An introduction R

Saneesh

2024-03-01

Contents

1 About **5**

1.1 How to use this book 5

1.2 Installation 6

1.3 Project 6

2 R operators **11**

2.1 Arithmetic operators 11

2.2 Functions 12

2.3 Reserved words 13

2.4 Variables 13

2.5 Documenting code 14

2.6 Relational / comparison operators in R 14

2.7 Data types 15

3 Installing R packages **17**

3.1 Package 17

4 Vectors & matrices **21**

4.1 Numeric vector 21

4.2 Mathematical operation 22

4.3 Indexing 22

4.4 Numeric matrices 23

4.5 Names for vectors, matrices and arrays 23

4.6 Characters or strings 24

5 List & Data frame 27

5.1 Making a data frame 27

6 Names functions 29

6.1 Loops and maps 30

7 Import data 31

7.1 Importing from .csv file 31

7.2 Read .csv 32

8 Data Visualization 35

8.1 Scatter plot 35

8.2 Box plot 38

9 Wordcloud 43

9.1 Wordcloud from data frame 43

9.2 Wordcloud from text data 47

10 Probability 53

10.1 create vectors 53

10.2 make a data frame, name it I gave df= data frame 53

10.3 calculating total sightings 54

10.4 Calculate the probability of seeing each species 54

Chapter 1

About

This is book is an introduction to R using `tidyverse` packages (Figure 1.1) and wrote using the `bookdown` package.

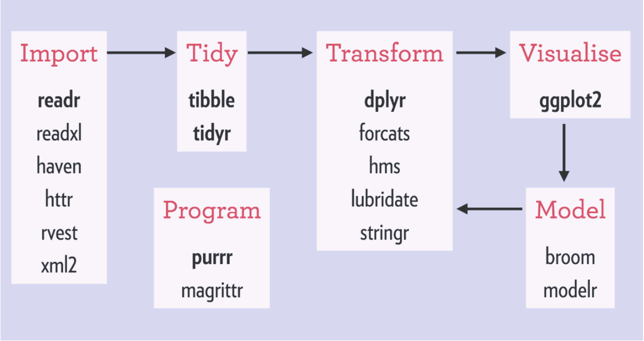


Figure 1.1: Tidyverse packages

1.1 How to use this book

Any text see inside the `grey` box is a `code`, you can copy that and paste in your `R` script.

```
sum(1+2)
```

```
## [1] 3
```

Any text or line starts with `## [1] 3` is an output of the code.

1.2 Installation

Download and install R. Follow the link below and choose the appropriate version.

Link: R-Windows
Link: R-Mac
Link: R-Ubuntu
Download and install RStudio. RStudio is an interface which facilitates coding in R. We recommend RStudio Desktop.

Link: RStudio

1.3 Project

RStudio projects make it straightforward to divide your work into multiple contexts, each with their own working directory, workspace, history, and source documents. 1. Make a `New Folder`

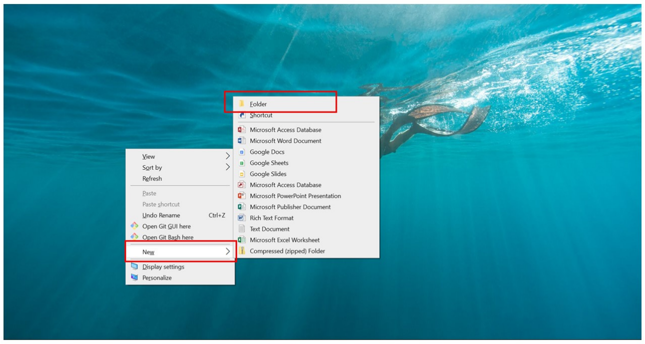


Figure 1.2: New Folder

E.g, I made a folder `TISS_R`

2. Open `RStudio` and start a `New Project`

Or from `File` > `New Project` Because we have a folder made with name `TISS_R` I am going to select `Existing Directory` if you don't have a folder select `New Directory` and select a location. The name of the directory or the folder will be the new project's name.

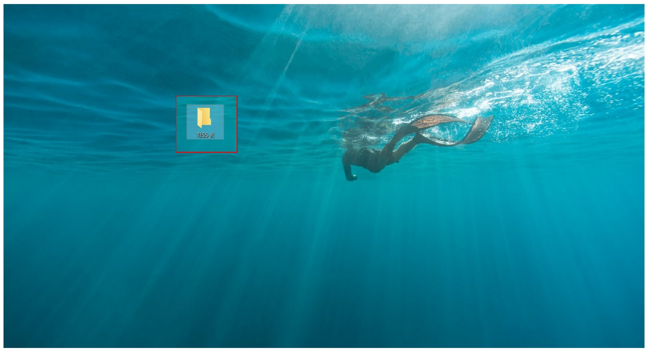
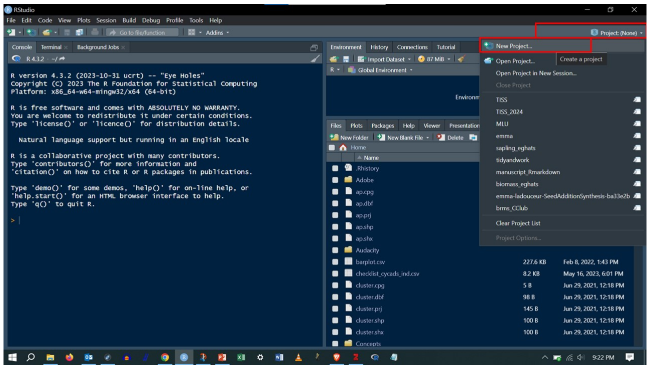


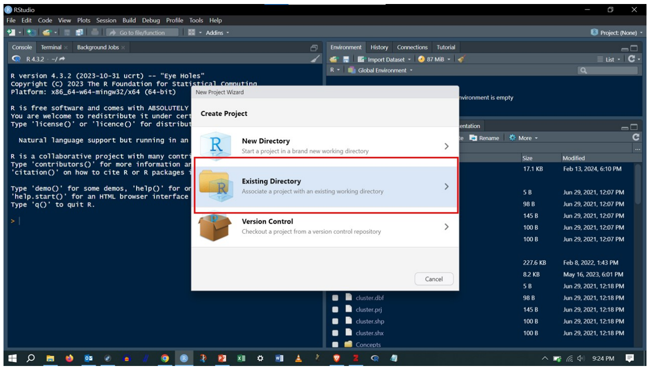
Figure 1.3: New Folder



```
R version 4.3.2 (2023-10-31 ucrt) -- "Eye Holes"
Copyright (C) 2023 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.

  Natural language support but running in an English locale

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.

>
```

barplot.csv	227.6 KB	Feb 8, 2022, 1:43 PM
checklist_cycads_ind.csv	8.2 KB	May 16, 2023, 6:01 PM
cluster.cpg	5 B	Jun 29, 2021, 12:18 PM
cluster.dbf	98 B	Jun 29, 2021, 12:18 PM
cluster.prj	145 B	Jun 29, 2021, 12:18 PM
cluster.shp	100 B	Jun 29, 2021, 12:18 PM
cluster.shx	100 B	Jun 29, 2021, 12:18 PM

Figure 1.4: New Project



```
R version 4.3.2 (2023-10-31 ucrt) -- "Eye Holes"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY
You are welcome to redistribute it under cert
Type 'license()' or 'licence()' for distribut

  Natural language support but running in an

R is a collaborative project with many contri
Type 'contributors()' for more information ar
'citation()' on how to cite R or R packages i

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

>
```

New Project Wizard

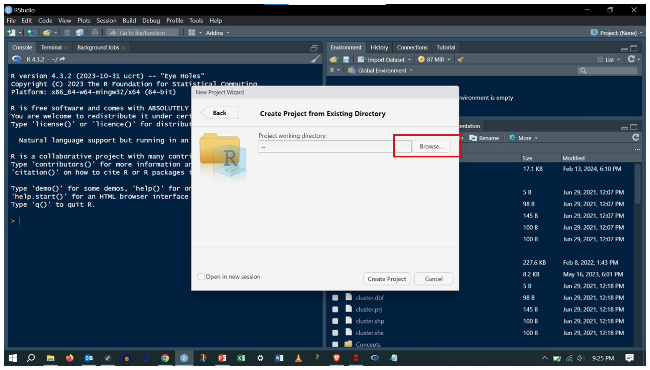
Create Project

New Directory
Start a project in a brand new working directory

Existing Directory
Associate a project with an existing working directory

Version Control
Checkout a project from a version control repository

Figure 1.5: Existing Directory



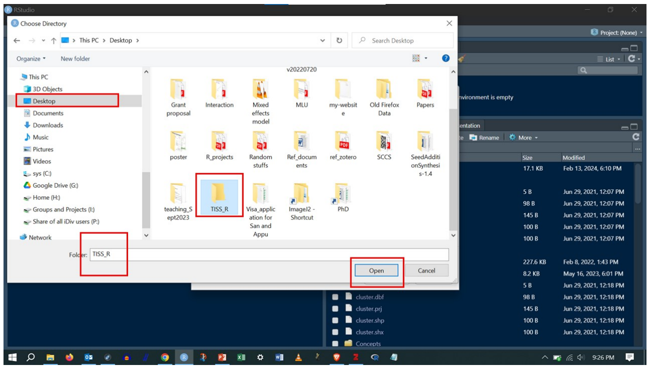
```
R version 4.3.2 (2023-10-31 ucrt) -- "Eye Holes"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY
You are welcome to redistribute it under cert
Type 'license()' or 'licence()' for distribut

  Natural language support but running in an

R is a collaborative project with many contri
Type 'contributors()' for more information ar
'citation()' on how to cite R or R packages i

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

> |
```

New Project Wizard

Create Project from Existing Directory

Project working directory: ~

Figure 1.6: Create

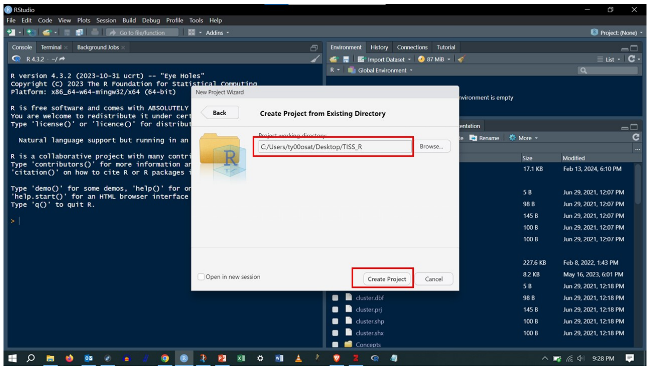


Choose Directory

This PC › Desktop

Folder: TISS_R

Figure 1.7: Open



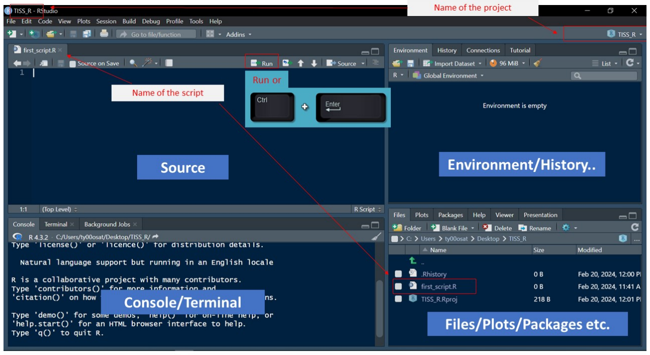
```
R version 4.3.2 (2023-10-31 ucrt) -- "Eye Holes"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY
You are welcome to redistribute it under cert
Type 'license()' or 'licence()' for distribut

  Natural language support but running in an

R is a collaborative project with many contri
Type 'contributors()' for more information ar
'citation()' on how to cite R or R packages i

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

> |
```

New Project Wizard

Create Project from Existing Directory

Project working directory: C:/Users/ty00osat/Desktop/TISS_R

Figure 1.8: Create Project



```
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how                              ns.

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.
```

Name	Size	Modified
..		
.Rhistory	0 B	Feb 20, 2024, 12:00 Pl
first_script.R	0 B	Feb 20, 2024, 11:41 A
TISS_R.Rproj	218 B	Feb 20, 2024, 12:01 Pl

Figure 1.9: Done: New Project

Chapter 2

R operators

2.1 Arithmetic operators

The R arithmetic operators allows us to do math operations, like sums, divisions or multiplications, among others. The following table summarizes all base R arithmetic operators.

Arithmetic operators
+ Plus
- Minus
* Multiplication
/ Division
^ Exponential
** Exponential

Basic operations

```
3 + 5   # 8
```

```
## [1] 8
```

```
8 - 3   # 5
```

```
## [1] 5
```

```
7 * 5   # 35
```

```
## [1] 35
```

```
1/2     # 0.5
```

```
## [1] 0.5
```

```
4 ^ 4   # 256
```

```
## [1] 256
```

```
4 ** 4  # 256
```

```
## [1] 256
```

```
5 %% 3  # 2
```

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

```
5 %/% 3 # 1
```

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

2.2 Functions

R has many built-in functions. The most common situation is that the function is called by its name using prefix notation, followed by round brackets that enclose the function's arguments (separated by commas if there are multiple arguments). For example, the function round takes a number and, by default, returns the closest integer:

```
# the function `round` takes a number as an argument and
# returns the closest integer (default)
round(0.6213)
```

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

`round` allows several arguments. It takes as input the number `x` to be rounded and another integer number digits which gives the number of `digits` after the comma to which `x` should be rounded. We can then specify these arguments in a function call of round by providing the named arguments.

```
round(0.6213, digits = 2)
```

```
## [1] 0.62
```

Example of another function, `sum`

```
sum(1, 2, 3)
```

```
## [1] 6
```

2.3 Reserved words

There are also reserved words you can’t use, like `TRUE`, `FALSE`, `NULL`, among others. You can see the full list of R reserved words typing `help(Reserved)` or `?Reserved`. It not advisable to use these words outside quotes (’ ’, “ “) or backtick (‘) quotes.

2.4 Variables

You can assign values to variables using three assignment operators:
`->`
`<-`
`=`

```
a <- 6       # assigns 6 to variable a
7 -> b       # assigns 7 to variable b
c = 3        # assigns 3 to variable c, use of = is not widly accepted!
d <-  a * b / c   # assigns 6 * 7 / 3 = 14 to variable d
d           # returns d
```

```
## [1] 14
```

Use `print()` function to see the output

```
print(a)
print(7)
print(c)
print(d)
```

2.5 Documenting code

It is good practice to document code with short but informative comments. Comments in R are demarcated with `#`.

```
round(             # call the function `round`
  sum(2.213*3.123),# find sum of 2.213*3.123
  digits = 2       # show the sum with two digits after the `.`
)
```

```
## [1] 6.91
```

In RStudio, you can use Command+Shift+C (on Mac) and Ctrl+Shift+C (on Windows/Linux) to comment or uncomment code, and you can use comments to structure your scripts. Any comment followed by —- is treated as a (foldable) section in RStudio.

```
# SECTION: variable assignments ----
x <- 6
y <- 7
```

```
# SECTION: some calculations ----

x * y
```

```
## [1] 42
```

2.6 Relational / comparison operators in R

Comparison or relational operators are designed to compare objects and the output of these comparisons are of type boolean. To clarify, the following table summarizes the R relational operators.

Relational operators in R
`>` Greater than
`<` Lower than
`>=` Greater or equal than
`<=` Lower or equal than
`==` Equal to
`!=` Not equal to

```
3 > 5  # FALSE
```

```
## [1] FALSE
```

```
3 < 5  # TRUE
```

```
## [1] TRUE
```

```
3 >= 5 # FALSE
```

```
## [1] FALSE
```

```
3 <= 5 # TRUE
```

```
## [1] TRUE
```

```
3 == 5 # FALSE
```

```
## [1] FALSE
```

```
3 != 5 # TRUE
```

```
## [1] TRUE
```

2.7 Data types

To learn a programming language entails to first learn something about what kinds of objects we will have to deal with. Let's therefore briefly go through the data types. We can assess the type of an object stored in variable `x` with the function `typeof(x)`.

```
x <- 1
typeof(x)
```

```
## [1] "double"
```

```
x <- "hello"
typeof(x)
```

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

```
x <- TRUE
typeof(x)
```

```
## [1] "logical"
```

```
x <- mean
typeof(x)
```

```
## [1] "closure"
```

It is possible to cast objects of one type into another type using functions `as.` in base R.

```
x <- as.numeric(1) # "double"
typeof(x)
```

```
## [1] "double"
```

```
x <- as.character(1) # "character"
typeof(x)
```

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

Chapter 3

Installing R packages

3.1 Package

An R package is a library of functions that have been developed to cover some needs or specific scientific methods that are not implemented in base R.

The Comprehensive R Archive Network (CRAN) is the official R packages repository, with thousands of free R packages available. Most of them have been developed by Data Scientists, Statisticians, Professors and researchers.

First, you need to look for the name of the package you want to install. You may want to research for your topic googling something like: 'graphics package R' or 'R package for time series'.

Once you decided what package to install, just call the `install.packages` function with the name of the package inside the parenthesis `()` with quotation marks `" "` or `' '`. As an example, we are going to install the `calendR` package, that allows creating monthly and yearly calendars, but you can install the package you prefer.

```
# install.packages('calendR')
```

Once you see a similar looking message in the Console "The downloaded binary packages are in"drive user downloaded_packages", it is good idea to add a comment `#` before install.packages like this `# install.packages('calendR')` to avoid installing the same package again and again!

After installation, you need to load the package if you want to access its functions. For that purpose, you can load it with the `library` function, specifying the package name with or without quotation marks.

```
library(calendR)
```

```
## ~~ Package calendR
## Visit https://r-coder.com/ for R tutorials ~~
```

Once loaded, you can use ? or the help function with the package name or the name of any function to see the documentation. You will also find useful examples to understand how the package works.

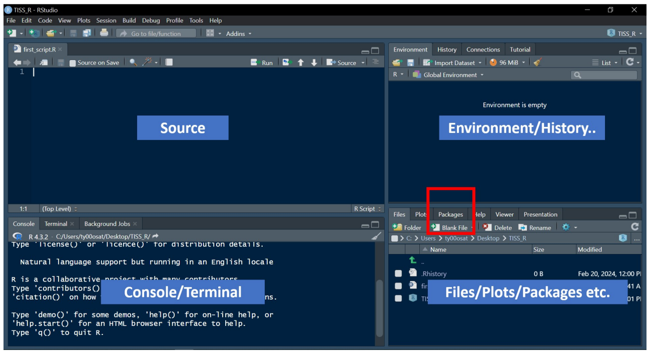
```
?calendR
help("calendR")
help(calendR)
```

In addition, you can find out where the packages are going to be installed calling the .libPaths() function.

```
.libPaths()
```

```
## [1] "C:/Users/ty00osat/Documents/R-4.3.2/library"
```

Alternatively, you can install R packages from the menu (Package).



```
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()
'citation()' on how                                    ns.

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.
```

TISS_R - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
first_script.R
Source on Save
Run
Source
Source
1:1
(Top Level)
R Script
Environment History Connections Tutorial
Import Dataset
96 MiB
List
Global Environment
Environment is empty
Environment/History..
Console Terminal Background Jobs
R 4.3.2 · C:/Users/ty00osat/Desktop/TISS_R/
Console/Terminal
Files Plots Packages Help Viewer Presentation
Folder
Blank File
Delete
Rename
C: > Users > ty00osat > Desktop > TISS_R
Name
Size
Modified
.Rhistory
0 B
Feb 20, 2024, 12:00 PM
Files/Plots/Packages etc.

Figure 3.1: Package

In RStudio go to Tools → Install Packages and in the Install from option select Repository (CRAN) and then specify the packages you want.

If you need to install several packages at once without writing the same function over and over again, you can make use of the c function within the install.packages function. Note that now the quotation marks are needed to specify the packages names.

```
install.packages(c("ggplot2", "dplyr"))
```

Once installed, you can get a list of all the functions in the package. If the package is on CRAN, you will find documentation in PDF format of all functions inside a page like https://cran.r-project.org/web/packages/package_name. Recall you can access this documentation in HTML format with the help function.

```
help(package = ggplot2)
```

Chapter 4

Vectors & matrices

R is essentially an array-based language. Arrays are arbitrary but finite-dimensional matrices.
* vectors = one-dimensional arrays
* matrices = two-dimensional arrays
* arrays = more-than-two-dimensional, it is important to keep in mind that arrays can contain objects of other types than numeric information (as long as all objects in the array are of the same type).

4.1 Numeric vector

```
x <- 1
print(x)
```

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

Vectors in general can be declared with the built-in function `c()`. To memorize this, think of concatenation or combination.

There are also helpful functions to generate sequences of numbers:

```
x <- seq(1:10)
x # or
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
print(x)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

4.2 Mathematical operation

Every mathematical operation can be expected to apply to a vector.

```
a <- c(2, 4, 6, 8)
a+1
```

```
## [1] 3 5 7 9
```

```
a*4
```

```
## [1]  8 16 24 32
```

```
a/0.5
```

```
## [1]  4  8 12 16
```

```
b <- c(8, 6, 4, 2)

a+b
```

```
## [1] 10 10 10 10
```

```
a*b
```

```
## [1] 16 24 24 16
```

4.3 Indexing

Indexing in R starts with 1, not 0.

```
x <- c(2, 4, 6, 8)   # this is now a 4-place vector
x
```

```
## [1] 2 4 6 8
```

```
x[2] # what is the entry in position 2 of the vector x?
```

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

```
x[1] # what is the entry in position 1 of the vector x?
```

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

4.4 Numeric matrices

```
x <- seq(1:10)
m1 <- matrix(x)
m1
```

```
##       [,1]
##  [1,]    1
##  [2,]    2
##  [3,]    3
##  [4,]    4
##  [5,]    5
##  [6,]    6
##  [7,]    7
##  [8,]    8
##  [9,]    9
## [10,]   10
```

```
m <- matrix(1:6, ncol = 3)
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

4.5 Names for vectors, matrices and arrays

The positions in a vector can be given names. These names of vector `x` positions are retrieved and set by the `names` function.

```
students <- c("Prachi", "Rishi", "Soumya")  # names of students
grades <- c(1.3, 2.7, 2.0)                 # a vector of grades

names(grades) <- students  # assign names
names(grades)              # retrieve names again: names assigned
```

```
## [1] "Prachi" "Rishi"  "Soumya"
```

4.6 Characters or strings

Strings are called characters in R. We will be stubborn and call them strings for most of the time here. We can assign a string value to a variable by putting the string in double-quotes.

```
institute <- 'TISS'
typeof(institute)
```

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

```
print(institute)
```

```
## [1] "TISS"
```

Create vectors of characters

```
chr_vector <- c("small", "medium", "large")
print(chr_vector)
```

```
## [1] "small"  "medium" "large"
```

```
chr_vector
```

```
## [1] "small"  "medium" "large"
```

```
typeof(chr_vector)
```

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

Factors are special vectors, which treat their elements as instances of a finite set of categories.

```
factor_vector <- as.factor(c("small", "medium", "large"))
print(factor_vector)
```

```
## [1] small  medium large
## Levels: large medium small
```

```
factor_vector
```

```
## [1] small  medium large
## Levels: large medium small
```

```
typeof(factor_vector)
```

```
## [1] "integer"
```

```
levels(factor_vector) # alphabetically ordered
```

```
## [1] "large"  "medium" "small"
```

A factor in R is a data structure used to represent a vector as categorical data. Therefore, the factor object takes a bounded number of different values called levels. Factors are very useful when working with character columns of data frames, for creating barplots and creating statistical summaries for categorical variables.

For plotting or other representational purposes, it can help to manually specify an ordering on the `levels` of a factor using the levels argument.

```
factor(factor_vector, levels = c('large', 'medium', 'small'))
```

```
## [1] small  medium large
## Levels: large medium small
```

```
levels(factor_vector) # ordered the way we want
```

```
## [1] "large"  "medium" "small"
```

```
days <- c("Friday", "Tuesday", "Thursday", "Monday", "Wednesday", "Monday",
          "Wednesday", "Monday", "Monday", "Wednesday", "Sunday", "Saturday")

# Levels in alphabetical order
my_factor <- factor(days)
my_factor
```

```
##  [1] Friday    Tuesday   Thursday  Monday    Wednesday Monday    Wednesday
##  [8] Monday    Monday    Wednesday Sunday    Saturday
## Levels: Friday Monday Saturday Sunday Thursday Tuesday Wednesday
```

Convert numeric to factor in R.
Suppose you have registered the birth city of six individuals with the following codification: 1. Mumbai 2. Chennai 3. Bangalore 3. Cochin
Hence, you will have something like the following data stored in a numeric vector:

```
# Sample data
city <- c(3, 2, 1, 4, 3, 2)
```

Now, you can call the `factor` to convert the data into factor and get it categorized for further analysis.

```
my_factor <- factor(city)
my_factor
```

```
## [1] 3 2 1 4 3 2
## Levels: 1 2 3 4
```

If the input vector is numeric, as in the previous section, the corresponding label (the city) is not reflected. In order to solve this issue, you can store the data in a factor object using the `factor` function and indicate the corresponding `labels` of the levels in the labels argument, in order to rename the factor `levels`.

```
# Setting the labels in the corresponding order
factor_cities <- factor(city, labels = c("Mumbai", "Chennai", "Bangalore", "Cochin"))

# Print the result
factor_cities
```

```
## [1] Bangalore Chennai   Mumbai    Cochin    Bangalore Chennai
## Levels: Mumbai Chennai Bangalore Cochin
```

Chapter 5

List & Data frame

Lists are very important in R because almost all structured data that belongs together is stored as lists. Objects are special kinds of lists. Data is stored in special kinds of lists, so-called data frames or so-called tibbles.

A data frame is base R's standard format to store data in. A data frame is a list of vectors of equal length. Data sets are instantiated with the function `data.frame`.

5.1 Making a data frame

```
name = c("saneesh", "sanusha", "appu", "jaru")
sex = c(2, 0, 5, 8)

df <- data.frame(name, sex)

df <- # df= data frame
  data.frame(name = c("saneesh", "sanusha", "appu", "jaru"),
             sex = c(2, 0, 5, 8))
```

Chapter 6

Names functions

The special operator supplied by base R to create new function is the keyword `function`. Here is an example of defining a new function with two input variables `x` and `y` that returns a computation based on these numbers. We assign a newly created function to the variable cool_function so that we can use this name to call the function later. Notice that the use of the return keyword is optional here. If it is left out, the evaluation of the last line is returned.

```
# define a new function
# - takes two numbers x & y as argument
# - returns x + y

sum_function <- function(x, y) {
  z = x + y
  return(z)
}

sum_function(x = 1, y = 4)
```

```
## [1] 5
```

```
sum_function(12, 4)
```

```
## [1] 16
```

```
# define a new function
# we know x and total we want to calculate percentage

percentage_function <- function(x, y) {
  percentage = (x / y)*100
  return(percentage)
}

percentage_function (360, 600)
```

```
## [1] 60
```

```
# we know percentage and total we want to calculate x

reverse_percentage_function <- function(x, y) {
  percentage = (x / 100)*y
  return(percentage)
}

reverse_percentage_function (60, 600)
```

```
## [1] 360
```

6.1 Loops and maps

Chapter 7

Import data

7.1 Importing from .csv file

Although creating data frames from existing data structures is extremely useful, by far the most common approach is to create a data frame by importing data from an external file. To do this, you'll need to have your data correctly formatted and saved in a file format that R is able to recognize. Fortunately for us, R is able to recognize a wide variety of file formats, although in reality you'll probably end up only using two or three regularly.
Saving files to import The easiest method of creating a data file to import into R is to enter your data into a spreadsheet using either Microsoft Excel or LibreOffice Calc and save the spreadsheet as a comma delimited file. To save a spreadsheet as a comma delimited file in Microsoft Excel or LibreOffice Calc select File -> Save as … from the main menu. You will need to specify the location (project folder) you want to save your file in the 'Save in folder' option and the name of the file in the 'Name' option. In the drop down menu located above the 'Save' button change the default 'Save as type:' to 'CSV(Comma delimited) (*.csv)'.

Once you've saved your data file in a suitable format we can now read this file into R.

If you don't have a `*.csv` don't worry, we will make one using the `write.csv` function.

```
df <- # df= data frame
  data.frame(name = c("saneesh", "sanusha", "appu", "jaru"),
             sex = c(2, 0, 5, 8),
             height = c(167, 158, 110, 50))

# copy the name of the data frame `df`
```

```r
write.csv(df, # copied name
          file= 'family.csv', # a new file will be made in the project folder with nam
          row.names = FALSE) # to avoid auto generated row names
```

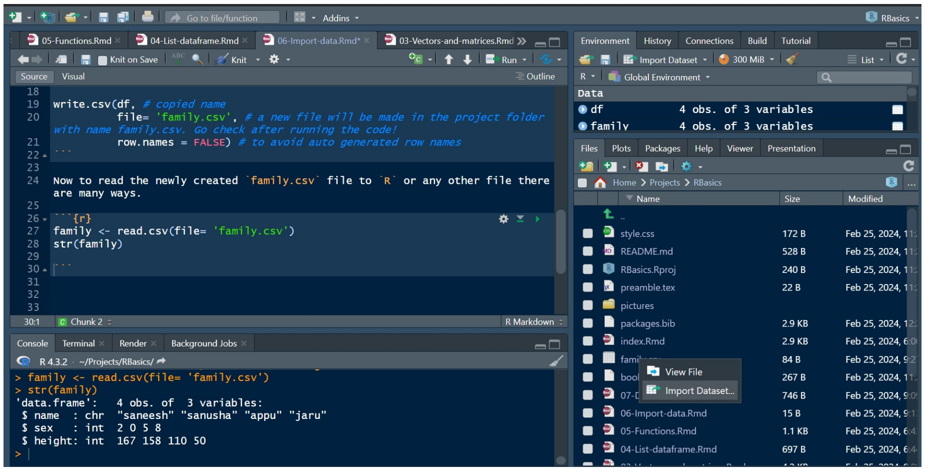
7.2 Read .csv

Now to read the newly created `family.csv` file to `R` or any other file there are many ways.

```r
family <- read.csv(file= 'family.csv')
str(family)
```

```
## 'data.frame':    4 obs. of  3 variables:
##  $ name  : chr  "saneesh" "sanusha" "appu" "jaru"
##  $ sex   : int  2 0 5 8
##  $ height: int  167 158 110 50
```

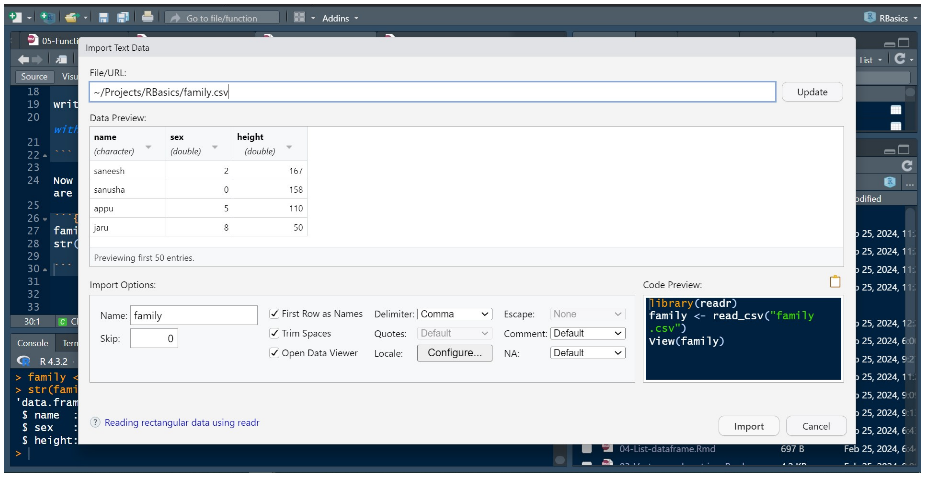
The other method is to import file from `Files`, `Plots`, `Packages` panel.
Go to `Files`, left click on the .csv file you want to import to `R`, `Import Dataset...` a new window will open, `Import Text Data`



```
write.csv(df, # copied name
          file= 'family.csv', # a new file will be made in the project folder
with name family.csv. Go check after running the code!
          row.names = FALSE) # to avoid auto generated row names
```

Now to read the newly created `family.csv` file to `R` or any other file there
are many ways.

```{r}
family <- read.csv(file= 'family.csv')
str(family)

```
```

```
> family <- read.csv(file= 'family.csv')
> str(family)
'data.frame':   4 obs. of  3 variables:
 $ name  : chr  "saneesh" "sanusha" "appu" "jaru"
 $ sex   : int  2 0 5 8
 $ height: int  167 158 110 50
>
```

```
Data
df          4 obs. of 3 variables
family      4 obs. of 3 variables
```

Name	Size	Modified
..		
style.css	172 B	Feb 25, 2024, 11:
README.md	528 B	Feb 25, 2024, 11:
RBasics.Rproj	240 B	Feb 25, 2024, 11:
preamble.tex	22 B	Feb 25, 2024, 11:
pictures		
packages.bib	2.9 KB	Feb 25, 2024, 12:
index.Rmd	2.9 KB	Feb 25, 2024, 6:0
fami	84 B	Feb 25, 2024, 9:2
bool	267 B	Feb 25, 2024, 11:
07-D	746 B	Feb 25, 2024, 9:0
06-Import-data.Rmd	15 B	Feb 25, 2024, 9:1
05-Functions.Rmd	1.1 KB	Feb 25, 2024, 6:4
04-List-dataframe.Rmd	697 B	Feb 25, 2024, 6:4

Figure 7.1: Files .csv file

Look at the data preview, import options (change `Name` if needed), click `Import`. or copy code from `code preview`, click on `cancel` and paste the code on the script panel and `Run`. If you don't have `readr` package `R` will show an error, don't worry read the message and `install.packages('readr')` will solve the problem, then run the code again.
```



Import Text Data

File/URL: ~/Projects/RBasics/family.csv

Data Preview:

name (character)	sex (double)	height (double)
saneesh	2	167
sanusha	0	158
appu	5	110
jaru	8	50

Previewing first 50 entries.

Import Options:

Name: family
Skip: 0

☑ First Row as Names
☑ Trim Spaces
☑ Open Data Viewer

Delimiter: Comma
Quotes: Default
Locale: Configure...

Escape: None
Comment: Default
NA: Default

Code Preview:

```
library(readr)
family <- read_csv("family
.csv")
View(family)
```

Figure 7.2: Files .csv file

Chapter 8

Data Visualization

Condensing intricate data into numerical summaries inevitably leads to some loss of information. Visualization, though it also involves some information loss, can be less so when executed effectively. A thorough data analysis should always begin with a phase where the analyst deeply familiarizes themselves with the data they're working with. This process of gaining a deep understanding is integral to data analysis, and visualization plays a crucial role in it.

The vague & defeasible rule of thumb of good data visualization (according to the Edward Tufte).

> "Communicate a maximal degree of relevant true information in a way that minimizes the recipient's effort of retrieving this information."

8.1 Scatter plot

`?geom_point`, try to to know more about geom_points The point geom is used to create scatterplots. The scatterplot is most useful for displaying the relationship between two continuous variables.

8.1.1 Let us make some data.

Imagine Shubham and I are planning to do a research on how Cashew and Teak respond to a new fertilizer we invented!!

To make the data we will use `ggplot2` and `dplyr`, we can use `tidyverse` so both packages will be loaded.

We also want to check if the trees in the fertilizer treatment can store more carbon. I don't know how scientifically correct this is but I am using this resource Carbon storage calculator. Based on this I wrote a function, read the `Chapter 6` for additional information on functions.

8.1.2 carbon calculator function

```
carbon_calc <- function(x) {
  dry_weight = x * 13.0933
  carbon = round(dry_weight / 2)
  return(carbon)
}

carbon_calc(1)
```

```
## [1] 7
```

```
# treatment (factor)
library(ggplot2)
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
```

```
# new df with treatment
set.seed(007) # the name is Bond, James Bond= 007, its just a number. If you want the
# new df with treatment

treatment <- c(rep('nofertilizer', 50),
               rep('fertilizer', 50))

name <- c(rep('cashew', 100),
          rep('teak', 100))
```

```r
gbh <- c(rnorm(50,
             mean = 15,
             sd = 4),
         rnorm(50,
               mean = 30,
               sd = 6))

df <- data.frame(treatment,
                 name,
                 gbh)

str(df)
```

```
## 'data.frame':    200 obs. of  3 variables:
##  $ treatment: chr  "nofertilizer" "nofertilizer" "nofertilizer" "nofertilizer" ...
##  $ name     : chr  "cashew" "cashew" "cashew" "cashew" ...
##  $ gbh      : num  24.1 10.2 12.2 13.4 11.1 ...
```

```r
newdf <- df %>%
  filter(gbh  >= 15)

co <- rnorm(length(newdf$gbh), carbon_calc(1), 1.5) # kg

newdf <- newdf %>%
  mutate(co2= gbh * co) # co2 is a new column based gbh

str(newdf)
```

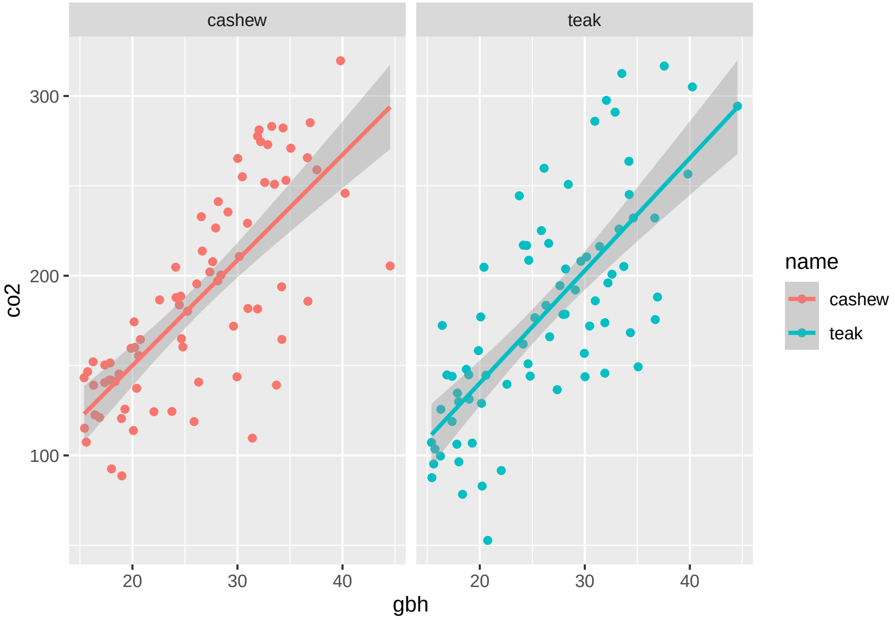
```
## 'data.frame':    154 obs. of  4 variables:
##  $ treatment: chr  "nofertilizer" "nofertilizer" "nofertilizer" "nofertilizer" ...
##  $ name     : chr  "cashew" "cashew" "cashew" "cashew" ...
##  $ gbh      : num  24.1 18 15.6 23.8 16.4 ...
##  $ co2      : num  188 142 107 124 123 ...
```

```r
names(newdf)
```

```
## [1] "treatment" "name"      "gbh"       "co2"
```

```r
ggplot(data = newdf, aes(x = gbh, y = co2, col= name)) +
  geom_point() +
  geom_smooth(method = 'lm') +
  facet_wrap(~ name)
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



```
# To know what facet_wrap is

# ?facet_wrap
# help(facet_wrap)
```

8.2 Box plot

A box and whisker plot or diagram (otherwise known as a boxplot), is a graph summarising a set of data. The shape of the boxplot shows how the data is distributed and it also shows any outliers. It is a useful way to compare different sets of data as you can draw more than one boxplot per graph. These can be displayed alongside a number line, horizontally or vertically.
Reading a Box and Whisker Plot
Interpreting a boxplot can be done once you understand what the different lines mean on a box and whisker diagram. The line splitting the box in two represents the median value. This shows that 50 % of the data lies on the left hand side of the median value and 50 % lies on the right hand side. The left edge of the box represents the lower quartile; it shows the value at which the first 25 % of the data falls up to. The right edge of the box shows the upper quartile; it shows that 25 % of the data lies to the right of the upper quartile value. The values at

which the horizontal lines stop at are the values of the upper and lower values of the data. The single points on the diagram show the outliers.
geom_boxplot function from ggplot2 compactly displays the distribution of a continuous variable and a categorical variable, see Figure (Boxplot)

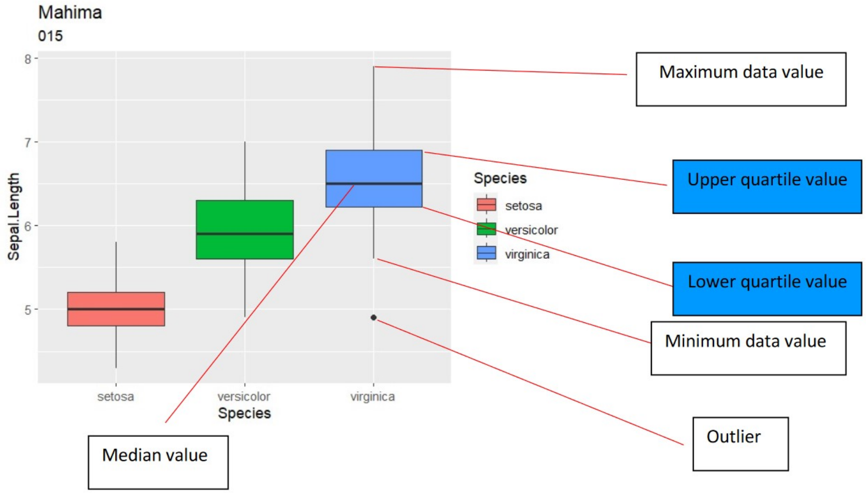
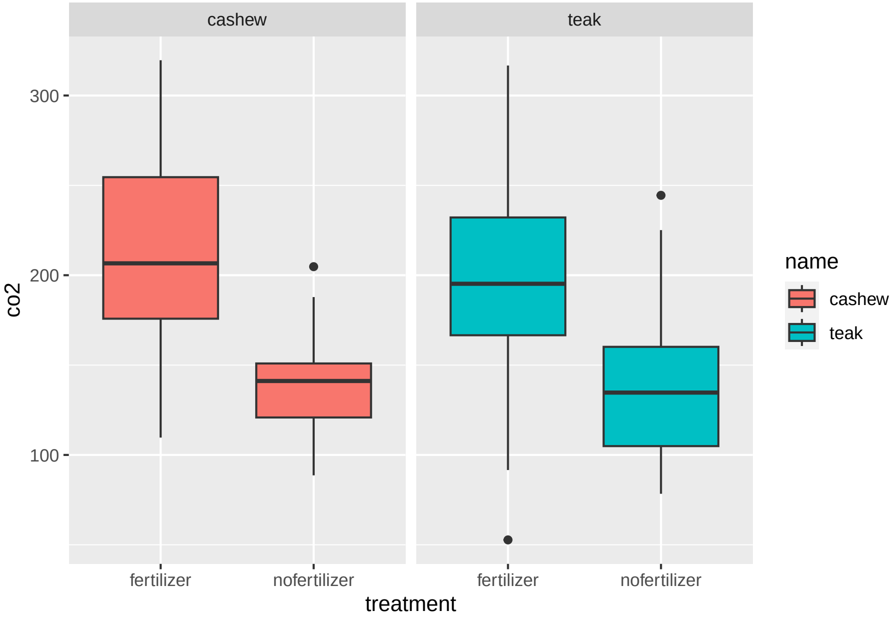
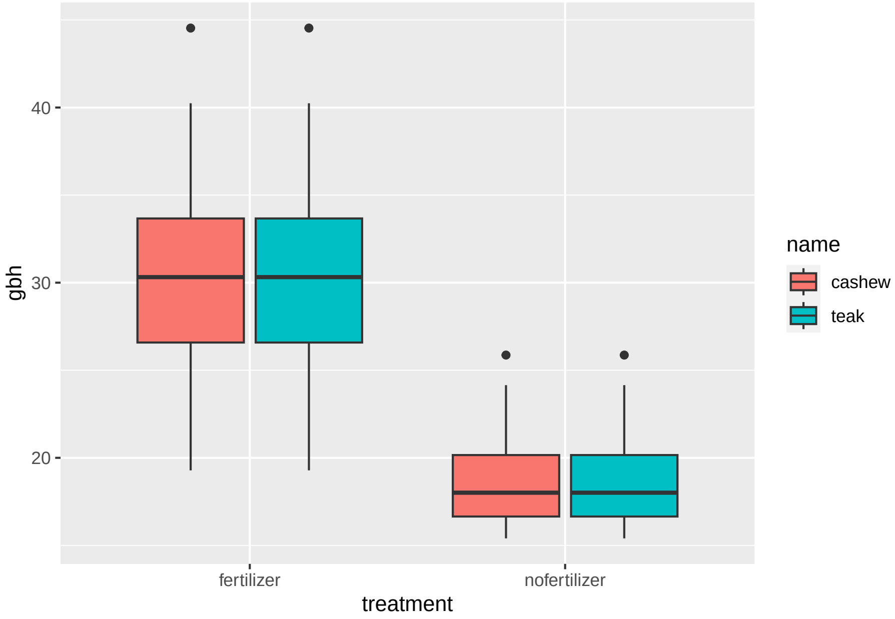


Figure 8.1: Boxplot

```
ggplot(data = newdf,
       aes(x = treatment,
           y = co2,
           fill = name)) +
  geom_boxplot() +
  facet_wrap( ~ name)
```



```
ggplot(data = newdf,
       aes(y = gbh,
           x = treatment,
           fill = name)) +
  geom_boxplot()
```


40
30
20
gbh
fertilizer
nofertilizer
treatment
name
cashew
teak

Chapter 9

Wordcloud

9.1 Wordcloud from data frame

```
# install.packages("dplyr")
# install.packages('wordcloud')

# add packages----
library(dplyr) # to clean data
library(wordcloud) # to make wordcloud
```

```
## Loading required package: RColorBrewer
```

9.1.1 Read data

```
words <- data.frame(Name= c(rep('James Bond', 30),
                    rep('Iron Man', 20),
                    rep('Spider Man', 5),
                    rep('Caption America', 4)),
                    Tag= c(rep('Spy', 30),
                               rep('Super hero', 29)),
                    Duty= c(rep('kill', 30),
                            rep('developer', 20),
                            rep('save', 5),
                            rep('save', 4)))

str(words) # display the internal structure `object`, here, our `data`
```

```
## 'data.frame':    59 obs. of  3 variables:
##  $ Name: chr  "James Bond" "James Bond" "James Bond" "James Bond" ...
##  $ Tag : chr  "Spy" "Spy" "Spy" "Spy" ...
##  $ Duty: chr  "kill" "kill" "kill" "kill" ...
```

```
head(words, 3) # change the number in the () and see more rows
```

```
##         Name Tag Duty
## 1 James Bond Spy kill
## 2 James Bond Spy kill
## 3 James Bond Spy kill
```

```
tail(words, 3)
```

```
##                Name        Tag Duty
## 57 Caption America Super hero save
## 58 Caption America Super hero save
## 59 Caption America Super hero save
```

```
# let us see what is the duty of our heros
names(words) # to see the column names of words
```

```
## [1] "Name" "Tag"  "Duty"
```

```
# to add count to the data
words1 <- words %>% count(Duty) # words1 is a new data frame we created from words
```

```
names(words1)
```

```
## [1] "Duty" "n"
```

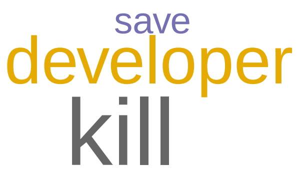
```
max(words1$n) # copy this number for max.words=
```

```
## [1] 30
```

```
# par(bg="black") # for black background
```

Before you plot, make sure that the `Files`, `Plot Packages` panel is maximized!! click on the rectangular button on top of the File, Plots, Packages window.

```
wordcloud(word = words1$Duty, # BroaderGoal from the words1
          freq = words1$n, # n from the words1 data frame
          min.freq = 1, # minimum frequency of a word
          max.words = 14, # maximum frequency of a word
          random.order=T,
          # colors = brewer.pal(6, 'GnBu') # remove comment from
          # these code one by one and see how color change
          # colors = brewer.pal(6, 'Reds')
          # colors = brewer.pal(6, 'Blues')
          # colors = brewer.pal(1, 'Reds')
          colors=brewer.pal(8, "Dark2")
          )
```



Let us see what are they known as

```
names(words) # to see the column names of words
```

```
## [1] "Name" "Tag"  "Duty"
```

```
# know the data
words2 <- words %>% count(Tag)

names(words2)
```

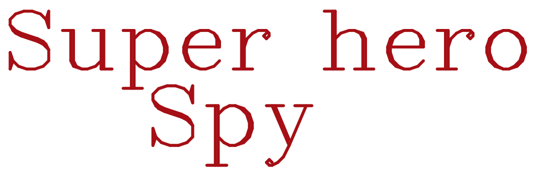
```
## [1] "Tag" "n"
```

```
max(words2$n) # copy this number for max.words=
```

```
## [1] 30
```

Before you plot, make sure that the plot window or Plot pane is maximized!! click on the rectangular button on top of the `File`, `Plots`, `Packages` window.

```
# par(bg="black") # for black background

wordcloud(word = words2$Tag, # BroderBackground from the words2
          freq = words2$n, # n from the words2 data frame
          min.freq = 1, # minimum frequency of a word
          max.words = 10, # maximum frequency of a word
          random.order=T,
          # vfont= c(family= 'gothic english', face= 'plain'), # fonts, remove comment
          # these code one by one and see how font and color change
          # vfont= c(family= 'script', face= 'plain'),
          # vfont= c(family= 'serif', face= 'plain'),
          # vfont= c(family= 'sans serif', face= 'plain'),
          vfont= c(family= 'serif', face= 'cyrillic'),
          # colors = brewer.pal(6, 'GnBu') # colors
          colors = brewer.pal(6, 'Reds')
          # colors = brewer.pal(6, 'Blues')
          # colors = brewer.pal(1, 'Reds')
          # colors=brewer.pal(8, "Dark2")
)
```



If the plot doesn't look good restart R from Session tab or press Ctrl+Shift+F10, then start from add packages :). If you want to change background bg= "black" or other colors do the above, restart. If you are happy with a plot, Export> Save as PDF> select PDF Size> change directory if you want.

9.2 Wordcloud from text data

```
# packages----
# install.packages('tm')
# install.packages('wordcloud')
```

Load packages

```
library(tm)
```

```
## Loading required package: NLP

##
## Attaching package: 'NLP'

## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```
library(wordcloud)
```

Text

```
mytext <- "India, a vibrant land where ancient traditions dance with modern aspiration
```

```
mycorpus <- Corpus(VectorSource(mytext))

mycorpus <- tm_map(mycorpus, removePunctuation)
```

```
## Warning in tm_map.SimpleCorpus(mycorpus, removePunctuation): transformation
## drops documents
```

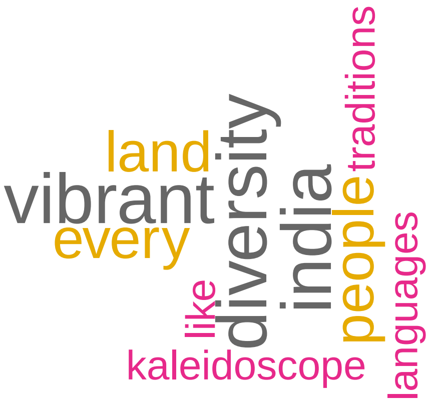
```
mycorpus <- tm_map(mycorpus, removeNumbers)
```

```
## Warning in tm_map.SimpleCorpus(mycorpus, removeNumbers): transformation drops
## documents
```

```
mycorpus <- tm_map(mycorpus, removeWords, stopwords("english"))
```

```
## Warning in tm_map.SimpleCorpus(mycorpus, removeWords, stopwords("english")):
## transformation drops documents
```

```
mywords <- TermDocumentMatrix(mycorpus)
m <- as.matrix(mywords)
v <- sort(rowSums(m), decreasing=TRUE)

#make wordcloud----
wordcloud(names(v), v, scale=c(3,0.5), min.freq = 1,
          max.words=10, random.order=FALSE, rot.per=0.35,
          colors=brewer.pal(8, "Dark2"))
```



```
# a word cloud for Indian budget
library(rvest)
```

```
##
## Attaching package: 'rvest'

## The following object is masked from 'package:readr':
##
##     guess_encoding
```

```
library(wordcloud)
library(tm)
library(Rgraphviz)
```

```
## Loading required package: graph

## Loading required package: BiocGenerics

##
## Attaching package: 'BiocGenerics'

## The following object is masked from 'package:NLP':
##
##     annotation
## The following objects are masked from 'package:dplyr':
##
##     combine, intersect, setdiff, union

## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':
##
##     anyDuplicated, aperm, append, as.data.frame, basename, cbind,
##     colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,
##     get, grep, grepl, intersect, is.unsorted, lapply, Map, mapply,
##     match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,
##     Position, rank, rbind, Reduce, rownames, sapply, setdiff, sort,
##     table, tapply, union, unique, unsplit, which.max, which.min

## Loading required package: grid
```

```
# install.packages("BiocManager")
# BiocManager::install("Rgraphviz")


# budget
# set path to the textfile
budget <- read.delim('budget_speech.txt', fileEncoding="UCS-2LE")
```

```
## Warning in read.table(file = file, header = header, sep = sep, quote = quote, :
## incomplete final line found by readTableHeader on 'budget_speech.txt'
```

```
# Remove punctuation and numbers
budget <- gsub("[[:punct:]0-9]", "", budget, useBytes = TRUE)

# covert text to lowercase
budget <- tolower(budget)

# remove stopwords
stopwords <- stopwords('SMART') # options 'english'

budget <- budget[!budget %in% stopwords]



# Create a corpus from the text
```

```
corpus <- Corpus(VectorSource(budget))

# Create a term-document matrix
tdm <- TermDocumentMatrix(corpus)

# Convert the tdm to a matrix and calculate word frequencies
matr <- as.matrix(tdm)
word_freq <- sort(rowSums(matr), decreasing = TRUE)
```

```
wordcloud(names(word_freq),
          word_freq, max.words = 50, # can change number of words
          random.order = FALSE,
          colors = brewer.pal(8, "Dark2"))
```

It will look something like this!

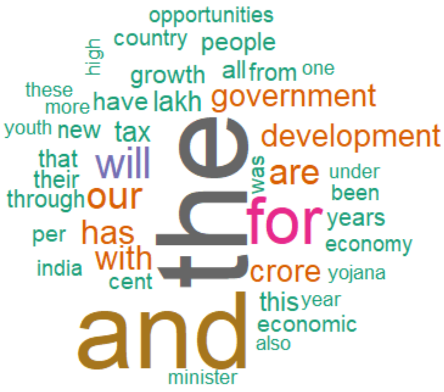


Figure 9.1: Budget word cloud

Chapter 10

Probability

```
library('dplyr', 'ggplot2')
```

```
set.seed(007) # to ensure that the random sampling results are reproducible.
```

10.1 create vectors

```
species= c('Tiger', 'Deer', 'Leopard')
tiger_reserve = c('Kanha', 'Bandipur')
sighting = c(1, 0) # 1 (for sighting) and 0 (for no sighting)
```

10.2 make a data frame, name it I gave df= data frame

```
df <- data.frame(
  species= sample(x = species, 20, replace = T), # replace= T, for repeated sampling (with replac
  tiger_reserve= sample(x = tiger_reserve, 20, replace = T),
  sighting= sample(x = sighting, 20, replace = T)
)
```

```
print(df)
```

```
##    species tiger_reserve sighting
## 1     Deer         Kanha        1
## 2  Leopard      Bandipur        0
## 3  Leopard         Kanha        1
## 4  Leopard      Bandipur        1
## 5     Deer      Bandipur        0
## 6  Leopard         Kanha        0
## 7     Deer      Bandipur        1
## 8     Deer      Bandipur        1
## 9  Leopard      Bandipur        1
## 10    Deer      Bandipur        1
## 11 Leopard         Kanha        0
## 12 Leopard      Bandipur        0
## 13    Deer      Bandipur        0
## 14 Leopard         Kanha        1
## 15    Deer         Kanha        0
## 16 Leopard      Bandipur        0
## 17 Leopard      Bandipur        1
## 18    Deer         Kanha        0
## 19    Deer         Kanha        0
## 20 Leopard         Kanha        0
```

10.3 calculating total sightings

```
sightings_total <- df %>%
  group_by(species, tiger_reserve) %>%
  summarize(total_sightings = sum(sighting))
```

```
## `summarise()` has grouped output by 'species'. You can override using the
## `.groups` argument.
```

10.4 Calculate the probability of seeing each species

```
probabilities <- sightings_total %>%
  mutate (probability = total_sightings / sum(total_sightings)) %>%
  mutate (percentage = probability*100)
```

```
library(report)
print(probabilities)
```

```
## # A tibble: 4 x 5
## # Groups:   species [2]
##   species tiger_reserve total_sightings probability percentage
##   <chr>   <chr>                   <dbl>       <dbl>      <dbl>
## 1 Deer    Bandipur                    3        0.75         75
## 2 Deer    Kanha                       1        0.25         25
## 3 Leopard Bandipur                    3        0.6          60
## 4 Leopard Kanha                       2        0.4          40
```

```
report::report_statistics(probabilities)
```

```
## Deer, Bandipur, n = 1; Kanha, n = 1
## Deer, n = 2, Mean = 2.00, SD = 1.41, Median = 2.00, MAD = 1.48, range: [1, 3], Skewness = 0.0
## Deer, n = 2, Mean = 0.50, SD = 0.35, Median = 0.50, MAD = 0.37, range: [0.25, 0.75], Skewness
## Deer, n = 2, Mean = 50.00, SD = 35.36, Median = 50.00, MAD = 37.06, range: [25, 75], Skewness
## Leopard, Bandipur, n = 1; Kanha, n = 1
## Leopard, n = 2, Mean = 2.50, SD = 0.71, Median = 2.50, MAD = 0.74, range: [2, 3], Skewness =
## Leopard, n = 2, Mean = 0.50, SD = 0.14, Median = 0.50, MAD = 0.15, range: [0.40, 0.60], Skewne
## Leopard, n = 2, Mean = 50.00, SD = 14.14, Median = 50.00, MAD = 14.83, range: [40, 60], Skewne
```

```
probabilities %>% ggplot(aes(x= species, y= probability, fill= tiger_reserve))+
  geom_bar(stat = 'identity')
```

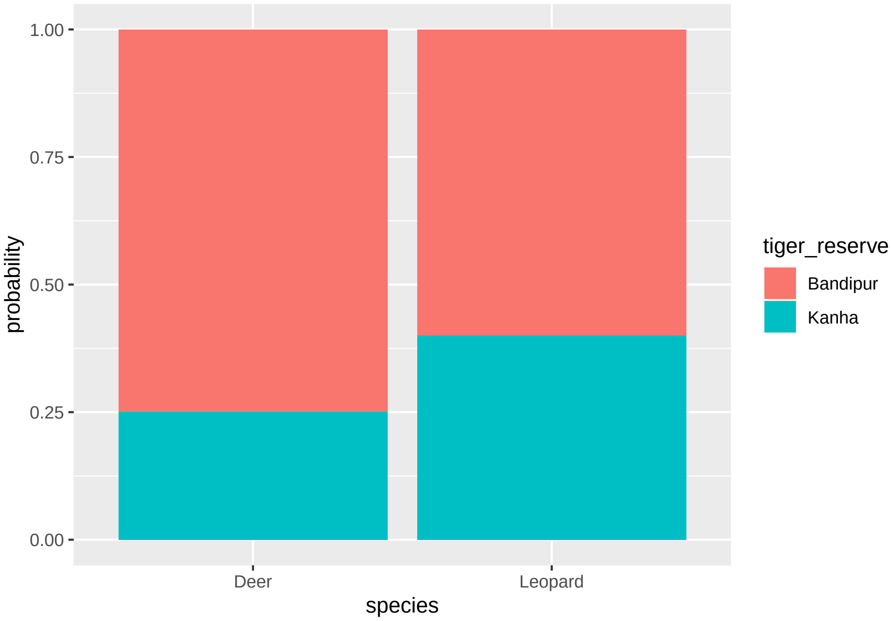


Figure 10.1: Probability