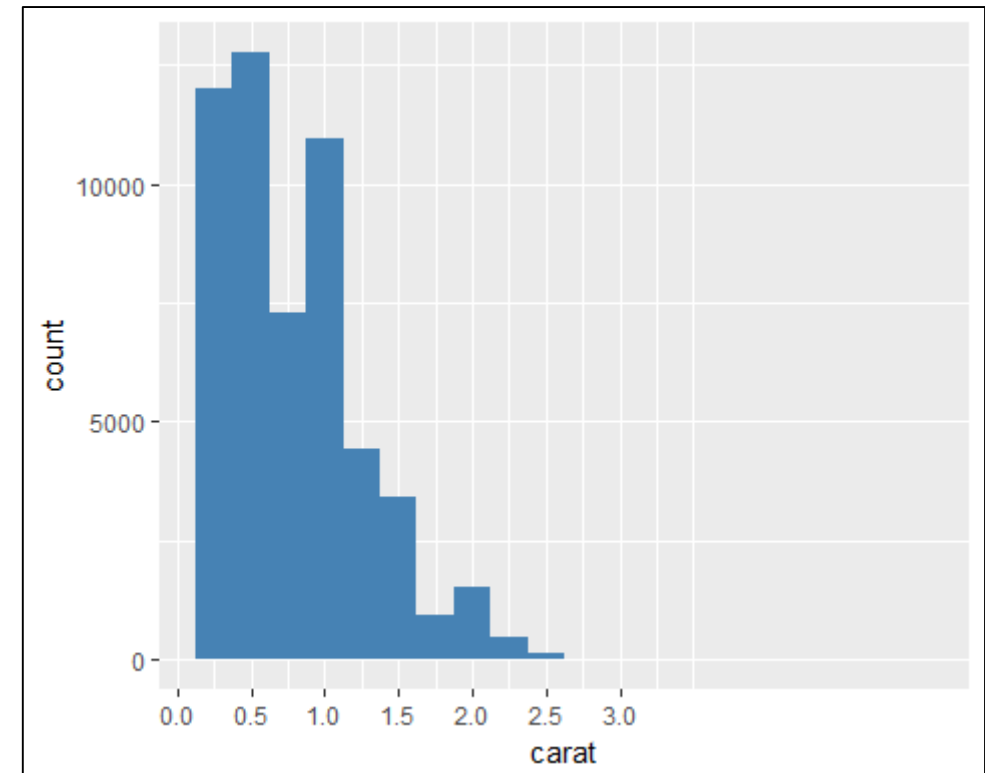
```
#Plot a barplot using ggplot function #
bp <- ggplot(diamonds, aes(clarity, fill = cut))
  + geom_bar() + theme(axis.text.x = element_text(
    angle = 70, vjust = 0.5))
bp
```



Histogram

- The same ggplot function is used to draw a Histogram

```
# Plot a histogram using ggplot function #
ggplot(diamonds, aes(x = carat)) + geom_histogram(
  binwidth = 0.25, fill = 'steelblue') + scale_x_
_continuous(breaks=seq(0,3, by=0.5))
```



dplyr()

- dplyr is a powerful R package to transform and summarize tabular data with rows and columns
- The dplyr package is more useful because it contains a set of functions that perform data manipulation operations such as filtering for rows, selecting specific columns, re-ordering rows, adding new columns and summarizing data.
- Function in dplyr package are easier compared with base R function i.e. split(), subset(), apply(), sapply(), lapply(), tapply() and aggregate()
- Functions in dplyr package:

Functions	Description
select()	select columns
filter()	filter rows
arrange()	re-order or arrange rows

Install dplyr and datasets package

- To use the function in dplyr package, Install it and load it using library() function
- Install and load “datasets” package to use the inbuilt dataset i.e. mtcars
- We have used in build dataset “mtcars” available in database package

```
#install dplyr package #
install.packages("dplyr")
#Load dplyr and datasets package #
library(dplyr)

##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
:
##
##      filter, lag
## The following objects are masked from 'package:base':
##
##      intersect, setdiff, setequal, union

library(datasets)
# Load inbuilt dataset mtcars using data() function #
data(mtcars)
# View Loaded dataset using View() function #
View(mtcars)
```

Example of functions in dplyr package

- Use select function in dplyr package to select cyl and drat column in data
- It will select only cyl and drat column in mtcars dataset.

```
# Apply select function from dplyr package #
select_mtcars <- select(mtcars, cyl, drat)
select_mtcars
```

```
##              cyl  drat
## Mazda RX4      6  3.90
## Mazda RX4 Wag  6  3.90
## Datsun 710      4  3.85
## Hornet 4 Drive  6  3.08
## Hornet Sportabout 8  3.15
## Valiant         6  2.76
## Duster 360      8  3.21
## Merc 240D       4  3.69
## Merc 230        4  3.92
## Merc 280        6  3.92
## Merc 280C       6  3.92
## Merc 450SE      8  3.07
## Merc 450SL      8  3.07
....  ....
.....
```

filter() function in dplyr package

- Filter function in dplyr package will filter the column based on the condition.
- Apply filter on mpg column in mtcars.
- filter observations having mpg greater than 20.

```
# Apply filter function from dplyr package #
filter(mtcars, mpg > 20)
```

##	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
## 1	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
## 2	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
## 3	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
## 4	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
## 5	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
## 6	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
## 7	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
## 8	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
## 9	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
## 10	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
## 11	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
## 12	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
## 13	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
## 14	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

arrange() function in dplyr package

- arrange() function in dplyr package used to arrange data in ascending order.
- Observations in mpg column is sorted in ascending order then internally wt is sorted with mpg.

Apply arrange function from dplyr package

arrange(mtcars, mpg, wt)

##		mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
## 1		10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
## 2		10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
## 3		13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
## 4		14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
## 5		14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
## 6		15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
## 7		15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
## 8		15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
## 9		15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
## 10		15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
## 11		16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
## 12		17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
## 13		17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
## 14		18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
## 15		18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
...

readr

- readr package is used to read various forms of data into R.
- readr package is written by Hadley Wickham
- This is fast, easy to use and consistent.
- In this package it returns a tibble which is a super charge version of a dataframe.
- It also conveniently shows the column classes.
- In this package character are never converted to factors so no more stringsAsFactors = FALSE
- This package can replace the traditional read.csv() and read.table() base R functions from utils package.
- Utils package is slower as compared to readr package
- Readr package is used to read Delimited files with read_csv().

readr

- Install and load readr package

```
# readr package #  
# install readr package #  
install.packages("readr")  
# Load readr package #  
library(readr)
```

readr

- Read a urbanpop dataset using read_csv() function
- In the output of read_csv() function it shows the class of each column

```
# Load urbanpop dataset using read_csv #  
read_csv("urbanpop.csv")  
  
## Parsed with column specification:  
## cols(  
##   country = col_character(),  
##   `1960` = col_integer(),  
##   `1961` = col_double(),  
##   `1962` = col_double(),  
##   `1963` = col_double(),  
##   `1964` = col_double(),  
##   `1965` = col_double(),  
##   `1966` = col_double()  
## )
```

readr

- In continuation of the above output,
- A tibble: 209 x 8 means that the data has 209 rows and 8 columns

```
## # A tibble: 209 x 8
##           country `1960`      `1961`      `1962`      `1963`
##           <chr>    <int>      <dbl>      <dbl>      <dbl>
## 1 Afghanistan  769308  814923.049  858521.698  903913.86
## 2 Albania      494443  511802.780  529438.851  547376.75
## 3 Algeria     3293999  3515147.548  3739963.007  3973289.13
## 4 American Samoa      NA    13660.298   14165.797   14758.93
## 5 Andorra       NA     8723.921    9700.346   10748.38
## 6 Angola       521205  548265.046  579695.370  612086.70
## 7 Antigua and Barbuda  21699   21635.051   21664.200   21740.74
## 8 Argentina  15224096 15545222.590 15912120.020 16282345.35
## 9 Armenia     957974  1008597.321  1061426.399  1115612.32
## 10 Aruba      24996   28139.757   28532.729   28763.12
## # ... with 199 more rows, and 3 more variables: `1964` <dbl>,
## #   `1965` <dbl>, `1966` <dbl>
```