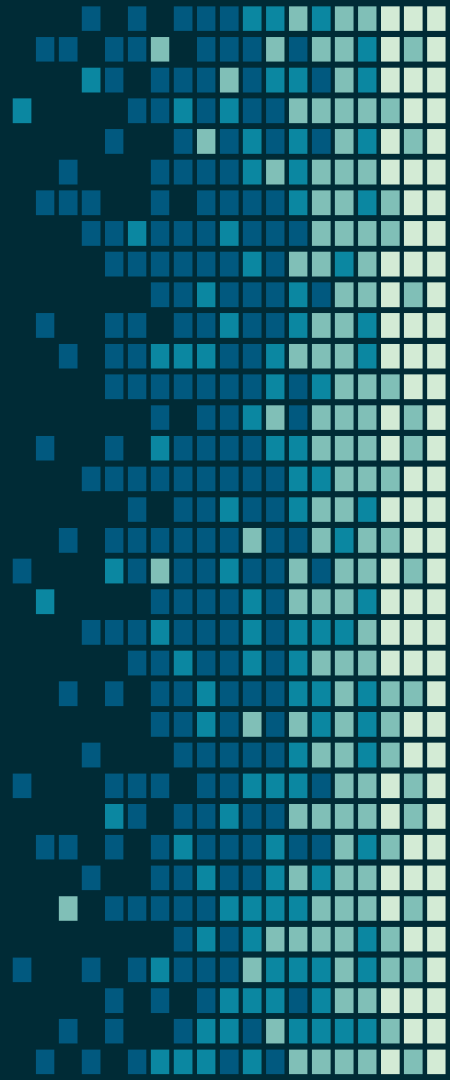


# Learning R the hard way

Maani Beigy, MD-MPH





Universal Council of Epidemiology (UCE)

Universal Scientific Education and Research Network (USERN)

Students' Scientific Research Center (SSRC)

Tehran University of Medical Sciences (TUMS)



# HELLO!

I am Maani Beigy

Who R U ?

<https://github.com/MaaniBeigy>

[https://www.researchgate.net/profile/Maani\\_Beigy](https://www.researchgate.net/profile/Maani_Beigy)

<https://stackoverflow.com/users/9555505/maanib?tab=profile>

<https://orcid.org/0000-0003-2963-3533>

[www.linkedin.com/in/maani-beigy-69454438](http://www.linkedin.com/in/maani-beigy-69454438)





# Introduction

Why R?



# Introduction: excellent

free available

Open-source

reproducible

importing and

manipulating

research

data

packages

statistical  
modeling

machine  
learning

visualization



Introduction: excellent 

Cutting edge  
tools

fantastic  
community

integrated development environment (IDE)

**RSTUDIO**

functional  
programming

create interactive websites:

Writing  
apps

**Shiny**

Metaprogramming facilities  
domain-specific languages (DSL)

SQL, mongoDB, C, C++



# Introduction: bad

more focused  
on results

Much of the  
R code

Metaprogramming  
a double-edged sword

Inconsistency  
Lots of changes

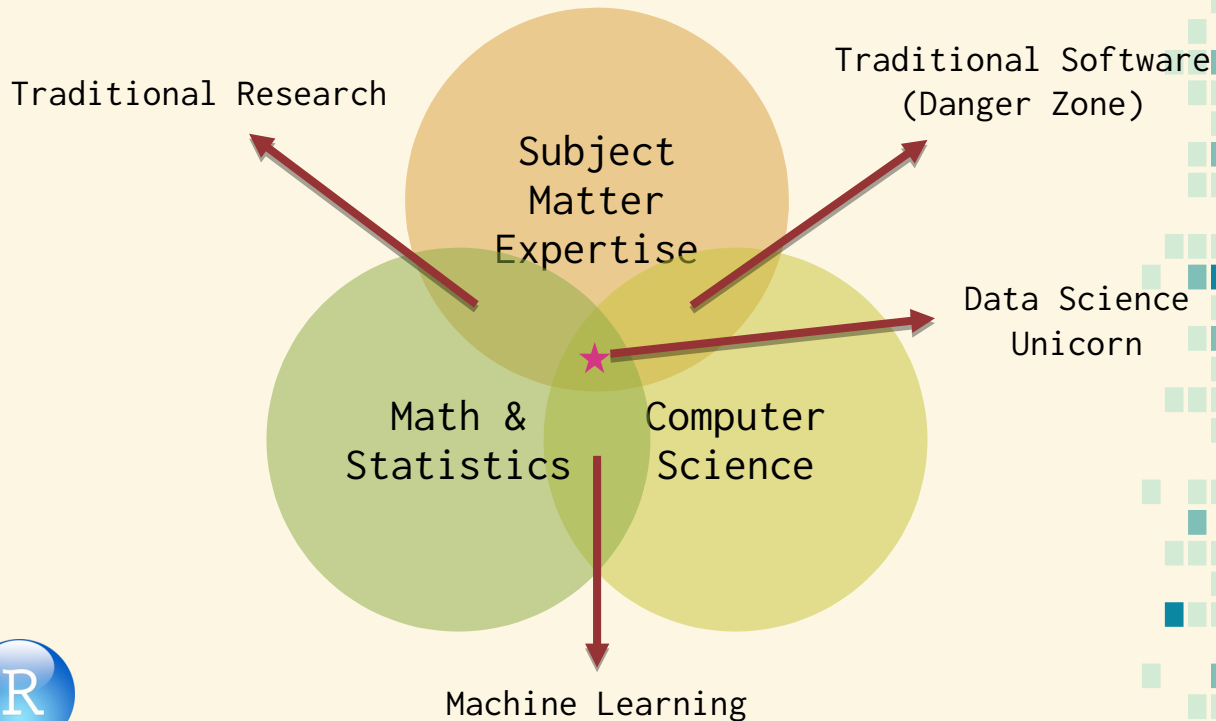
profligate user  
of memory

poorly written R code  
can be terribly slow



# Introduction:

## Data Science



# Goals

## Part 1



To provide an introduction to R software environment and programming

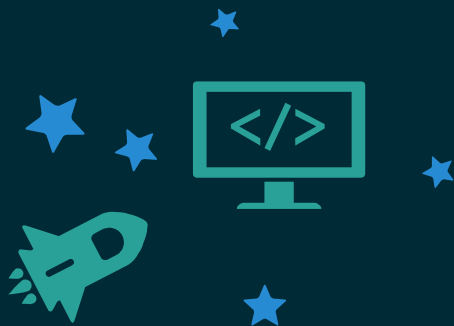
## Part 2



To provide a hands-on experience with R statistical analyses







# Part 1

R software environment and programming



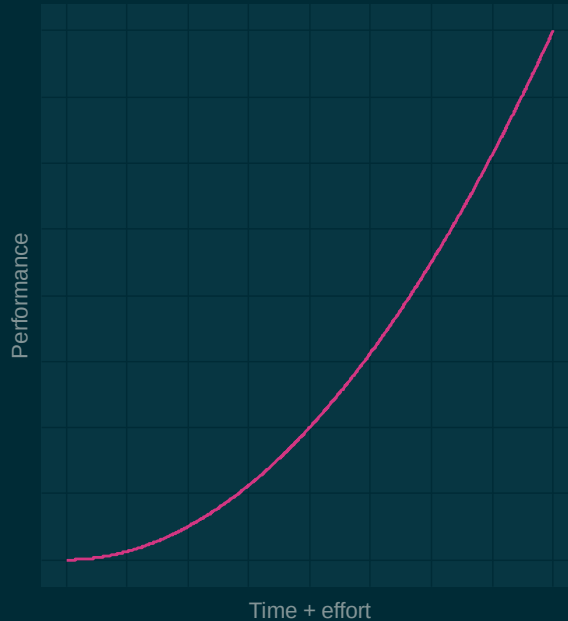
# What is R?

- › `Programming language` and `software` environment for statistical computing and graphics
- › Widely used among statisticians and data scientists
  - › Excellent statistical facilities
- › Nearly everything you may need in terms of statistics has already been programmed and made available in R
  - › either as part of the `main packages` or as a `user-contributed packages`



# Learning R

- › R has a **steep learning curve**



# History of R

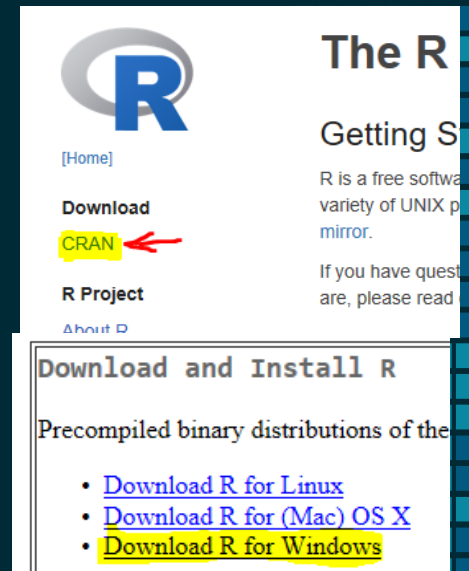
- › R is an implementation of **S programming** language.
- › S was created by John Chambers in 1976 at Bell Labs.
- › R: Open source platform similar to S developed by Robert Gentleman and Ross Ihaka (U of Auckland) during the 1990s.
- › Since 1997, R has been developed by the **R Development Core Team**.



# Download R

- › To obtain and install R on your computer:
  - › Go to the R homepage <https://www.r-project.org/>
  - › Click **CRAN** under Download
  - › Choose a “**mirror site**”
  - › Click **Download R for Windows**
  - › Click **base**
  - › Click [Download R 3.5.2 for Windows](#)

Note: Updated versions of R are available every few months.

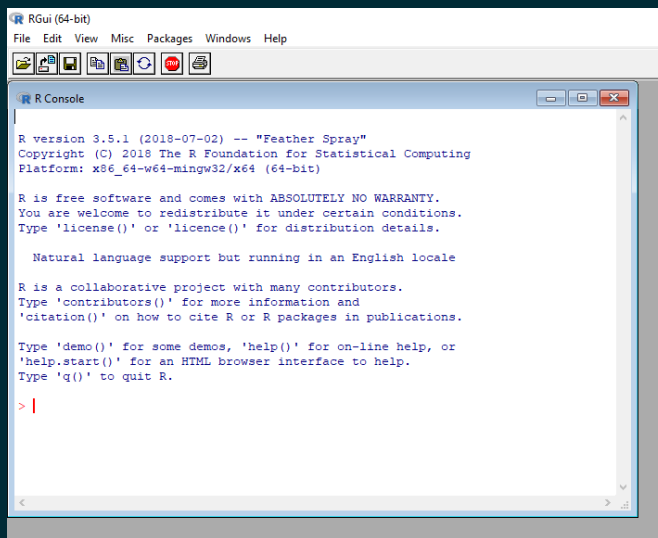


# Install and Use R

- › To install R click the downloaded file. The simplest procedure is to accept all default settings.
- › After installation, the R icon will appear on your desktop.
- › Launch R by double-clicking the icon.
- › You will see the following R Windows



# Install and Use R



The screenshot shows the RGui (64-bit) window with a menu bar (File, Edit, View, Misc, Packages, Windows, Help) and a toolbar. The R Console window is open, displaying the following text:

```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 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.

> |
```

- › Please type the following commands on R console and see what happens:
- › `citation()`




# Install and Use RStudio

**R Studio** Products Resources Pricing About Us Blogs

## Choose Your Version of RStudio

RStudio is a set of integrated tools designed to help you be more productive with R. It includes a console, syntax-highlighting editor that supports direct code execution, and a variety of robust tools for plotting, viewing history, debugging and managing your workspace. [Learn More](#) about RStudio features.



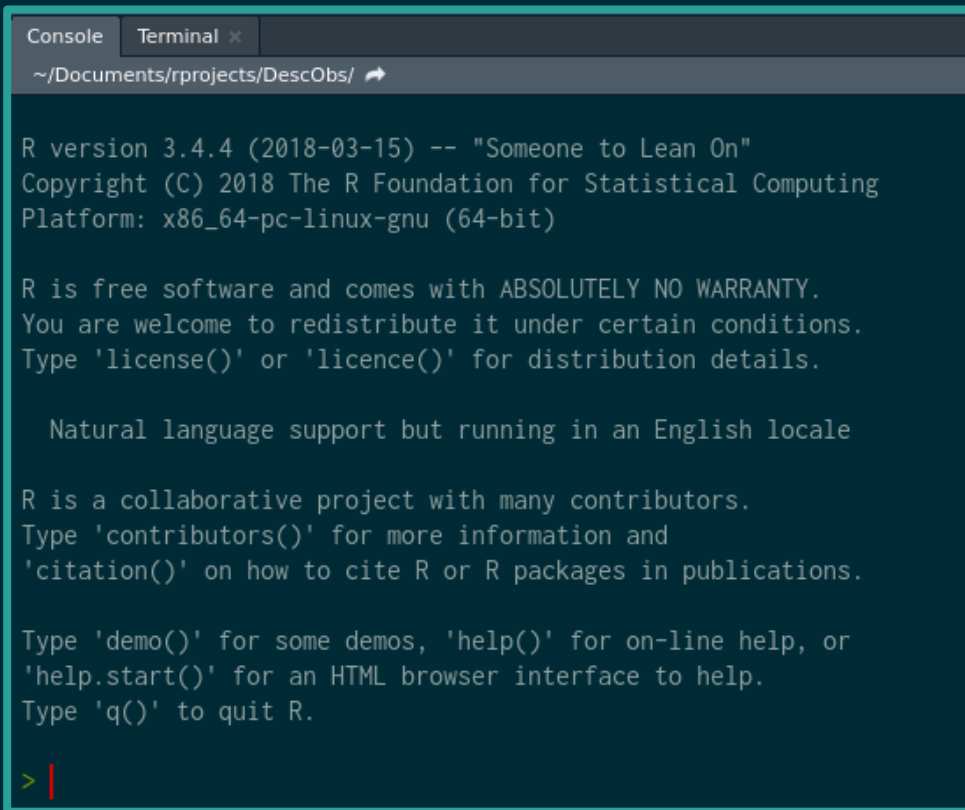
RStudio Desktop Open Source License	RStudio Desktop Commercial License	RStudio Server Open Source License	RStudio Server Pro Commercial License	RStudio Server Pro + RStudio Connect Commercial License
FREE	\$995 per year	FREE	\$9,995 per year	\$29,995 per year
<a href="#">DOWNLOAD</a>	<a href="#">BUY</a>	<a href="#">DOWNLOAD</a>	<a href="#">DOWNLOAD</a>	<a href="#">TALK</a>
<a href="#">Learn More</a>	<a href="#">Learn More</a>	<a href="#">Learn More</a>	<a href="#">Learn More</a>	<a href="#">Learn More</a>

Integrated Tools for R





# Console and Terminal

A screenshot of an R console window. The window has a title bar with 'Console' and 'Terminal x' tabs. The address bar shows the path '~/.Documents/rprojects/DescObs/'. The main area contains the R startup message, including the version (3.4.4), copyright (2018 The R Foundation), and platform (x86\_64-pc-linux-gnu). It also includes instructions on how to use R, such as typing 'license()' for distribution details, 'contributors()' for more information, and 'demo()' for some demos. The prompt '> |' is visible at the bottom.

```
Console Terminal x
~/Documents/rprojects/DescObs/ ↗

R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (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

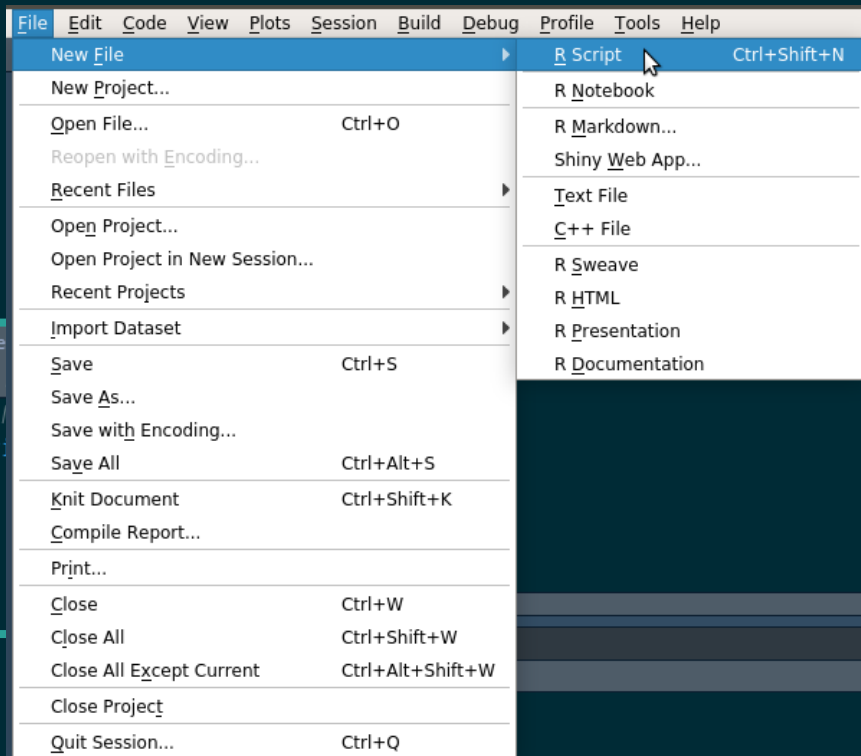
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

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.

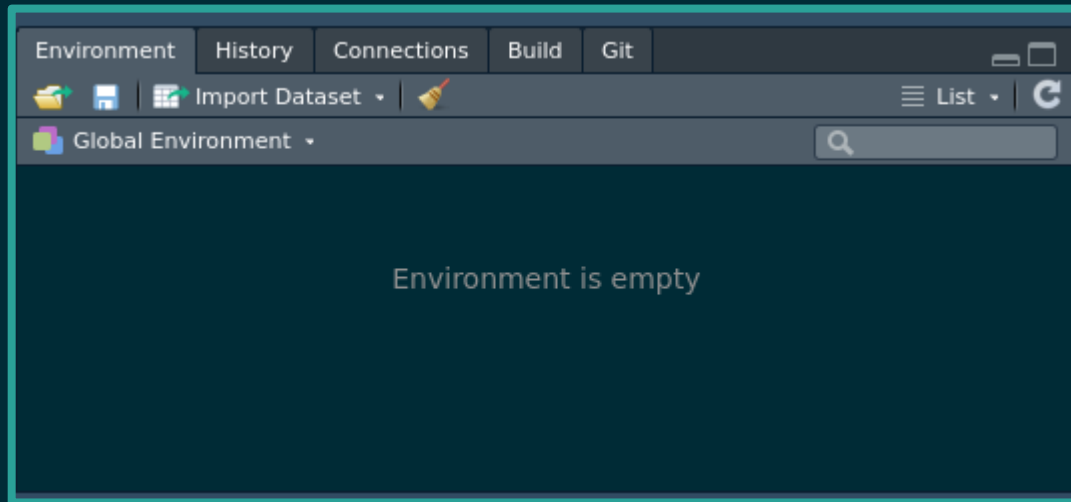
> |
```



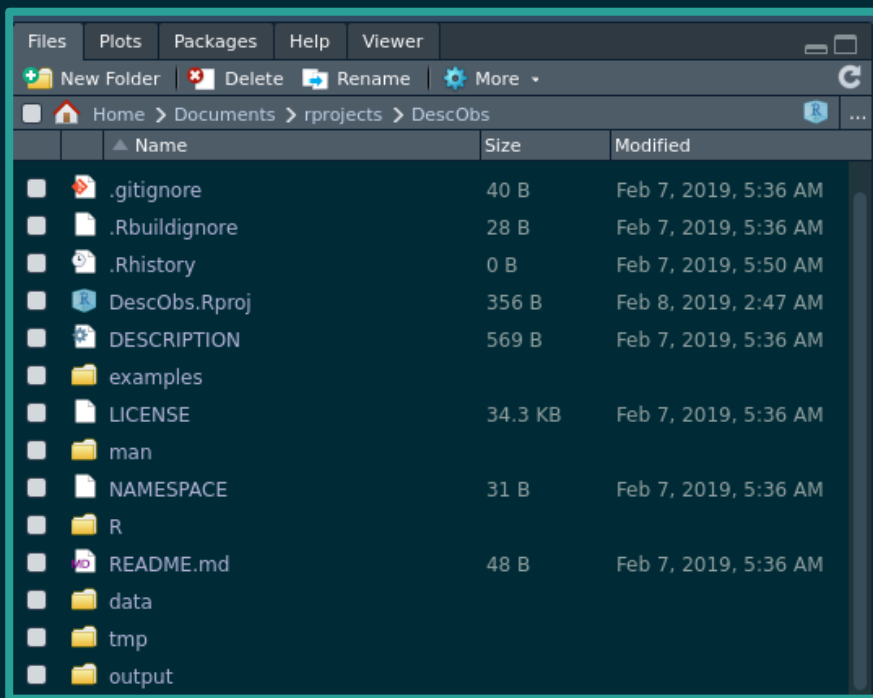
# Script



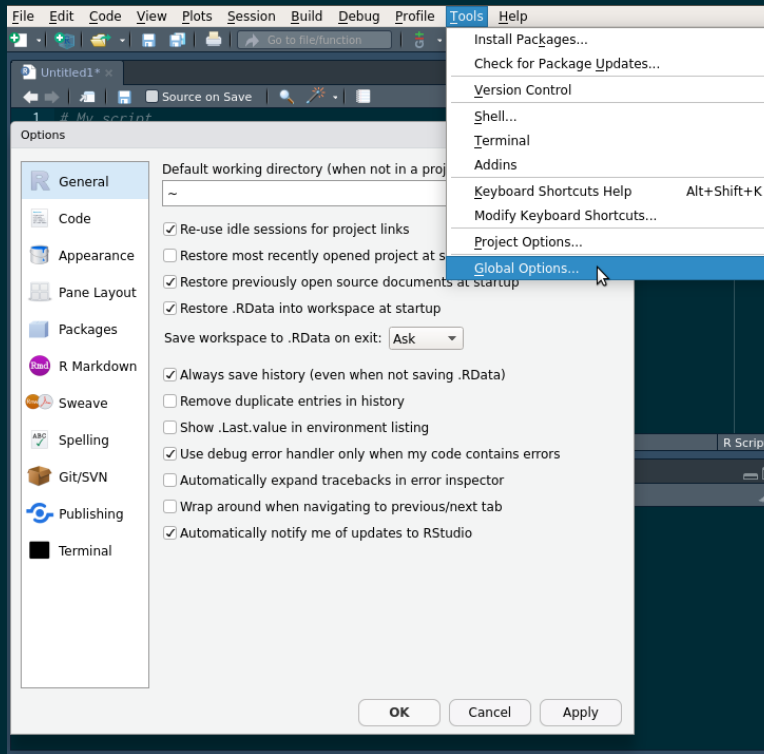
# Environment



# Outputs, Files, ...



# Setting and Options



# Hello World!

```
> print("hello world")  
[1] "hello world"
```



# help("")

- `help("boxplot")` or simply `?boxplot`

boxplot {graphics}

R Documentation

## Box Plots

### Description

Produce box-and-whisker plot(s) of the given (grouped) values.

### Usage

```
boxplot(x, ...)
```

```
## S3 method for class 'formula'
```

```
boxplot(formula, data = NULL, ..., subset, na.action = NULL,  
        drop = FALSE, sep = ".", lex.order = FALSE)
```

```
## Default S3 method:
```

```
boxplot(x, ..., range = 1.5, width = NULL, varwidth = FALSE,  
        notch = FALSE, outline = TRUE, names, plot = TRUE,  
        border = par("fg"), col = NULL, log = "",  
        pars = list(boxwex = 0.8, staplewex = 0.5, outwex = 0.5),  
        horizontal = FALSE, add = FALSE, at = NULL)
```

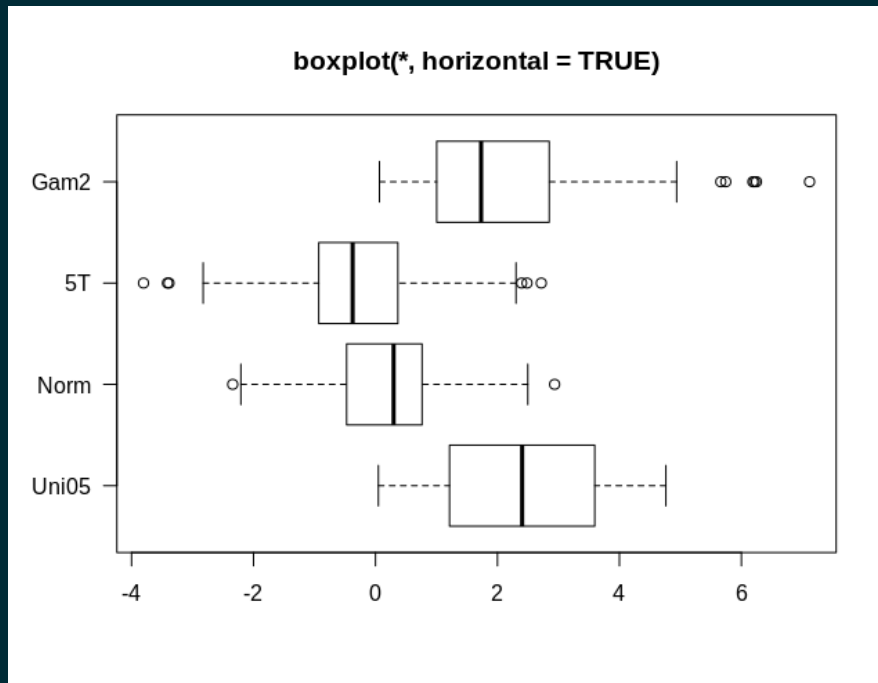
### Arguments

formula	a formula, such as <code>y ~ grp</code> , where <code>y</code> is a numeric vector of data values to be split into groups according to the grouping variable <code>grp</code> (usually a factor).
---------	---



# example("")

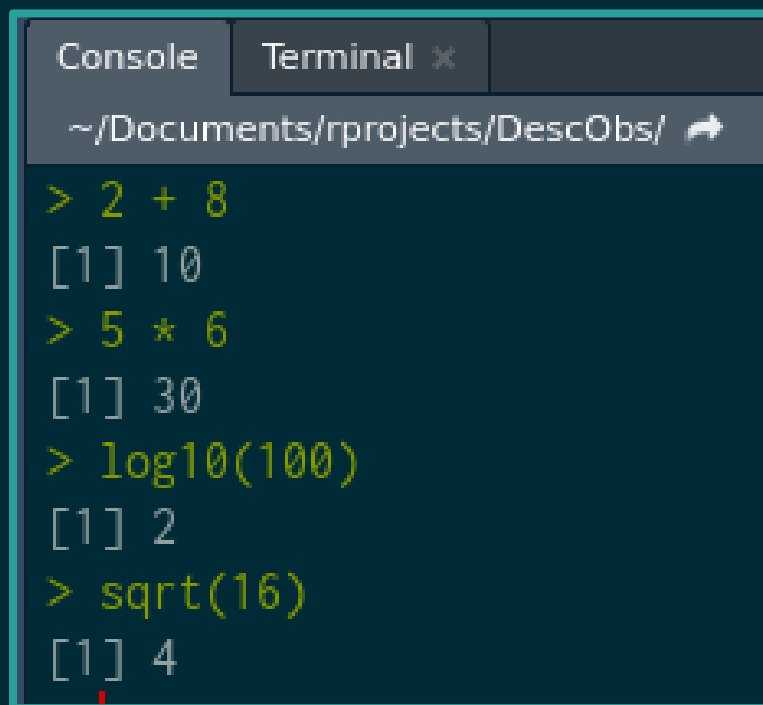
- `example("boxplot")`





# Typing Commands

?Arithmetic



```
Console Terminal x  
~/Documents/rprojects/DescObs/ ↵  
> 2 + 8  
[1] 10  
> 5 * 6  
[1] 30  
> log10(100)  
[1] 2  
> sqrt(16)  
[1] 4
```



# Typing Commands

Arithmetic {base}

R Documentation

## Arithmetic Operators

### Description

These unary and binary operators perform arithmetic on numeric or complex vectors (or objects which can be coerced to them).

### Usage

```
+ x  
- x  
x + y  
x - y  
x * y  
x / y  
x ^ y  
x %% y  
x %/% y
```

### Arguments

x,	numeric or complex vectors or objects which can be coerced to such, or other objects
y	for which methods have been written.



# Typing Commands

`+` `-` `*` `/` `^`

`==`

equal

`<`

smaller than

`>`

greater than

`<=`

smaller than or equal

`>=`

greater than or equal

`!=`

different

`&`

and

`|`

or



# Typing Commands

## quotient and remainder

- To determine the **quotient** and **remainder** of a division, use the **floor division** and **modulo** operators, respectively

```
> 20 %/% 6 # quotient by floor division (%/%)
```

```
[1] 3
```

```
> 20 %% 6 # remainder by modulo (%%)
```

```
[1] 2
```



# Assignment Operation

- To assign a value to a variable use `<-` not `=`

```
> weight <- 72
```

```
> weight
```

```
[1] 72
```

- To assign within a function use `=`

```
> mean(x = 1:100, na.rm = TRUE)
```

```
[1] 50.5
```



# Assignment Example

```
> weight <- 72
> height <- 182
> BMI <- (weight)/(height/100)^2
> BMI
[1] 21.73651
> bmi
Error: object 'bmi' not found
```



# Functions (if else)

- Write a function for BMI

```
> BMI <- function(weight, height) {  
  if (height > 2.5) {  
    stop('height is not in meters')  
  } else (weight)/(height)^2  
}
```



# Data storage modes

- "logical" TRUE or FALSE
- "integer" 6L
- "double" 6 or 6.0
- "character" "m"
- "list"
- "function"
- "complex" 3.141593+0i # rare
- "raw" 2c 00 # rare





# Data structures

dim	Homogeneous	Heterogeneous
1d	Atomic Vector	List
2d	Matrix	Data frame
nd	Array	



# Data structures

dim	Homogeneous	Heterogeneous
1d	Atomic Vector	List
2d	Matrix	Data frame
nd	Array	



# Data structures:

## Atomic Vector

```
> a <- 8.4  
> b <- "foo"  
> str(a)  
  num 8.4  
> str(b)  
  chr "foo"  
> class(a)  
[1] "numeric"  
> class(b)  
[1] "character"
```

```
> typeof(a)  
[1] "double"  
> typeof(b)  
[1] "character"
```



# Data structures:

## Atomic Vector

- Atomic vectors are usually created with `c()`, short for combine:

```
> dbl_var <- c(1, 2.5, 4.5)
```

*# With the L suffix, you get an integer rather than a double*

```
> int_var <- c(1L, 6L, 10L)
```

*# Use TRUE and FALSE (or T and F) to create logical vectors*

```
> log_var <- c(TRUE, FALSE, T, F)
```

```
> chr_var <- c("foo", "bar")
```



# Data structures:

Atomic Vector: `is.atomic()`

```
> dbl_var <- c(1, 2.5, 4.5)
> typeof(dbl_var)
[1] "double"
> int_var <- c(1L, 6L, 10L)
> typeof(int_var)
[1] "integer"
> log_var <- c(TRUE, FALSE, T, F)
> typeof(log_var)
[1] "logical"
> chr_var <- c("foo", "bar")
> typeof(chr_var)
[1] "character"
```



# Data structures:

## Numeric Vector

```
> is.numeric(dbl_var)
```

```
[1] TRUE
```

```
> is.double(dbl_var)
```

```
[1] TRUE
```

```
> is.integer(dbl_var)
```

```
[1] FALSE
```

```
> is.numeric(int_var)
```

```
[1] TRUE
```

```
> is.integer(int_var)
```

```
[1] TRUE
```

```
> is.atomic(int_var)
```

```
[1] TRUE
```



# Data structures:

## Coercion

```
> c <- c("a", 1)
> str(c)
chr [1:2] "a" "1"
> is.vector(c)
[1] TRUE
> is.atomic(c)
[1] TRUE
> is.character(c)
[1] TRUE
> is.factor(c)
[1] FALSE
```



# Data structures

dim	Homogeneous	Heterogeneous
1d	Atomic Vector	List
2d	Matrix	Data frame
nd	Array	





# Data structures:

## Lists or Recursive Vector

- You construct lists by using `list()`

```
> d <- list(  
  1:10,  
  "foo",  
  c(TRUE, FALSE, TRUE, TRUE),  
  C(7.2, 16.8)  
)
```

```
> str(d)
```

List of 4

\$ : int [1:10] 1 2 3 4 5 6 7 8 9 10

\$ : chr "foo"

\$ : logi [1:4] TRUE FALSE TRUE TRUE

\$ : num [1:2] 7.2 16.8



# Data structures:

## Lists or Recursive Vector

```
> e <- list(list(list(list(1:10))))  
> e <- list(  
  list(  
    list(  
      list(  
        1:10  
      )  
    )  
  )  
)
```



# Data structures:

## Lists or Recursive Vector

```
> str(e)
List of 1
 $ :List of 1
  ..$ :List of 1
   .. ..$ :List of 1
    .. .. ..$ : int [1:10] 1 2 3 4 5 6 7 8 9 10
```



# Data structures:

## Lists or Recursive Vector

```
> f <- list(  
  "foo", list(  
    "bar", list(  
      "spam", list(  
        "eggs", 1:10  
      )  
    )  
  )  
)
```



# Data structures:

## Lists or Recursive Vector

```
> str(f)
List of 2
 $ : chr "foo"
 $ :List of 2
  ..$ : chr "bar"
  ..$ :List of 2
   .. ..$ : chr "spam"
   .. ..$ :List of 2
    .. . . . $ : chr "eggs"
    .. . . . $ : int [1:10] 1 2 3 4 5 6 7 8 9 10
```



# Data structures:

## Lists or Recursive Vector

```
> is.vector(f)
[1] TRUE
> is.atomic(f)
[1] FALSE
> is.recursive(f)
[1] TRUE
> is.list(f)
[1] TRUE
```



# Data structures:

## Coercion

```
> g <- list(  
  list(1, 2),  
  c(3, 4)  
)  
> h <- c(  
  list(1, 2),  
  c(3, 4)  
)
```

- What is the difference?



# Objects

- R can be written as
  - Object Oriented Programming (OOP)
- To list all objects in your current workspace

```
> ls()
```

```
[1] "a" "b" "BMI" "c" "chr_var" "d" "dbl_var" "e" "f"  
[10] "g" "h" "height" "int_var" "log_var" "weight" "x" "y"
```

- To remove some objects that are no longer needed :

```
> rm(height, weight)
```

```
> ls()
```

```
[1] "a" "b" "BMI" "c" "chr_var" "d" "dbl_var" "e" "f"  
[10] "g" "h" "int_var" "log_var" "x" "y"
```





# Attributes

- All objects can have arbitrary additional attributes
  - used to store **metadata** about the object
- Use `structure()` function to return a new object with modified attributes

```
> i <- structure(  
  .Data = 1:10,  
  description = "This is a vector",  
  label = "id of people"  
)
```



# Attributes ...

```
> i
[1] 1 2 3 4 5 6 7 8 9 10
attr(,"description")
[1] "This is a vector"
attr(,"label")
[1] "id of people"

> str(i)
atomic [1:10] 1 2 3 4 5 6 7 8 9 10
- attr(*, "description")= chr "This is a vector"
- attr(*, "label")= chr "id of people"

> typeof(i)
[1] "integer"
```



# Attributes ...

```
> attr(i, "label")  
[1] "id of people"  
  
> attributes(i)  
$description  
[1] "This is a vector"  
  
$label  
[1] "id of people"
```



# Attributes ...

- Stable attributes
  - Names
    - a character vector giving each element a name
  - Dimensions
    - used to turn vectors into matrices and arrays
  - Class
    - used to implement the S3 object system



# Names

```
> k <- structure(  
  .Data = 1:4,  
  "names" = c("a", "b", "c", "d")  
)  
> k  
a b c d  
1 2 3 4  
> str(k)  
Named int [1:4] 1 2 3 4  
- attr(*, "names")= chr [1:4] "a" "b" "c" "d"  
> typeof(k)  
[1] "integer"
```



# Names ...

```
> l <- c(  
  a = 1, b = 2, c = 3, d = 4  
)  
> l  
a b c d  
1 2 3 4  
> str(l)  
Named num [1:4] 1 2 3 4  
- attr(*, "names")= chr [1:4] "a" "b" "c" "d"  
> typeof(l)  
[1] "double"  
> names(l)  
[1] "a" "b" "c" "d"
```



# Factors

- One important use of *attributes* is to define factors.
- A factor is a vector that can contain only predefined values
  - used to store *categorical* data.
- Factors are built on top of *integer* vectors using two attributes:
  - the class *“factor”*
    - which makes them behave differently from regular integer vectors
  - the *levels*
    - which defines the set of allowed values.



# Factors ...

```
> m <- factor(
  x = c(0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0),
  levels = c(0, 1),
  labels = c("male", "female"))
)
> class(m)
[1] "factor"
> typeof(m)
[1] "integer"
> str(m)
Factor w/ 2 levels "male","female": 1 2 1 1 1 2 2
2 1 2 ...
> summary(m)
male female
    7     6
```





# Dimensions

```
> n <- structure(  
  .Data = 1:10,  
  "dim" = c(2, 5),  
  "dimnames" = list(  
    c("foo", "bar"), c("a", "b", "c", "d", "e")  
  )  
)  
  
> n  
      a b c d e  
foo 1 3 5 7 9  
bar 2 4 6 8 10  
  
> class(n)  
[1] "matrix"  
  
> typeof(n)  
[1] "integer"
```



# Data structures

dim	Homogeneous	Heterogeneous
1d	Atomic Vector	List
2d	Matrix	Data frame
nd	Array	



# Matrices and arrays

- Adding a "dim" attribute to an atomic vector
  - allows it to behave like a *multi-dimensional array*.
- A special case of the array is the **matrix**,
  - which has 2 dimensions.

*# Two scalar arguments to specify rows and columns*

```
> o <- matrix(1:6, ncol = 3, nrow = 2)
```

*# One vector argument to describe all dimensions*

```
> p <- array(1:12, c(2, 3, 2))
```



# Matrices and arrays ...

*# Two scalar arguments to specify rows and columns*

```
> o <- matrix(1:6, ncol = 3, nrow = 2)
```

*# One vector argument to describe all dimensions*

```
> p <- array(1:12, c(2, 3, 2))
```

```
> o
```

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

```
> p
```

```
, , 1
```

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

```
, , 2
```

	[,1]	[,2]	[,3]
[1,]	7	9	11
[2,]	8	10	12



# Matrices and arrays ...

```
> o <- matrix(1:6, ncol = 3, nrow = 2)
> dim(o)  # gets the dimensions of matrix o
[1] 2 3
> o[2,1]  # subsets the item in row 2, column 1
[1] 2
> o[2,]   # subsets all items in row 2
[1] 2 4 6
> t(o)    # transposes matrix o
      [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
> o %*% t(o)  # multiplies matrix o by its transpose
      [,1] [,2]
[1,]   35   44
[2,]   44   56
```



# Data structures

dim	Homogeneous	Heterogeneous
1d	Atomic Vector	List
2d	Matrix	Data frame
nd	Array	



# Data frames

- A data frame is
  - the most common way of storing data in R
  - a **list** of equal-length **vectors**
  - 2-dimensional structure
  - shares properties of both **matrix** and **list**
- Create a data frame using **data.frame()**
  - which takes named vectors as input

```
> df <- data.frame(  
  id = 1:3,  
  gender = c("m", "f", "m"),  
  fbs = c(104, 98, 129)  
)
```

```
> str(df)
```

```
'data.frame':      3 obs. of  3 variables:
```

```
$ id      : int  1 2 3
```

```
$ gender: Factor w/ 2 levels "f","m": 2 1 2
```

```
$ fbs     : num  104 98 129
```



# Data frames ...

```
> df
  id gender fbs
1  1      m 104
2  2      f  98
3  3      m 129
> class(df)
[1] "data.frame"
> typeof(df)
[1] "list"
```





## Import Data frames

- import/export requires addressing the path
  - Importing with **absolute path**

```
> df <- read.csv( # comma separated text
  File = "/home/maanib/Documents/rprojects/teacheR/data/df.csv",
  fileEncoding = "UTF-8",
  header = TRUE # column names
)
```

```
> str(df)
'data.frame':      3 obs. of  4 variables:
 $ X      : int  1 2 3
 $ id     : int  1 2 3
 $ gender: Factor w/ 2 levels "f","m": 2 1 2
 $ fbs    : int  104 98 129
```



## Import Data frames

- import/export requires addressing the path
  - Importing with **relative path**
- Make a **config.R** in project's directory

- **Linux**

```
> dir <- "/home/maanib/Documents/rprojects/teacherR"
```

- **Windows**

```
> dir <- "E:\\Archive\\maanib\\rprojects\\teacherR"
```

```
> source(config.R)
```

```
> df <- read.csv( # comma separated text
```

```
  file = paste0( # the string concatenate function
```

```
    dir, # the directory address of project files
```

```
    "/data/df.csv" # the relative address
```

```
  ),
```

```
  fileEncoding = "UTF-8",
```

```
  header = TRUE # column names
```

```
)
```



## Import Data frames ...

```
> source(config.R)
> df <- read.table( # plain text document
  file = paste0(dir, "/data/df.txt"),
  fileEncoding = "UTF-8",
  header = TRUE # column names
)
```



## Export Data frames

```
> source(config.R)
> write.csv(  # comma separated text
  x = df,
  file = paste0(dir, "/output/df.csv"),
  fileEncoding = "UTF-8"
)
> write.table(  # plain text document
  x = df,
  file = paste0(dir, "/output/df.txt"),
  fileEncoding = "UTF-8",
  sep = "|",  # separator character
  col.names = TRUE  # column names
)
```



# Missing Values in R

- Values that are “Not Available” are called missing
  - They are indicated as **NA** in R

```
> p <- c(1, 2, NA, 25, 16)
```

```
> p
```

```
[1] 1 2 NA 25 16
```

```
> mean(p)
```

```
[1] NA
```

- Use **na.rm = TRUE** argument to strip **NAs** before the computation proceeds

```
> mean(p, na.rm = TRUE)
```

```
[1] 11
```



# Subsetting in R

- There are two main methods for subsetting
  - The first method deals with `[]` and `$`

```
> df[2, 4] # subsets the element in row 2 column 4
```

```
[1] 98
```

```
> df[, 3] # subsets the elements in column 3
```

```
[1] m f m
```

```
Levels: f m
```

```
> df$id # subsets all elements of id field
```

```
[1] 1 2 3
```

```
> df[which(df$gender == "m"), ] # subsets male gender
```

```
  X id gender fbs
```

```
1 1  1      m 104
```

```
3 3  3      m 129
```



## Subsetting in R ...

- There are two main methods for subsetting
  - The second method deals with `subset()` function

```
> subset( # subsets the element in row 2 column 4
```

```
  df,  
  row.names(df) == 2,  
  select = 4
```

```
)
```

```
  fbs
```

```
2  98
```

```
> subset(df, select = 3) # subsets the elements in column 3
```

```
> subset(df, select == "id") # subsets elements of id field
```

```
> subset(df, gender == "m") # subsets male gender
```

```
  X id gender fbs
```

```
1 1  1      m 104
```

```
3 3  3      m 129
```



# Conditional Expressions & Loops

```
if (condition1) {  
  statement1  
} else if (condition2) {  
  statement2  
} else {  
  statement3  
}
```

```
> if (height > 2.5) {  
  stop('height is not in meters')  
} else (weight)/(height)^2
```





# Conditional Expressions & Loops

```
for (i in 1:n) {  
  a group of statements  
}
```

- Example: Let us transform fbs values to diabetes categories

```
df$diabetes <- 0  
for (i in 1:length(df$id)) {  
  if (df$fbs[i] < 100) {  
    df$diabetes[i] <- "Normal"  
  } else if (df$fbs[i] >= 100 & df$fbs[i] < 125) {  
    df$diabetes[i] <- "Prediabetes"  
  } else {  
    df$diabetes[i] <- "Diabetes"  
  }  
}
```

```
> df
```

	X	id	gender	fbs	diabetes
1	1	1	m	104	Prediabetes
2	2	2	f	98	Normal
3	3	3	m	129	Diabetes



# Conditional Expressions & Loops

- Other loops:

- R while Loop

```
i <- 1
while (i < 6) {
  print(i)
  i = i + 1
}
```

- R next Statement

```
for (val in x) {
  if (val == 3) {
    next
  }
  print(val)
}
```

- Other loops:

- R break statement

```
for (val in x) {
  if (val == 3) {
    break
  }
  print(val)
}
```

- R repeat loop

```
x <- 1
repeat {
  print(x)
  x = x + 1
  if (x == 6) {
    break
  }
}
```



# Functions

- An efficient way *to do the same task repeatedly* is to create a **function**:

```
function(arglist) {  
  body  
}
```

- Example: convert centimeter to inch

```
cent.to.inch <- function(x) {  
  inch <- x * 0.393701  
  return(inch)  
}  
> cent.to.inch(20)  
[1] 7.87402
```





# Part 2

R statistical analyses



# Install Packages

- A package is a collection of **functions**, often for performing specific tasks.
  - Many scientists provide their novel methods as R packages and share with others
- For CRAN packages
  - `install.packages("package.name")`
- For github packages
  - First `install.packages("devtools")`
  - `devtools::install_github("package.name")`
- To load packages and use them
  - `library(package.name)` or `require(package.name)`
- To cite the packages
  - Use `citation("package.name")` to acquire the bibliographic information



# Univariate Statistics

- To provide the summary statistics of all the variables in a data set, use the `summary()`

```
> summary(iris) # summary of all variables
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
Min. :4.300	Min. :2.000	Min. :1.000	Min. :0.100	setosa :50
1st Qu.:5.100	1st Qu.:2.800	1st Qu.:1.600	1st Qu.:0.300	versicolor:50
Median :5.800	Median :3.000	Median :4.350	Median :1.300	virginica :50
Mean :5.843	Mean :3.057	Mean :3.758	Mean :1.199	
3rd Qu.:6.400	3rd Qu.:3.300	3rd Qu.:5.100	3rd Qu.:1.800	
Max. :7.900	Max. :4.400	Max. :6.900	Max. :2.500	

```
> summary(cars$speed) # summary of one variable
```

- Other useful functions for descriptive statistics

```
mean(), median(), sd(), max(), min(),  
IQR(), range(), quantile(), length()
```



# Measures of Association

```
> cor(trees$Girth, trees$Volume)
[1] 0.9671194

> cor(trees$Girth, trees$Volume, method = "spearman")
[1] 0.9547151

> cor.test(trees$Girth, trees$Volume)

Pearson's product-moment correlation

data: trees$Girth and trees$Volume
t = 20.478, df = 29, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.9322519 0.9841887
sample estimates:
      cor
0.9671194
```



# Measures of Association

```
> install.packages("scales")
> library(scales)
> r <- cor.test(trees$Girth, trees$Volume)
> pvalue(
  r$p.value,
  accuracy = 0.0001,
  decimal.mark = ".",
  add_p = TRUE
)
[1] "p<0.0001"
```





# Probability Distributions

- Probability density function (**d**)
  - `dnorm()`, `dchisq()`, `dt()`, `df()`
- Cumulative distribution function (**p**)
  - `pnorm()`, `pchisq()`, `pt()`, `pf()`
- Inverse cumulative distribution function (**q**)
  - `qnorm()`, `qchisq()`, `qt()`, `qf()`
- Random number generator (**r**)
  - `rnorm()`, `rchisq()`, `rt()`, `rf()`



# Probability Distributions ...

- Probability density function (**d**)

- `dnorm()`, `dchisq()`, `dt()`, `df()`

```
> z_scores <- seq(-3, 3, by = 0.1)
```

```
[1] -3.0 -2.9 -2.8 -2.7 -2.6 -2.5 -2.4 -2.3 -2.2 -2.1 -2.0 -1.9 -1.8 -1.7 -1.6 -1.5  
[17] -1.4 -1.3 -1.2 -1.1 -1.0 -0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1  0.0  0.1  
[33]  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1.0  1.1  1.2  1.3  1.4  1.5  1.6  1.7  
[49]  1.8  1.9  2.0  2.1  2.2  2.3  2.4  2.5  2.6  2.7  2.8  2.9  3.0
```

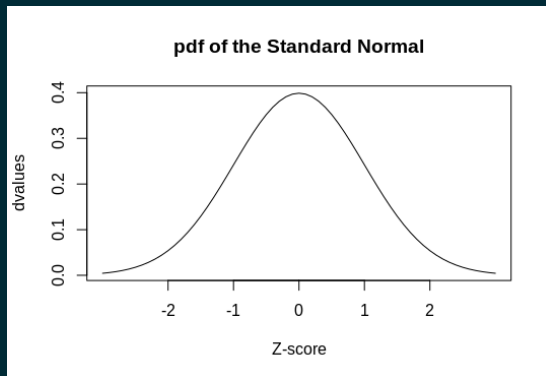
```
> dvalues <- dnorm(z_scores)
```

```
[1] 0.004432 0.005953 0.007915 0.010421 0.013583 0.017528 0.022395 0.028327 0.035475  
[10] 0.043984 0.053991 0.065616 0.078950 0.094049 0.110921 0.129518 0.149727 0.171369  
[19] 0.194186 0.217852 0.241971 0.266085 0.289692 0.312254 0.333225 0.352065 0.368270  
[28] 0.381388 0.391043 0.396953 0.398942 0.396953 0.391043 0.381388 0.368270 0.352065  
[37] 0.333225 0.312254 0.289692 0.266085 0.241971 0.217852 0.194186 0.171369 0.149727  
[46] 0.129518 0.110921 0.094049 0.078950 0.065616 0.053991 0.043984 0.035475 0.028327  
[55] 0.022395 0.017528 0.013583 0.010421 0.007915 0.005953 0.004432
```



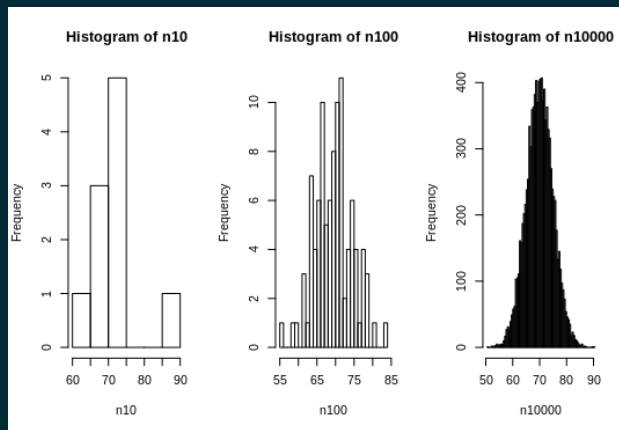
# Probability Distributions ...

```
> plot( # Plot where y = values and x = z_score indexes
      dvalues,
      xaxt = "n", # Don't label the x-axis
      type = "l", # Make it a line plot
      main = "pdf of the Standard Normal",
      xlab = "Z-score"
    )
# These commands label the x-axis
axis(1, at = which(dvalues == dnorm(0)), labels = c(0))
axis(1, at = which(dvalues == dnorm(1)), labels = c(-1, 1))
axis(1, at = which(dvalues == dnorm(2)), labels = c(-2, 2))
```



# Probability Distributions ...

```
> pnorm(1.96)
[1] 0.975
> qnorm(0.975)
[1] 1.96
> qnorm(0.975)
```



```
> n10 <- rnorm(10, mean = 70, sd = 5)
> n100 <- rnorm(100, mean = 70, sd = 5)
> n10000 <- rnorm(10000, mean = 70, sd = 5)
> par(mfrow = c(1, 3))
# The breaks argument specifies how many bars are in the histogram
> hist(n10, breaks = 5)
> hist(n100, breaks = 20)
> hist(n10000, breaks = 100)
```



# Checking the Normality Assumption

- Shapiro-Wilk Test

```
> shapiro.test(iris$Sepal.Width)
```

Shapiro-Wilk normality test

data: iris\$Sepal.Width

W = 0.98, p-value = 0.1

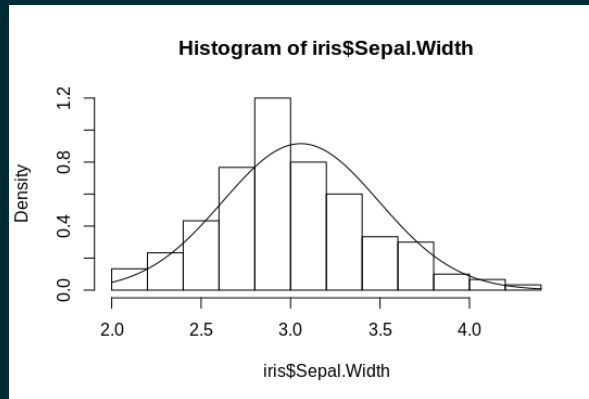
```
> hist(iris$Sepal.Width, freq = FALSE)
```

```
> curve(
```

```
  dnorm(x, mean(iris$Sepal.Width), sd(iris$Sepal.Width)),
```

```
  add = TRUE
```

```
)
```



# Analyzing tabular data

```
> str(infert)
> help(infert)
> install.packages("dplyr")
> require("dplyr")
> infert.labels <- list("0" = "Control", "1" = "Case")
> infert$status <- dplyr::recode(infert$case, !!!infert.labels)
> infertility.table <- table(
  infert$status, (infert$induced + infert$spontaneous)
)
```

	0	1	2	3
Case	7	34	32	10
Control	60	58	42	5

```
> chisq.test(infertility.table)
```

Pearson's Chi-squared test

data: infertility.table

X-squared = 27.048, df = 3, p-value = 5.751e-06



# Analyzing tabular data ...

```
> fit <- glm(  
  case ~ induced + spontaneous,  
  family = binomial(link = logit), # logistic regression  
  data = infert  
)  
> fit$rsquare <- 1 - (fit$deviance/fit$null.deviance)  
> summary(fit)
```

Coefficients:

	Estimate	Std. Error	z value	Pr(> z )	
(Intercept)	-1.7079	0.2677	-6.380	1.78e-10	***
induced	0.4181	0.2056	2.033	0.042	*
spontaneous	1.1972	0.2116	5.657	1.54e-08	***

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

```
> fit$rsquare  
[1] 0.1156308
```



# Analyzing tabular data ...

- We want **odds ratios** instead of coefficients

```
> install.packages("questionr")
```

```
> require("questionr")
```

```
> odds.ratio(fit)
```

	OR	2.5 %	97.5 %	p	
(Intercept)	0.18125	0.10462	0.2999	1.776e-10	***
induced	1.51912	1.01599	2.2824	0.04201	*
spontaneous	3.31085	2.21211	5.0865	1.543e-08	***

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1





# Comparison of Means: t-test

```
> iris2 <- iris[iris$Species %in% c(
  "versicolor", "virginica"
), ]
> iris2 <- iris2 <- subset(
  iris, Species == "versicolor" | Species == "virginica"
)
> t.test(Sepal.Width ~ Species, data = iris2)

Welch Two Sample t-test

data: Sepal.Width by Species
t = -3.2058, df = 97.927, p-value = 0.001819
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.33028364 -0.07771636
sample estimates:
mean in group versicolor  mean in group virginica
                2.770                2.974
```



# Comparison of Means: ANOVA

```
> levels(PlantGrowth$group)
[1] "ctrl" "trt1" "trt2"
> boxplot(weight ~ group, data = PlantGrowth)
> anova(aov(weight ~ group, data = PlantGrowth))
```

Analysis of

Response: weight

Deviance

group

Residuals 27

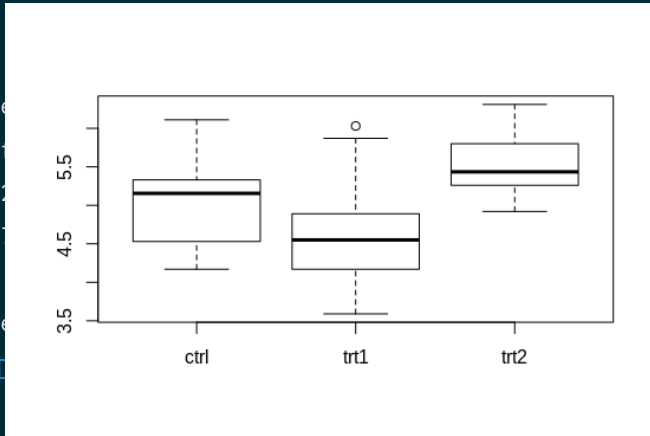
---

Signif. codes:

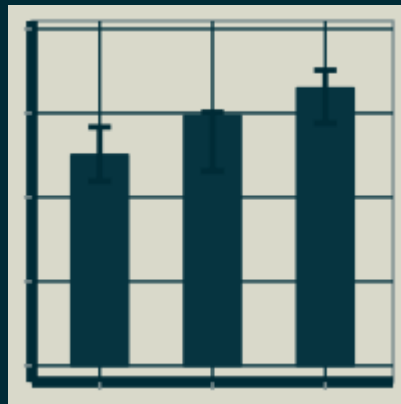
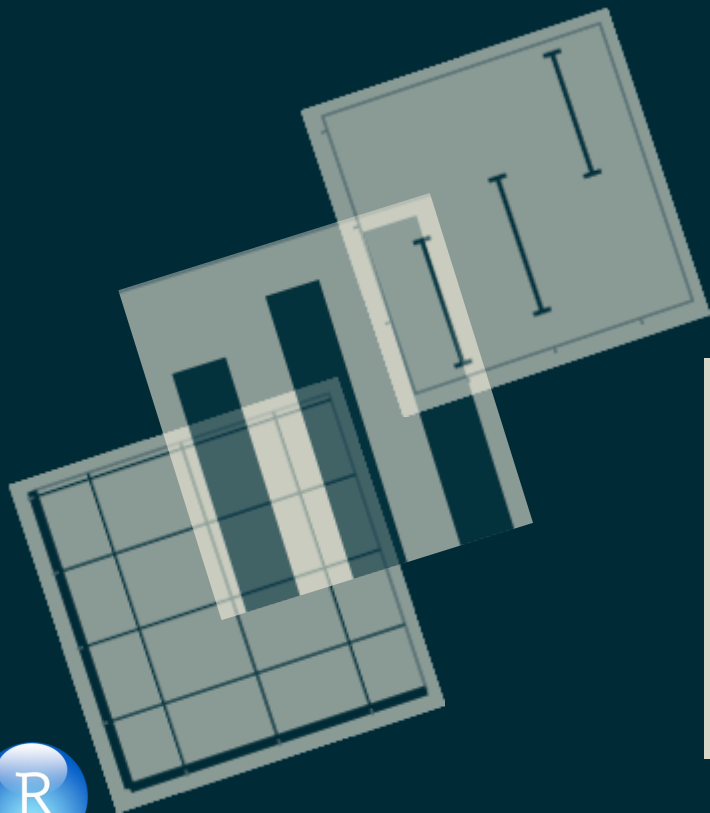
```
> TukeyHSD
```

\$group

	diff	lwr	upr	p adj
trt1-ctrl	-0.371	-1.0622161	0.3202161	0.3908711
trt2-ctrl	0.494	-0.1972161	1.1852161	0.1979960
trt2-trt1	0.865	0.1737839	1.5562161	0.0120064

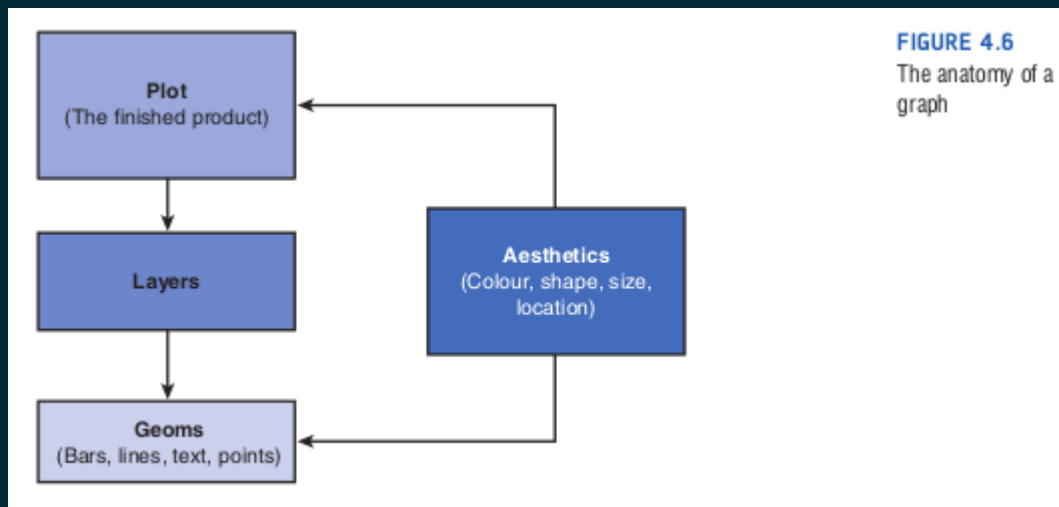


# Magical graphics in R



# Magical graphics in R

Many people migrate to R *only* because of its great *graphic capabilities*.



Field A, Miles J, Field Z. Discovering statistics using R. Sage publications; 2012.



# R Graphics: geometric options

- `geom_bar()`
  - creates a layer with bars representing different statistical properties
- `geom_point()`
  - creates a layer showing the data points
    - as you would see on a `scatter plot`.
- `geom_line()`
  - creates a layer that connects data points with a straight line
- `geom_smooth()`
  - creates a layer that contains a ‘smoother’
    - i.e., a line that summarizes the data as a whole rather than connecting individual data points



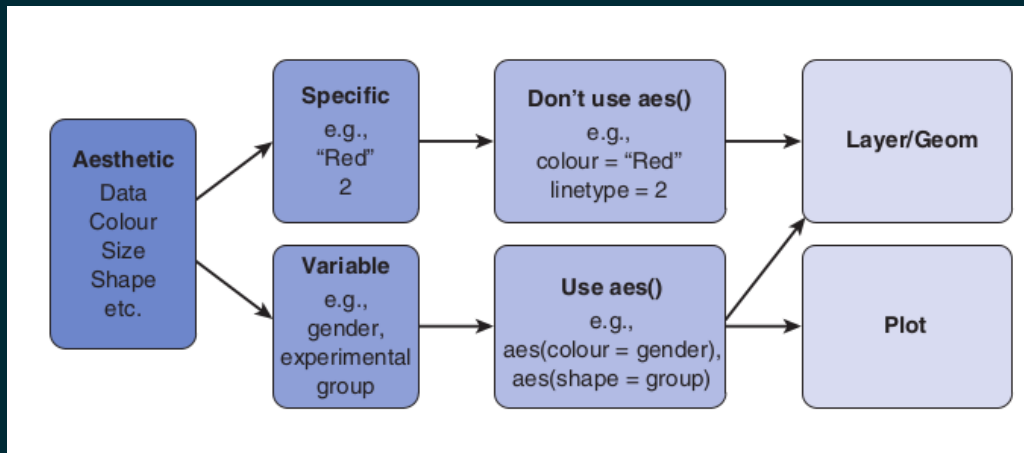
# R Graphics: geometric options

- `geom_histogram()`
  - creates a layer with a histogram on it
- `geom_boxplot()`
  - creates a layer with a box-whisker diagram
- `geom_text()`
  - creates a layer with text on it
- `geom_density()`
  - creates a layer with a density plot on it
- `geom_errorbar()`
  - creates a layer with error bars displayed on it
- `geom_hline()` and `geom_vline()`
  - create a layer with a user-defined horizontal or vertical line, respectively



# R Graphics: aesthetics

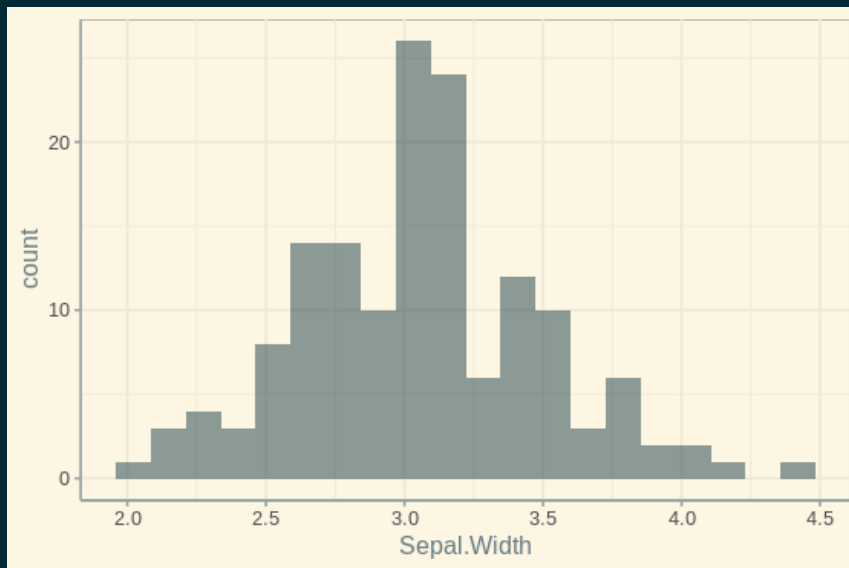
- `aes()`



Field A, Miles J, Field Z. Discovering statistics using R.  
Sage publications; 2012.

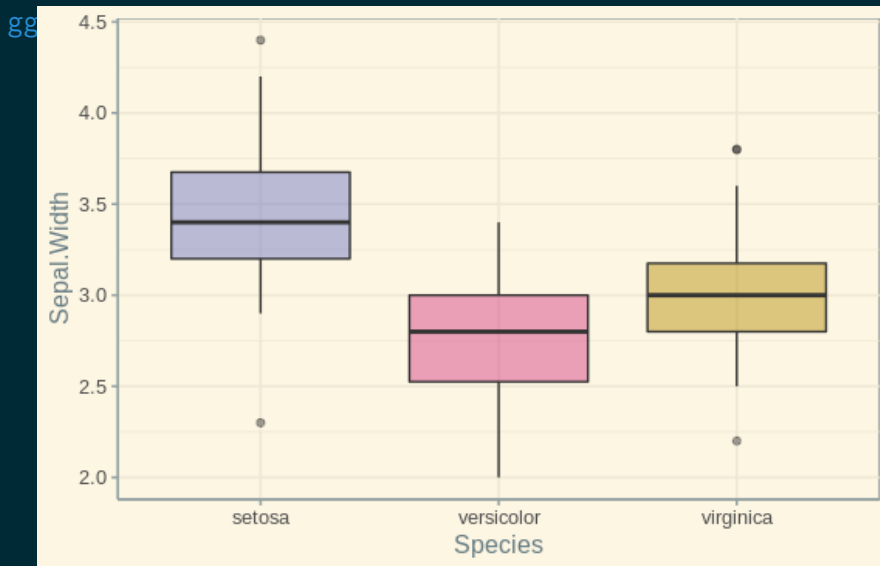


# R Graphics: example





# R Graphics: example

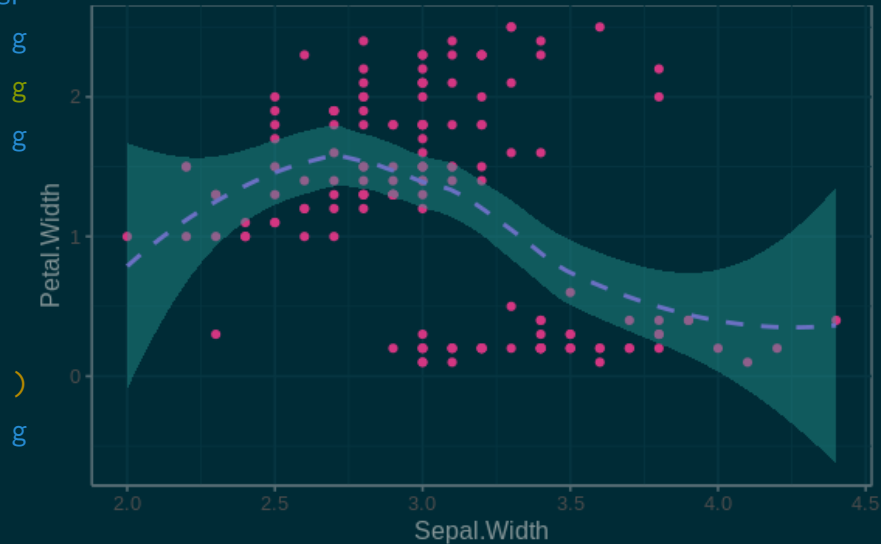


b58900")) +



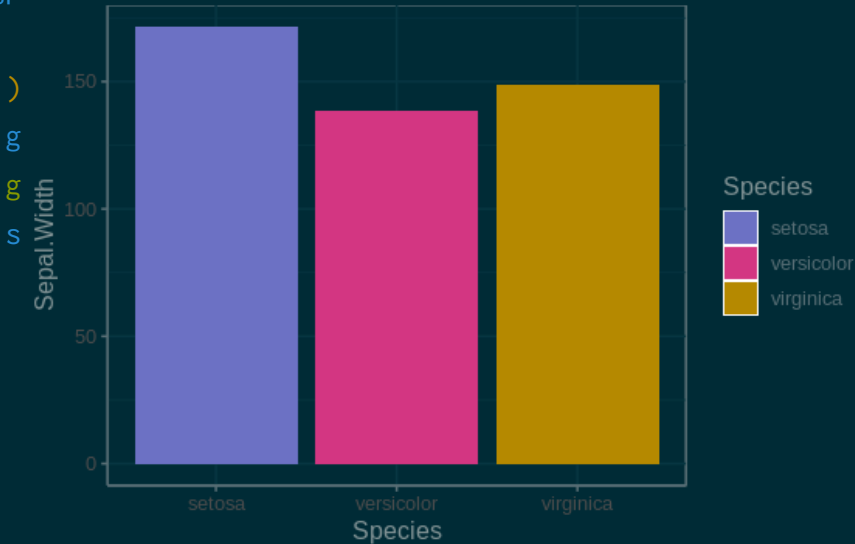
# R Graphics: example

```
ggplot(data = iris, aes(x = Sepal.Width, y = Petal.Width)) +
```



# R Graphics: example

ggplot2

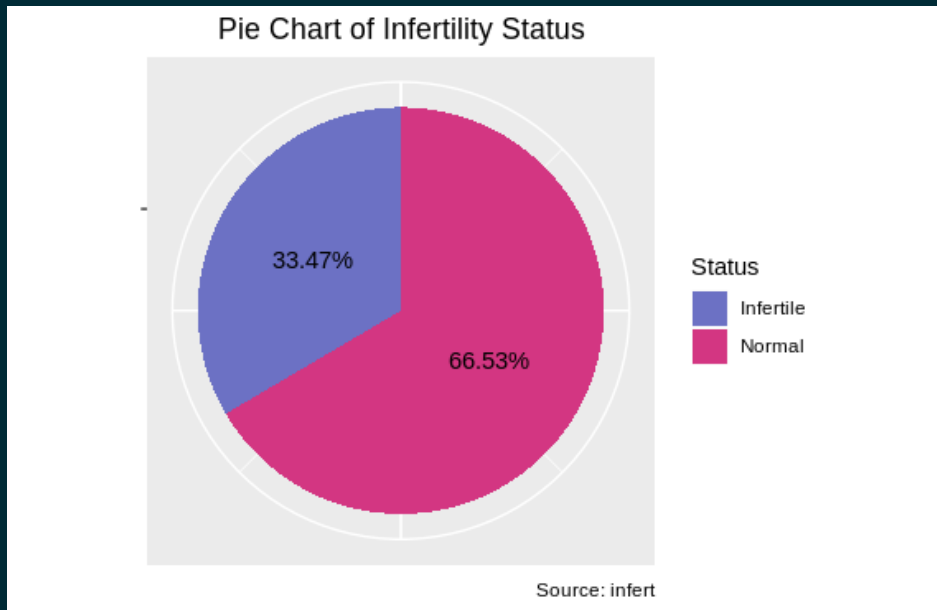


Species)

58900" ))



# R Graphics: exercise



# References

1. Wickham H. Advanced r. Chapman and Hall/CRC; 2014.
2. Keramat Nouri, Introductory R. Workshop Slides. 2018.
3. Field A, Miles J, Field Z. Discovering statistics using R. Sage publications; 2012.

