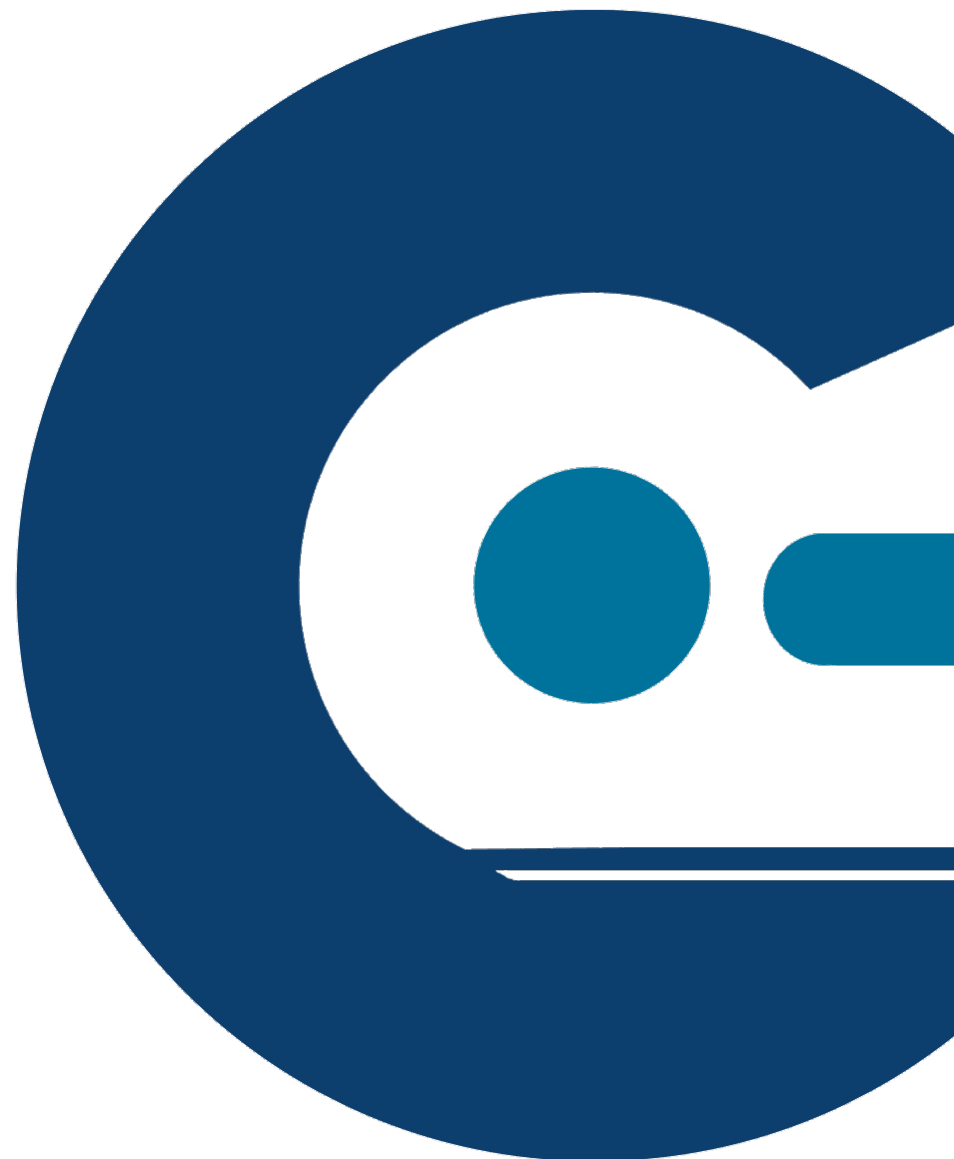


Introduction to R programming

February 8, 2023





- Morning session
 - What is R
 - Basic data types and commands
 - Matrices and data frames
 - Simple programming (loops and conditional statements)
- Afternoon
 - Visualizations (bar charts, box plots, scatter plots)
 - Basic statistics
 - Principal component analysis
 - Heatmaps

Introduction to R programming



- R is a programming language for statistical computing and data visualization.
 - Available for Windows, macOS and Linux platforms
- Free available at <https://www.r-project.org/>



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.

R console



To start R console in Windows,
click the R icon



```
R Console

R version 3.1.1 (2014-07-10) -- "Sock it to Me"
Copyright (C) 2014 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.

> |
```

To start R console in macOS or Linux,
run command “R” in terminal

```
ubuntu@ubuntu: ~$ R

R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"
Copyright (C) 2015 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.

[Previously saved workspace restored]

> |
```

The **>** is the R command prompt
Type **q()** to exit R console

RStudio



Free and open-source integrated development environment (IDE) for R

<https://www.rstudio.com/>

The screenshot shows the RStudio IDE interface with the following components and annotations:

- Code editor/data preview:** The top-left pane shows a script with three lines of R code:

```
1 a <- 1
2 b <- 2
3 a + b
```
- Variables, data, History of commands:** The top-right pane shows the 'Environment' tab with a table of variables:

Variable	Value
a	1
b	2
- Files, plots, packages, help:** The bottom-right pane shows a file explorer view of the 'Home' directory with a list of files and folders.
- R console to execute code:** The bottom-left pane shows the 'Console' tab with the R startup message and a prompt for user input.

General usage notes for R (1)



Use **TAB** to autocomplete filenames, tools/functions, and variables.
Double-tap TAB to get a list of all possible auto-completions.

Use **up/down arrows** to recall and scroll through previously entered commands.

Use **left/right arrows** to move through a command to edit it.

> is R command prompt in R console; just like \$ in Linux.

is used for comments (notes to yourself). Anything in the line after # will be ignored by the system.

There is no spellcheck, and **capitalization matters**, e.g. x is different from X.

Use **STOP** icon  to stop a command.

q() to quit R console or RStudio

General usage notes for R (2)



- **Variable** is used to store a value that can be changed in a computer program, e.g. `x = 1`; `x = 100`.
- **Function** is a piece of common codes that can be executed many times, e.g. `plot(x, y)`.
- Current R working directory on your computer:
`getwd()`: get current working directory.
`setwd()`: set current working directory.
- **Count/index** for R objects starts with 1, some other programming languages may use 0 as start.
- **Interactive** mode: run commands immediately in R console.
- **Script** mode: write R commands into a script file and execute the script.

Exercise Handouts



Instructions { Create a new script.

1. From the *File* menu select *New File > R Script*
2. Save file as “my_Rscript.R”

Code block { Write the following commands in code editor of R Studio and run them using icon **Run** in R Studio

```
# Performing simple calculations
10+6
```

Expected output {

```
## [1] 16
10*6
## [1] 60
10/6
## [1] 1.666667
2**4 # 2 to the power of 4; same as 2^4
```

Comment

When performing the exercises, you should only type the code blocks.

Note! Color of the text in your terminal may differ.
The color of the text has no effect on the execution of the code.

- What is the best way to organize your data?
 - Long vs. wide format
 - Attribute table vs. counts/data table
 - Value types:
 - Number vs. Text/Character vs. Logical/Boolean (T, F)
 - Continuous vs. categorical/discrete values

Group_1	Group_2	Group_3	Group_4
60.8	68.3	102.6	87.9
57.1	67.7	102.2	84.7
65	74	100.5	83.2
58.7	66.3	97.5	85.8
61.8	69.9	98.9	90.3

Wide format



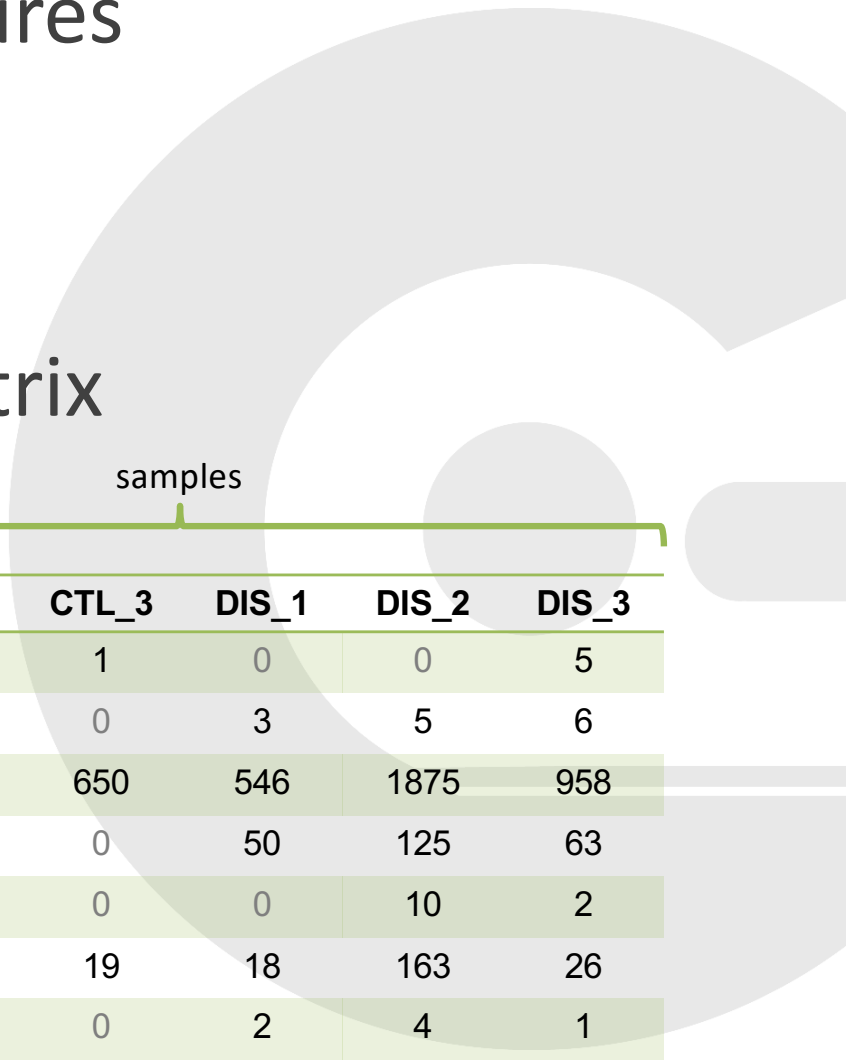
Sample	Weight	Group
SA1	68.3	1
SA2	68.3	2
SA3	102.6	3
SA4	87.9	4
SA5	57.1	1
SA6	67.7	2

Long format

Counts/data table



- Rows are observations/features
- Columns are samples
- Each cell is a count
- Can be thought of as 2D matrix



observations	samples					
	CTL_1	CTL_2	CTL_3	DIS_1	DIS_2	DIS_3
	ENSMUST00000196221.1	6	0	1	0	5
	ENSMUST00000179664.1	24	0	0	3	6
	ENSMUST00000178537.1	155	212	650	546	1875
	ENSMUST00000178862.1	0	0	0	50	125
	ENSMUST00000179520.1	60	0	0	10	2
	ENSMUST00000179883.1	205	19	19	18	163
	ENSMUST00000179932.1	11	1	0	2	4
	ENSMUST00000180001.1	0	83	7	47	14

Attribute table



- Each row is a different sample
- Attributes/factors/covariates are different columns
- Factors/covariates can be **continuous** or **discrete/categorical**

Sample	bwt	gestation	parity	age	height	weight	smoke
1	120	284	0	27	62	100	0
2	113	282	0	33	64	135	0
3	128	279	0	28	64	115	1
4	108	282	0	23	67	125	1
5	136	286	0	25	62	93	0
6	138	244	0	33	62	178	0
7	132	245	0	23	65	140	0
8	120	289	0	25	62	125	0
9	143	299	0	30	66	136	1

Breakdown of attribute table



- Each column can be thought of as a collection of values (vector)
- Each vector has a different data type (continuous number, discrete factor, etc.)
- The order (position) of values in each vector is the same for the same sample

Sample	bwt	gestation	parity	age	height	weight	smoke
1	120	284	0	27	62	100	0
2	113	282	0	33	64	135	0
3	128	279	0	28	64	115	1
4	108	282	0	23	67	125	1
5	136	286	0	25	62	93	0
6	138	244	0	33	62	178	0
7	132	245	0	23	65	140	0



Data types and basic commands

Learning Objectives

- Recognize different data types
- Manipulate (set/get) variables in R
- Execute basic R commands





- Numeric
1, 33, 1.7, 1e-3 (i.e. 0.003)
- Character
"hello world"
- Logical
TRUE (T), FALSE (F)

Variable name: start with letters, followed by digits, letters, . or _

For example...

```
> my.number <- 1.33 #numeric
> Modell.name <- "full model"
> is_duplicate <- T
> ls() #list objects in the memory
[1] "is_duplicate" "Modell.name"  "my.number"
```

What is R doing?



```
> 2+4  
[1] 6
```

The **[1]** means the first element of a vector
Even a single number in R is a vector, so “6” is a vector of size 1

```
> x <- 7
```

<- means “assign” in this case “assign the value 7 to the variable x”
Some use **=** for this purpose

```
> x + 19  
[1] 26
```

Adding 19 to “x” gives the expected value of 26

```
> X + 10
```

```
Error: object 'X' not found
```

X is not the same as x
R is **case sensitive**: upper case letters are
different to lower case letters

Some simple R commands



```
> 2+2
```

```
[1] 4
```

Result of the command

```
> 3^2
```

```
[1] 9
```

```
> sqrt(25)
```

```
[1] 5
```

```
> 2*(1+1)
```

```
[1] 4
```

```
> 2*1+1
```

Order of precedence

```
[1] 3
```

```
> exp(1)
```

```
[1] 2.718282
```

```
> log(2.718282)
```

```
[1] 1
```

Optional argument

```
> log(10, base=10)
```

```
[1] 1
```

```
> log(10
```

```
+
```

```
, base = 10)
```


```
[1] 1
```

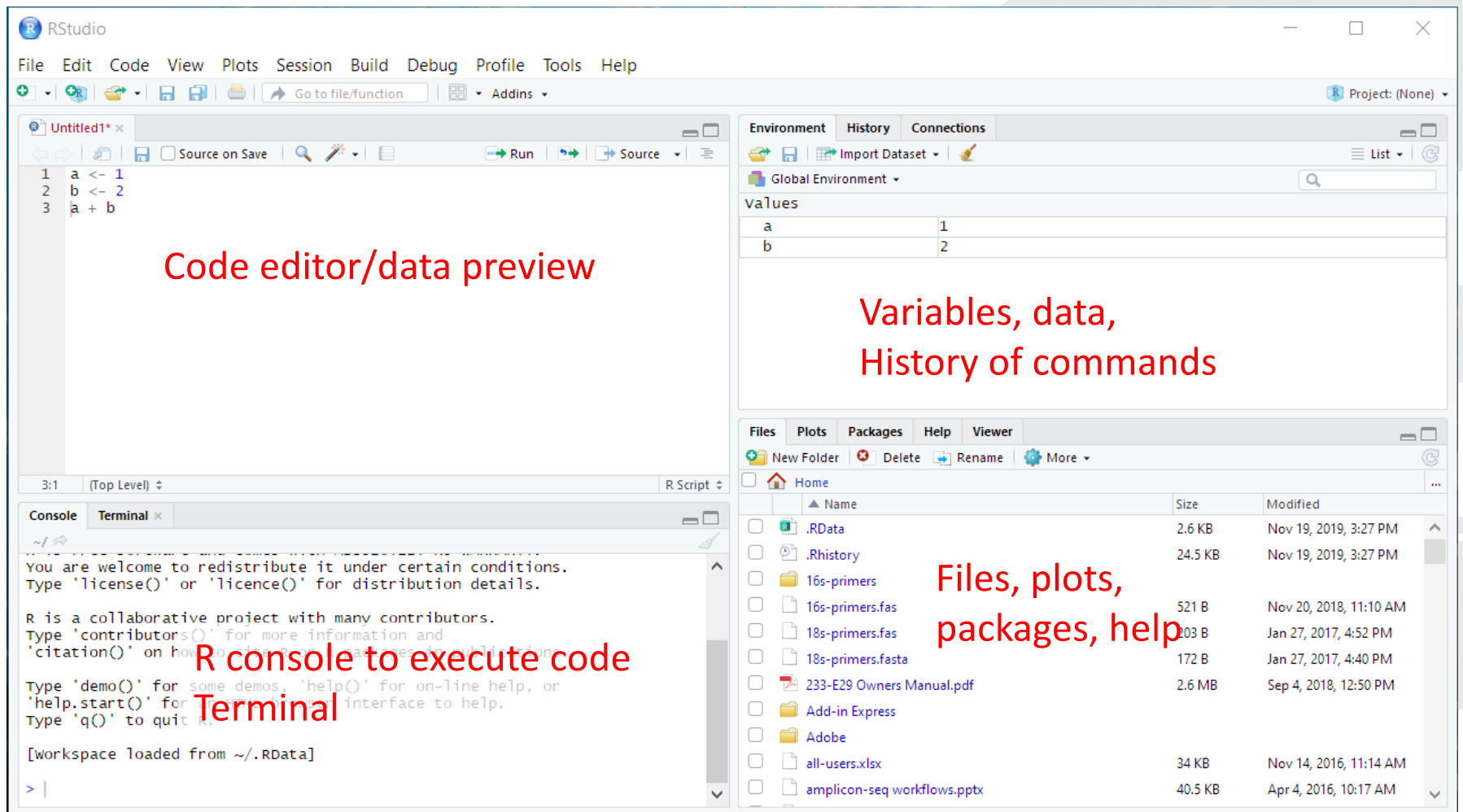
Incomplete command

EXERCISE 1: R studio and simple R commands



Type commands in code editor (top left)

To execute: click  **Run** or **command+return** (Mac) or **control+return** (PC) to run the command on the cursor's line, or a highlighted block of code



The screenshot shows the RStudio interface with four main panels. The top-left panel is the Code Editor, showing a script with three lines of R code: `1 a <- 1`, `2 b <- 2`, and `3 a + b`. The top-right panel is the Environment pane, showing the Global Environment with two variables: 'a' with value 1 and 'b' with value 2. The bottom-left panel is the Console, showing the R startup message and the prompt `>`. The bottom-right panel is the Files pane, showing a file explorer view of the home directory with various files and folders. Red text annotations are overlaid on the image to identify the panels: 'Code editor/data preview' points to the Code Editor, 'Variables, data, History of commands' points to the Environment pane, 'Files, plots, packages, help' points to the Files pane, and 'R console to execute code Terminal' points to the Console.

Code editor/data preview

Variables, data,
History of commands

Files, plots,
packages, help

R console to execute code
Terminal

A **vector** is a one-dimensional collection of the same type of objects.

- Use **c()** function to create a vector.
- “**c()**” is a function that concatenates values together into a vector.

```
> x <- c(7.8, 9.0, 7.1, 8.8)
```

```
> x
```

```
[1] 7.8 9.0 7.1 8.8
```

- Use **index 1** to access the first element of the vector. Square brackets “[]” specify the index.

```
> x[1]
```

```
[1] 7.8
```

- **index 1:3** means 1 to 3, i.e. 1,2,3

```
> x[1:3]
```

```
[1] 7.8 9.0 7.1
```

- **order** function returns the **indices** for the sorted vector

```
> order(x)
```

```
[1] 3 1 4 2
```

- create a sorted vector **ordered.x**

```
> (ordered.x <- x[order(x)])
```

```
[1] 7.1 7.8 8.8 9.0
```

Other ways to create vectors

- The “:” operator is used for consecutive numbers

```
> 1:10
```

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

- **seq** function allows more flexibility

```
> seq(from=1, to=10, by=2)
```

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

- print out values while assigning to variable using “()”

```
> (color1 <- c("red", "red", "red", "red"))
```

```
[1] "red" "red" "red" "red"
```

- **rep** function replicates "red" 4 times and "blue" 3 times

```
> color2 <- c(rep("red", 4), rep("blue", 3))
```

```
> color2
```

```
[1] "red" "red" "red" "red" "blue" "blue" "blue"
```

- **Create a logical vector**

```
> logical.vector <- c(T, F, T, F)
```



Vector – Checking data types



Check the type of vectors

```
> typeof(x)
[1] "double"
> typeof(1:10)
[1] "integer"
> typeof(color1)
[1] "character"
> typeof(logical.vector)
[1] "logical"
> is.numeric(x) # "double" and "integer" are both numeric
[1] TRUE
> is.character(x)
[1] FALSE
> is.character(color1)
[1] TRUE
> is.logical(logical.vector)
[1] TRUE
```

A **factor** is a one-dimensional collection of objects with a pre-defined/limited number of possible values (“categorical variable”).

- Use **factor** function to convert a vector into a factor

```
> data <- c(1,2,2,3,1,2,3,3,1,2,3,3,1)
```

```
> fdata <- factor(data)
```

```
> fdata
```

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

```
Levels: 1 2 3
```

- **levels** function provides access to the levels attribute

```
> levels(fdata)
```

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

- **levels** function can also be used to change the levels of a factor

```
> levels(fdata) <- c('I','II','III')
```

```
> fdata
```

```
[1] I   II  II  III I   II  III III I   II  III III I
```

```
Levels: I II III
```

- **Convert to a character vector**

```
> as.character(fdata)
[1] "I"    "II"   "II"   "III"  "I"    "II"   "III"  "III"  "I"
"II"   "III"  "III"  "I"
```

- **Create a factor directly**

```
> num <- factor(c(1:5,2,5),labels=c("I", "II","III","IV", "V"))
> num
[1] I    II   III  IV   V    II   V
Levels: I II III IV V
```

- **Create an ordered factor**

```
> ordered.num <- factor(1:5,labels=c("I","II","III","IV","V"),ordered=T)
> ordered.num
[1] I    II   III  IV   V
Levels: I < II < III < IV < V
```

- **Now, we can compare the elements.**

```
> ordered.num[1] < ordered.num[5]
[1] TRUE
```

A **list** is a generic vector containing other objects.

```
> n <- c(2, 3, 5)
> s <- c("aa", "bb")
> b <- c(TRUE, FALSE, TRUE)
# Create a list called x
> x <- list(n, s, b)
> x
[[1]]
[1] 2 3 5

[[2]]
[1] "aa" "bb"

[[3]]
[1] TRUE FALSE TRUE
```

- Use `[[]]` (double square bracket) to access an element/object

```
> x[[2]]
[1] "aa" "bb"
```

- Access the 2nd element of the vector `x[[2]]`. remember `x[[2]] = c("aa", "bb")`

```
> x[[2]][2]
```

```
[1] "bb"
```

- **Change "bb" to "cc"**

```
> x[[2]][2] <- "cc"
```

```
> x
```

```
[[1]]
```

```
[1] 2 3 5
```

```
[[2]]
```

```
[1] "aa" "cc"
```

```
[[3]]
```

```
[1] TRUE FALSE TRUE
```

- **Convert a list into a character vector**

```
> (unlist(x))
```

```
[1] "2"      "3"      "5"      "aa"     "cc"     "TRUE"   "FALSE"  "TRUE"
```



EXERCISE 2: Vector, factor, and list



A **vector** is a one-dimensional collection of the same type of objects.

A **factor** is a one-dimensional collection of objects with a limited number of different values (“categorical variable”).

A **list** is a generic vector containing other objects.



BREAK
We'll come back at 10:50

Matrix



A **matrix** is a collection of data elements arranged in a two-dimensional rectangular layout. Elements in a matrix have the **same data type**.

Example is a counts table (read counts for each gene in each sample)

- Use `matrix()` function to create a matrix

```
> d <- matrix(1:12, nrow=3, byrow=TRUE)
```

```
> d
```

```
      [,1] [,2] [,3] [,4]
[1,]     1     2     3     4
[2,]     5     6     7     8
[3,]     9    10    11    12
```

- Use `[row_index, column_index]` to access an element

```
> d[1,2]
```

```
[1] 2
```

- `row_index, column_index` can be range. We can get subset of matrix

```
> d[1:2, 1:3]
```

```
      [,1] [,2] [,3]
[1,]     1     2     3
[2,]     5     6     7
```

	CTL_1	CTL_2	CTL_3
ENSMUST196221.1	6	0	1
ENSMUST179664.1	24	0	0
ENSMUST178537.1	155	212	650
ENSMUST178862.1	0	0	0
ENSMUST179520.1	60	0	0
ENSMUST179883.1	205	19	19
ENSMUST179932.1	11	1	0
ENSMUST180001.1	0	83	7

- **Missing column_index means all columns**

```
> d[1:2, ]
```

	[,1]	[,2]	[,3]	[,4]
[1,]	1	2	3	4
[2,]	5	6	7	8

- **Missing row_index means all rows**

```
> d[, 1:2]
```

	[,1]	[,2]
[1,]	1	2
[2,]	5	6
[3,]	9	10



Matrix: Transposing



- **Get a subset of the matrix and save as “m”**

```
> m <- d[1:2, 1:3]
```

```
> m
```

```
      [,1] [,2] [,3]  
[1,]    1    2    3  
[2,]    5    6    7
```

- **Get the dimension of m, i.e. two rows, three columns**

```
> dim(m)
```

```
[1] 2 3
```

- **Use `t()` function to transpose**

```
> t(m)
```

```
      [,1] [,2]  
[1,]    1    5  
[2,]    2    6  
[3,]    3    7
```

```
> dim(t(m))
```

```
[1] 3 2
```



Matrix: Creating by combining vectors



- Use `cbind()` or `rbind()` function to create a matrix.
- All objects (vectors/matrices) need to have same length on orthologous dimension

```
> a <- 1:3  
> b <- c(7, 8, 9)
```

Put vectors a and b side by side, i.e. combine them column-wise

```
> x <- cbind(a, b)  
> x
```

```
      a b  
[1,] 1 7  
[2,] 2 8  
[3,] 3 9
```

Combine matrices row-wise. Note: x has two columns and additional vector has two elements.

```
> x <- rbind(x, c(4, 10))  
> x
```

```
      a b  
[1,] 1 7  
[2,] 2 8  
[3,] 3 9  
[4,] 4 10
```

- **Element-wise operation of matrix**

```
> x
```

```
      a  b
[1,] 1  7
[2,] 2  8
[3,] 3  9
[4,] 4 10
```

```
> x+x
```

```
      a  b
[1,] 2 14
[2,] 4 16
[3,] 6 18
[4,] 8 20
```

- **Convert a matrix to a vector**

```
> as.vector(x) # convert the matrix to a vector by column
```

```
[1] 1 2 3 4 7 8 9 10
```

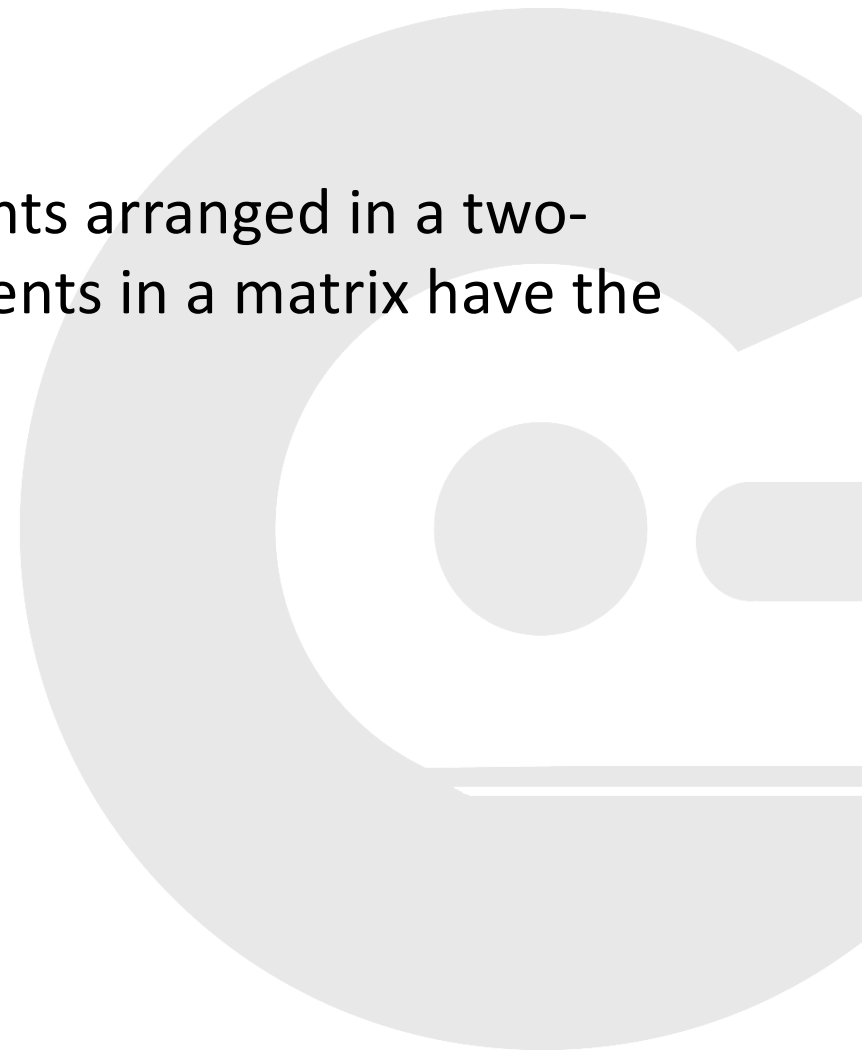
```
> as.vector(t(x)) # convert the matrix to a vector by row
```

```
[1] 1 7 2 8 3 9 4 10
```

EXERCISE 3: Matrix



A **matrix** is a collection of data elements arranged in a two-dimensional rectangular layout. Elements in a matrix have the **same data type**.



Revisiting the attribute table



- Each column can be thought of as collection of values (vector)
- Each vector has a different data type (continuous number, discrete factor, etc.)
- The order (position) of values in each vector is the same for the same sample
- Can store as **data frame**

Sample	bwt	gestation	parity	age	height	weight	smoke
1	120	284	0	27	62	100	0
2	113	282	0	33	64	135	0
3	128	279	0	28	64	115	1
4	108	282	0	23	67	125	1
5	136	286	0	25	62	93	0
6	138	244	0	33	62	178	0
7	132	245	0	23	65	140	0

A **data frame** is a table or a two-dimensional array-like structure. Each column contains equal-length vector or factor, but different column may have different data types.

Create a data frame

```
> employee <- c("John Doe", "Peter Gynn", "Jolie Hope")
> salary <- c(21000, 23400, 26800)
> startdate <- as.Date(c("2010-11-1", "", "2007-3-14"))
> employ_data <- data.frame(employee, salary, startdate)
> employ_data
```

	employee	salary	startdate
1	John Doe	21000	2010-11-01
2	Peter Gynn	23400	<NA>
3	Jolie Hope	26800	2007-03-14

```
> colnames(employ_data) # get column names
[1] "employee" "salary"    "startdate"
> rownames(employ_data) # get row names
[1] "1" "2" "3"
```

Data frame: Selecting subset



Use a column name to select a column.

```
> employ_data$employee  
[1] "John Doe"      "Peter Gynn"    "Jolie Hope"
```

Use [row_index, column_index] to select subset

```
> employ_data[1:2, 1:2]  
  employee salary  
1  John Doe  21000  
2 Peter Gynn  23400
```

Like matrix, missing column index means all columns

```
> employ_data[1:2, ]  
  employee salary  startdate  
1  John Doe  21000 2010-11-01  
2 Peter Gynn  23400      <NA>
```



Data frame: Selecting a subset



Select subset based on logical operation

```
> employ_data[employ_data$salary>23000,]
```

	employee	salary	startdate
2	Peter Gynn	23400	<NA>
3	Jolie Hope	26800	2007-03-14

Change the value of an element

```
> employ_data[1,1] <- "Robert Doe"
```

```
> employ_data
```

	employee	salary	startdate
1	Robert Doe	21000	2010-11-01
2	Peter Gynn	23400	<NA>
3	Jolie Hope	26800	2007-03-14



Data frame



Expand a data frame

```
> gender <- c("M", "M", "F")  
> employ_data <- cbind(employ_data, gender)  
> employ_data
```

	employee	salary	startdate	gender
1	Robert Doe	21000	2010-11-01	M
2	Peter Gynn	23400	<NA>	M
3	Jolie Hope	26800	2007-03-14	F

Select a column as a vector

```
> employ_data[[2]]  
[1] 21000 23400 26800
```

Selecting a column as a data frame (not a vector)

```
> employ_data[2]  
  salary  
1  21000  
2  23400  
3  26800
```



Data frame: Sorting the rows



Order a data frame on a column. Default is by increasing values.

```
> employ_data[order(employ_data$salary), ]
```

	employee	salary	startdate	gender
1	Robert Doe	21000	2010-11-01	M
2	Peter Gynn	23400	<NA>	M
3	Jolie Hope	26800	2007-03-14	F

Order the salary by decreasing values

```
> order(employ_data$salary, decreasing=T)
```

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

```
> employ_data[order(employ_data$salary, decreasing=T), ]
```

	employee	salary	startdate	gender
3	Jolie Hope	26800	2007-03-14	F
2	Peter Gynn	23400	<NA>	M
1	Robert Doe	21000	2010-11-01	M

EXERCISE 4: Data frame



A **data frame** is a table or a two-dimensional array-like structure. Each column contains equal-length vector or factor, but different column may have different data types.

Save history and R data



Save the history to a file

```
> history()
> savehistory(file = "my.Rhistory")
```

Save all objects in the current R session using save.image.

```
> save.image(file = "all.RData")
```

Save a particular set of objects using save then quit the session

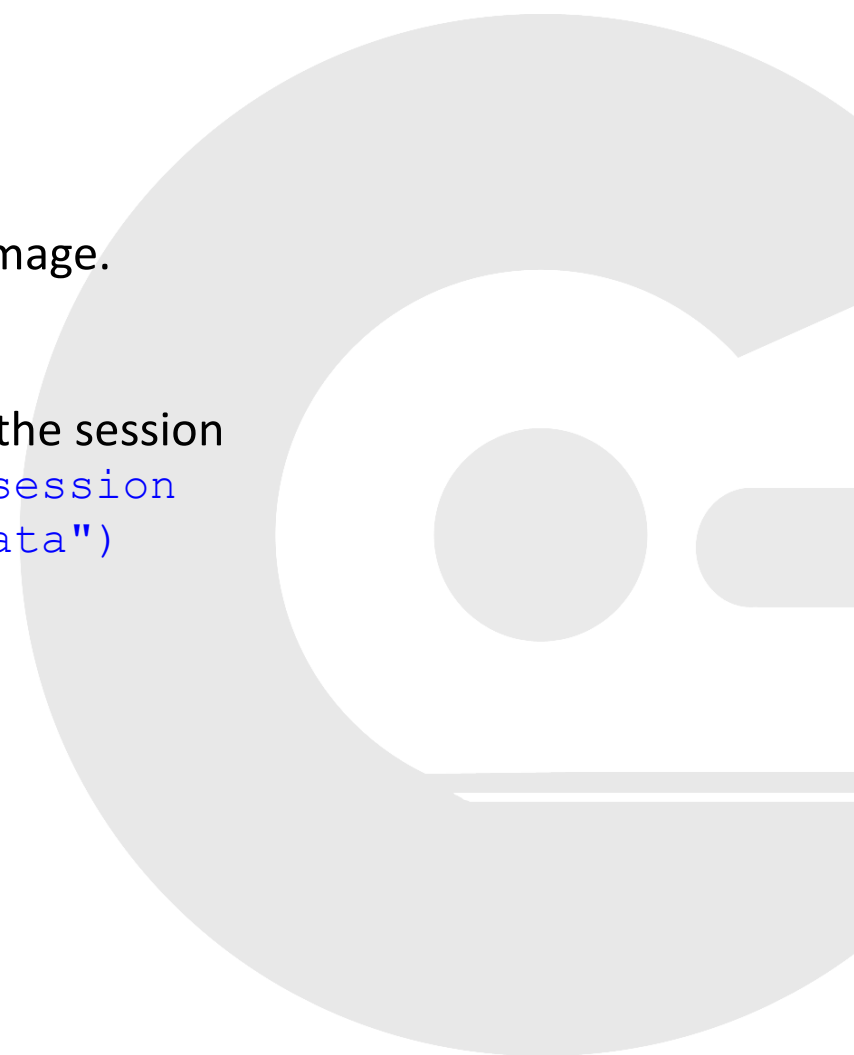
```
> ls() # list the objects in current R session
> save(employ_data, x, file="my_data.RData")
> q()
```

Load the history

```
> loadhistory(file = "my.Rhistory")
> history()
```

Load an R data file into a session using load.

```
> load(file = "my_data.RData")
```



EXERCISE 5: Save history and R data



Save the history into a file

Save all R objects into a file

Save some R objects into a file

Load history from a file

Load R objects from a file



If ... else ... statement



Example: Test if a number is positive or not

```
> x <- 5
```

- The commands between "{" and "}" means they are in the same block

```
> if (x > 0)
```

```
+ {
```

```
+ print(x)
```

```
+ print("Positive number")
```

```
+ }
```

```
[1] 5
```

```
[1] "Positive number"
```

```
> x <- -3
```

```
> if (x > 0)
```

```
+ {
```

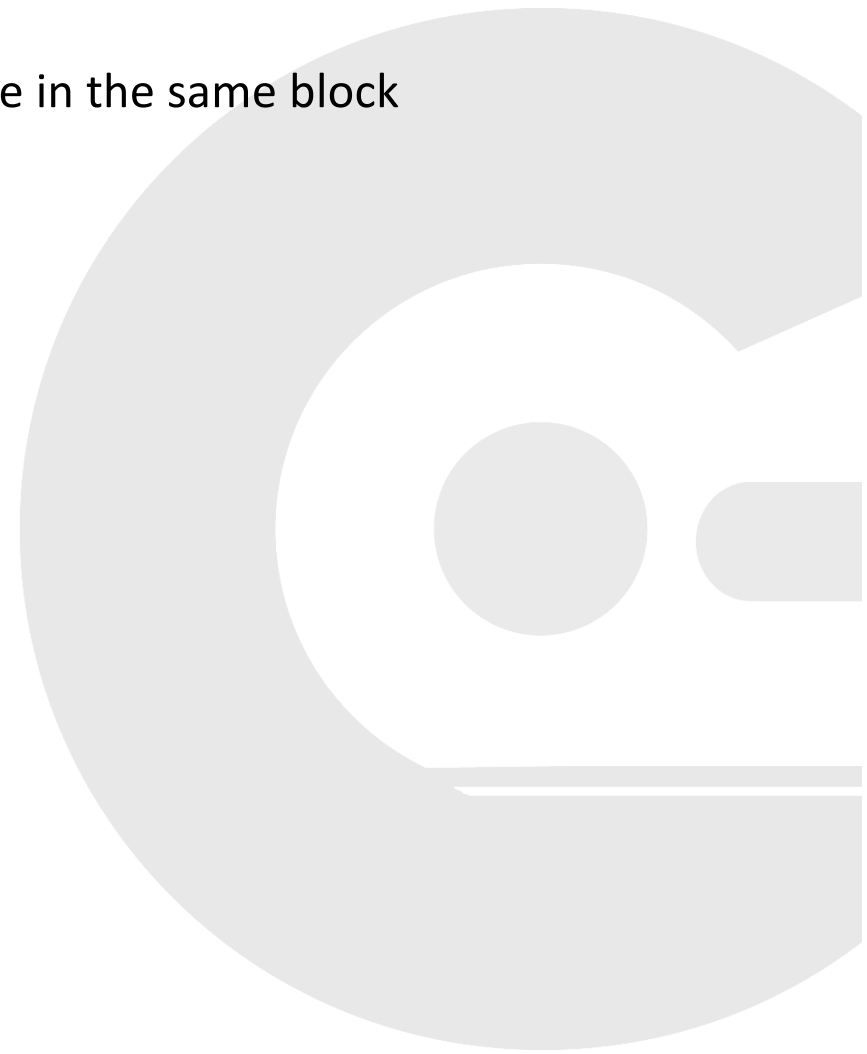
```
+ print("Positive number")
```

```
+ } else {
```

```
+ print("Negative number")
```

```
+ }
```

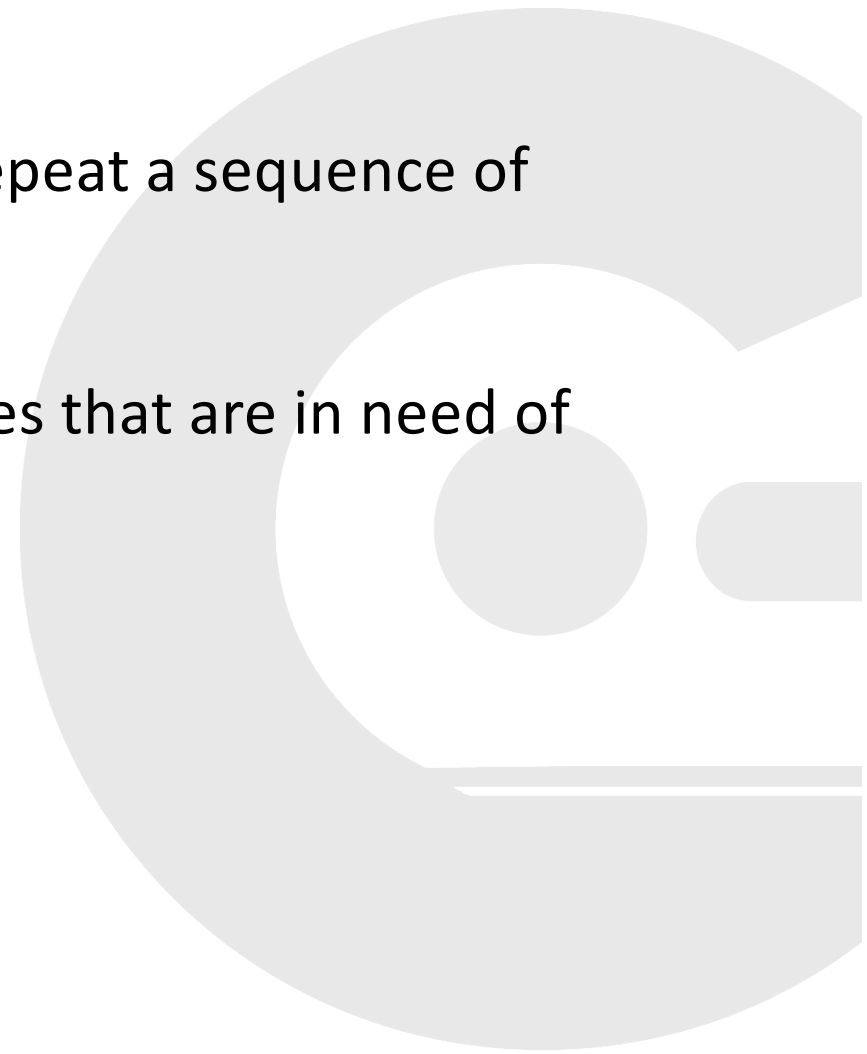
```
[1] "Negative number"
```





Conceptually, a **for** loop is a way to repeat a sequence of codes under certain conditions.

It allows us to automate parts of codes that are in need of repetition.



For loop



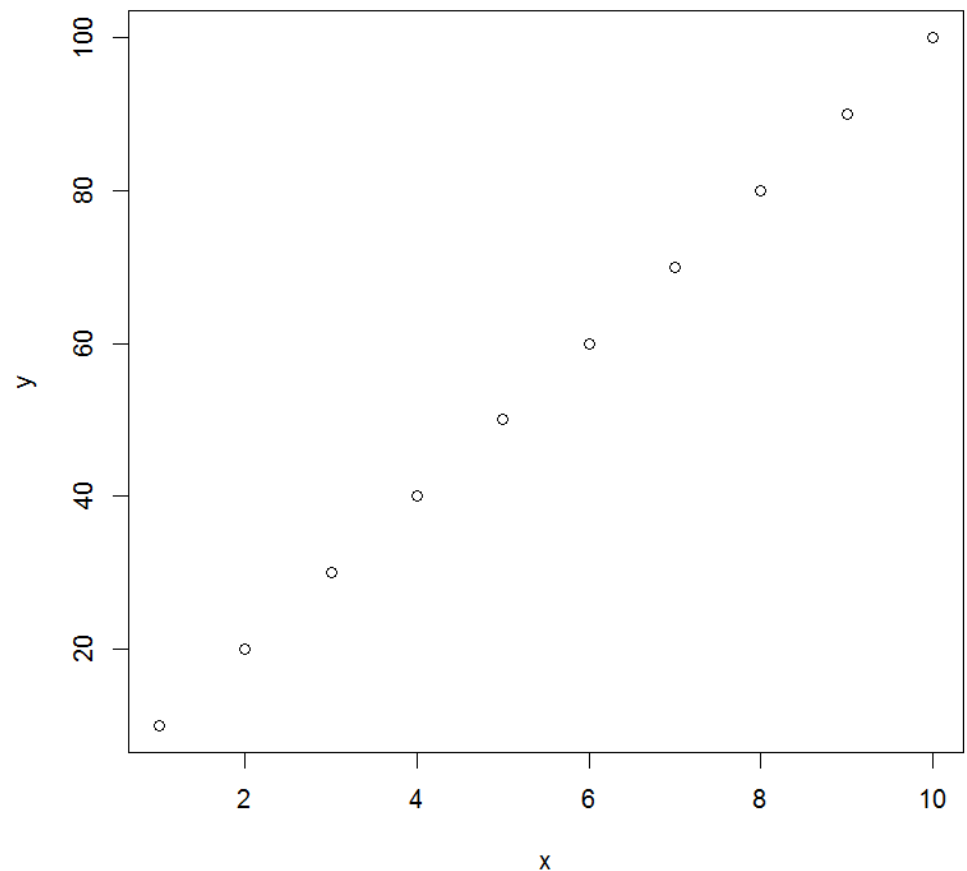
Variable `i` is assigned with each element of vector 1:10

```
> for (i in 1:10)
+ {
+     # paste is a function to combine character string together.
+     print(paste("Number:", i*10, sep=" "))
+ }
[1] "Number: 10"
[1] "Number: 20"
[1] "Number: 30"
[1] "Number: 40"
[1] "Number: 50"
[1] "Number: 60"
[1] "Number: 70"
[1] "Number: 80"
[1] "Number: 90"
[1] "Number: 100"
```



- Functions are predefined blocks of code that can be executed by name
- Calling functions...

```
> log2(8)
[1] 3
> x <- 1:10
> y <- x*10
> plot(x, y)
```



Help for functions



```
> help(plot) or ?plot
```

plot {graphics}

R Doc

Generic X-Y Plotting

Description

Generic function for plotting of R objects. For more details about the graphical parameter arguments, see [par](#).

For simple scatter plots, [plot.default](#) will be used. However, there are `plot` methods for many R objects, including [functions](#), [data.frames](#), [density](#) objects, etc. Use `methods(plot)` and the documentation for these.

Usage

```
plot(x, y, ...)
```

Arguments

- `x` the coordinates of points in the plot. Alternatively, a single plotting structure, function or *any R object with a plot method* can be provided.
- `y` the y coordinates of points in the plot, *optional* if `x` is an appropriate structure.
- `...` Arguments to be passed to methods, such as [graphical parameters](#) (see [par](#)). Many methods will accept the following arguments:

`type`

what type of plot should be drawn. Possible types are

- "p" for **p**oints,
- "l" for **l**ines,
- "b" for **b**oth,
- "c" for the lines part alone of "b",
- "o" for both 'overplotted',
- "h" for 'histogram' like (or 'high-density') vertical lines,
- "s" for stair steps,
- "S" for other steps: see 'Details' below

Function: Defining functions



Function is a piece of common code that can be executed many times. We can execute (call) functions written by others, or we can write our own functions.

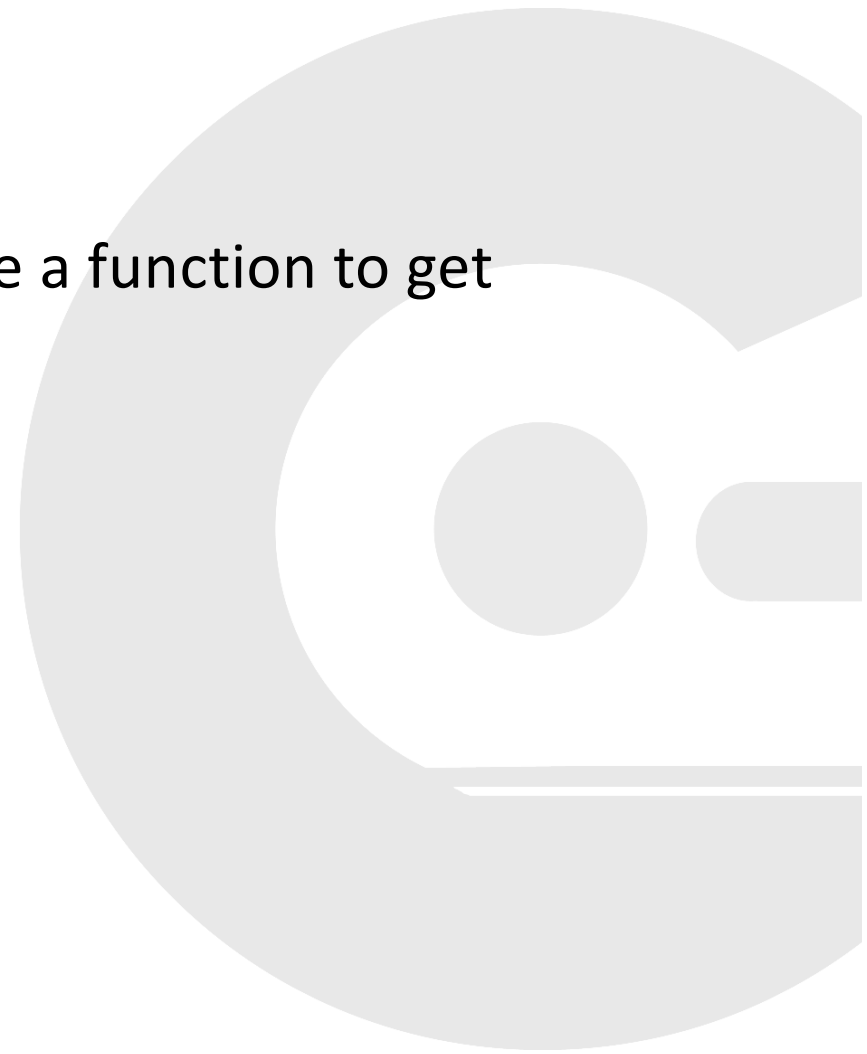
```
> average <- function(x)
+ {
+   sum <- 0
+   for (i in x)
+   {
+     sum <- sum + i
+   }
+   N <- length(x)
+   return (sum/N)
+ }
>
> a <- 1:100
> average(a)
[1] 50.5
```



EXERCISE 6: Write a user-defined function



Given a numeric vector, please write a function to get the maximum value of the vector.





LUNCH
We'll come back at 1 pm

Read and write a file



```
> bw_data <- read.table("birth_weight.txt", header=T)
```

Note: This can be a file on your computer, or a web URL, e.g. <http://...>

```
> head(bw_data)
```

	bwt	gestation	parity	age	height	weight	smoke
1	120	284	0	27	62	100	0
2	113	282	0	33	64	135	0
3	128	279	0	28	64	115	1
4	108	282	0	23	67	125	1

```
> ordered_bw_data <- data[order(bw_data$bwt), ]
```

```
> head(ordered_bw_data)
```

	bwt	gestation	parity	age	height	weight	smoke
923	55	204	0	35	65	140	0
860	58	245	0	34	64	156	1
781	62	228	0	24	61	107	0
1082	63	236	1	24	58	99	0

```
> write.table(ordered_bw_data, file="ordered_birth_weight.txt", sep="\t",  
quote=F, row.names=F)
```

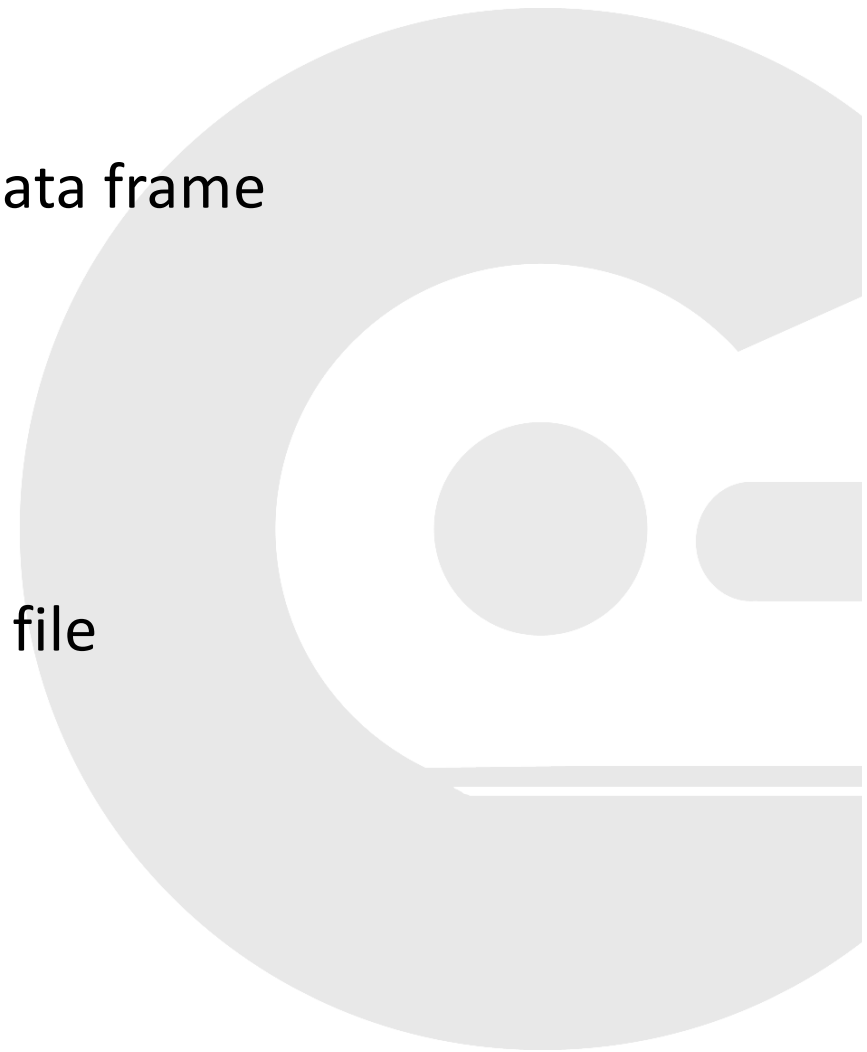
EXERCISE 7: Read and write a file



Read the file `birth_weight.txt` into a data frame

Sort the data frame by mother's age

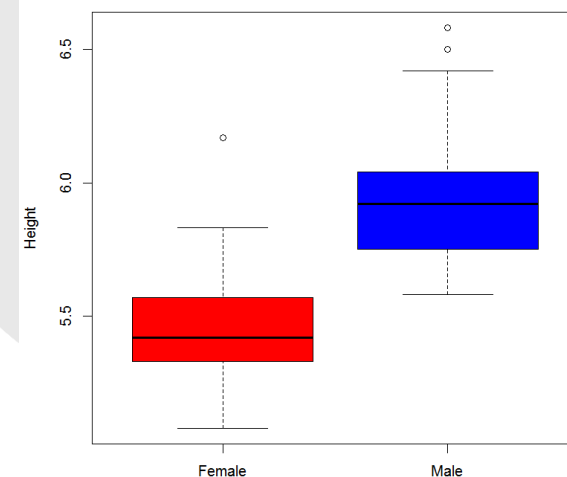
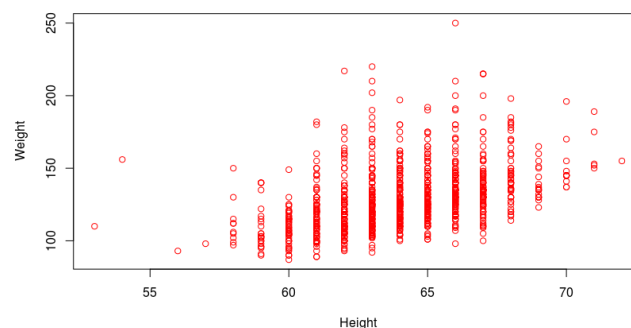
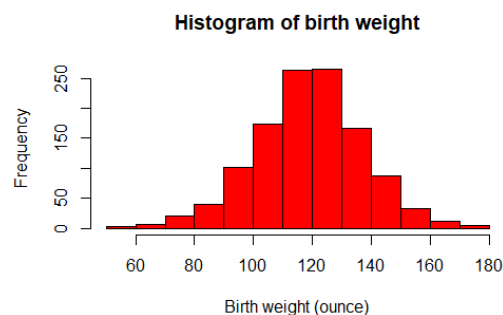
Write the sorted data frame to a new file



Basic Visualizations



- “Out of the box” R can create a number of basic visualizations (plots)
- More complex visualizations are possible and there are number of 3rd party libraries/packages that you can use to create these visualizations.



Basic Visualizations: Output



- By default, R will display the plot to a local plot window (if available)
- It is also possible to save plots to a file, e.g. PDF, PNG or TIFF file.
- Start saving plots to a file using a function, e.g. `pdf()` or `png()`
- Stop saving using `“dev.off()”`
- All graphics (plots) generated after “redirection” function will be saved to file until `“dev.off()”`
- For PDF, each plot will be a new page.

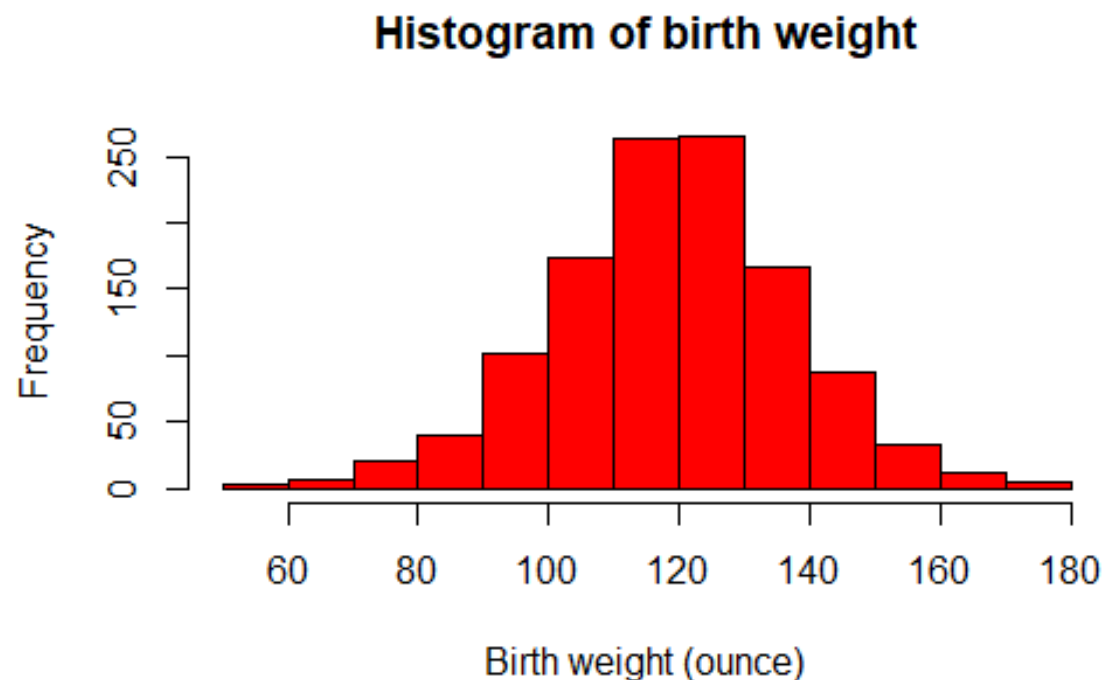
```
> pdf(file="my_plots.pdf")  
    ...various plot commands...  
> dev.off()
```

Basic Visualizations: Histogram



hist() function can be used to generate histogram. It will automatically bin a data.

```
> hist(bw_data$bwt,  
      main="Histogram of birth weight",  
      xlab="Birth weight (ounce)", col="red")
```



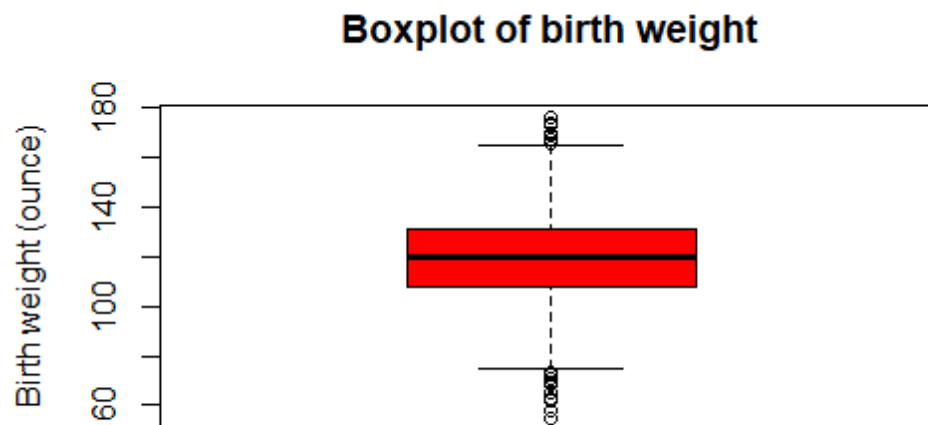
Basic visualization: Box plot



boxplot() function can be used to generate box plot.

Will calculate mean, quantiles, min and max of data provided

```
> boxplot(bw_data$bwt,  
  main="Boxplot of birth weight",  
  ylab="Birth weight (ounce)", col="red")
```

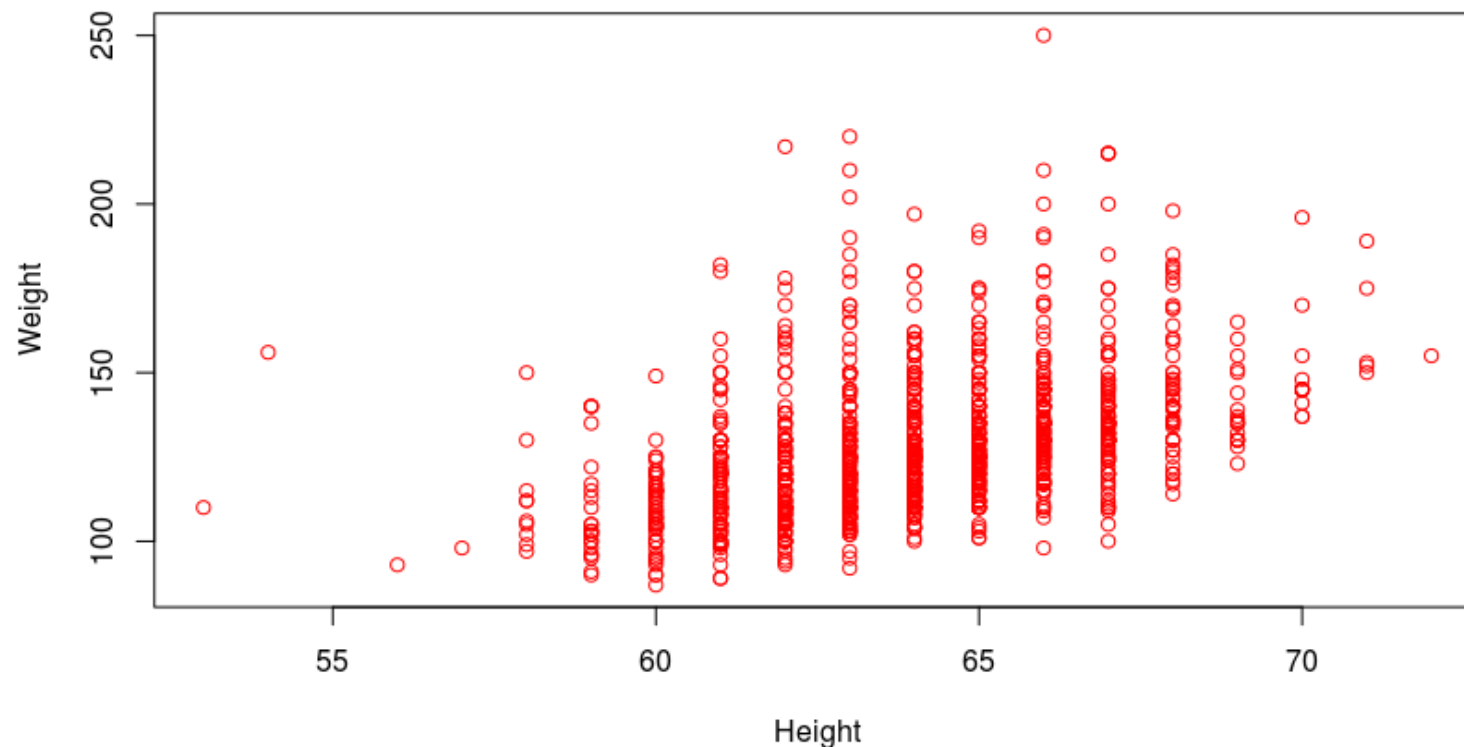


Basic visualization: Scatter plot



plot() will generate scatter plot, needs two vectors for x and y.

```
> plot(bw_data$height, bw_data$weight,  
       xlab="Height", ylab="Weight", col="red")
```



EXERCISE 8: Histogram, boxplot, and scatter plot

Read the file `birth_weight.txt` into a data frame

Create a histogram for mother's age and save as a PDF file

Create a boxplot for mother's age and save as a PDF file

Create a scatter plot for mother's weight vs baby's birth weight and save as a PDF file



Statistical tests

Descriptive statistics



```
> summary(bw_data$bwt)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
55.0	108.0	120.0	119.5	131.0	176.0

```
> mean(bw_data$bwt)
```

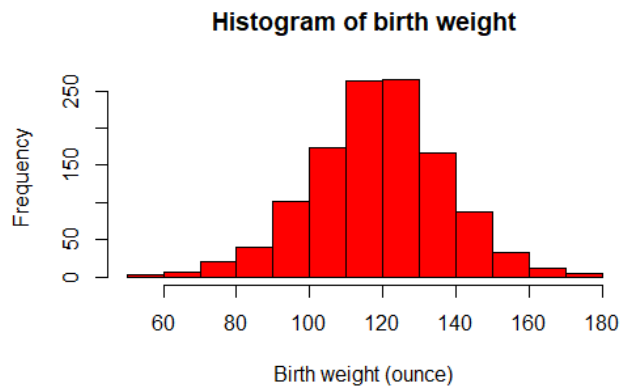
```
[1] 119.4625
```

```
> sd(bw_data$bwt)
```

```
[1] 18.32867
```

```
> hist(bw_data$bwt, main="Histogram of birth weight", xlab="Birth  
weight (ounce)", col="red")
```

The histogram of birth weight looks like a bell shape thus, it follows a normal distribution.



Student's *t*-test

Test if the means of two groups are equal

Assumption: normal distribution

Female Heights:

5.33	5.33	5.17	5.75	5.42	5.42	5.50	5.50	5.58
5.33	5.50	5.67	5.42	5.25	6.17	5.42	5.33	5.17
5.42	5.42	5.42	5.42	5.42	5.83	5.33	5.67	5.33
5.66	5.25	5.75	5.57	5.35	5.42	5.08	5.75	5.33
5.08								

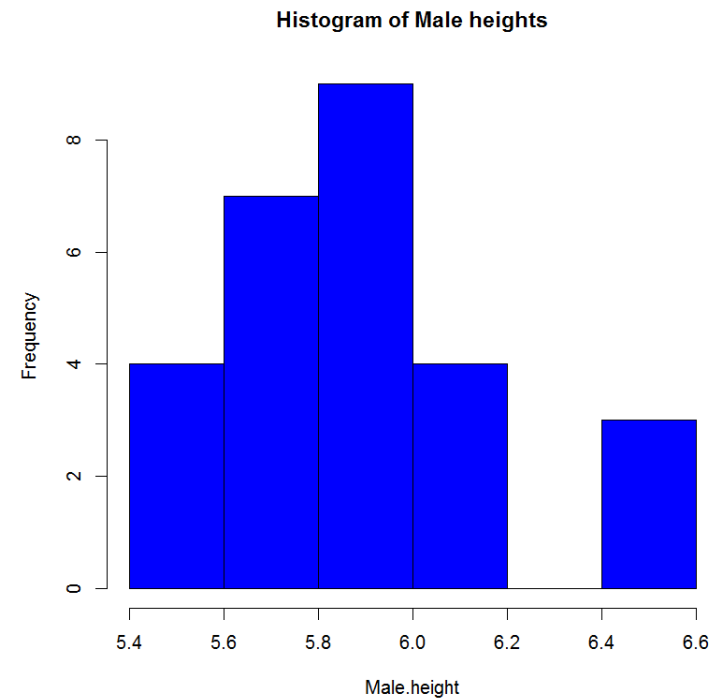
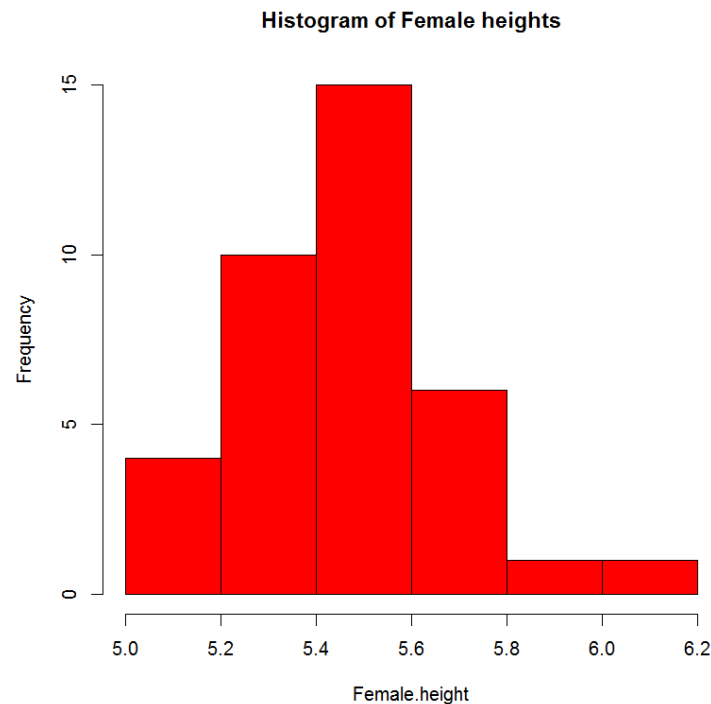
Male Heights:

5.75	5.92	6.17	6.08	5.58	5.92	6.00	5.75	5.92
5.75	5.75	5.83	6.58	6.00	5.75	6.42	6.50	6.17
6.00	5.67	5.58	5.83	5.58	5.58	6.08	5.67	6.00

T test: is female as tall as male?



```
> Female.height <- c(5.33,5.33,5.17,5.75,5.42,5.42,5.5,5.5,5.58,5.33,  
5.5,5.67,5.42,5.25,6.17,5.42,5.33,5.17,5.42,5.42,5.42,5.42,5.83,5.33,5  
.67,5.33,5.66,5.25,5.75,5.57,5.35,5.42,5.08,5.75,5.33,5.08)  
  
> Male.height <- c(5.75,5.92,6.17,6.08,5.58,5.92,6,5.75,5.92,5.75,5.75,  
5.83,6.58,6,5.75,6.42,6.5,6.17,6,5.67,5.58,5.83,5.58,5.58,6.08,5.67,6)  
  
> hist(Female.height, main="Histogram of Female heights", col="red")  
> hist(Male.height, main="Histogram of Male heights", col="blue")
```



T test: is female as tall as male?



```
> boxplot(Female.height, Male.height, col=c("red", "blue"), names=
c("Female", "Male"), ylab="Height")
> t.test(Female.height, Male.height)
```

Welch Two Sample t-test

data: Female.height and Male.height

t = -7.2727, df = 48.293, **p-value = 2.719e-09**

alternative hypothesis: true difference in means is not equal to 0

95 percent confidence interval:

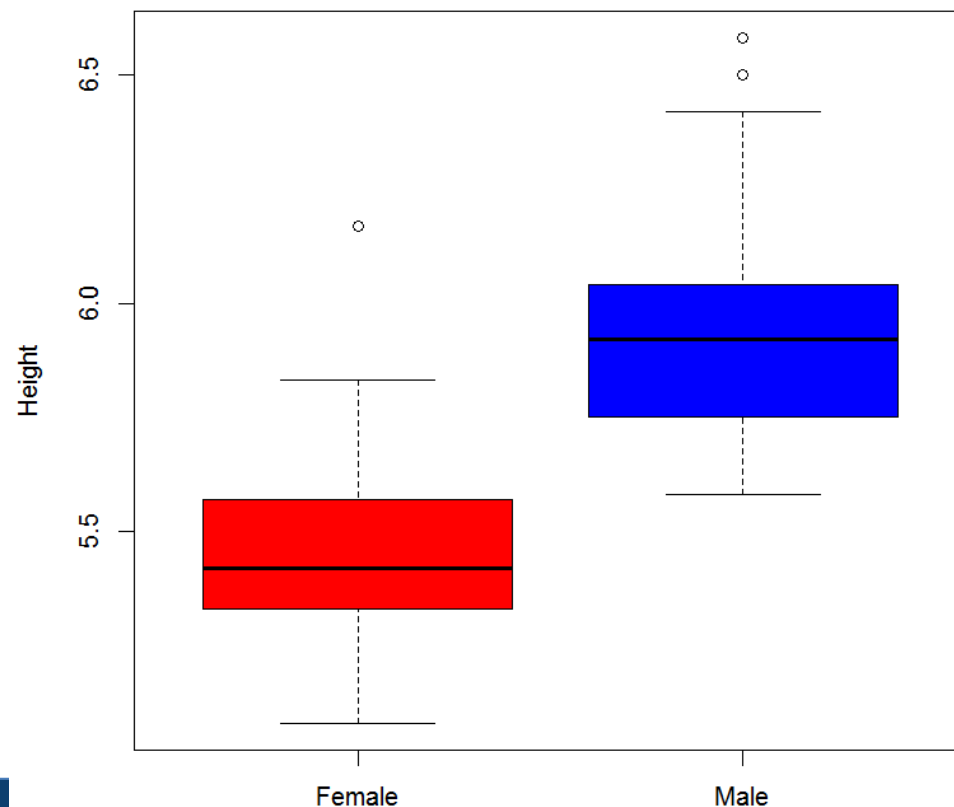
-0.5956498 -0.3376635

sample estimates:

mean of x mean of y

5.452973 5.919630

**p-value = 2.719e-09, which means
female and male have different
heights.**



ANOVA: ANalysis Of VAriance

Test if the means of groups (>2) are equal

Assumption: normal distribution

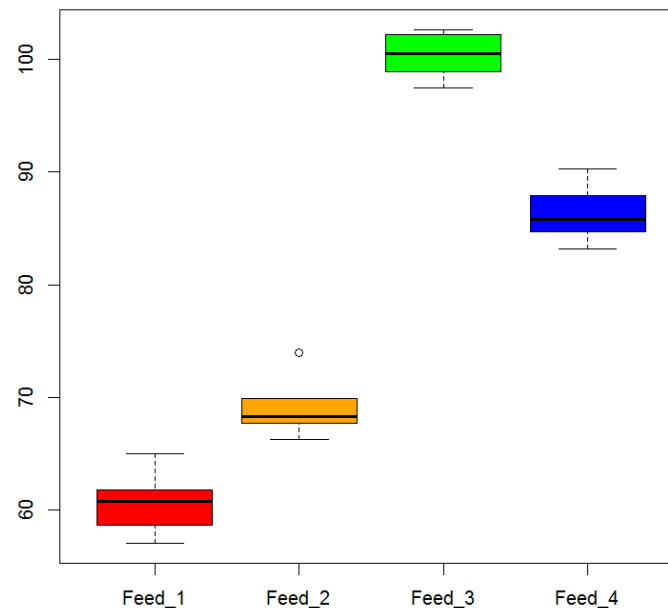
In one research study, 20 young pigs are assigned at random among 4 experimental groups. Each group is fed a different diet. (This design is a completely randomized design.) The data are the pigs' weights in kg after being raised on these diets for 10 months. We wish to ask whether mean pig weights are the same for all 4 diets.

Feed_1	Feed_2	Feed_3	Feed_4
60.8	68.3	102.6	87.9
57.1	67.7	102.2	84.7
65	74	100.5	83.2
58.7	66.3	97.5	85.8
61.8	69.9	98.9	90.3

ANOVA: pigs' weight with different diets



```
> pigs.w <- read.table("pigs.txt", header=T)
> pigs.w
  Feed_1 Feed_2 Feed_3 Feed_4
1  60.8   68.3  102.6   87.9
2  57.1   67.7  102.2   84.7
3  65.0   74.0  100.5   83.2
4  58.7   66.3   97.5   85.8
5  61.8   69.9   98.9   90.3
> boxplot(pigs.w, col=c("red", "orange", "green", "blue"))
```



ANOVA: pigs' weight with different diets



```
> library(reshape2)
> pigs.w.m <- melt(pigs.w)
```

Data transformation using **melt** function: it takes **wide-format data** and melts it into **long-format data**.

ANOVA function `aov` accepts the **long-format data**

```
> pigs.w.m
  variable value
1   Feed_1  60.8
2   Feed_1  57.1
3   Feed_1  65.0
4   Feed_1  58.7
5   Feed_1  61.8
6   Feed_2  68.3
7   Feed_2  67.7
8   Feed_2  74.0
9   Feed_2  66.3
10  Feed_2  69.9
11  Feed_3 102.6
12  Feed_3 102.2
13  Feed_3 100.5
14  Feed_3  97.5
15  Feed_3  98.9
16  Feed_4  87.9
17  Feed_4  84.7
18  Feed_4  83.2
19  Feed_4  85.8
20  Feed_4  90.3
```



	Feed_1	Feed_2	Feed_3	Feed_4
1	60.8	68.3	102.6	87.9
2	57.1	67.7	102.2	84.7
3	65.0	74.0	100.5	83.2
4	58.7	66.3	97.5	85.8
5	61.8	69.9	98.9	90.3

```
> colnames(pigs.w.m) <- c("group", "weight")
```

Aside: formula (~) in R



- R uses formulas with ~ in a variety of applications

```
outcome ~ factor1 + factor2 + factor1:factor2
```

```
outcome ~ factor1 * factor2
```

- Left-hand side is dependent variable (measurement)
- Right-hand side lists independent variables (experimental factors)
 - Interaction terms can be included with :
 - Complete model with *
- R automatically builds out linear models based on all levels of factors
- Data needs to be in long format
 - `outcome`, `factor1`, and `factor2` are column names from a data frame

ANOVA: pigs' weight with different diets



Make a boxplot to visualize (using formula)

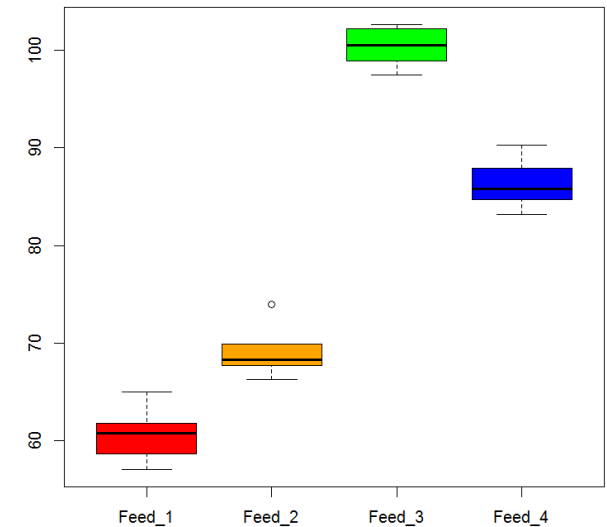
```
> boxplot(weight ~ group, data=pigs.w.m)
```

Perform ANOVA using aov function: fits an analysis of variance model (also using formula)

```
> summary(aov(weight ~ group, data=pigs.w.m))
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
group	3	4703	1567.7	206.7	5.28e-13 ***
Residuals	16	121	7.6		

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



p-value = 5.28e-13, which means that the mean weight in at least one group is different from the mean weight of the others.

EXERCISE 9: Student's t-test



Use the data in `birth_weight.txt`.

Make boxplots for baby's birth weight (`bwt`), for smoker and non-smoker, respectively.

Use T-test to test if the baby's birth weights are different between smoking mother and non-smoking mother.

Fisher's exact test



- Use Fisher's exact test when we have two categorical variables and we want to see if there is an association between the two categorical variables.
- It is valid even when the sample size is small.
- Below is a 2x2 contingency table for patients exposed to some pathogen.
- We want to test if there is an association between disease and exposure status.

	Diseased	Not diseased
Exposed	99	8
Not exposed	88	43

Fisher's exact test



```
> m <- matrix(c(99, 8, 88, 43), nrow=2, byrow=T)
> rownames(m) <- c("Exposed", "Not exposed")
> colnames(m) <- c("Diseased", "Not diseased")
> m
```

	Diseased	Not diseased
Exposed	99	8
Not exposed	88	43

Calculate odds ratio

```
> OR <- (99/8) / (88/43)
> OR
[1] 6.046875
> fisher.test(m)
```

Fisher's Exact Test for Count Data

data: m

p-value = 1.204e-06

alternative hypothesis: true odds ratio is not equal to 1

95 percent confidence interval:

2.608378 15.606815

sample estimates:

odds ratio

6.004836

p-value = 1.204e-06, which means that disease is associated with exposure status.

EXERCISE 10: Fisher's exact test



There are patients participating a clinical trial for two different therapies. Please test if there is an significant association between therapy and cure.

	Cured	Not cured
Therapy 1	34	12
Therapy 2	22	25

Adjust p-values for multiple tests



Example:

In one research study, researchers did microarray gene expression for disease samples vs healthy controls. There are 20,000 genes tested. 20,000 p-values are obtained from 20,000 statistical tests.

Problem:

We need to do multiple test correction, because the more statistical tests are made, the more likely we obtain significant p-values even if there are no difference between groups.

From 20,000 statistical tests on a random gene data set, I expect to get 1000 genes that pass a $p < 0.05$ threshold by chance alone. The FDR correction adjusts for this.

Multiple test correction:

After multiple test correction, we obtain adjusted p-values (aka False Discovery Rate, FDR, Benjamini-Hochberg corrected p-values, q-values).

pval is the vector of p-values

```
> FDR <- p.adjust(pval, method="BH")
```


Illustration of multiple testing



```
> pval <- vector() # create an empty vector
```

randomly generate values for wt and ko with the same means.

run t-test 10,000 times

```
> for (i in 1:10000)
+ {
+   wt <- rnorm(10, mean=5, sd=3)
+   ko <- rnorm(10, mean=5, sd=3)
+   pval[i] <- t.test(wt, ko)$p.value
+ }
```

how many pval < 0.05?

```
> length(pval[pval<0.05])
```

Lots of tests with p<0.05 (about 5% of the total)

[1] 525

```
> hist(pval)
```

```
> FDR <- p.adjust(pval, method="BH")
```

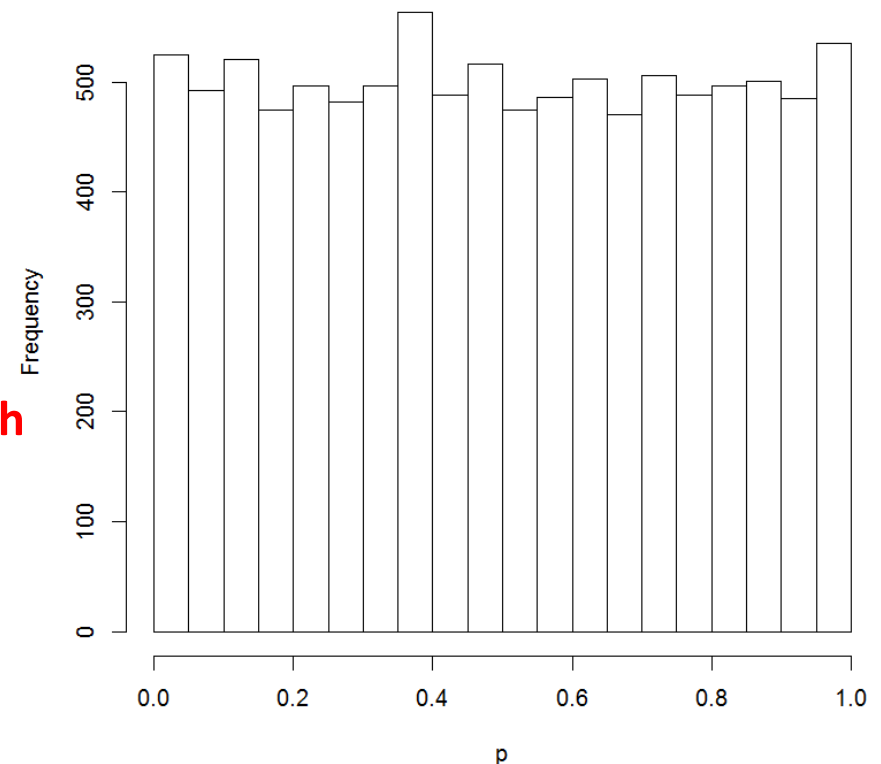
```
> FDR
```

[1] 0.9801946 0.9299517 ...

```
> summary(FDR)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.9058	0.9860	0.9863	0.9863	0.9990	0.9999

Histogram of p



**After FDR correction,
no more “significant”
results**



- Several terms are used interchangeably:
 - False Discovery Rate (FDR)
 - Q-value or q-value
 - Benjamini
 - Benjamini-Hochberg (or B-H)
 - Adjusted p-value (though this could refer to other p-value adjustments, like Bonferroni)

EXERCISE 11: False discovery rate



Randomly generate wt with mean 10 and standard deviation 3.

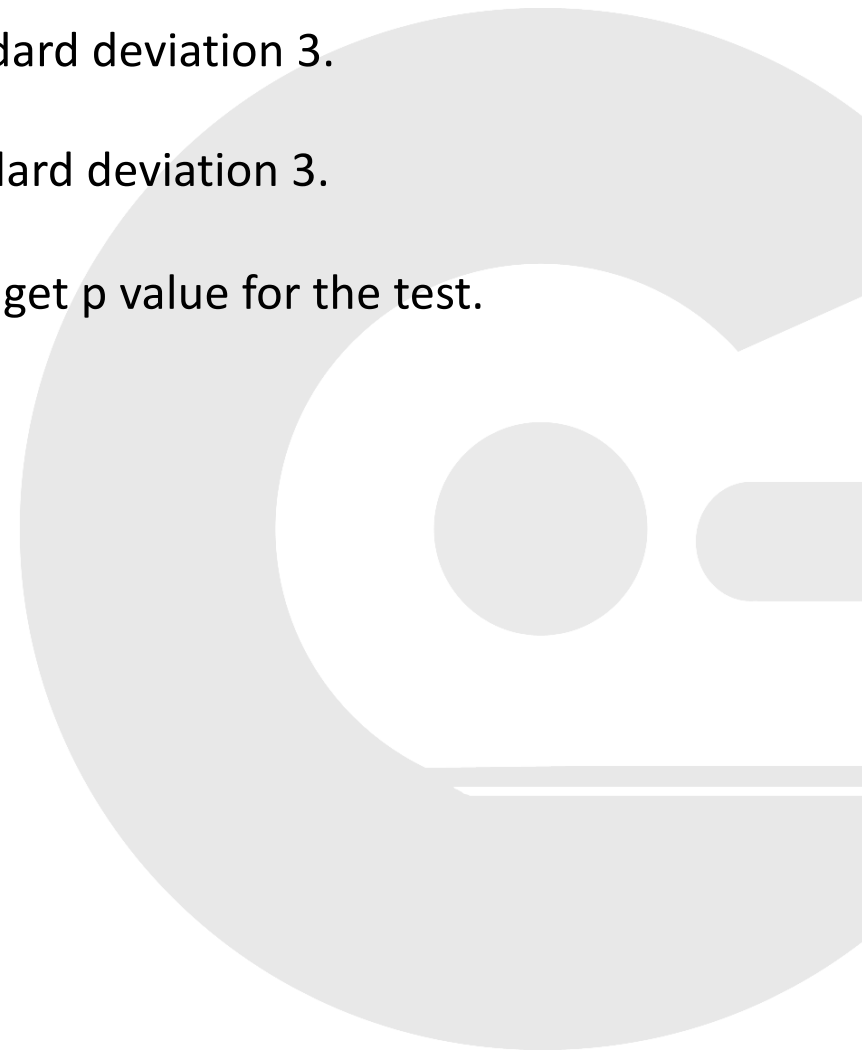
Randomly generate ko with mean 10 and standard deviation 3.

Use T-test to test if wt is different from ko, and get p value for the test.

Repeat this procedure 10000 times

Can you find how many $p < 0.05$?

Calculate false discovery rate.



BREAK

Please complete our workshop survey

<http://go.uic.edu/RICWorkshopSurvey>





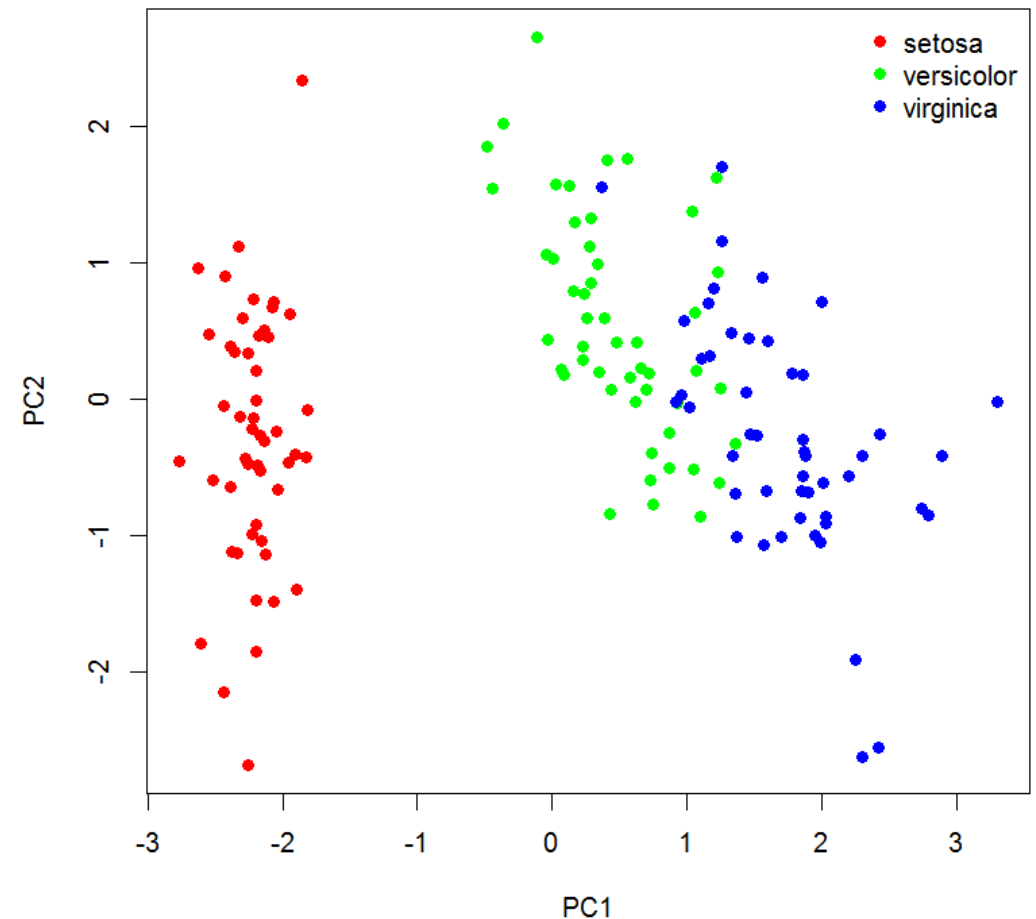
Clustering and data visualization

Principal component analysis (PCA)



Principal component analysis (PCA)

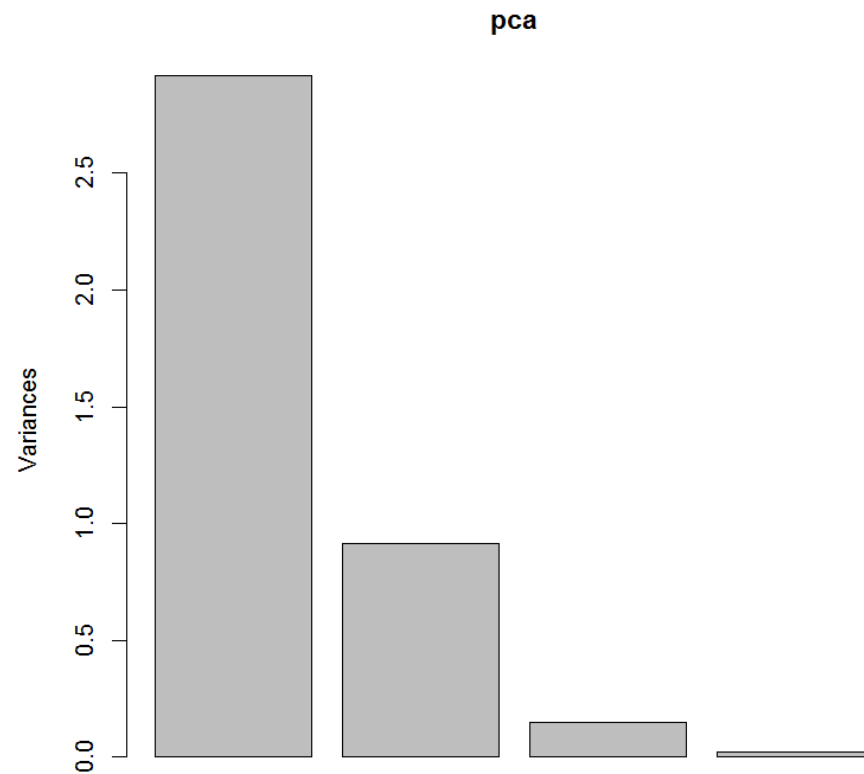
- Orthogonal transformation of data
- View high-dimension data: dimensionality reduction
- Quick visualization of sample similarity
- Batch effects
- Outliers



Principal component analysis (PCA)



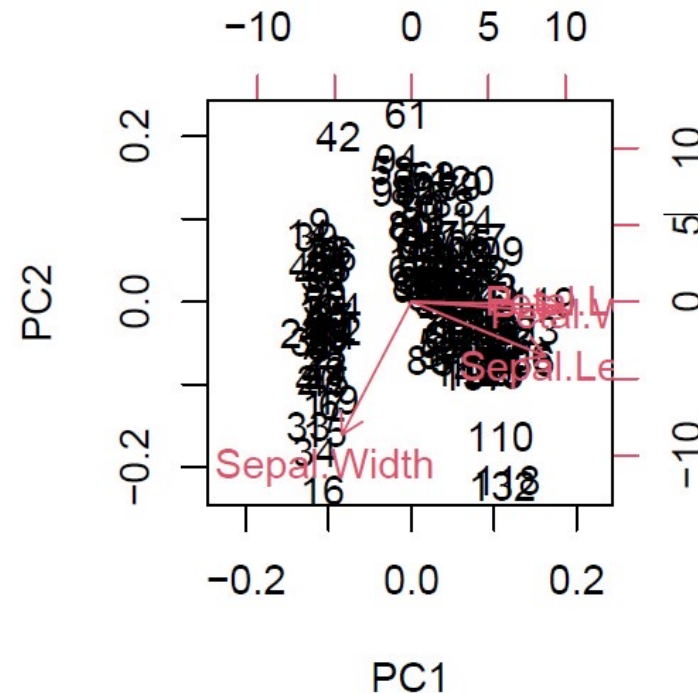
Scree plot shows the variance explained for each principal component.



Principal component analysis (PCA)



- A biplot represents both observations and variables on the same plot



Built-in dataset: iris



- Gives measurements in centimeters of petal length and width and sepal length and width
- 50 flowers from each of 3 species of iris
- The species are: *Iris setosa*, *versicolor*, and *virginica*

Sample R data for PCA



Load built-in data “iris”

```
> data(iris)
> head(iris)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

Get just the data (measured values) and species for the flowers

```
> ir <- iris[, 1:4]
> ir.species <- iris[, 5]
```

To examine variability of all numeric variables

sapply: traverse over a set of data like a list or vector, and calling the function for each item.

```
> sapply(ir, var)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
0.6856935	0.1899794	3.1162779	0.5810063

R code for PCA



```
> pca <- prcomp(ir, scale= TRUE)
```

scale=TRUE means transform all the attributes so that they are on equivalent scale (z-score)

Look at the **pca\$x** object: principal components

```
> pca$x
```

	PC1	PC2	PC3	PC4
[1,]	-2.25714118	-0.478423832	0.127279624	0.024087508
[2,]	-2.07401302	0.671882687	0.233825517	0.102662845
[3,]	-2.35633511	0.340766425	-0.044053900	0.028282305
[4,]	-2.29170679	0.595399863	-0.090985297	-0.065735340

Generate colors for each of the different species (help with plotting)

```
> ir.colors <- factor(ir.species)
```

```
> levels(ir.colors)
```

```
[1] "setosa"      "versicolor" "virginica"
```

```
> levels(ir.colors) <- c("red", "green", "blue")
```

```
> ir.colors
```

```
[1] red  red  red  red  red  red  red  red  red  red  red
red  red  red  red  red  red
```

```
...
```

```
[137] blue blue blue blue blue blue blue blue blue blue
blue blue blue blue
```

```
Levels: red green blue
```

PCA plot of PC1 vs PC2 and add a legend for the colors

```
> plot(pca$x[,1], pca$x[,2], col=colors, xlab="PC1", ylab="PC2",  
pch=16)  
> legend('topright', legend = c("setosa", "versicolor",  
"virginica"),  
+       col = c("red", "green", "blue"), pch = 16, bty = 'n')
```

Check the percent variance for each PC

```
> summary(pca)
```

Importance of components:

	PC1	PC2	PC3	PC4
Standard deviation	1.7084	0.9560	0.38309	0.14393
Proportion of Variance	0.7296	0.2285	0.03669	0.00518
Cumulative Proportion	0.7296	0.9581	0.99482	1.00000

Scree plot to visualize the relative proportion of variance

```
> screeplot(pca)
```

Biplot to visualize observations and variables

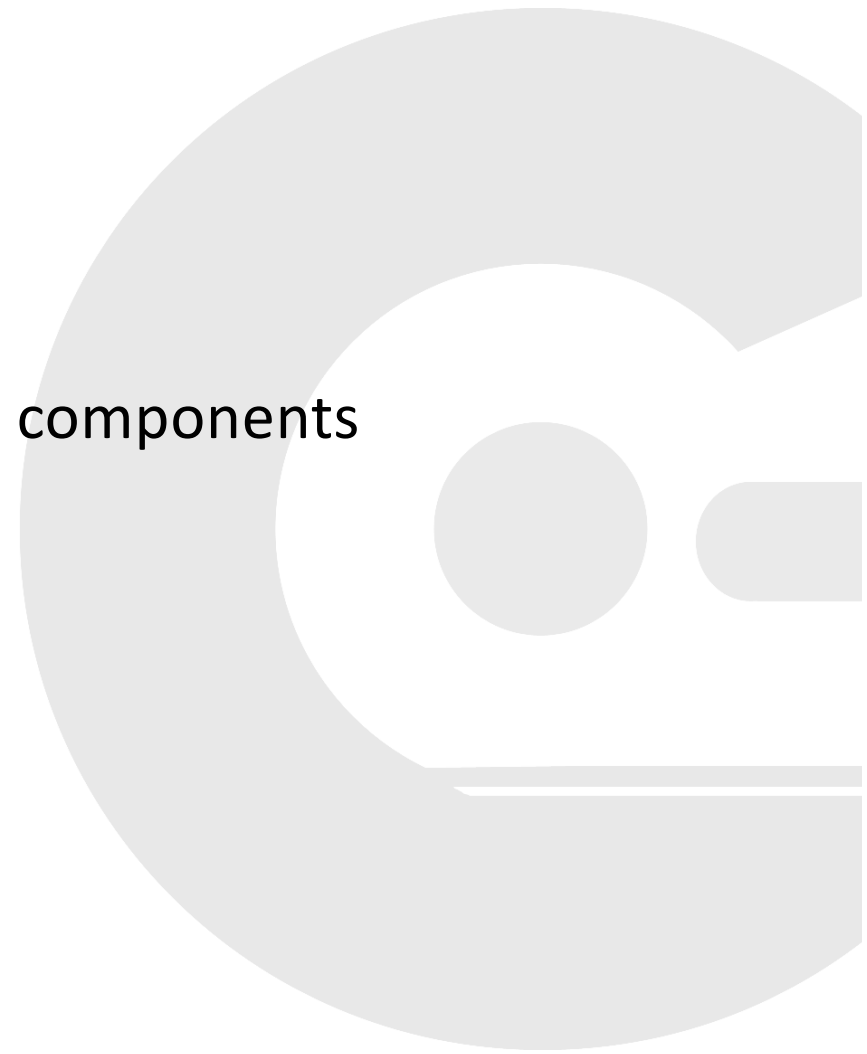
```
> biplot(pca)
```

EXERCISE 12: PCA



Create a PCA plot using iris data

Create a scree plot for the principal components



R package repositories



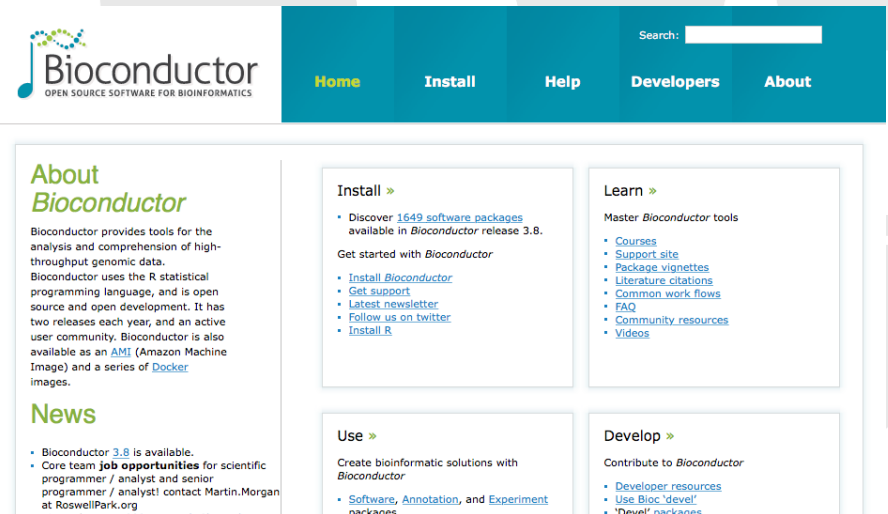
- CRAN

- <https://cran.r-project.org>
- Generic R packages



- Bioconductor

- <http://bioconductor.org>
- Bioinformatics packages



Install and load packages



R packages include reusable R functions, the documentation that describes how to use them, and sample data.

Check if the packages are already installed

```
# A %in% B means if elements of vector A are in the vector B
> c("ggplot2", "ComplexHeatmap") %in% rownames(installed.packages())
[1] FALSE FALSE
```

Install CRAN (Comprehensive R Archive Network) packages:

```
> install.packages("ggplot2")
Installing package into 'home/ubuntu/R/x86_64-pc-linux-gnu-library/3.2'
...
> library(ggplot2) #load ggplot2 library, so that you can start using it
```

Install Bioconductor packages:

```
> if (!requireNamespace("BiocManager")) install.packages("BiocManager")
> BiocManager::install(c("ComplexHeatmap"))
...
Installing package(s) 'ComplexHeatmap'
> library(ComplexHeatmap)
```

EXERCISE 13: Install CRAN and Bioconductor packages



Check if two packages "ggplot2" and "ComplexHeatmap" are installed or not.

If they are not installed, please install them.

Heatmap and Clustering

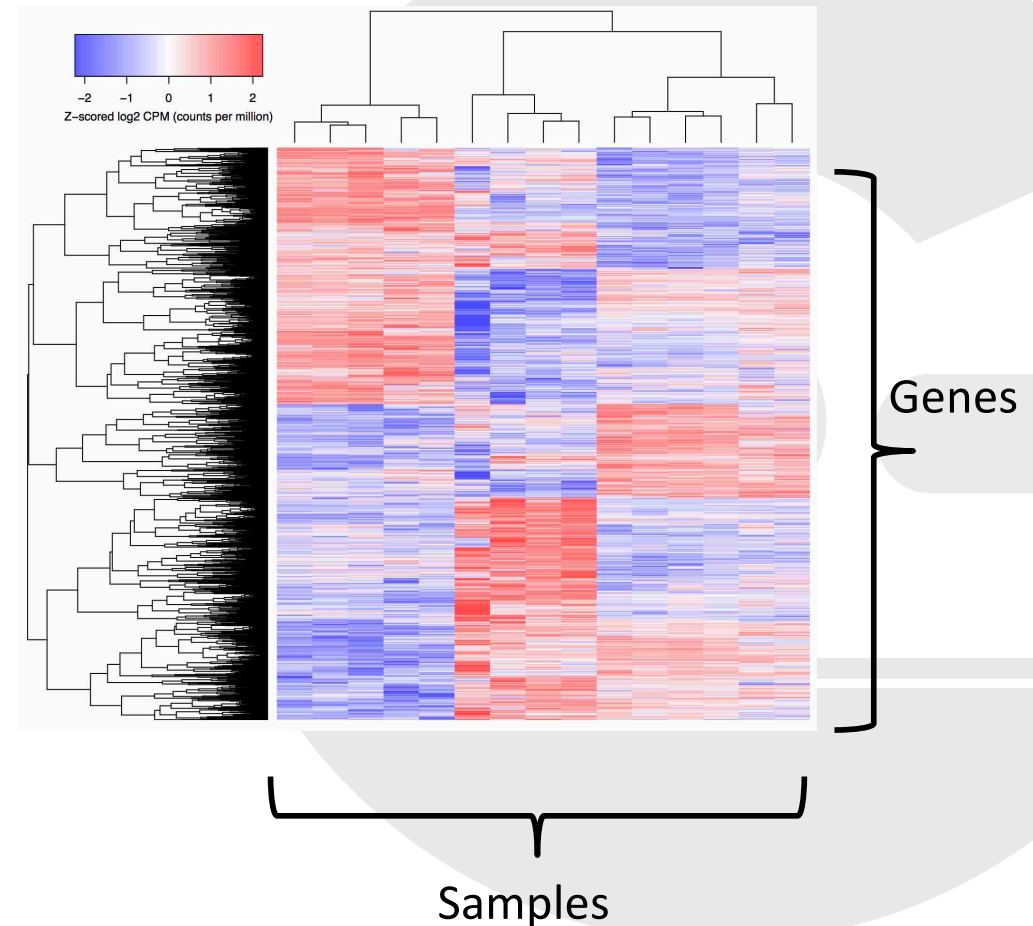


Why use heatmap/clustering:

- Find significant patterns in the data
- Hierarchical clustering + heatmaps

Data transformations:

- Select significant features, e.g. differentially expressed genes
- Log-scale?
- Z-score transformation for rows or columns?



R code for Heatmap



Use ComplexHeatmap package

```
> library(ComplexHeatmap)
```

Load data

```
> data(iris)
```

```
> head(iris)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa

Subset data frame to only include measured values

```
> ir <- iris[, 1:4]
```

z-score transformation on columns so that we can see relative differences

```
> std.ir <- scale(ir)
```

Generate basic heatmap

```
> Heatmap(std.ir)
```

Generate heatmap with title

```
> Heatmap(std.ir, name = "Heatmap of iris")
```

EXERCISE 14: Heatmap



Create a heatmap using iris data

Try different heatmap options





- Use `data()` to view list of preloaded datasets.
- Use `?` or `help()` to get information about a preloaded dataset.
 - e.g. `?mtcars` or `help(mtcars)`
- Other tutorials in R...



- Advanced R
 - Manipulation of data frames: combining, subsetting, filtering
 - Use of apply functions versus for loops
 - (Better) plots in ggplot2
 - More statistical tests
- *Many of these topics will be useful for later workshops*



- Advanced Statistics in EdgeR
 - Application of edgeR package for differential statistics in RNA-seq, ChIP-seq, ATAC-seq, metagenomics
 - Visualizations with PCA and Heatmaps
 - *Application of data frame manipulation from advanced R session*



- Single-cell RNA-seq
 - Quantification, QC, and feature selection
 - Clustering
 - Data visualization and statistical analysis
 - Cell marker analysis
 - Pseudotime analysis
 - *Application of data frame manipulation, visualizations, apply statements, and statistics from advanced R session*



- Pathway analysis
 - Pathway databases and online resources
 - Statistical methods
 - Result visualization
 - Network/molecular interaction analysis
 - *Application of data frame manipulation and visualizations from advanced R session*