

## Introduction to R program

เอกสารนี้จัดทำขึ้นเพื่อใช้เป็นสื่อการสอนในงาน RNA-seq analysis workshop

ซึ่งมีจุดประสงค์ขึ้นเพื่อแนะนำการใช้ R เบื้องต้น เพื่อนำไปใช้ในการวิเคราะห์ข้อมูลของ RNA-seq

ในการใช้ R เพื่อทำการวิเคราะห์ RNA-seq นั้น ผู้ใช้งานจำเป็นจะต้องมีความรู้เรื่อง basic R ต่างๆ เล็กน้อย เพื่อที่จะได้ใช้งานได้อย่างไม่ติดขัด

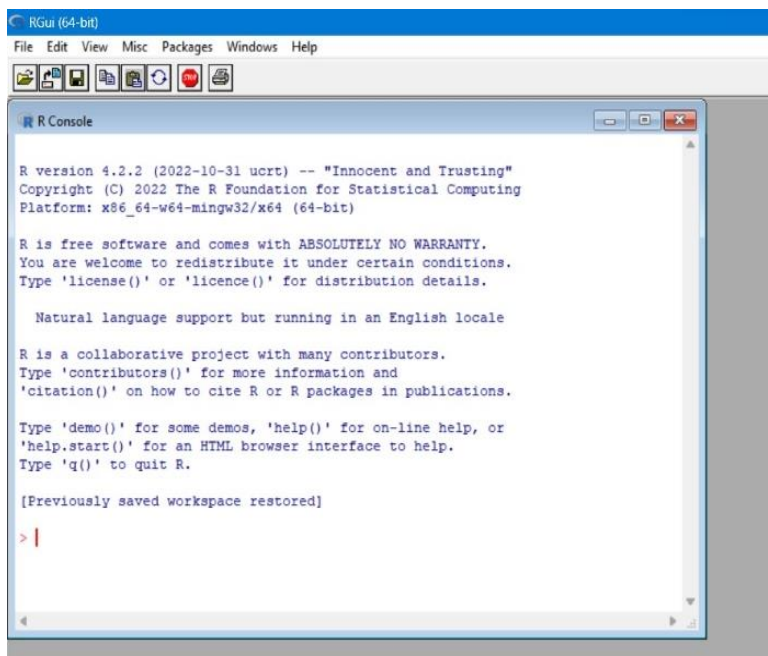
Online version: [https://tmrc.psu.ac.th/RNAseq/\\_book/index.html](https://tmrc.psu.ac.th/RNAseq/_book/index.html)

### R installation

### R console

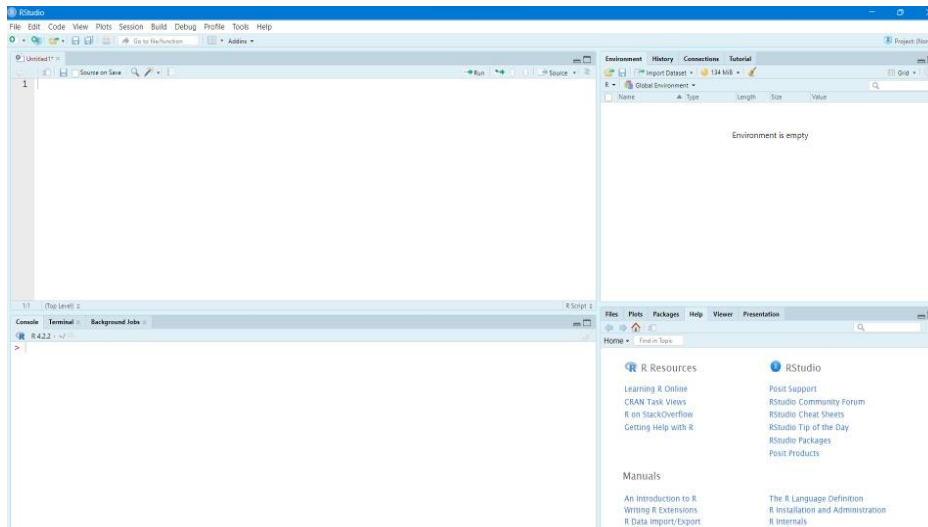
ผู้ที่ต้องการใช้ R สามารถดาวน์โหลดโปรแกรม ได้ที่ <https://cran.r-project.org/bin/windows/base/> โดยตัว

R console จะมีหน้าต่างดังภาพ



## Rstudio

อย่างไรก็ตาม การใช้งาน R ด้วยโปรแกรมนี้จะใช้งานค่อนข้างยาก โดยส่วนใหญ่ผู้ใช้งานจะต้องดาวน์โหลด IDE (integrated development environment) มาอำนวยความสะดวกในการเขียนคำสั่ง ซึ่ง IDE ที่ได้รับความนิยมมากที่สุด คือ Rstudio สามารถดาวน์โหลดได้ที่ <https://posit.co/download/rstudio-desktop/>



นี่คือหน้าต่าง default ของ Rstudio โดยส่วนประกอบหลักคือ

1. **Text editor** มุมซ้ายบน คือ ที่ๆ เราจะเขียน script ไว้เพื่อ run
2. **Environment** มุมขวาบน คือ ส่วนที่เก็บข้อมูล variable ต่างๆ ที่เรา assign
3. **R console** มุมซ้ายล่าง คือ ส่วนที่ R ทำงานจริงๆ ซึ่งก็คือ ตัว R console ที่เราโหลดมาตอนแรกนั่นเอง
4. **ส่วน Output** ที่จะมีไว้แสดงที่อยู่ของไฟล์ รูปภาพที่ render ออกมา และ อื่นๆ ตามที่เราจะปรับแต่ง

เราสามารถเขียนไว้ script ไว้ที่ text editor และกด run คำสั่งแต่ละบรรทัดได้โดยการกด Ctrl + Enter

**ยินดีด้วย!** ตอนนี้ท่านก็สามารถเริ่มใช้งาน R ได้แล้ว

## Basic R

### Basic operation

เราสามารถใช้ R ในการคำนวณต่างๆ ได้ เช่น บวก ลบ คูณ หาร ยกกำลัง เป็นต้น

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

3-2
## [1] 1

3*2
## [1] 6

3/2
## [1] 1.5

3^2
## [1] 9

log(3)
## [1] 1.098612

sqrt(3)
## [1] 1.732051

3==3 # ตรวจสอบว่าข้อมูลเหมือนกันหรือไม่
## [1] TRUE
```

## Variable

### Variable assignment

R สามารถเก็บข้อมูลต่างๆ ไว้ในตัวแปรได้ เพื่อที่สามารถนำมาใช้ในภายหลัง  
โดยการเก็บตัวแปรนั้นจะใช้เครื่องหมาย <-

```
x <- 2
x

## [1] 2
```

```

y <- 3
y
## [1] 3

x+y # เราสามารถนำตัวแปรมาทำ operation ได้ตามปกติ
## [1] 5

x*y
## [1] 6

x <- 5 # การลงข้อมูลในตัวแปรเดิมจะเป็นการลบตัวแปรเก่า
x
## [1] 5

hellothisisRNAseqworkshop <- (x+y)^(x-y) # สามารถตั้งชื่ออะไรก็ได้ราบใดที่ไม่เว้นวรรค
hellothisisRNAseqworkshop
## [1] 64

```

### Type of variable

R นั้นสามารถรองรับตัวแปรต่างๆ ได้หลากหลาย ซึ่งเป็นได้ทั้ง ตัวเลข หรือตัวอักษร หรือแม้กระทั่งเก็บหลายข้อมูลภายในตัวแปรเดียวได้

```

x <- "Hello world" # ตัวอักษร
x
## [1] "Hello world"

y <- c(1,2,3,4) # เก็บหลายตัวข้อมูลในตัวแปรเดียว
y
## [1] 1 2 3 4

z <- list(c(1,2,3), 4, c("hello world", "I love R")) # เก็บข้อมูลในรูปแบบ list
z
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] 4
##
## [[3]]
## [1] "hello world" "I love R"

class(x) # เราสามารถเช็คชนิดของตัวแปรได้โดยใช้ function class()
## [1] "character"

```

ลักษณะตัวแปรต่างๆ ใน R มีดังนี้

ชนิด	ตัวอย่าง	คำอธิบาย
numeric	1, 2.3, 5	จำนวนจริง รวมทศนิยม
integer	1, 2, 3	จำนวนเต็ม เป็น subset ของ numeric
complex	1i	จำนวนเชิงซ้อน
character	“สวัสดี”, “Hello world”	ตัวอักษร ต้องอยู่ในเครื่องหมาย ” ”
factor	“a”, “b”, “c”	คล้าย character แต่มีจำนวนตัวแปรจำกัด
logical	TRUE, FALSE	ตามหลักตรรกศาสตร์
vector	c(1,2,3)	หลายข้อมูลใน 1 ตัวแปร โดยต้องเป็นตัวแปรชนิดเดียวกัน
list	list(1, c(1,3,4), “Hello”)	หลายข้อมูลใน 1 ตัวแปร โดยไม่จำเป็นต้องเป็นตัวแปรชนิดเดียวกัน
dataframe	data.frame(x=3, y=2)	ตาราง

## Matrix and Dataframe

เนื่องจาก R นั้นเป็นโปรแกรมที่ส่วนมากใช้ในการวิเคราะห์ทางสถิติ

ซึ่งเกี่ยวข้องกับข้อมูลส่วนใหญ่จะถูกเก็บในรูปของตาราง R จึงมีตัวแปรที่เก็บข้อมูลในรูปของตารางโดยเฉพาะ เรียกว่า matrix และ dataframe ซึ่งเราจะใช้เป็นหลักในการวิเคราะห์ข้อมูลใน R

```
mat <- matrix(c(1,2,3,4), nrow=2)
mat

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

class(mat)

## [1] "matrix" "array"

df <- data.frame(x=c(3,4),y=c(2,5),z=c(4,7))
df

##    x y z
## 1 3 2 4
## 2 4 5 7

class(df)

## [1] "data.frame"
```

โดยตารางนั้นจะประกอบด้วยสองส่วนหลักๆ คล้าย excel spreadsheet ได้แก่

- Column (คอลัมน์): คือ ข้อมูลในแนวตั้ง ซึ่งแถวบนสุดจะเป็นชื่อ column นั้นๆ
- Row (แถว): คือ ข้อมูลในแนวนอน

โดย matrix นั้น สามารถเก็บ variable ในรูปแบบเดียวกันได้เท่านั้น แต่ dataframe สามารถเก็บข้อมูลต่างชนิดรวมกันได้ โดยมีข้อแม้ว่า column เดียวกัน จะต้องเป็นข้อมูลชุดเดียวกัน

## Subset

เราสามารถดึงข้อมูลแค่บางส่วนออกมาจาก vector, list, matrix หรือ dataframe ได้ เรียกว่าการ subset

```
x <- c("a","b","c","d")
x[3] # subset โดยระบุตำแหน่ง

## [1] "c"

x[1:3] # subset หลายตำแหน่ง

## [1] "a" "b" "c"

x[c(1,3)] # subset หลากหลายตำแหน่งแบบจำเพาะ

## [1] "a" "c"

y <- list(c(1,2,3), c("a","b","c"))
y[1] # subset list ตามตำแหน่ง (จะได้ list ย่อยออกมา)

## [[1]]
## [1] 1 2 3

y[[1]] # ดึงข้อมูลที่อยู่ใน list ออกมา

## [1] 1 2 3
```

ในส่วนของ matrix และ dataframe นั้น เราสามารถ subset ตามตำแหน่งได้ โดยการระบุ row และ column ตามลำดับ

```
mat

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

mat[1,2] # 1st row, 2nd column

## [1] 3
```

```
df
##    x y z
## 1 3 2 4
## 2 4 5 7

df[1,3] # 1st row, 3rd column
## [1] 4
```

ในส่วนของ dataframe นั้น เราสามารถ subset ได้โดยใช้ชื่อของ column อีกด้วย

```
df["x"] # subset เป็น column ย่อย
##    x
## 1 3
## 2 4

df[["x"]] # subset ข้อมูลที่อยู่ใน column นั้น
## [1] 3 4

df[[2, "x"]] # ระบุดัวย
## [1] 4

df$x # เหมือนกัน df[["x"]]
## [1] 3 4
```

## R function

function (ฟังก์ชัน) คือ ชุดของคำสั่งที่จะสั่งการให้ R ทำงานตามจุดประสงค์ที่เราตั้งไว้ โดยตัว function นั้นจะประกอบไปด้วย

- function ที่มีมาพร้อมกับ R ตั้งแต่ต้น (base R function)
- function ที่ผู้นิพนธ์ท่านอื่นเขียนไว้ และรวบรวมมาเป็น ชุดของ function เรียกว่า package
- function ที่เราเขียนขึ้นมาเอง

## Anatomy of function

function นั้นประกอบด้วย 4 ส่วน คือ 1. Function name (ชื่อฟังก์ชัน) 2. Argument (รายละเอียดของฟังก์ชัน) 3. Function body (รายละเอียดของฟังก์ชัน) 4. Return (ผลลัพธ์ของฟังก์ชัน)

ยกตัวอย่างฟังก์ชันหา ค่าเฉลี่ยของข้อมูล

```
find_mean <- function(x, y){
  (x + y)/2
}

find_mean(2, 3)
## [1] 2.5

find_mean(3, 5)
## [1] 4
```

จะเห็นว่า **function** นี้รับข้อมูล 2 ตัวแปร คือ **x** และ **y** ซึ่งเราจะต้องแทนค่าที่เราต้องการลงไป ใน **function** หลังจากนั้น **function** จะทำการประมวลผลและส่งผลลัพธ์กลับมา

ในผู้เริ่มต้น ส่วนใหญ่เรามักจะไม่ใช้ function ที่เขียนขึ้นมาเองมากนัก เนื่องจาก basic operation ส่วนใหญ่จะมีผู้นิพนธ์ขึ้นมาให้แล้ว

## Base R function

Base R function คือ function ที่ติดกับ R มาตั้งแต่แรก ซึ่งเราสามารถเรียกใช้ได้เลยโดยไม่ต้องทำการเรียก package ขึ้นมาก่อน

```
max(c(1,2,4,5,5,68)) # find max value
```



```
## [1] 68
min(c(1,4,5,6,-20)) # find min value
## [1] -20
mean(c(1,2,3,4)) # find mean
## [1] 2.5
median(c(1,2,5,3,4)) # find median
## [1] 3
unique(c(1,1,1,1,2,2,4,5,5,6,7,8)) # display only unique values
## [1] 1 2 4 5 6 7 8
```

ในส่วนของการ manipulate dataframe นั้น คำสั่งต่างๆ ที่น่ารู้มีดังนี้

```
df <- data.frame(x=c(3,3,6,7,8,9),y=c(2,5,8,1,2,3),z=c(4,7,9,4,7,8))
df

##   x y z
## 1 3 2 4
## 2 3 5 7
## 3 6 8 9
## 4 7 1 4
## 5 8 2 7
## 6 9 3 8

head(df, 5) # ดู 5 แถวแรก

##   x y z
## 1 3 2 4
## 2 3 5 7
## 3 6 8 9
## 4 7 1 4
## 5 8 2 7

tail(df, 5) # ดู 5 แถวล่าง

##   x y z
## 2 3 5 7
## 3 6 8 9
## 4 7 1 4
## 5 8 2 7
## 6 9 3 8

rowMeans(df) # หาค่า mean แต่ละแถว
## [1] 3.000000 5.000000 7.666667 4.000000 5.666667 6.666667
```

```
colMeans(df) # หาค่า mean แต่ละ columns
```

```
##      x      y      z
## 6.0  3.5  6.5
```

```
rownames(df) # ชื่อแถว
```

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

```
colnames(df) # ชื่อ column
```

```
## [1] "x" "y" "z"
```

สามารถดู base R function ทั้งหมดได้ที่ <https://stat.ethz.ch/R-manual/R-devel/library/base/html/00Index.html>

ถ้าเราต้องการดูว่า function นั้นใช้งานอย่างไร ให้ใส่เครื่องหมาย? หน้า function นั้น

## Tidyverse



Tidyverse เป็น package ซึ่งนิพนธ์โดย Haley Wickham และคณะ โดย function ส่วนใหญ่ใน tidyverse นั้นเกี่ยวข้องกับการปรับแต่งข้อมูลจาก dataframe ซึ่งจะอำนวยความสะดวกให้เราสามารถทำงานได้มากขึ้นกว่าการใช้ base R ข้อเสียของ tidyverse นั้น อาจจะทำให้ run ช้ากว่า และมีปรับแต่งให้ตรงกับการใช้งานจำเพาะได้ยากกว่า แต่สำหรับผู้ที่ไม่ใช่ R hardcore นั้น tidyverse ถือว่าเป็น package ที่อำนวยความสะดวกได้อย่างดีเยี่ยม

โดย tidyverse นั้นจะเป็น package ใหญ่ และจะแบ่งเป็นหลาย package ย่อยๆ ได้อีก โดยเราสามารถเรียกใช้ทั้งหมดได้ หรือ เรียกใช้แค่ package ย่อย

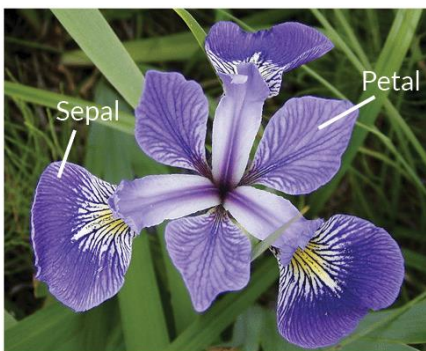
## dplyr

dplyr คือ package ย่อยของ tidyverse ซึ่งทำหน้าที่ในส่วน dataframe manipulation ทำให้เราสามารถดึงตารางออกมาได้อย่างอิสระ

การใช้งาน package ข้างบนนั้นจะต้อง install ก่อน และเมื่อใช้งาน จะต้องใช้คำสั่ง library

```
# install.packages("tidyverse") รันคำสั่งนี้ก่อนถ้ายังไม่เคย install
library(dplyr) # ต้อง run ทุกครั้งที่ใช้งาน
```

ในกรณีนี้จะใช้ข้อมูลตัวอย่าง iris เพื่อสาธิตการใช้ dplyr โดย iris เป็นข้อมูลของความยาวกลีบของพันธุ์ดอกไม้ต่างๆ



**Iris Versicolor**



**Iris Setosa**



**Iris Virginica**

รูปจาก: <https://www.datacamp.com/tutorial/machine-learning-in-r>

```
df <- iris # โหลด dataframe ตัวอย่างที่ติดมากับ base R
head(df, 5)
```

##	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

function หลักๆ ของ dplyr จะเกี่ยวข้องกับ data manipulation เป็นส่วนใหญ่

ในที่นี้จะแนะนำที่จำเป็นต้องใช้ในบ่ออื่น

- glimpse() มีไว้ดูภาพรวมข้อมูล



```
## Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          6.3          3.3          6.0          2.5 virginica
## 2          5.8          2.7          5.1          1.9 virginica
## 3          7.1          3.0          5.9          2.1 virginica
## 4          6.3          2.9          5.6          1.8 virginica
## 5          6.5          3.0          5.8          2.2 virginica
```

# เลือกแถวที่ *Species = setosa*, *Sepal.Length = 5.4*

```
df %>%
  filter(Species == "setosa" & Sepal.Length == 5.4) %>% head(5)
```

```
## Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.4          3.9          1.7          0.4 setosa
## 2          5.4          3.7          1.5          0.2 setosa
## 3          5.4          3.9          1.3          0.4 setosa
## 4          5.4          3.4          1.7          0.2 setosa
## 5          5.4          3.4          1.5          0.4 setosa
```

# เลือกแถวที่ *Sepal.Length = 5.1* หรือ *4.9*

```
df %>% filter(Sepal.Length == 5.1 | Sepal.Length == 4.9) %>% head(10)
```

```
## 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          4.9          3.1          1.5          0.1 setosa
## 4          5.1          3.5          1.4          0.3 setosa
## 5          5.1          3.8          1.5          0.3 setosa
## 6          5.1          3.7          1.5          0.4 setosa
## 7          5.1          3.3          1.7          0.5 setosa
## 8          4.9          3.1          1.5          0.2 setosa
## 9          4.9          3.6          1.4          0.1 setosa
## 10         5.1          3.4          1.5          0.2 setosa
```

สังเกตว่าจะเห็นเครื่องหมาย `%>%` ซึ่งใน R เราจะเรียกว่า “pipe operator” เป็นสิ่งที่เป็เอกลักษณ์ใน R ซึ่งส่งผลให้สามารถ run operation ได้ต่อๆ กัน เพื่อให้อ่านได้ง่าย

# เลือกแถวที่ *Species = setosa* คอลัมน์ *Sepal.Length*

```
df %>%
  filter(Species == "setosa") %>%
  select(Sepal.Length) %>% head(5)
```

```
## Sepal.Length
## 1          5.1
## 2          4.9
## 3          4.7
## 4          4.6
## 5          5.0
```

# เหมือนกับข้างบน แต่ไม่ใช่ *pipe operator* จะทำความเข้าใจได้ยากกว่า

```
select(filter(df, Species == "setosa"), Sepal.Length) %>% head(5)
```

```
## Sepal.Length
## 1      5.1
## 2      4.9
## 3      4.7
## 4      4.6
## 5      5.0

# ใช้แค่ base R solution จะไม่สามารถดึงออกมาเป็น dataframe ได้
df[df["Species"] == "setosa", "Sepal.Length"]

## [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5
## [20] 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0 5
## [39] 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0
```

บรรทัดสุดท้าย สำหรับ dataframe จะไม่สามารถดึงมาทั้ง column ได้ ซึ่งจะต้องใช้ข้อมูลอีกแบบ (tibble) แต่จะไม่พูดถึง ณ ที่นี้

**Note:** การ subset โดย dplyr นั้นสามารถทำใน dataframe/tibble เท่านั้น ไม่สามารถทำใน matrix ได้ (ต้องใช้วิธีของ base R)

- ในส่วนการเรียงข้อมูลนั้นจะใช้ function arrange()

```
df %>%
  arrange(Sepal.Length) %>% head(5) # เรียง Sepal.Length จากน้อยไปมาก

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1      4.3      3.0      1.1      0.1 setosa
## 2      4.4      2.9      1.4      0.2 setosa
## 3      4.4      3.0      1.3      0.2 setosa
## 4      4.4      3.2      1.3      0.2 setosa
## 5      4.5      2.3      1.3      0.3 setosa

df %>%
  arrange(desc(Sepal.Length)) %>% head(5) # เรียง Sepal.Length จากมากไปน้อย

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1      7.9      3.8      6.4      2.0 virginica
## 2      7.7      3.8      6.7      2.2 virginica
## 3      7.7      2.6      6.9      2.3 virginica
## 4      7.7      2.8      6.7      2.0 virginica
## 5      7.7      3.0      6.1      2.3 virginica
```

- เราสามารถจัดกลุ่มตัวแปรได้โดยใช้ group\_by() โดยมักจะใช้คู่กับ summarize()

```
df %>%
  group_by(Species) %>% # จัดกลุ่มตาม Species
```

```

summarize(Sepal.Length = sum(Sepal.Length), Sepal.Width = sum(Sepal.Width))
#รวมความยาวทั้งหมด

## # A tibble: 3 × 3
##   Species    Sepal.Length Sepal.Width
##   <fct>         <dbl>         <dbl>
## 1 setosa         250.           171.
## 2 versicolor    297.           138.
## 3 virginica     329.           149.

```

## ggplot2

ggplot2 คือ package ย่อยอีกตัวของ tidyverse ซึ่งใช้สำหรับการ plot graph

## Anatomy of ggplot

```

ggplot(data = data, aes(x = x, y = y, col = col, fill = fill)) +
  geom_xxx() +
  theme_xxx()

```

- aes คือ aesthetic ซึ่งหมายถึงการ map ข้อมูลของเราเข้ากับตำแหน่งของกราฟ
  - x = แกน x, y = แกน y
  - col = สี, fill = สีพื้นหลัง
- geom\_xxx() คือ การกำหนดว่าเราต้องการที่จะ plot กราฟอะไร
  - geom\_point() = scatterplot
  - geom\_line() = lineplot
  - geom\_boxplot() = boxplot
- theme\_xxx() คือ การกำหนด theme ของกราฟ เช่น theme\_bw(), theme\_classic()
- และยังมีการปรับแต่งอื่นๆ ได้อีกมาก สามารถศึกษาได้ที่ <https://ggplot2.tidyverse.org/reference/>

## Scatterplot

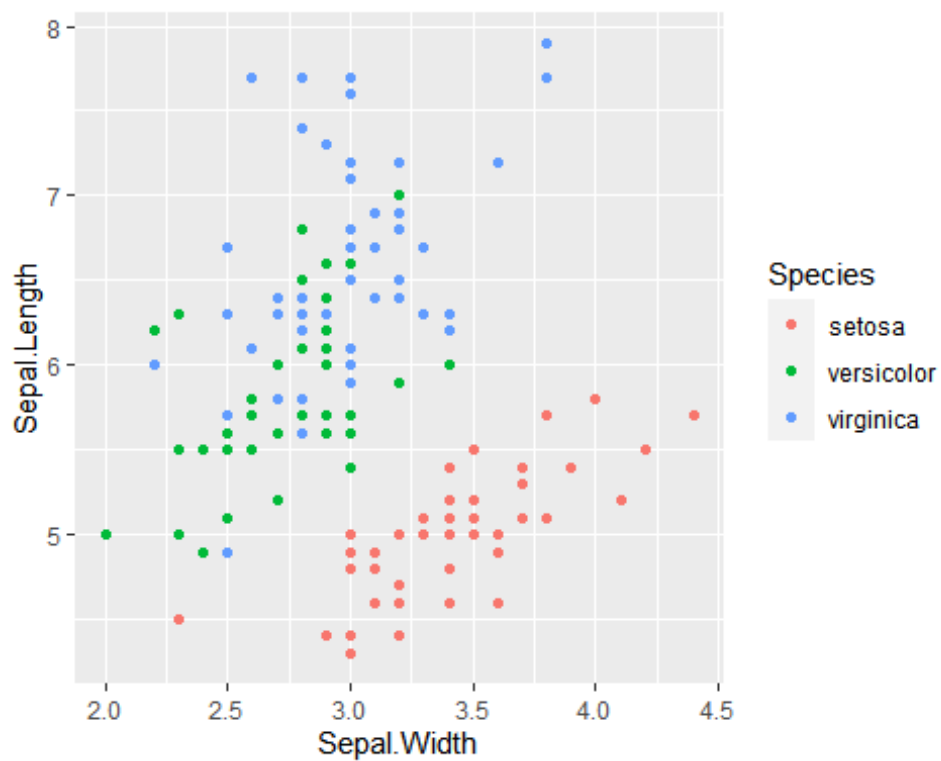
```

# install.packages("tidyverse") รันคำสั่งนี้ก่อนถ้ายังไม่เคย install
library(ggplot2)

ggplot(df, aes(x = Sepal.Width, y = Sepal.Length, col = Species)) + geom_poin
t()

```

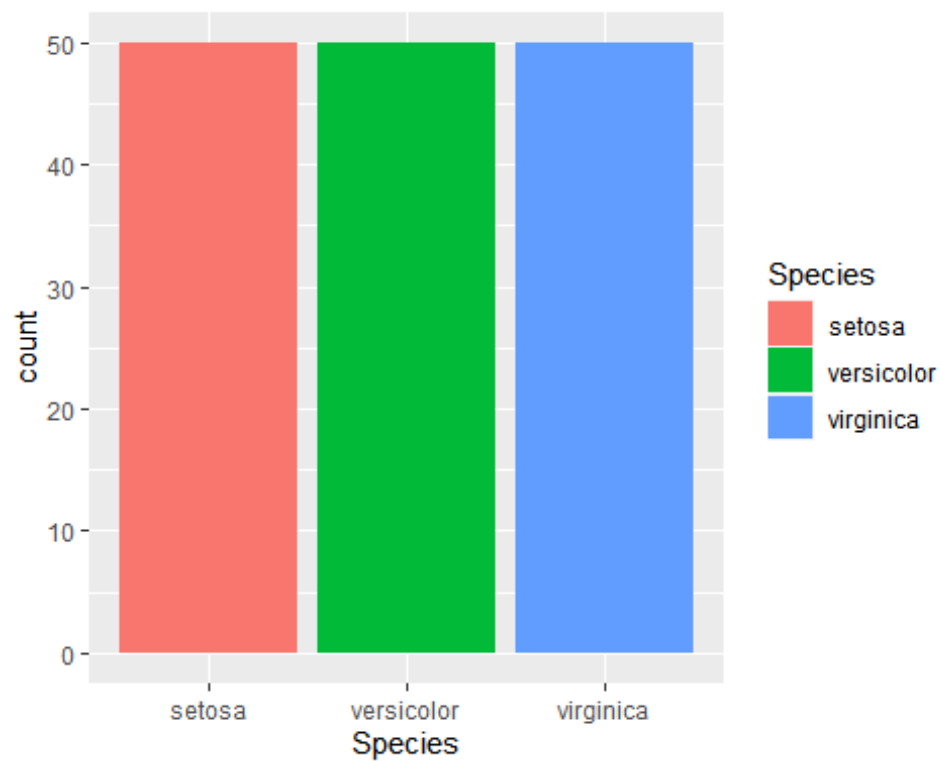




## Barchart

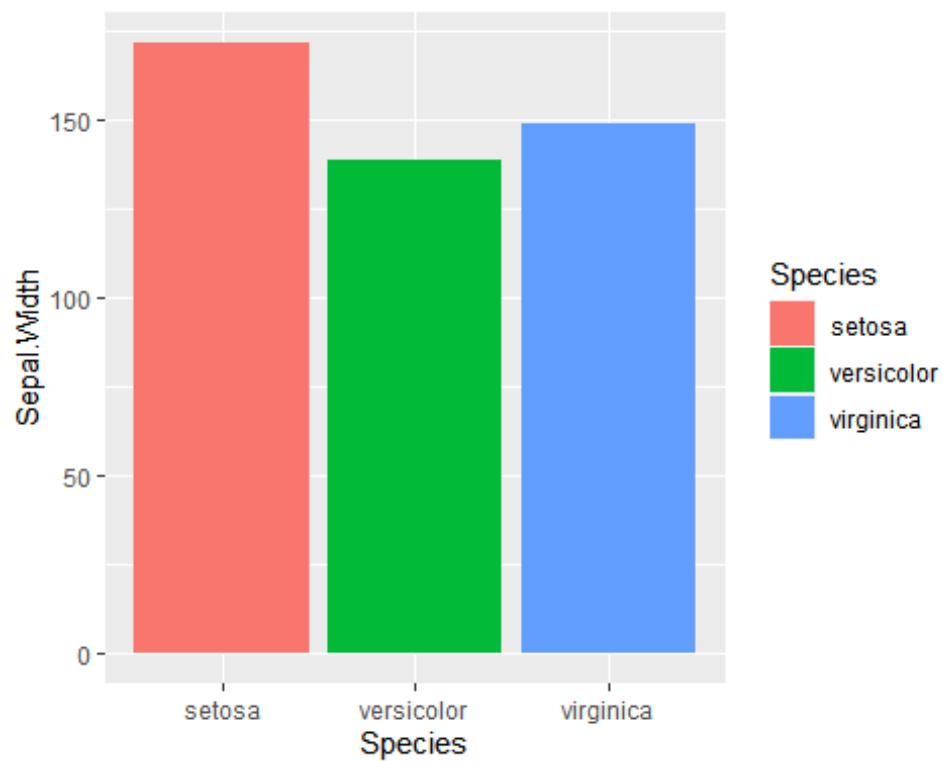
ใช้สำหรับนับจำนวนของ column นั้น ไม่มีค่า y

```
ggplot(df, aes(x = Species, fill = Species)) + geom_bar() # fill ไว้สำหรับแบ่งสีใน bar chart
```



ส่วน `geom_col()` จะรับค่า `y` ด้วย โดยข้อมูล `x` ที่ซ้ำกันจะถูกนำมารวมกัน (สามารถปรับแต่งได้เพิ่มเติม)

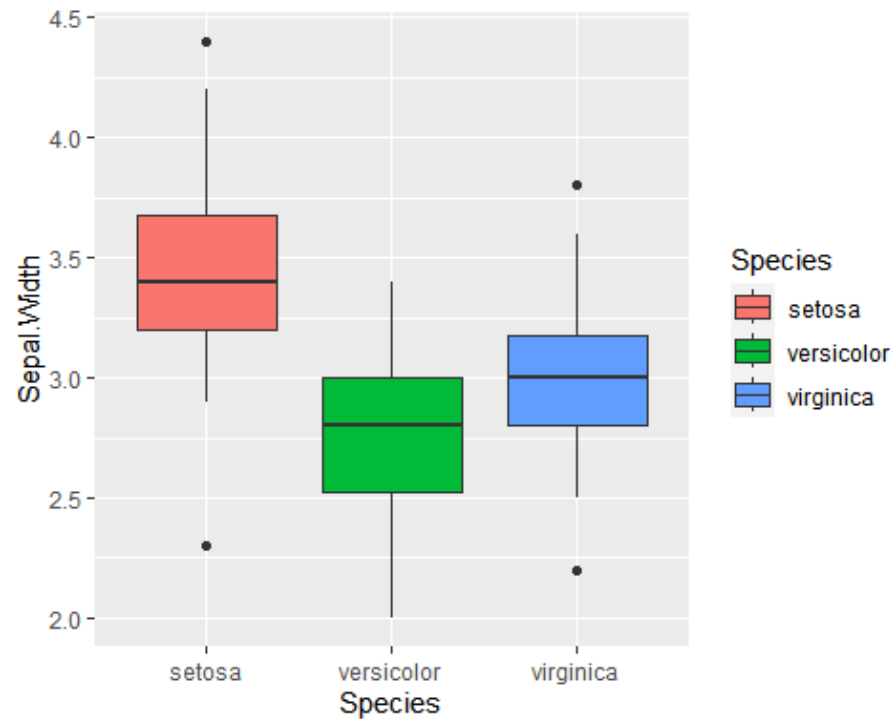
```
ggplot(df, aes(x = Species, y = Sepal.Width, fill = Species)) + # fill ไว้สำหรับแบ
งสีใน barchart
  geom_col()
```



## Boxplot

ทำการสร้าง box plot

```
ggplot(df, aes(x = Species, y = Sepal.Width, fill = Species)) +  
  geom_boxplot()
```

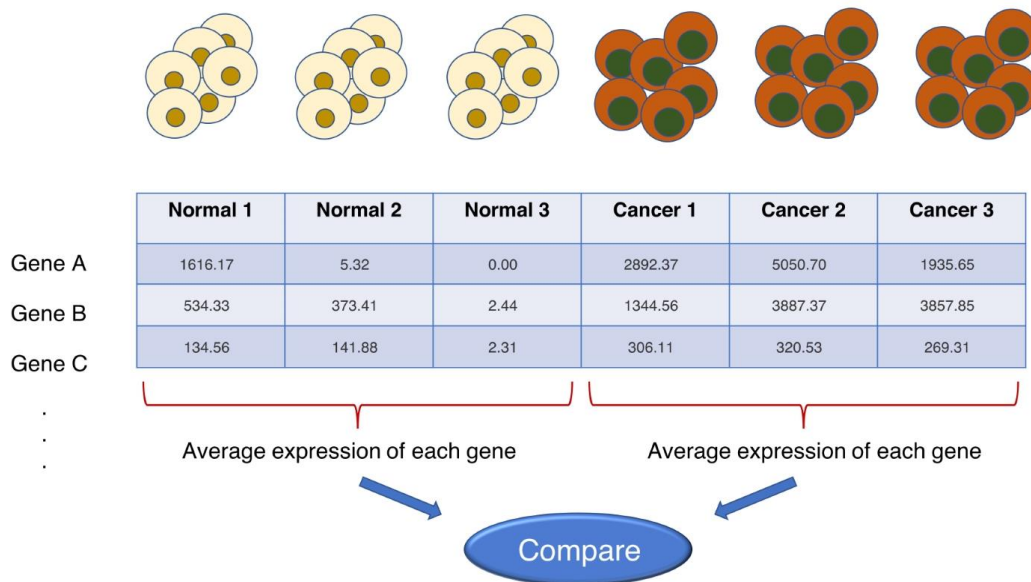


## RNA-seq data analysis

### What is differential gene expression?

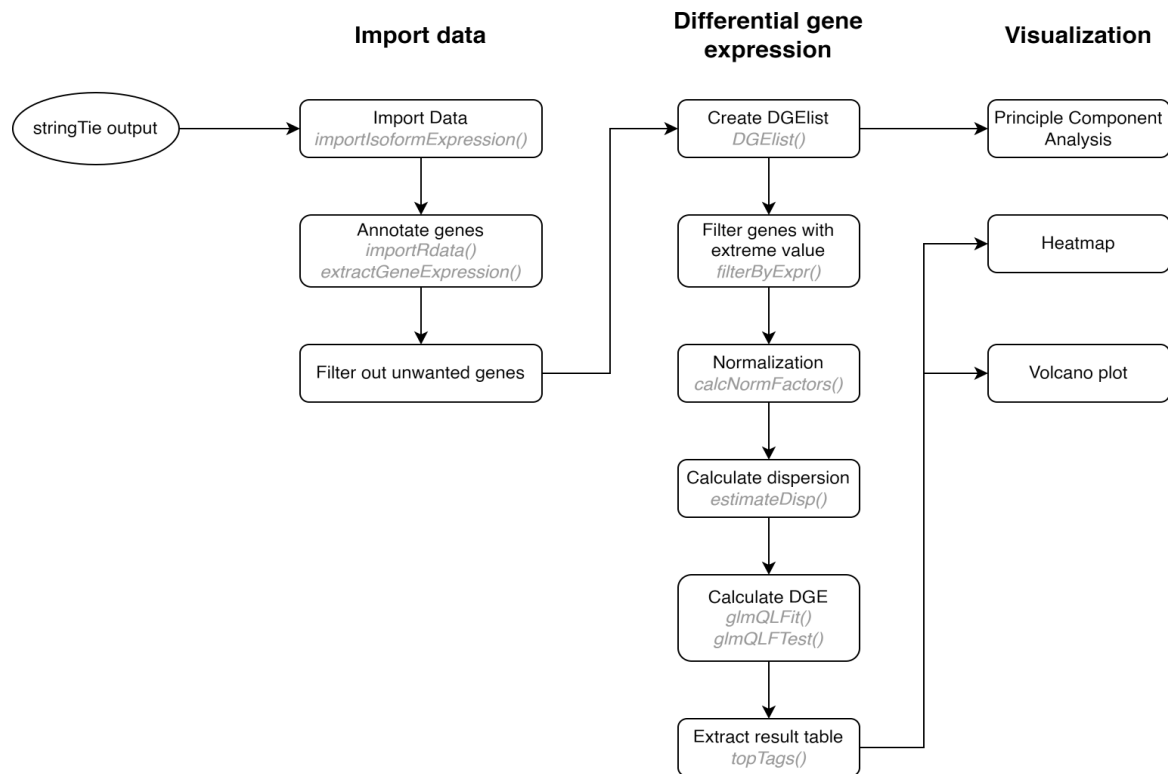
Differential gene expression คือ การหาความแตกต่างของการแสดงออกของ gene

ระหว่างกลุ่มตัวอย่างสองกลุ่มขึ้นไป เพื่อให้ได้ผลลัพธ์ว่ามี gene ตัวใดตัวหนึ่งแสดงออกมากหรือน้อยกว่าผิดปกติ เมื่อเทียบกับกลุ่มอื่นๆ โดยค่าที่นำมาใช้เปรียบเทียบนั้น จะได้มาจากขั้นตอน quantification














## Differential gene expression workflow

หลังจากเราได้ข้อมูล Gene expression quantification จาก StringTie แล้ว  
 เราจะนำข้อมูลมาผ่านกระบวนการต่างๆ เพื่อหาความแตกต่างของ gene แต่ละกลุ่ม



## Import data

ในการที่จะนำไฟล์ RNA analysis เข้าสู่ R นั้น จำเป็นที่จะต้องเตรียมข้อมูลให้เหมาะกับ function ที่เราจะใช้ในการอ่านข้อมูล โดยในแต่ละ sample นั้น จะประกอบด้วยไฟล์ .gtf และ .ctab ที่ได้จากการวิเคราะห์ก่อนหน้านี้

 N1	12/02/2023 16:10	File folder	
 N2	12/02/2023 16:10	File folder	
 N3	12/02/2023 16:10	File folder	
 T1	12/02/2023 16:10	File folder	
 T2	12/02/2023 16:10	File folder	
 T3	12/02/2023 16:10	File folder	
 Merge_full.gtf	12/02/2023 17:34	GTF File	394,439 KB
 e_data.ctab	12/02/2023 09:03	CTAB File	55,704 KB
 e2t.ctab	12/02/2023 09:03	CTAB File	28,337 KB
 ExNC02.gtf	12/02/2023 09:03	GTF File	384,988 KB
 i_data.ctab	12/02/2023 09:04	CTAB File	17,932 KB
 i2t.ctab	12/02/2023 09:04	CTAB File	22,511 KB
 t_data.ctab	12/02/2023 09:04	CTAB File	37,902 KB

## Import quantification

โดยเราจะใช้ function `importIsoformExpression` ในการนำข้อมูลเข้าให้อยู่ในรูปของ dataframe

```
stringTie_quant <- importIsoformExpression(
  parentDir = "./Source/bladder",
  addIsoformIdAsColumn = FALSE,
  readLength = 150
)

## Step 1 of 3: Identifying which algorithm was used...
## The quantification algorithm used was: StringTie
## Found 6 quantification file(s) of interest
## Step 2 of 3: Reading data...
## reading in files with read_tsv

## 1 2 3 4 5 6
## Step 3 of 3: Normalizing abundance values (not counts) via edgeR...
## Done

head(stringTie_quant$abundance, 10)
```

	N1	N2	N3	T1	T2
T3					
## MSTRG.24.1	0.7207532	0.00000000	0	0.00000000	0.00000000
## MSTRG.24.2	0.0000000	1.669273712	0	0.00000000	0.00000000
## MSTRG.24.4	0.0000000	0.493513530	0	0.00000000	0.00000000

```

00
## MSTRG.24.5      0.0000000 0.006439551 0 0.000000000 0.00000000 0.000000
00
## MSTRG.24.3      0.3298251 1.521147453 0 0.004002289 0.00000000 0.000000
00
## MSTRG.24.6      1.7884048 1.259405992 0 0.000000000 0.01820757 0.000000
00
## ENST00000456328.2 0.5592946 0.131153423 0 0.049252195 0.01158960 0.000000
00
## ENST00000450305.2 0.0000000 0.000000000 0 0.000000000 0.00000000 0.000000
00
## MSTRG.26.1      0.0000000 0.000000000 0 1.063266717 0.18918203 0.424279
10
## MSTRG.26.4      0.0000000 0.000000000 0 0.000000000 0.04498711 0.094081
83

```

จะเห็นว่าในไฟล์นั้นประกอบด้วย ส่วนแถว ซึ่งเป็นชื่อ isoform ของ RNA นั้นๆ และ ส่วนคอลัมน์ ซึ่งเป็นชื่อของ sample ที่เราศึกษา โดยข้อมูลแต่ละจุดคือ ค่าของ expression ที่ได้จากการวิเคราะห์

## Make a design matrix

หลังจากนั้น เราต้องสร้าง condition matrix ซึ่งประกอบด้วย แต่ละ sample ที่ต้องการศึกษา และ condition ของตัวอย่างนั้น ซึ่งในที่นี้เราจะแบ่งเป็นสองกลุ่ม ก็คือ Normal และ Tumor

```

design <- data.frame(
  sampleID = colnames(stringTie_quant$abundance),
  condition = gsub(".{1}$", "", colnames(stringTie_quant$abundance)) # Remove
number
)

design
##   sampleID condition
## 1      N1         N
## 2      N2         N
## 3      N3         N
## 4      T1         T
## 5      T2         T
## 6      T3         T

```

## Create a list of files

หลังจากนั้นเราจะต้องรวมไฟล์เข้ากันกับ annotation file ซึ่งจะทำให้การ annotate ชื่อ gene นั้น จาก Ensemble format เป็น gene id



```

switch_analyze_Rlist <- importRdata(
  isoformCountMatrix = stringTie_quant$counts,
  isoformRepExpression = stringTie_quant$abundance,
  designMatrix = design,
  isoformExonAnnoation = "./Source/bladder/BCaMerge.gtf",
)

## comparison estimated_genes_with_dtu
## 1      N vs T      4768 - 7948

names(switch_analyze_Rlist)

## [1] "isoformFeatures"      "exons"                "conditions"
## [4] "designMatrix"          "sourceId"              "isoformCountMatrix"
## [7] "isoformRepExpression" "runInfo"                "isoformRepIF"

```

สังเกตว่าภายใน 1 list นั้นจะประกอบด้วยหลายหัวข้อ ซึ่งเราสามารถดึงออกมาใช้ได้ด้วย operator \$

## Extract gene count matrix

ต่อไปเราจะใช้แค่ gene count matrix จาก list ที่เราสร้างขึ้นมา

```

gene_count <- extractGeneExpression(
  switch_analyze_Rlist,
  extractCounts = TRUE # set to FALSE for abundances
)

head(gene_count, 10)

```

##		gene_id	gene_name	N1	N2	N3	T
1							
## 1	ENSG000000000003.15	TSPAN6	1616.16632	5.323296	0.000000	2892.3669	
8							
## 2	ENSG0000000000419.14	DPM1	534.33260	373.409670	2.438447	1344.5619	
3							
## 3	ENSG0000000000457.14	SCYL3	134.55807	141.880574	2.312303	306.1116	
5							
## 4	ENSG0000000000460.17	C1orf112	68.44894	47.264863	17.260795	117.6624	
5							
## 5	ENSG0000000000938.13	FGR	40.13006	3177.326075	62.927494	10.9845	
1							
## 6	ENSG0000000000971.16	CFH	801.43355	8.169078	5.188639	1304.6613	
5							
## 7	ENSG0000000001036.14	FUCA2	246.33240	104.995451	5.653375	1192.9117	
6							
## 8	ENSG0000000001084.13	GCLC	374.82061	77.947080	6.817594	4565.3556	
4							
## 9	ENSG0000000001167.15	NFYA	128.75396	215.173089	0.000000	526.5242	
2							

```
## 10 ENSG0000001460.18      STPG1  105.25642    27.359621   3.458716   253.4063
9
##           T2           T3
## 1    5050.6964  1935.65287
## 2    3887.3732  3857.84694
## 3     320.5272   269.31308
## 4     207.5864    72.52807
## 5     192.8986   109.53769
## 6   13678.3377  2457.62369
## 7    3405.1256  1728.34830
## 8    1549.1920  3010.92600
## 9     783.7322   664.80964
## 10    370.8472   179.81425
```

จะเห็นว่าขณะนี้เรามีทั้ง gene\_id และ gene\_name แล้ว

## Filter out lncRNA

ต่อไป เราจะนำรายชื่อของ RNA ที่เราไม่สนใจออกไป ซึ่งในที่นี้คือ long-noncoding RNA

ซึ่งมักจะไม่ถูกเปลี่ยนไปเป็นโปรตีน แต่จะใช้สำหรับ function อื่นๆ ในร่างกาย

ก่อนอื่น เราต้อง import file ที่มีการ annotate ชนิดของ RNA เข้ามาใน R ก่อน โดยใช้ function

```
rtracklayer::import()
```

```
V38.gtf <- rtracklayer::import("./Source/gencode.v38.annotation.gtf")
unique(V38.gtf$gene_type)

## [1] "transcribed_unprocessed_pseudogene" "unprocessed_pseudogene"
## [3] "miRNA"                             "lncRNA"
## [5] "protein_coding"                     "processed_pseudogene"
## [7] "snRNA"                             "transcribed_processed_pseudogen
e"
## [9] "misc_RNA"                          "TEC"
## [11] "transcribed_unitary_pseudogene"     "snoRNA"
## [13] "scaRNA"                            "rRNA_pseudogene"
## [15] "unitary_pseudogene"                 "polymorphic_pseudogene"
## [17] "pseudogene"                        "rRNA"
## [19] "IG_V_pseudogene"                   "scRNA"
## [21] "IG_V_gene"                         "IG_C_gene"
## [23] "IG_J_gene"                         "sRNA"
## [25] "ribozyme"                          "translated_processed_pseudogene"
"
## [27] "vault_RNA"                         "TR_C_gene"
## [29] "TR_J_gene"                         "TR_V_gene"
## [31] "TR_V_pseudogene"                   "translated_unprocessed_pseudoge
ne"
## [33] "TR_D_gene"                         "IG_C_pseudogene"
```

```
## [35] "TR_J_pseudogene"      "IG_J_pseudogene"
## [37] "IG_D_gene"            "IG_pseudogene"
## [39] "Mt_tRNA"              "Mt_rRNA"
```

จะเห็นว่ามีชนิดของ RNA มากมายหลายชนิดในไฟล์นี้ เราจะทำการเลือกชื่อ RNA ที่เราไม่สนใจ ซึ่งก็คือ lncRNA มากรองข้อมูลในส่วนที่เราไม่ต้องการออกในไฟล์ต้นฉบับของเรา

หลังจากนั้นเราจะนำ column gene\_id ออก และเปลี่ยน gene\_name ให้เป็นชื่อแถว

**Note:** ในที่ข้อมูลนี้เราจะทำการตัด RNA ที่มีชื่อซ้ำออกไป เพื่อให้ง่ายแก่การสอน

ซึ่งในการวิเคราะห์จริงอาจจะต้องใช้วิธีอื่นในการวิเคราะห์ชื่อ RNA ที่ซ้ำกันใน sample เดียวกัน

```
lncRNA_subset <- V38.gtf$gene_type == "lncRNA"
lncRNA <- V38.gtf[lncRNA_subset]$gene_name

gene_count_no_lncRNA <- gene_count %>% filter(!(gene_name %in% unique(lncRNA)))
head(gene_count_no_lncRNA, 10)
```

```
##           gene_id gene_name      N1      N2      N3      T
1
## 1  ENSG0000000003.15    TSPAN6 1616.16632  5.323296  0.000000 2892.3669
8
## 2  ENSG00000000419.14      DPM1  534.33260 373.409670  2.438447 1344.5619
3
## 3  ENSG00000000457.14     SCYL3  134.55807 141.880574  2.312303  306.1116
5
## 4  ENSG00000000460.17  C1orf112   68.44894  47.264863 17.260795  117.6624
5
## 5  ENSG00000000938.13      FGR   40.13006 3177.326075 62.927494   10.9845
1
## 6  ENSG00000000971.16      CFH  801.43355   8.169078  5.188639 1304.6613
5
## 7  ENSG00000001036.14     FUCA2  246.33240 104.995451  5.653375 1192.9117
6
## 8  ENSG00000001084.13     GCLC  374.82061  77.947080  6.817594 4565.3556
4
## 9  ENSG00000001167.15     NFYA  128.75396 215.173089  0.000000  526.5242
2
## 10 ENSG00000001460.18    STPG1  105.25642  27.359621  3.458716  253.4063
9
##           T2           T3
## 1  5050.6964 1935.65287
## 2  3887.3732 3857.84694
## 3   320.5272  269.31308
## 4   207.5864   72.52807
## 5   192.8986  109.53769
## 6 13678.3377 2457.62369
```

```
## 7 3405.1256 1728.34830
## 8 1549.1920 3010.92600
## 9 783.7322 664.80964
## 10 370.8472 179.81425
```

### ## Get only count matrix

```
count_matrix <- gene_count_no_lncRNA %>%
  distinct(gene_name, .keep_all = TRUE) %>% # Remove duplicate gene_name
  column_to_rownames("gene_name") %>%
  select(-gene_id) %>%
  as.matrix
```

```
head(count_matrix, 10)
```

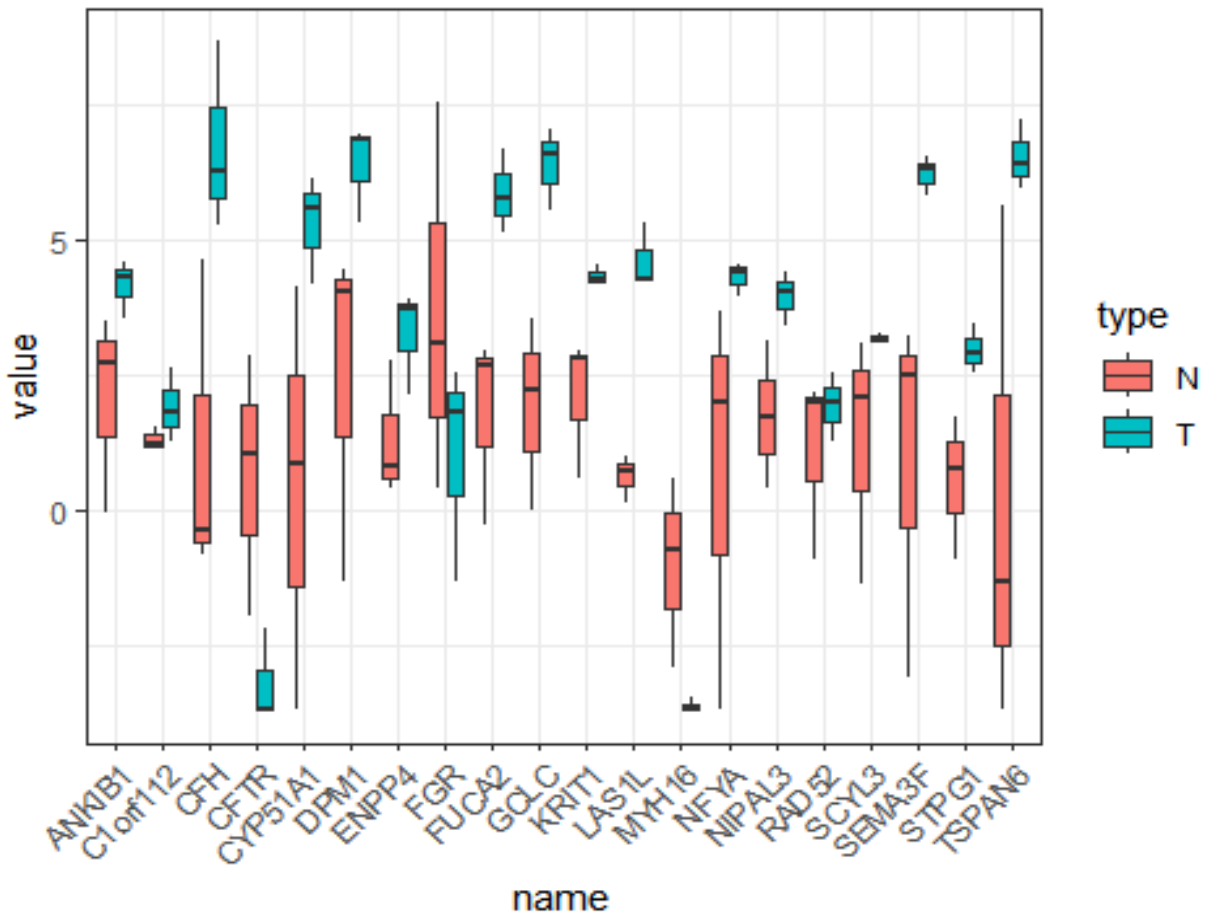
##		N1	N2	N3	T1	T2	T3
##	TSPAN6	1616.16632	5.323296	0.000000	2892.36698	5050.6964	1935.65287
##	DPM1	534.33260	373.409670	2.438447	1344.56193	3887.3732	3857.84694
##	SCYL3	134.55807	141.880574	2.312303	306.11165	320.5272	269.31308
##	C1orf112	68.44894	47.264863	17.260795	117.66245	207.5864	72.52807
##	FGR	40.13006	3177.326075	62.927494	10.98451	192.8986	109.53769
##	CFH	801.43355	8.169078	5.188639	1304.66135	13678.3377	2457.62369
##	FUCA2	246.33240	104.995451	5.653375	1192.91176	3405.1256	1728.34830
##	GCLC	374.82061	77.947080	6.817594	4565.35564	1549.1920	3010.92600
##	NFYA	128.75396	215.173089	0.000000	526.52422	783.7322	664.80964
##	STPG1	105.25642	27.359621	3.458716	253.40639	370.8472	179.81425

เมื่อลองนำข้อมูลมาสร้าง boxplot อย่างง่าย จะพบว่าหลาย gene ที่มีความแตกต่างกัน

ซึ่งต่อไปเราจะนำมาเข้าสู่กระบวนการหา differential gene expression เพื่อดูว่ามี gene

ใดบ้างที่มีความแตกต่างกันระหว่างสองกลุ่มอย่างมีนัยสำคัญ

```
count_matrix %>%
  edgeR::cpm(log=TRUE) %>%
  head(20) %>%
  t %>%
  as.data.frame() %>%
  rownames_to_column("type") %>%
  tidyr::pivot_longer(-type) %>%
  mutate(type = gsub("\\d", "", type)) %>%
  ggplot(aes(x = name, y = value, fill = type)) + geom_boxplot() +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust=1))
```



## Differential gene expression analysis

ก่อนที่เราจะทำการ visualize ข้อมูลนั้น เราจะต้องทำการวิเคราะห์ก่อนว่า RNA ไหนที่มีการแสดงออกระหว่างสองกลุ่มที่แตกต่างอย่างมีนัยสำคัญ

โดยเราจะเริ่มจากการสร้าง design matrix ซึ่งบ่งบอกว่าใครอยู่กลุ่มไหน

```
library(edgeR)

colnames(count_matrix)

## [1] "N1" "N2" "N3" "T1" "T2" "T3"

group <- design$condition
diff_design <- model.matrix(~0+group)
diff_design

##   groupN groupT
## 1      1      0
## 2      1      0
```

```
## 3      1      0
## 4      0      1
## 5      0      1
## 6      0      1
## attr(,"assign")
## [1] 1 1
## attr(,"contrasts")
## attr(,"contrasts")$group
## [1] "contr.treatment"
```

สิ่งที่เราเห็นคือ design matrix ของกลุ่มที่เราต้องการ โดยหมายเลข 1 คือตัวบ่งบอกว่า sample เราอยู่ในกลุ่มนั้นๆ โดยในที่นี้ sample 1-3 จะอยู่ในกลุ่ม Normal ส่วน sample 4-6 จะอยู่ในกลุ่ม Tumor

## Normalization

หลังจากนั้น เราจะต้องทำการ normalize ค่าการแสดงออกของ RNA เนื่องจากการ run RNA seq ในแต่ละ sample นั้น สภาวะของเครื่องอาจมีความแตกต่างกันบ้างเล็กน้อย ส่งผลให้ค่า signal intensity พื้นหลังนั้นไม่เท่ากัน

```
dge <- DGEList(counts=count_matrix, group=group)

keep <- filterByExpr(dge, group=group,min.count=2, min.prob=0.5)

dge <- dge[keep,]

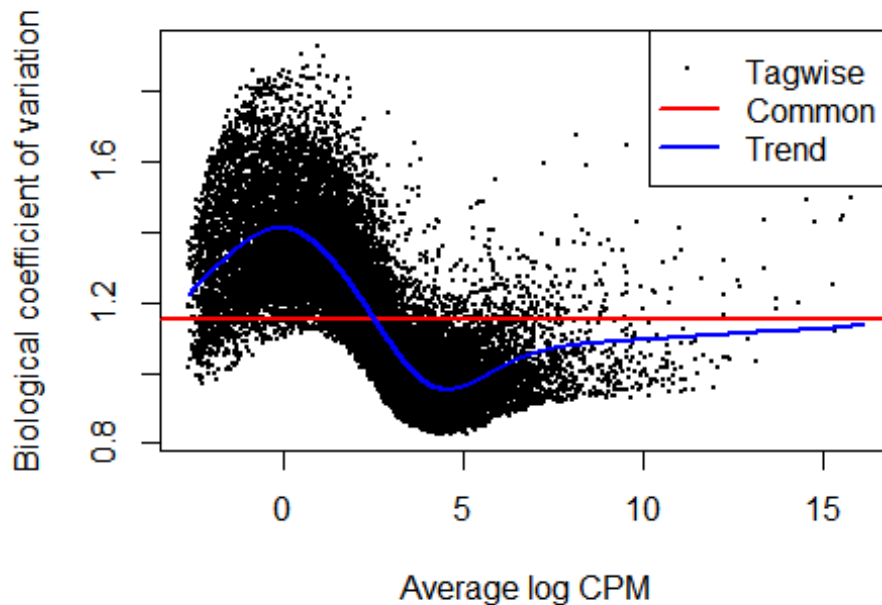
# Calculate normalization factor
genexp <- calcNormFactors(dge)

# GLM Common dispersion
genexp <- estimateGLMCommonDisp(dge, diff_design)

# Estimate GLM trended dispersions
genexp <- estimateGLMTrendedDisp(genexp, diff_design)

# Tagwise dispersion of each gene
genexp <- estimateGLMTagwiseDisp(genexp, diff_design)

plotBCV(genexp)
```



```
head(genexp$counts)
```

##		N1	N2	N3	T1	T2	T3
##	TSPAN6	1616.16632	5.323296	0.000000	2892.36698	5050.6964	1935.65287
##	DPM1	534.33260	373.409670	2.438447	1344.56193	3887.3732	3857.84694
##	SCYL3	134.55807	141.880574	2.312303	306.11165	320.5272	269.31308
##	C1orf112	68.44894	47.264863	17.260795	117.66245	207.5864	72.52807
##	FGR	40.13006	3177.326075	62.927494	10.98451	192.8986	109.53769
##	CFH	801.43355	8.169078	5.188639	1304.66135	13678.3377	2457.62369

หลังจาก normalize แล้ว เราจะทำการวิเคราะห์ differential gene expression

โดยการวิเคราะห์ทางสถิติที่เรียกว่า negative binomial generalized log-linear model

ซึ่งโดยสรุปคร่าวๆ คือการเปรียบเทียบ average log RNA expression ระหว่างสองกลุ่ม แต่ซับซ้อนกว่าเพื่อลดผลบวกหลง

**Note:** package ที่นิยมใช้ในปัจจุบัน ได้แก่ limma, edgeR, และ DEseq

โดยจะมีความแตกต่างกันเล็กน้อยในส่วนของการวิเคราะห์ทางสถิติ สำหรับผู้ที่สนใจสามารถศึกษาเพิ่มเติมได้ที่

<https://www.biostars.org/p/284775/>

หลังจากนั้นเราจะใช้ function topTags() เพื่อทำการดึงตารางผลของ differential RNA expression ออกมา

```
fit <- glmQLFit(genexp, diff_design)
```

```

genediff <- glmQLFTest(fit, contrast=c(-1,1))

# ALL genes
all_gene <- topTags(genediff, n = Inf, p.value = 1, adjust.method = "fdr")

all_gene$table %>%
  rownames_to_column("gene_name") %>%
  readr::write_csv("all_gene.csv")
# Only significant value
sig_gene <- topTags(genediff, n = Inf, p.value = 0.05,
                    adjust.method = "fdr", sort.by = "logFC")

# Total differentiated gene
summary(decideTests(genediff))

##           -1*groupN 1*groupT
## Down                1792
## NotSig              17504
## Up                   939

# Summary table
(sig_gene$table)

##           logFC      logCPM      F      PValue
FDR
## HBD          -18.273967  9.562807862 46.650297 8.545057e-12 1.729092e
-07
## ENSG10010139367.1 -17.649666  8.938617764 45.143875 1.842839e-11 1.864493e
-07
## AQP9          -15.704149  7.349606642 40.412758 2.066846e-10 1.099827e
-06
## CXCR1         -14.963082  6.252996881 39.713938 2.955267e-10 1.099827e
-06
## MEFV          -14.752717  6.042819257 39.573130 3.176099e-10 1.099827e
-06
## FPR2          -14.641776  5.931976369 39.521492 3.261161e-10 1.099827e
-06
## ADGRE3        -13.942470  5.233551274 39.032417 4.188918e-10 1.210896e
-06
## ALAS2         -13.895256  8.159710331 36.205749 1.783055e-09 2.405342e
-06
## ADGRG3        -13.615974  4.907601442 38.336422 5.982575e-10 1.451147e
-06
## PROK2         -13.564091  5.955987834 36.981569 1.197829e-09 1.897911e
-06
## MT-ATP8       13.558617 13.119447847 32.113716 1.458995e-08 7.380690e
-06
## GLT1D1        -13.536189  4.828029253 38.087774 6.795202e-10 1.451147e
-06
## FCAR          -13.505521  4.797405480 37.982573 7.171468e-10 1.451147e

```



```

-06
## FCGR3B          -13.473789  8.385197697 35.019121 3.277923e-09 3.158513e
-06
...

```

โดยจากตาราง จะพบว่ามีค่าต่าง ๆ โดยที่เราสนใจมักจะเป็น

- logFC ซึ่งก็คือ fold change ของ RNA expression ระหว่างกลุ่ม Normal vs Tumor
- pvalue โดยเรามักจะต้องปรับผลเพื่อลดภาวะผลบวกหลงออกไปด้วย เราจึงใช้ column FDR ไม่ใช่ PValue

## Data Visualization

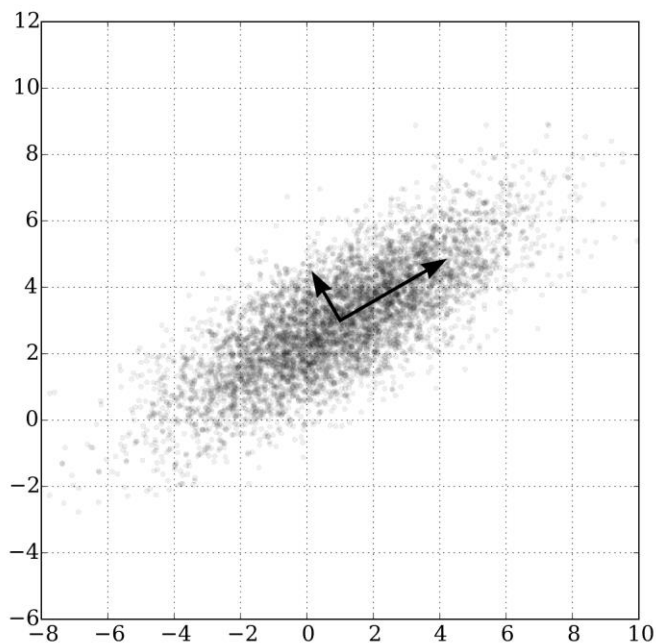
### Principal Component Analysis (PCA)

PCA คือการลดมิติของปริมาณข้อมูลลงเพื่อทำให้เกิดความง่ายขึ้นในการวิเคราะห์ โดยใช้หลักการรวมข้อมูลแบบ linear combination ที่มีความแปรปรวนใกล้เคียง ซึ่งโดยทั่วไปแล้ว

จะนำมาใช้ในการดูความแตกต่างกันของลักษณะข้อมูลในแต่ละกลุ่มแบบคร่าวๆ

หรือใช้ในการค้นหาความผิดปกติของข้อมูลที่เกิดจากสถานะที่ต่างกัน (batch effect)

โดยที่ข้อมูลที่มีลักษณะใกล้เคียงกันจะอยู่ในตำแหน่งที่ใกล้เคียงกัน



รูปจาก: [https://en.wikipedia.org/wiki/Principal\\_component\\_analysis](https://en.wikipedia.org/wiki/Principal_component_analysis)

#### Requirement

- ข้อมูลควรมีการถูก normalized โดยอาจจะ centered (ทำให้ scale เริ่มต้นที่ 0) หรือไม่ก็ได้
- ต้องไม่มี missing value

```
library(PCAtools)
```

```
## Loading required package: ggrepel
```

```
##
## Attaching package: 'PCAtools'

## The following objects are masked from 'package:stats':
##
##      biplot, screeplot

# Calculate log-counts-per-million
logcpm <- cpm(dge, prior.count = 2, log = TRUE)

# Create a metadata table
metadata <- data.frame(row.names = colnames(logcpm),
                       group = c(rep(1,3), rep(2,3)))

(metadata)

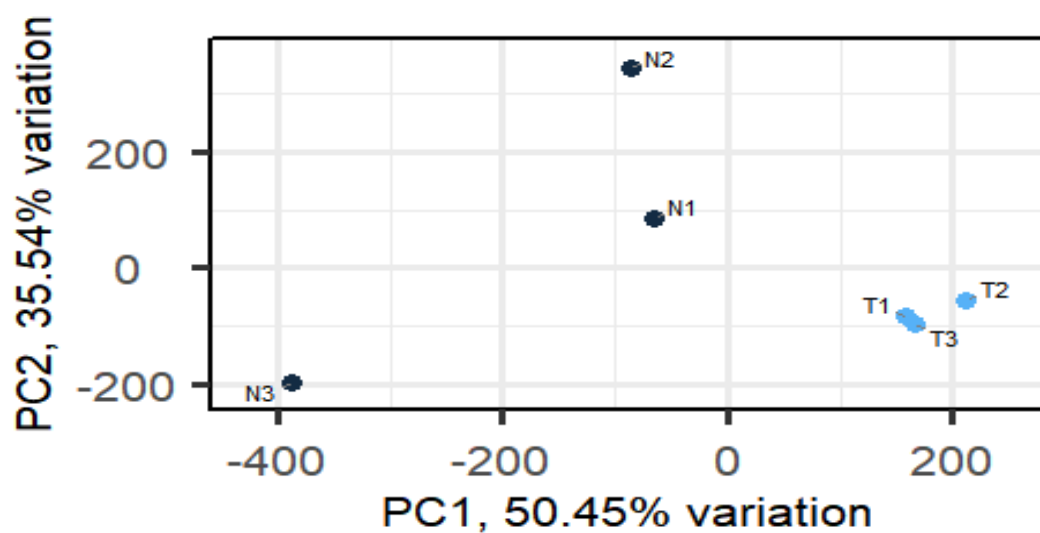
##      group
## N1      1
## N2      1
## N3      1
## T1      2
## T2      2
## T3      2
```

โดยการแปลผล PCA นั้น ควรดูไล่ไปที่ละแกน (มิติ 1 -> มิติ 2 ไม่ใช่ดู 2 มิติพร้อมกัน)

```
# Perform PCA analysis
pc <- pca(logcpm, metadata = metadata, removeVar = 0.1)

## -- removing the lower 10% of variables based on variance

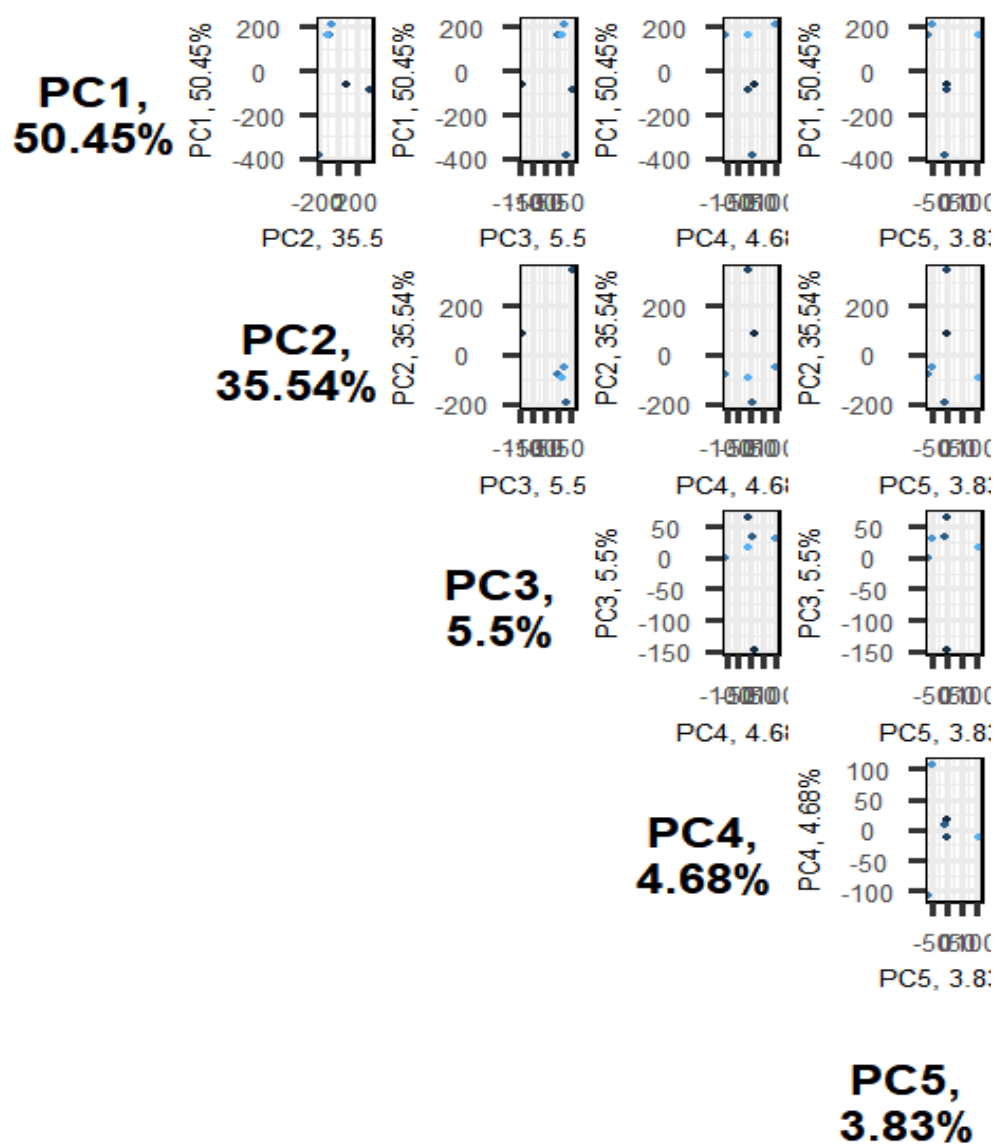
# Create PCA plot
biplot(pc, colby = "group")
```



จะเห็นได้ว่า ในส่วนของ T1, T2 และ T3 นั้นค่อนข้างเกาะกลุ่มกัน แต่ N นั้น มีความแตกต่างกันพอสมควรในทั้งสองมิติ

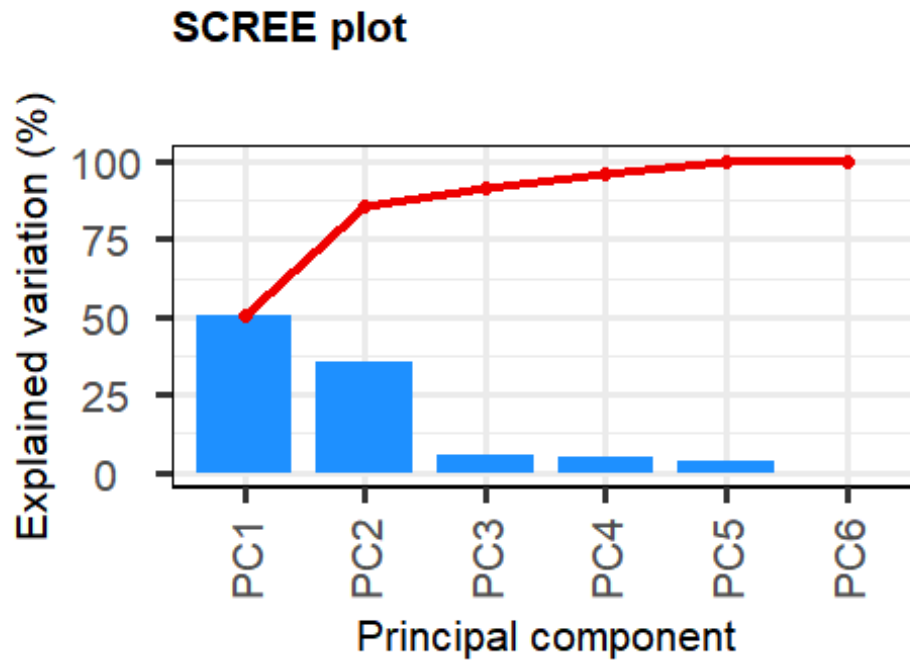
แม้ว่าในกราฟจะมีแค่ 2 มิติ แต่โดยที่จริงแล้วมิตินั้นจะโดนลดลงเหลือ  $n$  มิติ

```
pairsplot(pc)
```



ซึ่งเราสามารถดูความมากน้อยของผลกระทบของในแต่ละมิติได้โดยใช้ Scree plot โดยมิติแรกจะมีผลมากกว่ามิติหลังเสมอ

```
screepplot(pc)
```



ในส่วนของคุณข้อมูลเชิงลึกของ PCA สามารถศึกษาเพิ่มเติมได้ในเอกสารแนบ:

[http://www.cs.otago.ac.nz/cosc453/student\\_tutorials/principal\\_components.pdf](http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf)

ตัวอย่างการใช้งานเพิ่มเติม:

<https://www.bioconductor.org/packages/release/bioc/vignettes/PCAtools/inst/doc/PCAtools.html>

## Heatmap

Heatmap คือการเปลี่ยนข้อมูลที่มีให้อยู่ในรูปของสี ซึ่งจะแสดงความแตกต่างตามค่าที่มากหรือน้อย โดยการสร้าง heatmap นั้นจะใช้ข้อมูลดิบ (ก่อนทำ differential expression)

ซึ่งจะทำให้เห็นภาพรวมของข้อมูลแต่จะไม่ให้ข้อมูลความแตกต่างทางด้านสถิติมากนัก

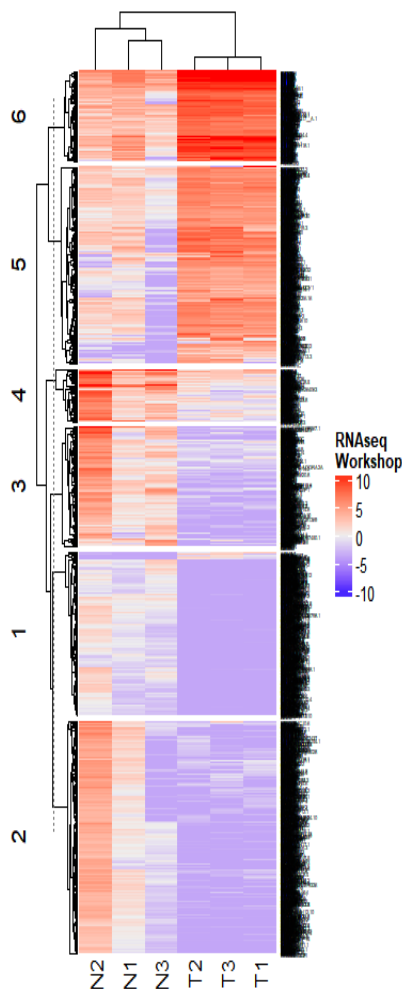
ซึ่งโดยปกติถ้านำข้อมูลทั้งหมดมาสร้าง heatmap จะทำให้รูปมีขนาดใหญ่เกินไป ดังนั้น

เรามักจะกรองข้อมูลที่เราต้องการจะนำเสนอก่อนที่จะนำมาสร้าง

```
library(ComplexHeatmap)

# Filter only significant DEGs gene (from EdgeR)
DEGGene <- logcpm[rownames(sig_gene$table),]

Heatmap(DEGGene, row_km = 6, name = "RNAseq\nWorkshop",
        row_names_gp = gpar(fontsize = 3))
```



ตัวอย่างการใช้งานเพิ่มเติม: <https://jokergoo.github.io/ComplexHeatmap-reference/book/>

## Volcano plot

Volcano plot คือกราฟที่แสดงความแตกต่างของการแสดงออกของ RNA ระหว่างสองกลุ่ม โดยมีแกน x คือ log fold change และ y คือ  $-\log_{10}(\text{p-value})$  เหตุผลที่แกน y ต้องเป็น  $-\log_{10}(\text{p-value})$  เพื่อที่จะปรับค่า p-value ที่เป็นทศนิยมนั้นให้อยู่ในหลักจำนวนเต็ม ซึ่งจะทำให้ได้กราฟที่มีรูปร่างคล้ายภูเขาไฟหัวกลับ

```
library(EnhancedVolcano)
```

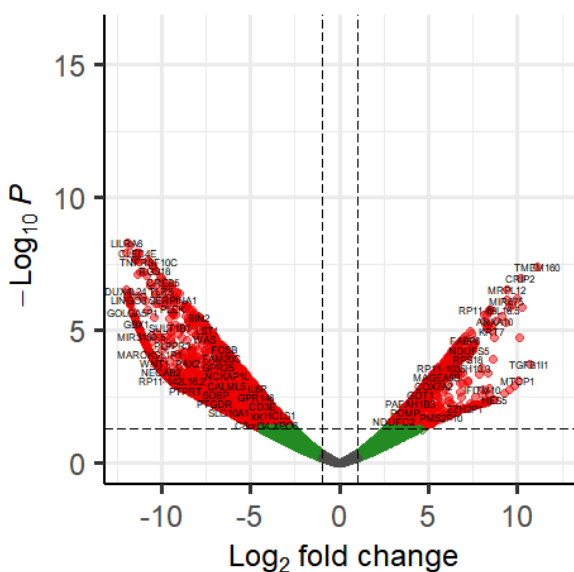
```
# Create Volcano plot
```

```
EnhancedVolcano(all_gene$table, lab = rownames(all_gene$table), x = "logFC",  
y = "PValue",  
xlim = c(-12, 12), labSize = 2.0, pCutoff = 0.05,  
title = "RNAseq workshop", max.overlaps = 50)
```

### RNAseq workshop

EnhancedVolcano

● NS ● Log<sub>2</sub> FC ● p-value and log<sub>2</sub> FC



total = 20235 variables

ค่าที่ cut-off ที่เราสนใจนั้นมักจะเป็นที่  $\log_{2}FC > 1-2$ , และ  $-\log_{10}(\text{p-value}) > 1.3-2$  ( $\text{p-value} < 0.01-0.05$ )



ตัวอย่างการใช้งานเพิ่มเติม:

<https://bioconductor.org/packages/release/bioc/vignettes/EnhancedVolcano/inst/doc/EnhancedVolcano.html>