Quantitative skills for genomic data analysis (1): R programming basics

BBMS 3009: Genome Science (First Semester, 2021)

Dr. Yuanhua Huang / 黃淵華
School of Biomedical Sciences &
Department of Statistics and Actuarial Science

香 港 大 學
THE UNIVERSITY OF HONG KONG

Today's learning objectives (R programming)

1. Common data types
2. Common data structures and their indexing

3. Read and write tables
4. Functions and packages
5. Basic plotting and ggplot2

6. Scientific / statistical computing

Why introducing R

- R is open-source and free!

- R is popular and powerful
  - General scientific computing
  - A lot of functions for statistical modelling and inference
  - Great support for **genomic data**. Bioconductor Package platform: https://www.bioconductor.org

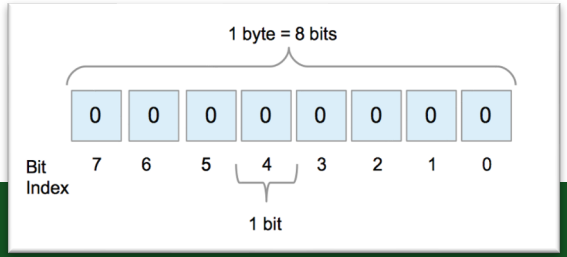
- R produces **nice plots** in a relatively easy way

- Excel is a great tool; R can do better and more elegant

1. Data type (and memory)

- numeric (also called double)
  - `x <- c(1.0, 2.0, 5.0, 7.0)`
- integer
  - `y <- c(1L, 2L, 5L, 7L)`
- logical
  - `z <- c(TRUE, TRUE, TRUE, FALSE)`
- character
  - `w <- c("aa", "bb", "5", "7")`

Check memory size with **`object.size()`**



1 kilobyte (KB) = 1024 bytes

R console

```
> x <- c(1.0, 2.0, 5.0, 7.0)
> y <- c(1L, 2L, 5L, 7L)
> z <- c(TRUE, TRUE, TRUE, FALSE)
> w <- c("aa", "bb", "5", "7")
>
> typeof(x)
[1] "double"
>
> object.size(x)
80 bytes
>
> object.size(rep(x, 1000))
32048 bytes
>
> object.size(rep(y, 1000))
16048 bytes
>
> object.size(rep(z, 1000))
16048 bytes
>
> object.size(rep(w, 1000))
32272 bytes
```

2. Data structures (1)

- A data structure is a data organization, management, and storage format that enables efficient access and modification

- **Vector**
  - `x <- c(1, 2, 5, 7)`
  - `x <- rep(3, 5)`
  - `x <- 1:12 # integer`
- **Matrix**
  - `A <- matrix(1:12, nrow=3)`
  - `B <- matrix(1:12, nrow=3, byrow=TRUE)`
  - `colnames(A) <- c("C1","C2","C3","C4")`
  - `rownames(A) <- c("R1","R2","R3")`

R console

```
> x <- 1:12
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11  12

> A <- matrix(1:12, nrow=3)
> A
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> class(A)
[1] "matrix" "array"
> dim(A)
[1] 3 4

> B <- matrix(1:12, nrow=3, byrow=TRUE)
> B
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

Data structures (1) - index

- Indexing vector and matrix
  - `x[3];  x[2:5];  x[c(2, 5, 6)];`
  - `x[c(TRUE, FALSE, FALSE, TRUE)]`

  - `A[1, 2]; A[1,c(2, 3)]; A[1, 2:4]; A[1, ]`
  - `A[1, "C3"];  A["R1", ];`
  - `A[1, 2:4, drop=FALSE]`
  - `A[c(TRUE,FALSE,TRUE), 2:4]`

- Modify values
  - `A[1, 2:4] <- c(-3, -5, 20)`

R console

```
> x <- 1:12
> x[c(2, 5, 6)]
[1] 2 5 6

> A <- matrix(1:12, nrow=3)
> colnames(A) <- c("C1","C2","C3","C4")
> rownames(A) <- c("R1","R2","R3")

> A[1, 2]
[1] 4

> A["R1", 2:4]
C2 C3 C4
 4  7 10
> dim(A["R1", 2:4])
NULL

> A[1, 2:4, drop=FALSE]
   C2 C3 C4
R1  4  7 10
> dim(A[1, 2:4, drop=FALSE])
[1] 1 3
```

Data structures (2): List and Data Frame

- **List**: a list of any data structure: value, vector, matrix, etc.
  - `x <- list(2.5, TRUE, 1:3)`
  - `x <- list("a" = 2.5, "b" = TRUE, "c" = 1:3)`
  - `x[[3]]; x[["c"]]; x$c`

```
> str(x)
List of 3
 $ a: num 2.5
 $ b: logi TRUE
 $ c: int [1:3] 1 2 3
```

- **Data Frame**: a special type of List.
  - A list of vectors with the same length.
  - Widely used as a rectangular data with flexible data type (like Excel)
  - `df <- data.frame("SN" = 1:2, "Age" = c(21,15), "Name" = c("John","Dora"))`
  - **`View(df)`**
  - `df$Age[2]`

```
> df
 SN Age Name
1  1  21 John
2  2  15 Dora

> class(df)
[1] "data.frame"
> typeof(df)
[1] "list"
```

Data Structure (3): factor vs vector

- **Factor**: a data structure used for fields that takes only predefined, finite number of values (categorical data)
  - Vector: `x = c("single", "married", "married", "single")`
  - Factor: `y = factor(c("single", "married", "married", "single"))`

```
> y <- factor(c("single", "married", "married", "single"))
> y
[1] single  married married single
Levels: married single
```

- Benefits of factor
  - As categorical data type, it uses half of the memory comparing to character and double
  - The order of the levels can be manipulated

```
> typeof(x)
[1] "character"
> typeof(y)
[1] "integer"
```

```
> z <- factor(c("single", "married", "married", "single") , levels=c("single", "married"))
> z
[1] single  married married single
Levels: single married
```

Today's learning objectives (R programming)

1. Common data types
2. Common data structures and their indexing

3. **Read and write tables**
4. **Functions and packages**
5. **Basic plotting and ggplot2**

6. Scientific / statistical computing

3. Read and write files

- Read table in .csv file (comma separated values) or .tsv file (tab-separated values)
  - `read.table()`
  - manual: https://stat.ethz.ch/R-manual/R-devel/library/utils/html/read.table.html
  - `df = read.table(csv_FILE_path, sep=",")`
  - `df = read.table(tsv_FILE_path, sep="\t")`

- Write data frame or matrix to .csv or .tsv file
  - `write.table()`
  - manual: https://stat.ethz.ch/R-manual/R-devel/library/utils/html/write.table.html
  - `write.table(df, out_csv_FILE_path, sep=",")`
  - `write.table(df, out_tsv_FILE_path, sep="\t")`

Read and write files - examples

- Download the file from Moodle:
- Lecture handouts --> Dr YH Huang --> R-materials.zip

```
> df = read.table("~/Desktop/R-materials/SRP029880.colData.tsv", sep="\t")
> df
                source_name         group
CASE_1      metastasized cancer      CASE
CASE_2      metastasized cancer      CASE
CASE_3      metastasized cancer      CASE
CASE_4      metastasized cancer      CASE
CASE_5      metastasized cancer      CASE
CTRL_1      normal colon             CTRL
CTRL_2      normal colon              CTRL
CTRL_3      normal colon             CTRL
CTRL_4      normal colon             CTRL
CTRL_5      normal colon             CTRL

> df$frozen <- c(1, 1, 0, 0, 0, 1, 1, 0, 0, 0)
> write.table(df, "~/Desktop/R-materials/SRP029880.colData.add_frozen.tsv", sep="\t", quote=FALSE)
```

4. Functions and packages

- A **function** is a set of statements organized together to perform a specific task. Many lines of codes are packed into one function & it's reusable.
- R has many build-in functions; users can also create their own functions.
  - Check how to use: `?function_name()`; `help(function_name)`
- Example functions:
  - Input values → output values: `mean(x); max(x);`
  - Input values → Do some thing: `write.table()`

- A **package** is a bundle of **functions**, data, documentation, and tests, and is easy to share with others
  - How to use functions in a package:
    - `library(package_name); function_name()`
    - `package_name::function_name()`

`mean()` refers to `base::mean()`
`write.table()` refers to `utils::write.table()`

Install package

- It depends on where the package is stored. Please refers to the documentation of the specific package you want to install and use.

- CRAN (the Comprehensive R Archive Network): **main platform**
  - For example: `install.packages("ggplot2")`

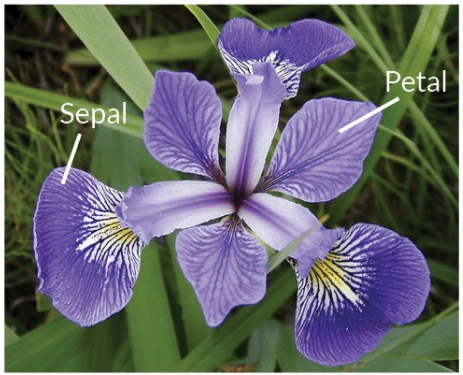
- Bioconductor: primarily for biology related packages
  - For example: `BiocManager::install("DESeq2")`

- Source codes or on GitHub:
  - For example: `devtools::install_github("tidyverse/ggplot2")`

5. Plotting – build-in dataset

- Using a build-in dataset for illustration: iris (4 flower features in 3 plants)
- Ronald Fisher in his 1936 paper
- Species:
  - Setosa
  - Versicolor
  - Virginica

```
> 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
```



Iris Versicolor

Iris Setosa

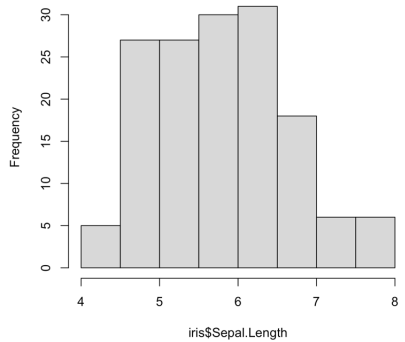
Iris Virginica

Plotting with build-in functions

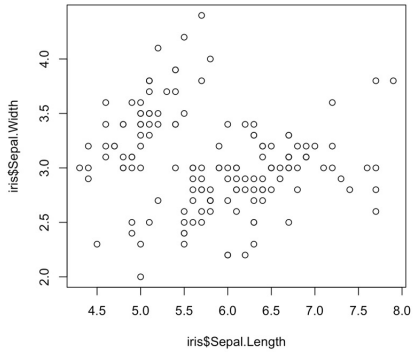
- 1D visualization
  - Histogram: `hist()`
- 2D visualization
  - Scatter plot: `plot()`
  - Box plot: `boxplot()`

```
x1 <- iris$Sepal.Length[iris$Species == "setosa"]
x2 <- iris$Sepal.Length[iris$Species == "versicolor"]
x3 <- iris$Sepal.Length[iris$Species == "virginica"]
```

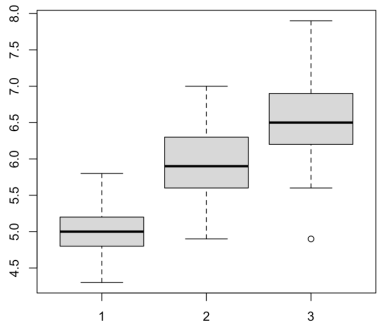
```
hist(iris$Sepal.Length)
```




```
plot(x=iris$Sepal.Length,
     y=iris$Sepal.Width)
```



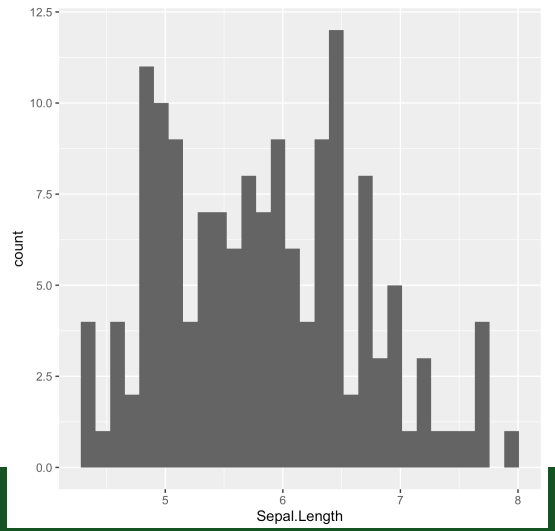

```
boxplot(x1, x2, x3)
```



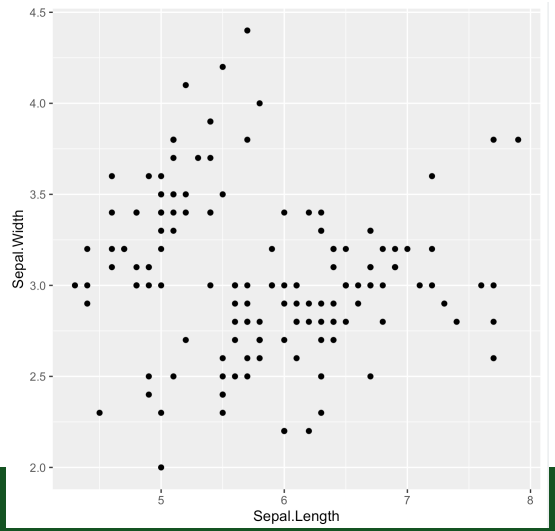
Plotting with ggplot2 package

- ggplot2 is new type of visualization, supporting various of plots
- Easy to arrange, save, and combine plots
- Gallery: https://www.r-graph-gallery.com/

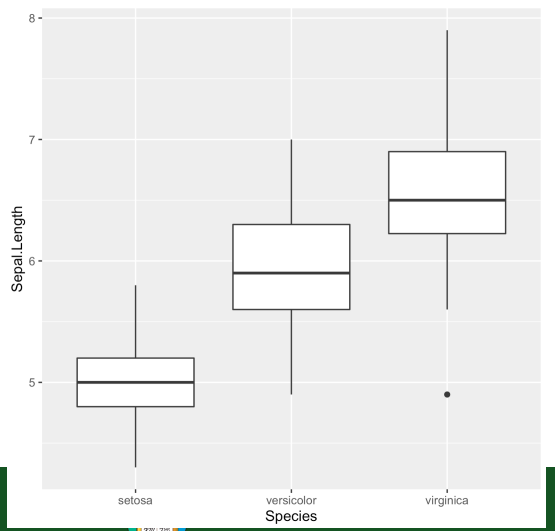
```
ggplot(iris, aes(x=Sepal.Length)) +
    geom_histogram()
```




```
ggplot(iris, aes(x=Sepal.Length,
                 y=Sepal.Width)) +
    geom_point()
```




```
ggplot(iris, aes(x=Species,
                 y=Sepal.Length)) +
    geom_boxplot()
```



Today's learning objectives (R programming)

1. Common data types
2. Common data structures and their indexing

3. Read and write tables
4. Functions and packages
5. Basic plotting and ggplot2

6. **Scientific / statistical computing**

6. Operators

Basic operation

- `6 * 5`
- `3 / 5`
- `3 %% 5`
- `3 > 5`

Operator ordering

- `2 + 6 * 5`
- `2 + (6 * 5)`
- `(2 + 6) * 5`

Quiz:

- `5 * 2 > 4`

Table. Operator Precedence in R

Operator	Description	Associativity
^	Exponent	Right to Left
-x, +x	Unary minus, Unary plus	Left to Right
%%	Modulus	Left to Right
*, /	Multiplication, Division	Left to Right
+, –	Addition, Subtraction	Left to Right
<, >, <=, >=, ==, !=	Comparisons	Left to Right
!	Logical NOT	Left to Right
&, &&	Logical AND; & for element-wise	Left to Right
|, ||	Logical OR; | for element-wise	Left to Right
->, ->>	Rightward assignment	Left to Right
<-, <<-	Leftward assignment	Right to Left
=	Leftward assignment	Right to Left

If you are not sure about a certain ordering, use brackets!

Functions for statistics

A few examples:

- Correlations: `cor(iris$Sepal.Length, iris$Petal.Length); cor.test()`
- Hypothesis testing: `t.test(x2, x3)`
- Regression:
  - `fit <- lm(y ~ x1 + x2 + x3, data=mydata)`
  - `summary(fit) # show results`
- We will go through the theory and practice next session

Additional resources: https://www.statmethods.net/stats/index.html

Resources and future self-learning

Online tutorials (the first one is probably enough):

- https://www.datamentor.io/r-programming/
- https://compgenomr.github.io/book/Rintro.html
- https://holab-hku.github.io/R-workshop/introduction-to-r.html

Recommended topics for future self-learning

1. For loop:
   - https://www.r-bloggers.com/2015/12/how-to-write-the-first-for-loop-in-r/
2. If … else:
   - https://www.r-bloggers.com/2019/06/how-to-use-if-else-statements-and-loops-in-r/
3. Write your own functions

Useful utility functions for your future use

- Matrix / data frame related:
  - `nrow(); ncol(); rownames(); colnames();`
  - `colSums(); rowSums(); colMeans(), rowMeans()`

- Summarize vector / matrix / data frame:
  - `summary()`
  - `table()`

- Patterns:
  - `match()`

Exercise (homework)

- **Make this attached plot with instructions:**
  1. Download and read data from Moodle:
     - Lecture handouts --> Dr YH Huang --> R-materials.zip
     - Unzip it and find the file: Diff_Expression_results.tsv
  2. Manipulate the dataframe
     - Add a column "log2FC_capped" for capping log2FoldChange to [-5, +5]
     - Add a column "is_DE" for "padj" < 0.05
  3. Plot it with the following ggplot2 script

```
ggplot(df, aes(x=log10(baseMean),
               y=log2FoldChange_capped)) +
  geom_point(aes(color=is_DE)) +
  scale_color_manual(values = c("red", "grey"))
```