Outline

0. Contact info

Yen-Hsieh Chen

yhchen@ibms.sinica.edu.tw

1. Course intro and prerequisite

1.1. course brief intro

This course is designed for R beginner, biologist, and both.

Through this course, participants will understand R basics, visualization & graph formatting, public (cancer) data retrieving, parallel computing basics, and biological analysis (i.e. differential gene expression analysis followed by gene set enrichment analysis). The course will starts from and heavily rely on *tidyverse* R package, which is a beginner-friendly stepping stone entering R world.

[*Highly Recommended*] All course contents are applied in *ipynb* format and participants may download or try to edit the code online for different results.

1.2. codes in the cloud

Course contents are now ready for either cloud presentatin at Github or demonstration via Binder.

For participants prefering to try the demonstration instead of installing the whole R environment locally, please refer to this Binder link and have the Binder preparing the online Jupyter environment (it may take whiles to build up).

1.3. local environment prerequisite

Anaconda(optional but recommend), R or RStudio

> To have R programming environment, direct installatio†n of R/RStudio or a virtural environment like Anaconda containing R/RStudio are both feasible. Installation steps of these items are platform-dependent (Linux/Mac/Windows).

1.4. local R package requirements

uncomment to install if you don't have packages installed

In [ ]:

```
#install.packages('palmerpenguins')
library(palmerpenguins)

#install.packages('tidyverse')
library(tidyverse)
#install.packages('ggpubr')
library(ggpubr)

#install.packages('UCSCXenaTools')
library(UCSCXenaTools)

#if (!require('BiocManager', quietly = TRUE))
#    install.packages('BiocManager')
#BiocManager::install("edgeR")
library(edgeR)
#BiocManager::install("clusterProfiler")
library(clusterProfiler)
#BiocManager::install("org.Hs.eg.db")
library(org.Hs.eg.db)
#BiocManager::install("enrichplot")
library(enrichplot)
```

2. Useful informations

2.1. R/RStudio from Anaconda

- Anaconda installation on Linux/Mac/Windows
- Using R with Anaconda
- Using RStudio with Anaconda
- Using R in Jupyter
- Faster solver of Anaconda: Mamba

2.2. online R resources

2.2.1. basic/interactive R courses

- DataCamp

2.2.2. biology-relevant R packages

- *Bioconductor* project
- *Neuroconductor project*
- *Statistical tools for high-throughput data analysis (STHDA)*

2.2.3. eBooks

- Cookbook for R
- R for Data Science
- R for Statistical Learning
- R for Graduate Students
- Computational Genomics with R
- Fundamentals of Data Visualization

2.2.4. visualization guidance

- R Graph Gallery
- interactive htmlwidgets for R
- from Data to Viz
- Tufte in R
- ggplot2 extensions
- paletteer gallery

3. R foundamentals

3.1. basic data type

Numbers (integer/float), *characters (string)*, *boolean (TRUE and FALSE)* are the primitive elements, namely **vectors**, in R environment and also common in other programming languages with interchangeable namings. Under most cases, any object/variable would be *character* either quoted by single (`'X'`) and double quatation (`"X"`) marks. Furthermore, to check a R object is certain class, *`is.X()`* functions are default in R programming returning `TRUE/FALSE` logical statement, e.g. *`is.character()`* , *`is.numeric()`* , etc. Examples are listed below and we also can check the class of an object via *`class()`* function.

In [1]:
```
class('X')
```

'character'

In [2]:
```
class(100)
```

'numeric'

In [3]:
```
is.numeric(123)
```

TRUE

In [4]:
```
is.logical(FALSE)
```

TRUE

> [Tips] To understand argument/parameters and detailed information for a certain function, e.g. `t.test()`, you may use `args(t.test)`, `help(t.test)` or `??t.test` for advanced information.

3.2. basic data structures

As mentioned basic *vecto`*, other traditional structures in R includes:

- *list*: vector-like 1D structure and component could be different data types
- *matrix*: 2D vector and all elements should be identical data type
- *data.frame*: table-like object, similar with *matrix* while columns could have different data types

3.2.1. *list*

In [5]:
```
list(
    A = 'X',
    B = c(1,3,5),
    C = matrix(1:3, nrow =1, ncol = 3)
) -> demo_list

print(demo_list)
```

```
$A
[1] "X"

$B
[1] 1 3 5

$C
     [,1] [,2] [,3]
[1,]    1    2    3
```

In [6]:
```
length(demo_list)
```

3

In [7]:
```
names(demo_list)
```

'A' · 'B' · 'C'

In [8]:
```
demo_list$B
```

1 · 3 · 5

In [9]:
```
demo_list[[2]]
```

1 · 3 · 5

Using function *list()* , you may concatenate multiple items with different data types as object `demo_list` . Also, by functions *length()* , object `demo_list` is confirmed to be a vector with 3 elements within. And to retrieve specific element from *list*, either by element naming (`demo_list$B`) or index (`demo_list[[2]]`) method are feasible.

3.2.2. *matrix*

In [10]:
```
demo_matrix_chr = matrix(letters[1:15], byrow = TRUE, nrow = 3)
print(demo_matrix_chr)
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,] "a"  "b"  "c"  "d"  "e"
[2,] "f"  "g"  "h"  "i"  "j"
[3,] "k"  "l"  "m"  "n"  "o"
```

In [11]:
```
demo_matrix_num = matrix(1:15, byrow = FALSE, nrow = 3)
```

In [12]:
```
rownames(demo_matrix_num) <- c('X', 'Y', 'Z')
colnames(demo_matrix_num) <- letters[1:5]
```

In [13]:
```
print(demo_matrix_num)
```

```
  a b c  d  e
X 1 4 7 10 13
Y 2 5 8 11 14
Z 3 6 9 12 15
```

In [14]:
```
colnames(demo_matrix_num)[c(3,5)] = c('new_C', 'new_E')
print(demo_matrix_num)
```

```
  a b new_C  d new_E
X 1 4     7 10    13
Y 2 5     8 11    14
Z 3 6     9 12    15
```

In [15]:
```
print(dim(demo_matrix_num))
```

```
[1] 3 5
```

In [16]:
```
print(demo_matrix_num[3,2])
```

```
[1] 6
```

For class *matrix* and *data.frame* below, since these two are table-like classes, you may add/change the column/row name via *`colnames`* and *`rownames()`* functions, even more, a specific column name could be altered by combining index.

To have the dimension of either *matrix* or *data.frame*, *`dim()`* function is often used and the results are twon numbers while the first one is **row count** and the other is **column count**. Also, to access or change the specific element within *matrix* or *data.frame*, even *list*, the coordinate system is applied while the representation would be like `table[row_index, column_index]` in 2D objects (*matrix* and *data.frame*), `vec[index]` in 1D array/vector and `List[[index]]` in 1D *list* object. Such coordinate system is commonly used in various programming languages including R.

3.2.3. *data.frame*

In [17]:

```
demo_df <- data.frame(
    col_num = 1:5,
    col_str = LETTERS[1:5],
    row.names = paste('row', letters[1:5], sep = '_')
)
print(demo_df)
```

```
      col_num col_str
row_a       1       A
row_b       2       B
row_c       3       C
row_d       4       D
row_e       5       E
```

In [18]:

```
demo_df$add_col = demo_df$col_num*1.3-5
demo_df[,4] = sqrt(demo_df$col_num)
```

In [19]:

```
print(demo_df)
```

```
      col_num col_str add_col       V4
row_a       1       A    -3.7 1.000000
row_b       2       B    -2.4 1.414214
row_c       3       C    -1.1 1.732051
row_d       4       D     0.2 2.000000
row_e       5       E     1.5 2.236068
```

In [20]:

```
str(demo_df)
```

```
'data.frame':   5 obs. of  4 variables:
 $ col_num: int  1 2 3 4 5
 $ col_str: chr  "A" "B" "C" "D" ...
 $ add_col: num  -3.7 -2.4 -1.1 0.2 1.5
 $ V4     : num  1 1.41 1.73 2 2.24
```

In [21]:

```
print(demo_df[,-1])
```

```
      col_str add_col       V4
row_a       A    -3.7 1.000000
row_b       B    -2.4 1.414214
row_c       C    -1.1 1.732051
row_d       D     0.2 2.000000
row_e       E     1.5 2.236068
```

In [22]:

```
print(demo_df[,'col_num'])
```

```
[1] 1 2 3 4 5
```

Structure of *data.frame* is similar with *matrix* presenting table-like form. The manupulation, coordinate system, naming, all are identical with *matrix* except not all elements within are restrcited to single data type, which is effective and widely used for downstream analysis. *Data.frame* is common however its drawback are (1) poor readability for high dimention data (either large column or row count) and (2) high occupacy for large data comparing to *matrix* with same dimension as following comparison.

The first disadvantage is solved by class *tibble* from *tidyverse* package, which will be discussed in later section, and the second one is hard to solve since it relates to the nature of R environment and data itself.

In [23]:

```
object_matrix = matrix(1:10000, nrow = 20)
object_dataframe = as.data.frame(object_matrix)

object.size(object_matrix)
object.size(object_dataframe)
```

```
40216 bytes
124608 bytes
```

3.3. variable assignment

Variable is a container storing data values and by assigning the value to a variable, such variable is created and stored in current R enviroment. Not till 2001, the assignment operator is stricted to arrow symbol (<–) but nowadays equation symbol (=) is accepted as well. These two assignment operators has their own supporters and you may find the advantages of <– from this article.

In [24]:

```
A = 100
print(A)
```

```
[1] 100
```

In [25]:
```
B <- 100
print(B)
```

```
[1] 100
```

In [26]:
```
100 -> C
print(C)
```

```
[1] 100
```

A/B/C here are the named variables we created with assigned value/data and both arrows (–> and <–) clearly represent the assignment directions while = has the its own direction, i.e. variable *A* **is** 100.

> Please keep in mind for careful variable naming since it would increase/decrease your code readability plus the naming started from numbers/special symbols are generally avoided unless quoted by back quotes. Also, complicated or meaningless namings, e.g. *ThisCouse_startsfrom_2pmto4pm_byYHC* or *tfadef.gaerre*, are avoided.

In [27]:
```
1hundred = 100
```

```
Error in parse(text = x, srcfile = src): <text>:1:2: unexpected symbol
1: 1hundred
     ^
Traceback:
```

In [28]:
```
$100 = 100
```

```
Error in parse(text = x, srcfile = src): <text>:1:1: 未預期的 '$'
1: $
    ^
Traceback:
```

In [29]:
```
`$100` = 100
print(`$100`)
```

```
[1] 100
```

3.4. basic arithmetic and logical operators

Following examples will cover basic arithmetic (addition, subtraction, multiplication and division) and logical operations.

In [30]:
```
A <- 10
B <- 100
C <- 100
```

In [31]:
```
# addition
print(A+B)
```

```
[1] 110
```

In [32]:
```
# subtraction
print(A-B)
```

```
[1] -90
```

In [33]:
```
# multiplication
print(A*B)
```

```
[1] 1000
```

In [34]:
```
# division
print(A/B)
```

```
[1] 0.1
```

In [35]:
```
# equal to
print(A == B)
```

```
[1] FALSE
```

In [36]:
```
# not equal to
print(A != B)
```

```
[1] TRUE
```

In [37]:
```
# (equal to or) larger than
print(A > B)
print(A >= B)
```

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

In [38]:
```
# (equal to or) smaller than
print(A < B)
print(A <= B)
```

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

In [39]:
```
# and
print(A == C & B == C)
```

```
[1] FALSE
```

In [40]:
```
# or
print(A ==C | B == C)
```

```
[1] TRUE
```

In [41]:
```
# is a member or not
print(C %in% c(A,B))
```

```
[1] TRUE
```

3.5. condition and iteration

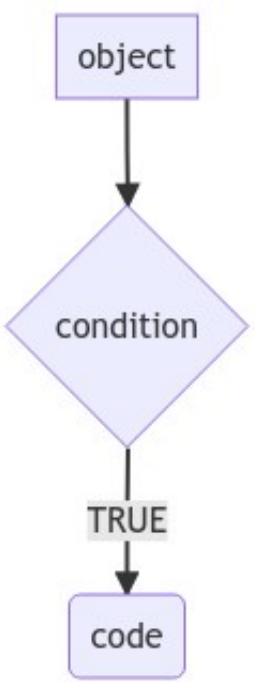
Condition and iteration are two ways to control your code flow when executing complicated programs and scenarios would be often like: (a) execute a code depending on the condition, e.g. a specific gene is expression or not, and (b) execute the code repeatly many times, e.g. make the addition process 3 times in a row.

Basic conditions are included in previous logical operators hence in this part we will focus on another condition flow control (*f-else* statement) and common iterations including *for/while-loop* and *apply* family.

3.5.1. *if-else*

The main purpose of *if-else* statement is to permit a code's running or not based on the certain condition, whereas the condition could be singular or multiple and the flow structure could be linear or nested. The basic concepts of *if-else* statement are listed as followed.

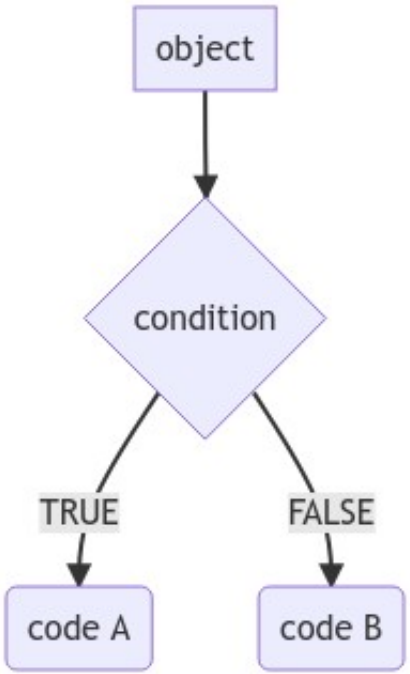
simple *if*



In [ ]:

```
if(the condition is TRUE){
    run code
}
```

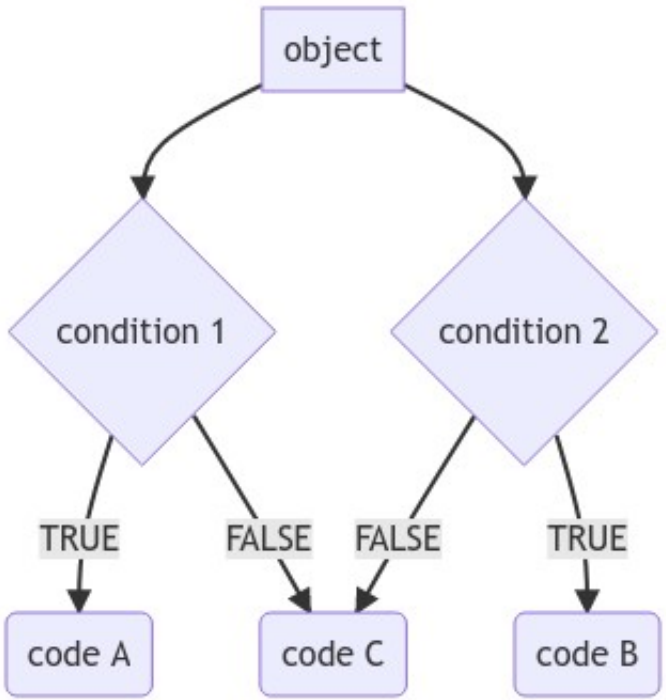
if-else with 1 condition



In [ ]:

```
if(the condition is TRUE){
    run code A
} else {
    run code B
}
```

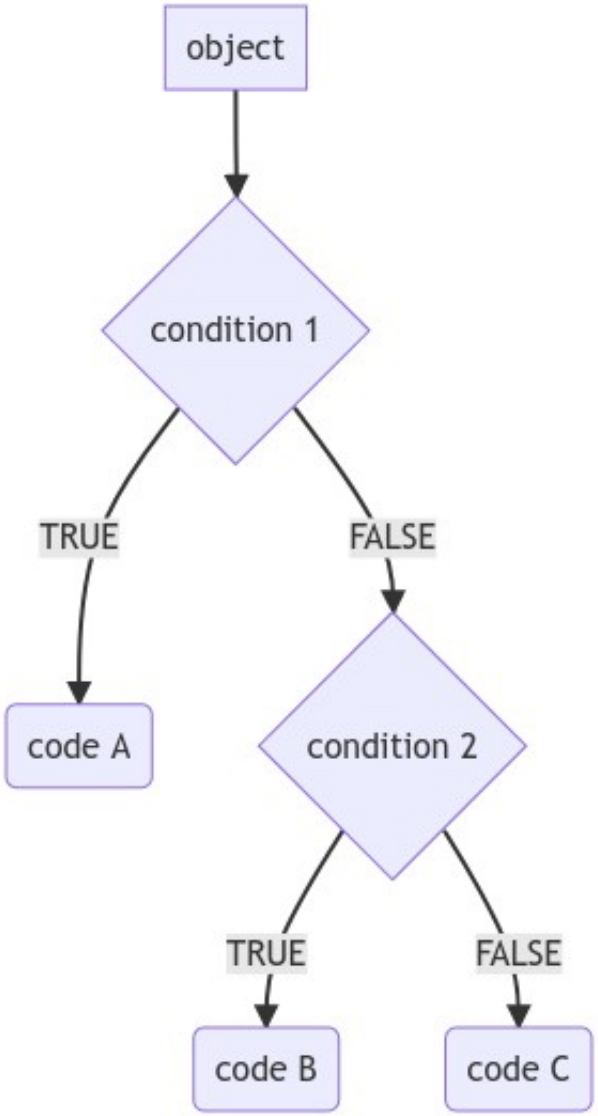
if-else with 2 condition, linear



In [ ]:

```
if(the condition_1 is TRUE){
    run code A
} else if(the condition_2 is TRUE){
    run code B
} else {
    run code C
}
```

if-else with 2 condition, nested

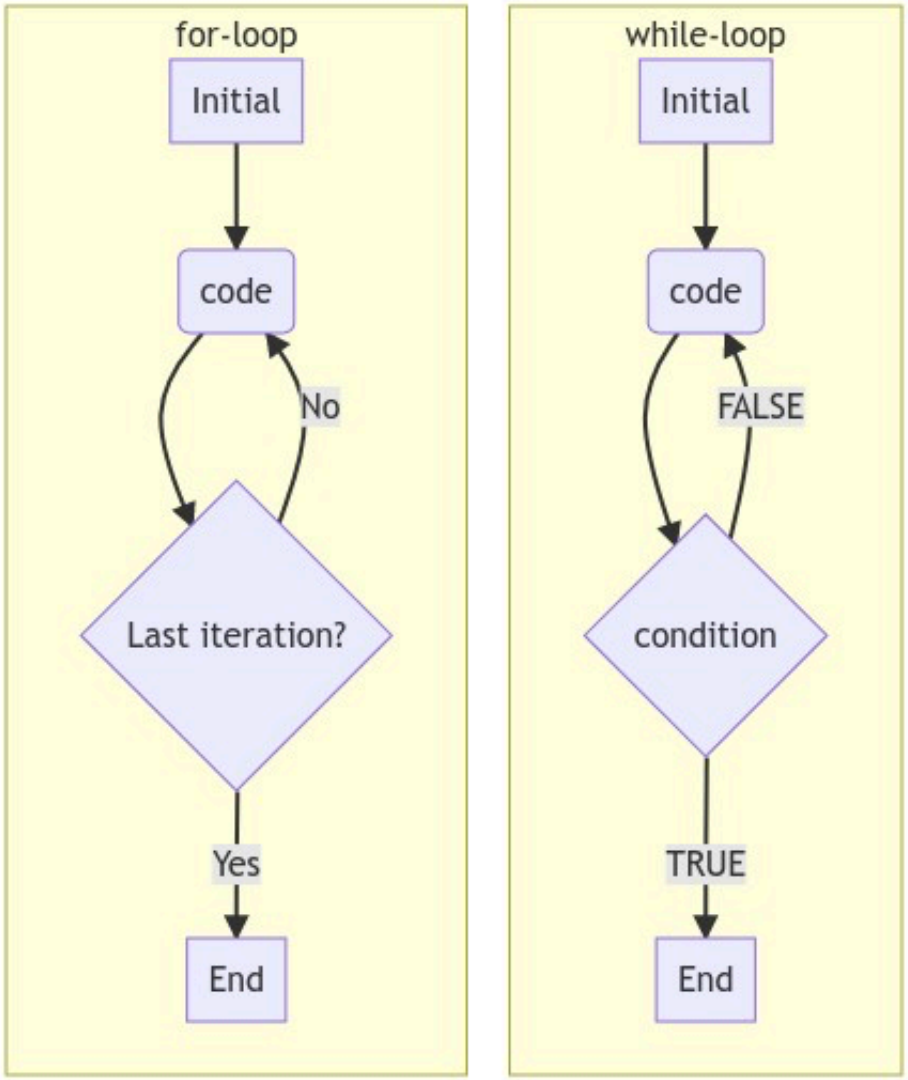


In [ ]:

```
if(the condition_1 is TRUE){
    run code A
} else {
    if(the condition_2 is TRUE){
        run code B
    } else {
        run code C
    }
}
```

3.5.2. *for/while* loop

The perupose and concept of *for* and *while* loop are similar that they both execute the code repeatly but the former one are limited within a range of iterations while the latter one will stop under certain condition. As followed diagram and code examples you may understand the difference.



for-loop

In [ ]:

```
iteration_range = 1:N
for(iteration in iteration_range){
    run code
}
```

while-loop

In [ ]:
```
index = 1
while(index <= 50){
    run code
}
```

3.5.3. *apply* family

Like *for/while-loop*, *apply* family functions from basic R are designed to perform functions repetitively on a *list* of components (e.g. *vector*, *data.frame*, *matrix*, etc.) and are composed of members including `apply()` , `lapply()` , `sapply()` , `vapply()` , `mapply()` , `rapply()` , and `tapply()` functions. The usage of which *apply* member depends on your input and output and here we will demonstrate `lapply()` function, which idea and syntax structure is similar to *for-loop*.

In [ ]:
```
iteration_range = 1:N
lapply(iteration_range,
    run code
)
```

As demosntrated here, the syntax structure is nearly identical with *for-loop* that the code chunck runs repeatly within the iteration. However, the overall performance of `lapply()` is better than *for-loop* and when it comes to preserving the output, the syntax would be different.

> Given a range of numbers, how to take the square root of them?

In [42]:
```
# for-loop
numbers = 1:15
res_forloop = c()
for(n in numbers){
    res_forloop[n] = sqrt(n)
}
```

In [43]:
```
str(res_forloop)
```

```
 num [1:15] 1 1.41 1.73 2 2.24 ...
```

In [44]:
```
# lapply
numbers = 1:15
res_lapply = lapply(numbers, sqrt)
# res_lapply = lapply(numbers, {sqrt})
```

In [45]:
```
str(res_lapply)
```

```
List of 15
 $ : num 1
 $ : num 1.41
 $ : num 1.73
 $ : num 2
 $ : num 2.24
 $ : num 2.45
 $ : num 2.65
 $ : num 2.83
 $ : num 3
 $ : num 3.16
 $ : num 3.32
 $ : num 3.46
 $ : num 3.61
 $ : num 3.74
 $ : num 3.87
```

The output as you can see are identical except the result format is either in *vector*(`res_forloop`) or *list* (`res_lapply`) format respectively. To achieve the *vector* format, one could use another family named *`sapply()`* and for details about the *apply* family, please find this link for more information.

3.6. vectorization

Vectorization is the parallel process of a function on multiple values at a time and no need to loop through each element, making an more efficient and concise code to execute and read comparing to non-vectorized format as demonstrated with following examples.

> Given two numeric vectors, how to make the pair-wise addition of the elements within?

3.6.1. non-vectorized

In [46]:
```
vector_A = matrix(1:15, nrow = 5)
vector_B = matrix(11:25, nrow = 5)

result = matrix(nrow = 5, ncol = 3) # empty 5x3 matrix
for(i in 1:nrow(vector_A)){
    for(j in 1:ncol(vector_A)){
        result[i,j] <- vector_A[i,j]+vector_B[i,j]
    }
}

print(result)
```

```
     [,1] [,2] [,3]
[1,]   12   22   32
[2,]   14   24   34
[3,]   16   26   36
[4,]   18   28   38
[5,]   20   30   40
```

3.6.2. vectorized

In [47]:
```
vector_A = matrix(1:15, nrow = 5)
vector_B = matrix(11:25, nrow = 5)

print(vector_A+vector_B)
```

```
     [,1] [,2] [,3]
[1,]   12   22   32
[2,]   14   24   34
[3,]   16   26   36
[4,]   18   28   38
[5,]   20   30   40
```

3.7. package install and load

Several repositories are providing R packages including *CRAN*, *BioConductor* and *Github*. Installation syntax is different between these repositories.

In [ ]:
```
# CRAN
install.packages('specific_pkg')

# Bioconductor
if (!require('BiocManager', quietly = TRUE))
    install.packages('BiocManager')

BiocManager::install('specific_pkg')

# Github
if (!require('devtools', quietly = TRUE))
    install.packages('devtools')

devtools::install_github('DeveloperName/PackageName')
```

To load an installed package, either two ways below are avaliable.

In [ ]:
```
library(specific_pkg)
require(specific_pkg)
```

To check the installed packages or the version of the installed package, one may use the following lines to have the result.

In [ ]:
```
installed.packges()
packageVersion('specific_pkg')
```

3.8. function

Even though R and other repositories are providing built-in and convinient packages to keep you fron reinventing the wheel, one may encounter scenarios that a user-defined function is prefered. The basic/simple syntax of function is listed.

In [48]:

```
demo_func <- function(){
    print('hello world')
}

demo_func()
```

```
[1] "hello world"
```

A practical user-defined functions would require one/multiple parameters to fit user need and (sometimes) would need printed messages during the process, return the result, or even save the intermediate results as example followed-up.

> Given a range of numbers, how to take the *integre of the specific position* from its square root result and save into a vector?

In [49]:

```
sqrt_dec = function(x, dec_pos = 1){
    if(x>=0){
        res_sqrt = sqrt(x)
        if(res_sqrt %% 1 == 0){
            message('input is a square number')
            return(0)
        } else {
            message('.... processing')
            res_chr = as.character(res_sqrt)
            res_split = unlist(strsplit(res_chr, split ='\\.'))
            res_dec = unlist(strsplit(res_split[[2]], split =''))
            res_final = as.numeric(res_dec[dec_pos])
            res_final
            return(res_final)
        }
    } else {
        message('incorrect input: positive number required')
        return(NA)
    }
}
```

In [50]:
```r
# check arguments
args(sqrt_dec)
```

```
function (x, dec_pos = 1)
NULL
```

In [51]:
```r
numbers = c(10, 256, 101.11, -100, -1.3)
res = sapply(numbers, sqrt_dec, dec_pos = 5)
print(res)
```

```
.... processing

input is a square number

.... processing

incorrect input: positive number required

incorrect input: positive number required

[1]  7  0  4 NA NA
```

With the example above, we have parameter `dec_pos` indicating the decimal position with default *1* in our user-defined function. Also, to determine the input is a square number (`res_sqrt %%1 == 0`) or incorrect input (NOT `x>=0`), nested *if-else* statement is applied as well. Furtheremore, we also have built-in *`unlist()`* function to convert the *list* object from the *`strsplit()`* output. Lastly, with our final result would be a *vector* object, it is named `res` by applying the *`sapply()`* function on all input numers interatively.

3.9. data input/output

Using a built-in data from R environment is an easy task but often you need to work with your own data in various format, e.g. *.txt*, *.csv*, *.xlsx*, etc. R already have built-in functions dealing with some of these file type, including *`read.csv()`* , *`read.delim()`* and *`read.table()`* , but the drawback are the limited file type and the input speed. With the convinient `tidyverse` package providing the solutions, you may use *`read*()`_* functions (listed here and here) with your file accordingly and seamlessly as built-in R functions.

> [**Tips**] For either *tidyverse* or basic R functions importing files of different formats, both local files or online files (direct file link *https://path_to_the_file.csv* for example) are allowed.

4. Tidyverse: pipe and tibble

In [ ]:
```
library(tidyverse)
```

4.1. Pipe

Pipe(%>%), developed by Stefan Milton Bache (*magrittr* R package), is an useful tool to concatenate discrete steps with a clear expression and readability. The strength of pipe, comparing to traditional work flow, will be demonstrated with follow-up hypothetical scenario.

> Given a range of numbers (e.g. 1 to 10), how to take the square root and round to two decimal places?

4.1.1. traditional *step-wise* approach

As intermediate object is created at each step, these independent objects will be applied to another step and with numerous steps, accumulating named objects will occupy hardware memory and hamper workflow readability.

In [53]:
```
numbers = 1:10
num_sqr = sqrt(numbers)
num_sqr_digit2 = format(num_sqr, digits = 2)
print(num_sqr_digit2) # string form
num_sqr_digit2_res = as.numeric(num_sqr_digit2)
print(num_sqr_digit2_res) # numeric form
```

```
[1] "1.0" "1.4" "1.7" "2.0" "2.2" "2.4" "2.6" "2.8" "3.0" "3.2"
[1] 1.0 1.4 1.7 2.0 2.2 2.4 2.6 2.8 3.0 3.2
```

4.1.2. traditional *one-sentence* approach

For the hypothetical scenario, workflow could be written into one-sentence format but the nested structure, which organize the workflow in a flashback, will increase the reading difficulty as well.

In [54]:
```
numbers = 1:10
as.numeric(
    format(
        sqrt(numbers),
        digits = 2
    )
)
# or written in this way
#as.numeric(format(sqrt(numbers), digits = 2))
```

1 · 1.4 · 1.7 · 2 · 2.2 · 2.4 · 2.6 · 2.8 · 3 · 3.2

4.1.3. tidy approach with pipe

Supported by the `%>%` , the workflow is formated as direct tone and increase the readability plus clear syntax expression. Furthermore, `%>%` is also feasible in user-defined function, putting these assembly together could make your codes efficient than traditional ways.

In [55]:
```
numbers = 1:10
numbers %>%
    sqrt() %>% # take the square root
    format(digits = 2) %>% # round to two decimal
    as.numeric() # convert result from string to numeric format
```

1 · 1.4 · 1.7 · 2 · 2.2 · 2.4 · 2.6 · 2.8 · 3 · 3.2

alternative expression

In [56]:

```
# user-defined function
sqrt_num = function(number){
    sqrt(number) %>%
        format(digits = 2) %>%
        as.numeric()
}

numbers = 1:10
sqrt_num(numbers)
```

1 · 1.4 · 1.7 · 2 · 2.2 · 2.4 · 2.6 · 2.8 · 3 · 3.2

4.2. Tibble

matrix and *data.frame* are two common R classes presenting table-like format as in MS Excel. These two classes though are similar but the contens in **all cells** within *matrix* should be identical, e.g. either all numeric or character; *data.frame* is a hybrid form of *matrix* taking columns with different classes, i.e. number+number, number+character, and character+character.

Tibble (*tbl_df* or *tbl*) is a branch of traditional *data.frame* as it accepts various classess in columns, e.g. formula, *tbl_df*, images, etc., which greatly increase the flexibility.

To demonstrate the difference between *matrix*, *data.frame* and *tbl_df*, we will use classic *iris* dataset as example.

> *iris* flower data set is the best known dataset introduced by the British statistician and biologist Ronald Fisher in 1936. The dataset consists of 50 instances from each of 3 iris classes (*Iris setosa, Iris virginica and Iris versicolor*) and 4 measurements (the length and the width of the sepals and petals). For its paradigm in statstics, this dataset is already included in R environment.

4.2.1. class *data.frame*

In [57]:

```
data(iris)
class(iris)
```

'data.frame'

In [58]:
```
str(iris)
```

```
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

In [59]:
```
head(iris)
```

A data.frame: 6 × 5

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
	<dbl>	<dbl>	<dbl>	<dbl>	<fct>
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

As illustrated by `str(iris)` (alternative `glimpse(iris)`), the default class of *iris* dataset is *data.frame* composed of 150 samples (row) and 5 features/attributes (column) while *Species* is the only non-numeric column. For the following part, we will compare the difference between *matrix*, *data.frame* and *tbl_df* via *iris* dataset.

4.2.2. class *matrix*

full dataset

In [60]:
```
matrix_iris = as.matrix(iris)
matrix_iris[1:6,]
```

A matrix: 6 × 5 of type chr

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

In [61]:
```
str(matrix_iris)
```

```
chr [1:150, 1:5] "5.1" "4.9" "4.7" "4.6" "5.0" "5.4" "4.6" "5.0" "4.4" ...
- attr(*, "dimnames")=List of 2
 ..$ : NULL
 ..$ : chr [1:5] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" ...
```

numeric part of dataset

In [62]:
```
matrix_iris_num = iris[,-5] # remove character-containing column
matrix_iris_num[1:6,]
```

A data.frame: 6 × 4

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

In [63]:
```
str(matrix_iris_num)
```

```
'data.frame':   150 obs. of  4 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
```

4.2.3. class *tbl_df*

iris conversion

In [64]:
```
as_tibble(iris) %>% head(4)
```

A tibble: 4 × 5

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
<dbl>	<dbl>	<dbl>	<dbl>	<fct>
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa

In [65]:

```
as_tibble(iris) %>% str()
```

```
tibble [150 × 5] (S3: tbl_df/tbl/data.frame)
 $ Sepal.Length: num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num [1:150] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num [1:150] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

With _ `as_tibble()` _ function, a *data.frame* object could be seamlessly converted to *tbl_df* form. To construct a new *tbl_df* object from scratch, one can apply the concept below.

In [66]:

```
S.length = iris$Sepal.Length
S.width = iris$Sepal.Width
new_tbl = tibble(
    SLength = S.length,
    SWidth = S.width,
    # suppose we want to multiply Sepal length by Sepal width
    `Length*Width` = SLength*SWidth,
    class = iris$Species)
str(new_tbl)
```

```
tibble [150 × 4] (S3: tbl_df/tbl/data.frame)
 $ SLength     : num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ SWidth      : num [1:150] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Length*Width: num [1:150] 17.8 14.7 15 14.3 18 ...
 $ class       : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

flexibility of *tbl_df*

Unlike the limitation of traditional *matrix* and *data.frame* that only *number/character/factor* are accepeted, components within *tbl_df* could be other format as the example below, which mostly are stored as list.

In [67]:
```
mixture_tbl =
    tibble(
        string = letters[1:5],
        number = 1:5,
        mixture = list(
            A = c('AS', 'IBMS'), # character vector
            B = c(5,6,8,100), # number vector
            C = function(x){x^2}, # user-defined function
            D = matrix(LETTERS[1:10], nrow = 2, ncol = 5), # simple matrix
            E = y~x1*(x2-3)) # formula
        )
```

In [68]:
```
mixture_tbl
```

A tibble: 5 × 3

string	number	mixture
<chr>	<int>	<named list>
a	1	AS , IBMS
b	2	5, 6, 8, 100
c	3	function (x) , {, x^2, }
d	4	A, B, C, D, E, F, G, H, I, J
e	5	y ~ x1 * (x2 - 3)

In [69]:
```
str(mixture_tbl$mixture)
```

```
List of 5
 $ A: chr [1:2] "AS" "IBMS"
 $ B: num [1:4] 5 6 8 100
 $ C:function (x)
  ..- attr(*, "srcref")= 'srcref' int [1:8] 8 17 8 32 17 32 8 8
  .. ..- attr(*, "srcfile")=Classes 'srcfilecopy', 'srcfile' <environment: 0x7fbcca4d8998>
 $ D: chr [1:2, 1:5] "A" "B" "C" "D" ...
 $ E:Class 'formula'  language y ~ x1 * (x2 - 3)
  .. ..- attr(*, ".Environment")=<environment: 0x7fbcca570908>
```

Though *tbl_df* follows the rules that elements within each column should be identical, e.g. column *string* and *number*, it accepts different data type mostly in *list* structure as column *mixture* as the example.

5. Data processing/visualization

Tidy data is an importnat stepping stone for R beginners for data analysis since avaliable data nowadays are mostly structured in MS Excel-like format, which table is consist of features columns (e.g. gene expression, height, histological status, etc.) and incidance rows (e.g. patients, sample collections, etc.). For more complicated relationational tables, please find here for more examples and demonstration. *Palmer penguins* data will be included here for demonstration.

> Palmer penguins data were collected and made available by Dr. Kristen Gorman and the Palmer Station, Antarctica LTER, a member of the Long Term Ecological Research Network. The palmerpenguins package contains two datasets. One is called **penguins**, and is a simplified version of the raw data **penguins_raw**. This data is an alternative for classic *iris* data since it is much closer to real world data, which is complicated, contains time series data, and having incomplete data in some observations.

5.1. data overview

In [70]:

```
library(palmerpenguins)
data(package = 'palmerpenguins')
```

Data sets

A data.frame: 2 × 3

Package	Item	Title
<chr>	<chr>	<chr>
palmerpenguins	penguins	Size measurements for adult foraging penguins near Palmer Station, Antarctica
palmerpenguins	penguins_raw (penguins)	Penguin size, clutch, and blood isotope data for foraging adults near Palmer Station, Antarctica

5.1.1. penguins_raw

In [71]:
```
glimpse(penguins_raw)
head(penguins_raw)
```

```
Rows: 344
Columns: 17
$ studyName             <chr> "PAL0708", "PAL0708", "PAL0708", "PAL0708", "PAL…
$ `Sample Number`       <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1…
$ Species               <chr> "Adelie Penguin (Pygoscelis adeliae)", "Adelie P…
$ Region                <chr> "Anvers", "Anvers", "Anvers", "Anvers", "Anvers"…
$ Island                <chr> "Torgersen", "Torgersen", "Torgersen", "Torgerse…
$ Stage                 <chr> "Adult, 1 Egg Stage", "Adult, 1 Egg Stage", "Adu…
$ `Individual ID`       <chr> "N1A1", "N1A2", "N2A1", "N2A2", "N3A1", "N3A2", …
$ `Clutch Completion`   <chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "No", …
$ `Date Egg`            <date> 2007-11-11, 2007-11-11, 2007-11-16, 2007-11-16,…
$ `Culmen Length (mm)`  <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34…
$ `Culmen Depth (mm)`   <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18…
$ `Flipper Length (mm)` <dbl> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190,…
$ `Body Mass (g)`       <dbl> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 34…
$ Sex                   <chr> "MALE", "FEMALE", "FEMALE", NA, "FEMALE", "MALE"…
$ `Delta 15 N (o/oo)`   <dbl> NA, 8.94956, 8.36821, NA, 8.76651, 8.66496, 9.18…
$ `Delta 13 C (o/oo)`   <dbl> NA, -24.69454, -25.33302, NA, -25.32426, -25.298…
$ Comments              <chr> "Not enough blood for isotopes.", NA, NA, "Adult…
```

A tibble: 6 × 17

studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Body Mass (g)	Sex	D (c
<chr>	<dbl>	<chr>	<chr>	<chr>	<chr>	<chr>	<chr>	<date>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>	<
PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1	Yes	2007-11-11	39.1	18.7	181	3750	MALE	
PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2	Yes	2007-11-11	39.5	17.4	186	3800	FEMALE	8.94
PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1	Yes	2007-11-16	40.3	18.0	195	3250	FEMALE	8.3(
PAL0708	4	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A2	Yes	2007-11-16	NA	NA	NA	NA	NA	
PAL0708	5	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A1	Yes	2007-11-16	36.7	19.3	193	3450	FEMALE	8.7(
PAL0708	6	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A2	Yes	2007-11-16	39.3	20.6	190	3650	MALE	8.66

5.1.2. penguins

In [72]:

```
glimpse(penguins)
head(penguins)
```

```
Rows: 344
Columns: 8
$ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adel…
$ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgerse…
$ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, …
$ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, …
$ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186…
$ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, …
$ sex               <fct> male, female, female, NA, female, male, female, male…
$ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007…
```

A tibble: 6 × 8

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
<fct>	<fct>	<dbl>	<dbl>	<int>	<int>	<fct>	<int>
Adelie	Torgersen	39.1	18.7	181	3750	male	2007
Adelie	Torgersen	39.5	17.4	186	3800	female	2007
Adelie	Torgersen	40.3	18.0	195	3250	female	2007
Adelie	Torgersen	NA	NA	NA	NA	NA	2007
Adelie	Torgersen	36.7	19.3	193	3450	female	2007
Adelie	Torgersen	39.3	20.6	190	3650	male	2007

5.2. data processing

Data processing (or wrangling) is often the step after the data collection and right before analysis/visualization. The main object of this step is to clean-up the real world messy data into an organized form, including to handle missing data, simplify naming and content, or even to create a specific attribute (feature). Here we would demonstrate the process by converting `penguins_raw` data to `penguins` data in multiple steps by using functions from *tidyverse* package, where you may found other useful packages/functions to fit your research link.

5.2.1. raw data processing

Comparing `penguins_raw` and `penguins` you may found the big difference in many ways, e.g. column removal, column renaming, upper/lower case conversion, time stamp summarization, content manipulation

In [73]:
```
# retain certain columns
names(penguins_raw)
names(penguins)
col_keep = c('Species', 'Island', 'Culmen Length (mm)', 'Culmen Depth (mm)', 'Flipper Length (mm)', 'Body Mass (g)
```

'studyName' · 'Sample Number' · 'Species' · 'Region' · 'Island' · 'Stage' · 'Individual ID' · 'Clutch Completion' · 'Date Egg' · 'Culmen Length (mm)' · 'Culmen Depth (mm)' · 'Flipper Length (mm)' · 'Body Mass (g)' · 'Sex' · 'Delta 15 N (o/oo)' · 'Delta 13 C (o/oo)' · 'Comments'

'species' · 'island' · 'bill_length_mm' · 'bill_depth_mm' · 'flipper_length_mm' · 'body_mass_g' · 'sex' · 'year'

In [74]:
```
# compare "Species" column
penguins_raw$Species %>% unique()
penguins$species %>% unique()
```

'Adelie Penguin (Pygoscelis adeliae)' · 'Gentoo penguin (Pygoscelis papua)' · 'Chinstrap penguin (Pygoscelis antarctica)'

Adelie · Gentoo · Chinstrap

▶ **Levels**:

In [75]:
```
# compare "Island" column
penguins_raw$Island %>% unique()
penguins$island %>% unique()
```

'Torgersen' · 'Biscoe' · 'Dream'

Torgersen · Biscoe · Dream

▶ **Levels**:

In [76]:

```
# compare "Sex" column
penguins_raw$Sex %>% unique()
penguins$sex %>% unique()
```

'MALE' · 'FEMALE' · NA

male · female · <NA>

▶ **Levels**:

In [77]:

```
# compare "Date" column
penguins_raw$`Date Egg` %>% unique()
penguins$year %>% unique()
```

2007-11-11 · 2007-11-16 · 2007-11-15 · 2007-11-09 · 2007-11-12 · 2007-11-10 · 2007-11-13 · 2007-11-19 · 2008-11-06 · 2008-11-09 · 2008-11-15 · 2008-11-13 · 2008-11-11 · 2008-11-14 · 2008-11-08 · 2008-11-02 · 2008-11-07 · 2008-11-17 · 2008-11-05 · 2008-11-10 · 2009-11-09 · 2009-11-15 · 2009-11-20 · 2009-11-12 · 2009-11-17 · 2009-11-18 · 2009-11-22 · 2009-11-16 · 2009-11-21 · 2009-11-23 · 2009-11-10 · 2009-11-13 · 2009-11-14 · 2007-11-27 · 2007-11-18 · 2007-11-29 · 2007-12-03 · 2008-11-04 · 2008-11-03 · 2009-11-25 · 2009-12-01 · 2009-11-27 · 2007-11-26 · 2007-11-21 · 2007-11-28 · 2007-11-22 · 2007-11-30 · 2008-11-25 · 2008-11-24 · 2009-11-19

2007 · 2008 · 2009

With initial comparison, now we can find the main difference between `penguins_raw` and `penguins` data are:

- only some columns are retained and renamed
- *Culmen* columns are renamed as *bill* columns
- *Species* column is simplified
- *Sex* column are in low case
- *Date Egg* column is simplified as year only
- *Character* columns are converted to *factor* columns
- Two columns (*Body Mass* and *year* columns) are integer

In [78]:

```
# user-defined function to simplify Species
species_convert = function(str){
    if(str == 'Adelie Penguin (Pygoscelis adeliae)'){
        return('Adelie')
    } else if(str == 'Gentoo penguin (Pygoscelis papua)'){
        return('Gentoo')
    } else{
        return('Chinstrap')
    }
}
```

In [79]:

```
penguins_process =
    penguins_raw %>%
    # retain column
    select(all_of(col_keep)) %>%
    # rename column
    rename(
        'species' = 'Species', # new_name = old_name
        'island' = 'Island',
        'sex' = 'Sex',
        'body_mass_g' = 'Body Mass (g)',
        'year' = 'Date Egg',
        'flipper_length_mm' = 'Flipper Length (mm)',
        'bill_length_mm' = col_keep[3],
        'bill_depth_mm' = col_keep[4],
    ) %>%
    mutate(
        # simplify Species column
        species = map_chr(species, function(str) species_convert(str)),
        # convert Sex column
        sex = map_chr(sex, tolower),
        # simplify Date column
        year = map_dbl(year, function(date){
            tmp =
                as.character(date) %>%
                str_split(pattern = '-') %>%
                unlist()
            return(as.numeric(tmp[[1]]))
        })
    ) %>%
    # convert character columns as factor columns
    mutate_if(is.character, factor) %>%
    # convert "body mass" and "year" columns as integer
    mutate_at(all_of(c('body_mass_g', 'year')), as.integer)
```

In [80]:

```
head(penguins_process)
```

A tibble: 6 × 8

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
<fct>	<fct>	<dbl>	<dbl>	<dbl>	<int>	<fct>	<int>
Adelie	Torgersen	39.1	18.7	181	3750	male	2007
Adelie	Torgersen	39.5	17.4	186	3800	female	2007
Adelie	Torgersen	40.3	18.0	195	3250	female	2007
Adelie	Torgersen	NA	NA	NA	NA	NA	2007
Adelie	Torgersen	36.7	19.3	193	3450	female	2007
Adelie	Torgersen	39.3	20.6	190	3650	male	2007

In [81]:
```
head(penguins)
```

A tibble: 6 × 8

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
<fct>	<fct>	<dbl>	<dbl>	<int>	<int>	<fct>	<int>
Adelie	Torgersen	39.1	18.7	181	3750	male	2007
Adelie	Torgersen	39.5	17.4	186	3800	female	2007
Adelie	Torgersen	40.3	18.0	195	3250	female	2007
Adelie	Torgersen	NA	NA	NA	NA	NA	2007
Adelie	Torgersen	36.7	19.3	193	3450	female	2007
Adelie	Torgersen	39.3	20.6	190	3650	male	2007

5.2.2. miscellaneous techniques for data wrangling

In [82]:
```
# How many studies in each year?
count(penguins, year)
```

A tibble: 3 × 2

year	n
<int>	<int>
2007	110
2008	114
2009	120

In [83]:

```
# What's the distribution of species in each island?
penguins %>% count(species, island)
```

A tibble: 5 × 3

species	island	n
<fct>	<fct>	<int>
Adelie	Biscoe	44
Adelie	Dream	56
Adelie	Torgersen	52
Chinstrap	Dream	68
Gentoo	Biscoe	124

In [84]:

```
# How many penguins' body mass are larger than 3700 g?
penguins %>%
    filter(body_mass_g>3700) %>%
    nrow()
```

225

In [85]:

```
# What's the largest size of flipper in each species?
penguins %>%
    group_by(species) %>%
    summarize(max_flipper = max(flipper_length_mm, na.rm = T))
```

A tibble: 3 × 2

species	max_flipper
<fct>	<int>
Adelie	210
Chinstrap	212
Gentoo	231

In [86]:

```
# What’s the distribution of bill’s length/depth ratio of each species?
penguins_billstat =
    penguins %>%
    mutate(bill_ratio = bill_length_mm/bill_depth_mm) %>%
    group_by(species) %>%
    nest() %>%
    mutate(stat = purrr::map(data, function(d){summary(d$bill_ratio)}))
penguins_billstat$stat
```

```
[[1]]
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  1.640   2.011   2.137   2.120   2.238   2.450       1

[[2]]
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  2.566   3.066   3.167   3.176   3.285   3.613       1

[[3]]
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.351   2.560   2.662   2.654   2.728   3.258
```

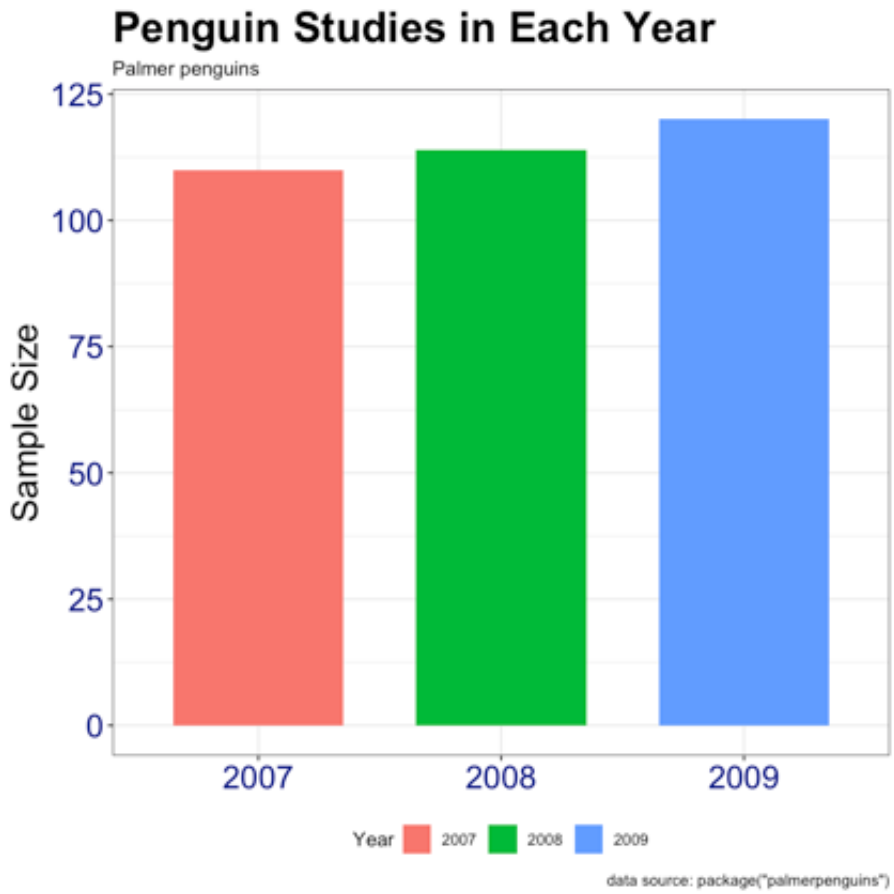
5.3. data visualization

Data visualization is the graphical representation of information and data by using visual concepts like charts, graphs, and maps. Through converting into human-sensible images, symbols, colors, or textures, data used to be hard to intepret are transformed and the information transparency are increased. Several systems are supporting R graphics and *ggplot2* included in *tidyverse* package is one of the package that is a good point to start since its *grammer of graphic* and user communities are truly user-friendly. Also, numerous extensions are developing and pushing the boundary of data visualization to various research fields (e.g. genomic/neuron science/phylogenetic/image analysis) instead of limited in statistical analysis. You may find scientific packages from these resources: *Bioconductor* project, *Neuroconductor project*, and useful ggplot2 extensions to fit your need. And to have a meaningful visualization result, one may need to define the question of interest, target audience, and the key information to give. For this section, we will demonstrate common visualizations in exploratory data analysis using *ggplot2*, *ggpubr* and *viridis* packages for efficient publication-ready and colorblind-friendly graphs.

5.3.1. vertical bar chart via standard *ggplot2*

In [87]:

```
# How many studies in each year?
penguins %>%
    count(year) %>%
    # visualization
    ggplot() +
        geom_bar(
            mapping = aes(x = factor(year), y = n, fill =
        factor(year)),
            stat = "identity",
            width = 0.7
            ) +
        # plot title/axis label setting
        labs(
            title = 'Penguin Studies in Each Year',
            subtitle = 'Palmer penguins',
            caption = 'data source: package("palmerpenguins")',
            x = 'Study Year',
            y = 'Sample Size', # change axis title
            fill = 'Year' # change legend title
            ) +
        # plot theme setting
        theme_bw() +
        theme(
            legend.position = 'bottom', # change legend position
            plot.title = element_text(face = 'bold', size = 24), # change font and size
            axis.text = element_text(color = 'navy', size = 18), # change color and size
            axis.title.x = element_blank(),
            axis.title.y = element_text(size = 20) # hide x-axis title
            )
```

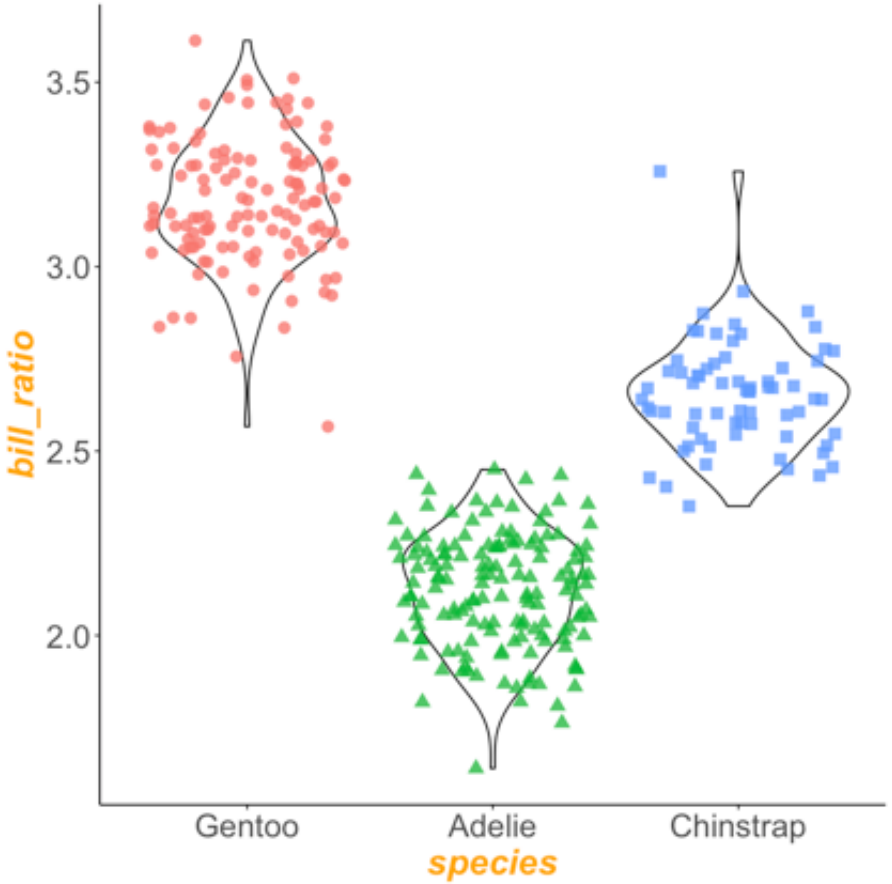


5.3.2. violin plot via standard *ggplot2*

In [88]:

```
# What's the distribution of bill's length/depth ratio of each species?
penguins %>%
    mutate(
        bill_ratio = bill_length_mm/bill_depth_mm,
        # reorder species by the median of body mass
        species = fct_reorder(.f = species, .x = body_mass_g, .fun =
        median, na.rm = T, .desc = T)
        ) %>%
    ggplot(aes(x = species, y = bill_ratio)) +
        geom_violin(color = 'black') +
        geom_jitter(aes(color = species, shape = species), alpha = 0.8, size = 3) +
        theme_classic() +
        theme(
            legend.position = 'none',
            axis.title = element_text(face = 'bold.italic', size = 20,  color = 'orange'),
            axis.text = element_text(size = 18)
            )
```

```
Warning message:
“Removed 2 rows containing non-finite values (stat_ydensity).”
Warning message:
“Removed 2 rows containing missing values (geom_point).”
```



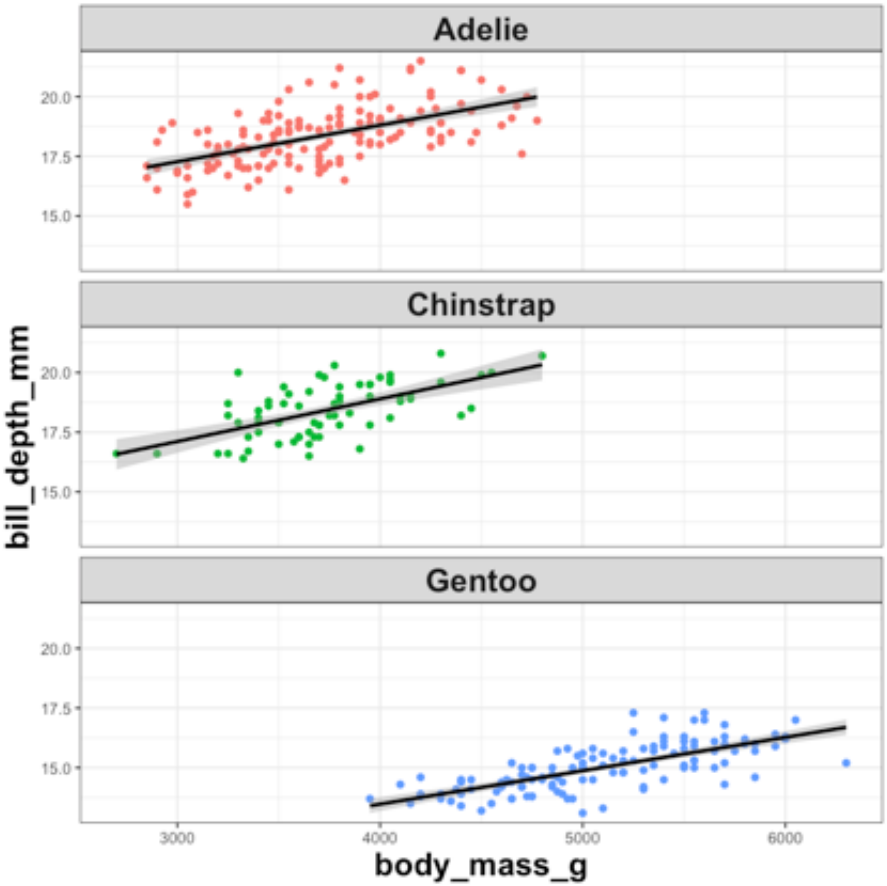
5.3.3. scatter plot with linear relations via standard *ggplot2*

In [89]:

```
# What's the relationship between body mass and other feature in each species?
penguins %>%
    ggplot(aes(x = body_mass_g, y = bill_depth_mm, color = species)) +
    geom_point() +
    geom_smooth(method = 'lm', color = 'black') +
    facet_wrap(~species, nrow = 3, ncol = 1) +
    theme_bw() +
    theme(
        legend.position = 'none',
        axis.title = element_text(face = 'bold', size = 18),
        strip.text = element_text(face = 'bold', size = 18)
        )
```

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

Warning message:
“Removed 2 rows containing non-finite values (stat_smooth).”
Warning message:
“Removed 2 rows containing missing values (geom_point).”
```



5.3.4. density plot via *ggpubr* & *ggsci*

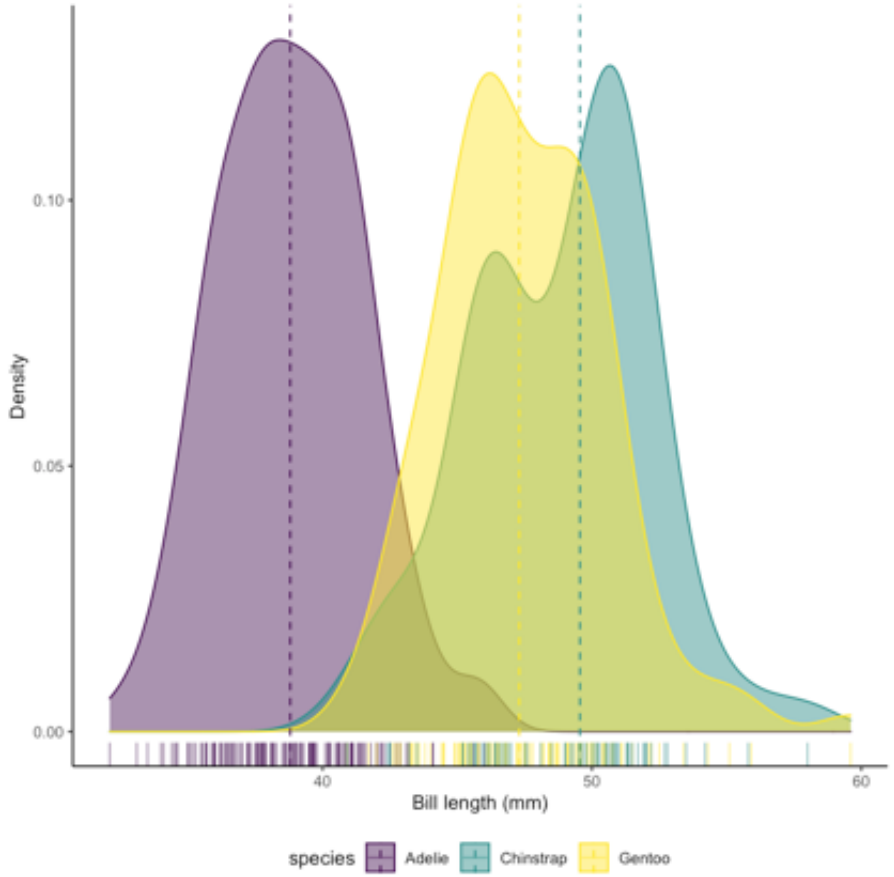
In [90]:

```
library(ggpubr)
library(ggsci)
```

In [91]:

```
# How to understand the distribution of culmen length between penguin speccies?
ggdensity(
    penguins,
    x = "bill_length_mm", # target variable (feature)
    add = "median", # additional/optional group average (mean)
    rug = TRUE, # add marginal rug for data distribution
    color = "species", # group stratification reference
    fill = "species", # group stratification reference
) +
    scale_fill_viridis_d() + # colorblind-friendly coloring
    scale_color_viridis_d() + # colorblind-friendly coloring
    labs(x = 'Bill length (mm)', y = 'Density') +
    theme_classic() +
    theme(legend.position = 'bottom')
```

```
Warning message:
“Removed 2 rows containing non-finite values (stat_density).”
```

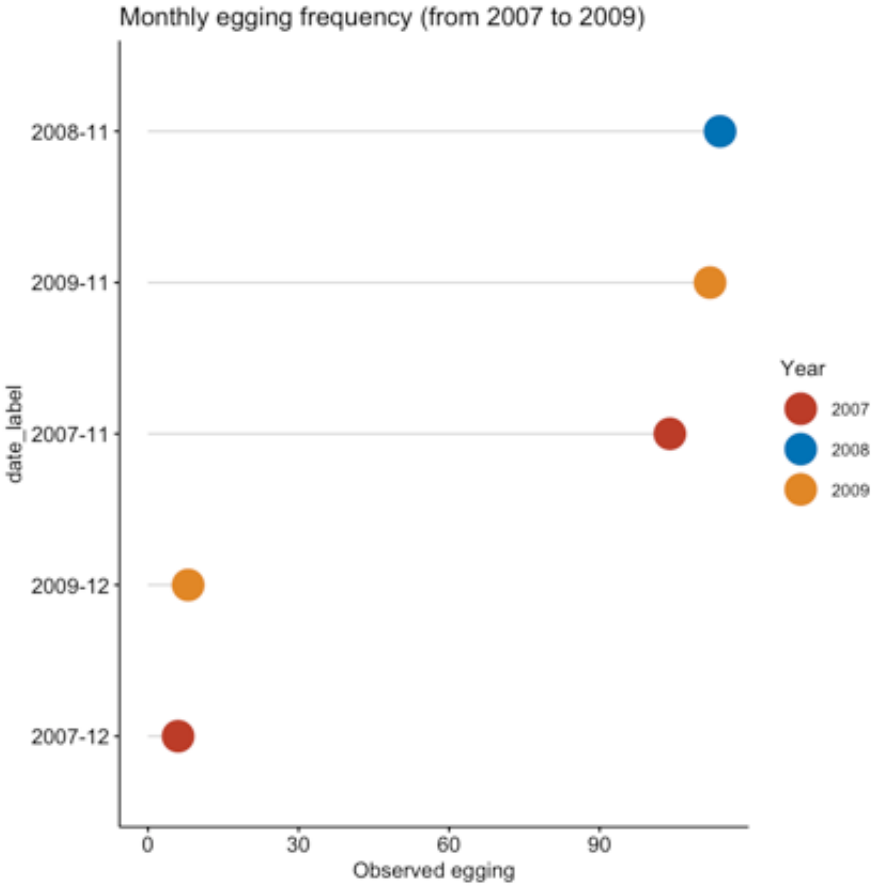


5.3.5. dot plot via *ggpubr* & *ggsci*

In [93]:

```
# What's the egging frequency of these penguins in each month by years?
penguins_raw %>%
    mutate(
        yr_egg = map_dbl(`Date Egg`, function(date){
            tmp =
                as.character(date) %>%
                str_split(pattern = '-') %>%
                unlist()
            return(as.numeric(tmp[[1]]))
        }),
        month_egg = map_dbl(`Date Egg`, function(date){
            tmp =
                as.character(date) %>%
                str_split(pattern = '-') %>%
                unlist()
            return(as.numeric(tmp[[2]]))
        })
    ) %>%
    #dplyr::select(Species, yr_egg, month_egg)
    group_by(yr_egg, month_egg) %>%
    summarize(count = n()) %>%
    mutate(
        date_label = paste(yr_egg, month_egg, sep = '-'),
        yr_egg = as.factor(yr_egg) # convert numeric `yr_egg` to factor for grouping
    ) %>%
    # dot plot
    ggdotchart(
        x = "date_label", y = "count",
        color = "yr_egg", # color group by `yr_egg`
        sorting = "descending", # sort value in descending order
        add = "segments", # add segments from y = 0 to dots
        rotate = TRUE, # rotate vertically
        dot.size = 8,
        ) +
        ggsci::scale_color_nejm() + # using NEJM coloring
        labs(
            title = 'Monthly egging frequency (from 2007 to 2009)',
            color = 'Year', X = 'Month', y = 'Observed egging') +
        theme(legend.position = 'right')
```

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



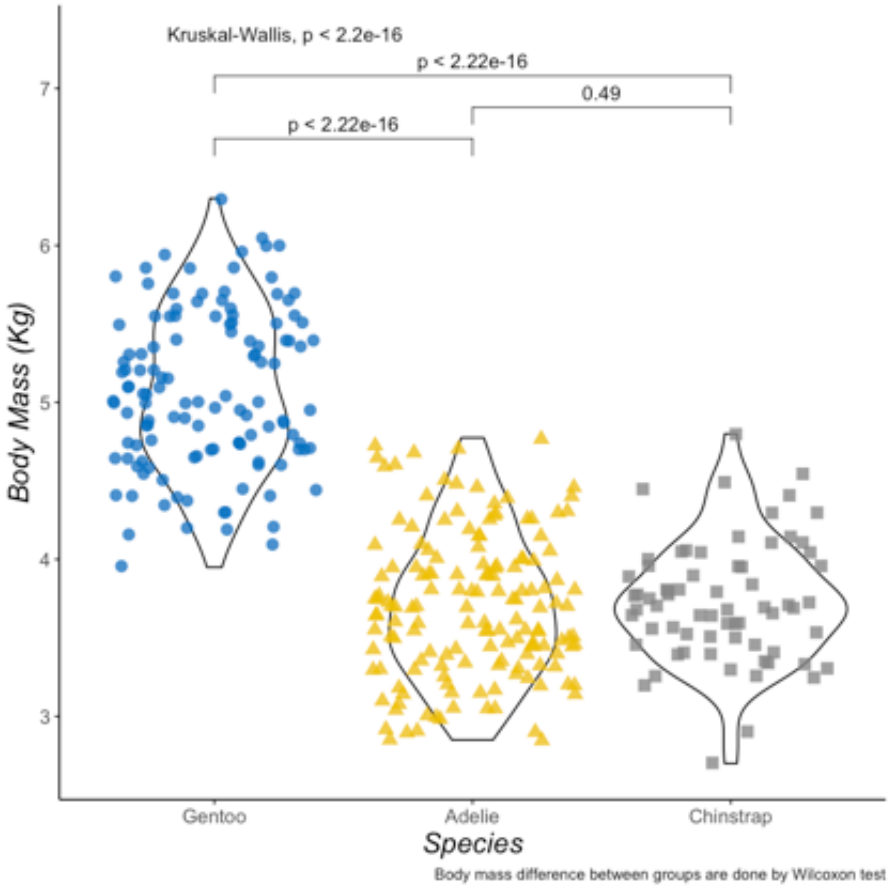
5.3.6. advanced violin plot via *ggpubr* & *ggsci*

In [94]:

```
# What's the distribution of body mass of each species? Please provide statistical significance as well.
compares = list(c('Adelie', 'Gentoo'), c('Adelie', 'Chinstrap'), c('Gentoo', 'Chinstrap'))

penguins %>%
    mutate(
        # reorder species by the median of body mass
        species = fct_reorder(.f = species, .x = body_mass_g, .fun =
        median, na.rm = T, .desc = T),
        body_mass_kg = body_mass_g/1000
        ) %>%
    ggplot(aes(x = species, y = body_mass_kg)) +
        geom_violin(color = 'black') +
        geom_jitter(aes(color = species, shape = species), alpha = 0.8, size = 3) +
        theme_classic() +
        theme(
            legend.position = 'none',
            axis.title = element_text(size = 16, face = 'italic'),
            axis.text = element_text(size = 11)
            ) +
    ggsci::scale_color_jco() + # using JCO coloring
    labs(
        x = 'Species', y = 'Body Mass (Kg)',
        caption = 'Body mass difference between groups are done by Wilcoxon test'
    ) +
    # add statistical results between species
    stat_compare_means(
        method = 'wilcox.test', # can change to *t.test*
        comparisons = compares, # combination of group comparisons
        label.y = c(6.5, 6.7, 6.9) # position for significance labling
    )+
    # add global anova p-value
    stat_compare_means(label.y = 7.3)
```

```
Warning message:
“Removed 2 rows containing non-finite values (stat_ydensity).”
Warning message:
“Removed 2 rows containing non-finite values (stat_signif).”
Warning message:
“Removed 2 rows containing non-finite values (stat_compare_means).”
Warning message:
“Removed 2 rows containing missing values (geom_point).”
```

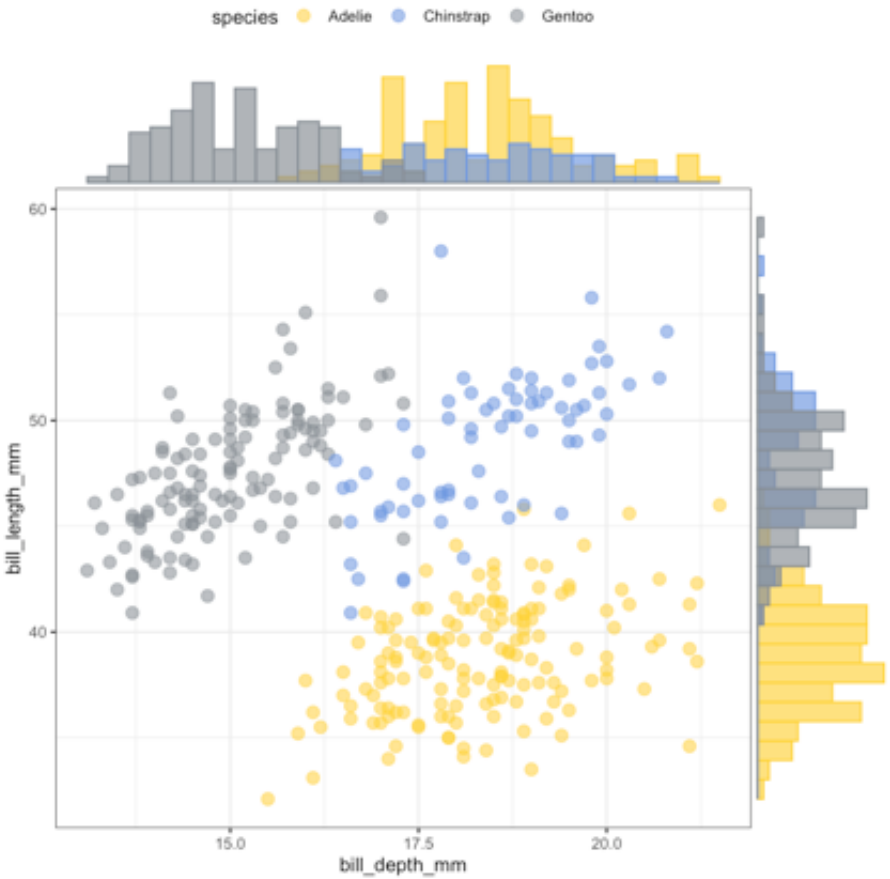


5.3.7. advanced scatter plot via *ggpubr* & *ggsci*

In [95]:

```
# How to visualize the bill's depth and length relationships between penguin species?
ggscatterhist(
    penguins,
    x = "bill_depth_mm", y = "bill_length_mm",
    color = "species", fill = 'species',
    size = 3, alpha = 0.6,
    palette = "simpsons",
    margin.plot = "histogram",
    ggtheme = theme_bw()
)
```

```
Warning message:
“Removed 2 rows containing missing values (geom_point).”
Warning message:
“Removed 2 rows containing missing values (geom_point).”
Warning message:
“Removed 2 rows containing non-finite values (stat_bin).”
Warning message:
“Removed 2 rows containing missing values (geom_point).”
Warning message:
“Removed 2 rows containing non-finite values (stat_bin).”
```



6. Cancer omic resources

Various databases and web tools are providing public cancer data and some of them are also visualizing summary plots and advanced survival analysis upon user query. And among numerous cancer data contributors, the Cancer Genome Atlas (TCGA) is one of widely recognized by researcher and also archived by databases including Genomic Data Commons (GDC), cBioPortal, Broad GDAC Firehose, UCSC Xena. Furthermore, apart from direct download, these sources and numerous R packages (RTCGAToolbox, TCGAbiolinks, cBioPortalData, etc.) are providing access for cancer data. With this section, we will use *UCSC Xena* for followed demonstration.

> Xena is a genomic data accessing-hub hosted by UCSC providing the collection of other public databases (TCGA, Broad Institute, ICGC, GTex, CCLE, etc.). Users can access, download, analyze data like cancer or single-cell datasets directly on the website or locally on personal device for the data deployed on Xena are tier-3 public data. And *UCSCXenaTools* is the R package providing the accession inside R environment for convenient analysis workflow.

6.1. Xena overview & searching

6.1.1. overview

Xena datasets viewpage

In [ ]:
```
library(tidyverse)
library(UCSCXenaTools)
data(XenaData)
```

In [3]:
```
head(XenaData)
```

XenaHosts <chr>	XenaHostNames <chr>	XenaCohorts <chr>	XenaDatasets <chr>	SampleCount <int>	DataSubtype <chr>
https://ucscpublic.xenahubs.net	publicHub	Breast Cancer Cell Lines (Neve 2006)	ucsfNeve_public/ucsfNeveExp_genomicMatrix	51	gene expressio
https://ucscpublic.xenahubs.net	publicHub	Breast Cancer Cell Lines (Neve 2006)	ucsfNeve_public/ucsfNeve_public_clinicalMatrix	57	phenotype
https://ucscpublic.xenahubs.net	publicHub	Glioma (Kotliarov 2006)	kotliarov2006_public/kotliarov2006_genomicMatrix	194	copy numbe
https://ucscpublic.xenahubs.net	publicHub	Glioma (Kotliarov 2006)	kotliarov2006_public/kotliarov2006_public_clinicalMatrix	194	phenotype
https://ucscpublic.xenahubs.net	publicHub	Lung Cancer CGH (Weir 2007)	weir2007_public/weir2007_genomicMatrix	383	copy numbe
https://ucscpublic.xenahubs.net	publicHub	Lung Cancer CGH (Weir 2007)	weir2007_public/weir2007_public_clinicalMatrix	383	phenotype

6.1.2. searching6

OR

In [4]:

```
XenaScan(
    pattern = 'Glioblastoma|gene expression',
    ignore.case = T) %>%
    head(3)
```

XenaHosts <chr>	XenaHostNames <chr>	XenaCohorts <chr>	XenaDatasets <chr>	SampleCount <int>	DataSubtype <chr>	La <ch
https://ucscpublic.xenahubs.net	publicHub	Breast Cancer Cell Lines (Neve 2006)	ucsfNeve_public/ucsfNeveExp_genomicMatrix	51	gene expression	Neve (Line g expres
https://ucscpublic.xenahubs.net	publicHub	Breast Cancer (Chin 2006)	chin2006_public/chin2006Exp_genomicMatrix	118	gene expression	G expres
https://ucscpublic.xenahubs.net	publicHub	Melanoma (Lin 2008)	lin2008_public/lin2008Exp_genomicMatrix	95	gene expression	Lin G

AND

In [5]:

```
XenaScan(
    pattern = 'Glioblastoma.*gene expression|gene expression.*Glioblastoma',
    ignore.case = T) %>%
    head(3)
```

XenaHosts <chr>	XenaHostNames <chr>	XenaCohorts <chr>	XenaDatasets <chr>	SampleCount <int>	DataSubtype <chr>	Label <chr>	
https://tcga.xenahubs.net	tcgaHub	TCGA Glioblastoma (GBM)	TCGA.GBM.sampleMap/HT_HG-U133A	539	gene expression array	AffyU133a	g
https://tcga.xenahubs.net	tcgaHub	TCGA Glioblastoma (GBM)	TCGA.GBM.sampleMap/HiSeqV2_percentile	172	gene expression RNAseq	IlluminaHiSeq percentile	g
https://tcga.xenahubs.net	tcgaHub	TCGA Glioblastoma (GBM)	TCGA.GBM.sampleMap/HiSeqV2_PANCAN	172	gene expression RNAseq	IlluminaHiSeq pancan normalized	g

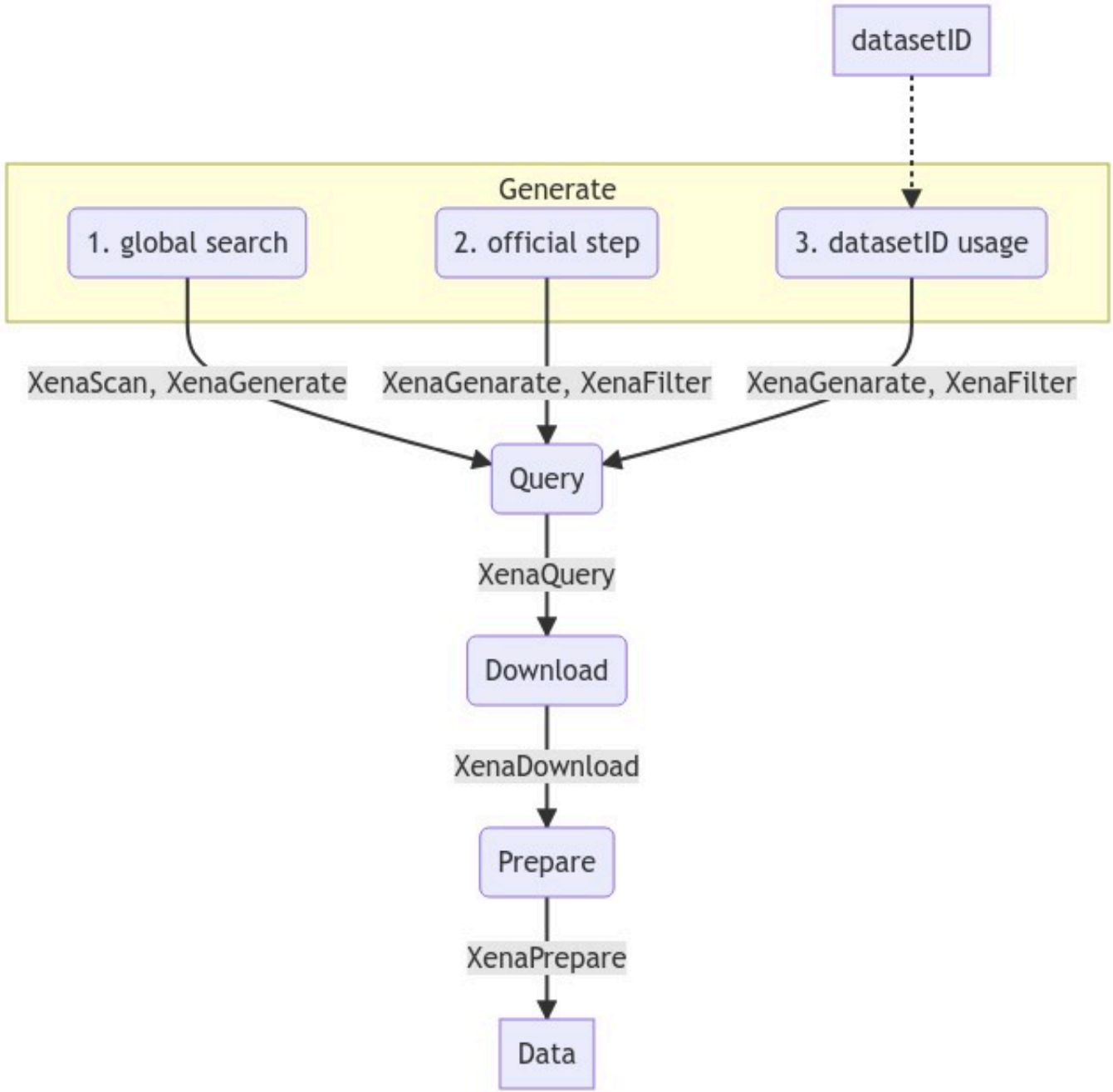
grep

In [6]:

```
XenaData %>%
    filter(
        grepl(XenaHostNames, pattern = 'tcga', ignore.case = T),
        grepl(DataSubtype, pattern = 'gene expression', ignore.case = T),
        grepl(AnatomicalOrigin, pattern = 'brain', ignore.case = T)
    ) %>%
    head(3)
```

XenaHosts	XenaHostNames	XenaCohorts	XenaDatasets	SampleCount	DataSubtype	Label	
<chr>	<chr>	<chr>	<chr>	<int>	<chr>	<chr>	
https://tcga.xenahubs.net	tcgaHub	TCGA Lower Grade Glioma (LGG)	TCGA.LGG.sampleMap/HiSeqV2_percentile	530	gene expression RNAseq	IlluminaHiSeq percentile	g
https://tcga.xenahubs.net	tcgaHub	TCGA Lower Grade Glioma (LGG)	TCGA.LGG.sampleMap/HiSeqV2_PANCAN	530	gene expression RNAseq	IlluminaHiSeq pancan normalized	g
https://tcga.xenahubs.net	tcgaHub	TCGA Lower Grade Glioma (LGG)	TCGA.LGG.sampleMap/HiSeqV2	530	gene expression RNAseq	IlluminaHiSeq	g

6.2. access Xena via *UCSCXenaTools* package


datasetID
Generate
1. global search
2. official step
3. datasetID usage
XenaScan, XenaGenerate
XenaGenarate, XenaFilter
XenaGenarate, XenaFilter
Query
XenaQuery
Download
XenaDownload
Prepare
XenaPrepare
Data

6.2.1. step 1: generate

option 1: global search-based

In [ ]:
```
file_op1 =
    XenaScan(pattern = 'Glioblastoma.*gene expression|gene expression.*Glioblastoma', ignore.case = T) %>% # searc
    XenaGenerate()
file_op1@datasets
```

option 2: official steps

In [ ]:
```
file_op2 =
    XenaGenerate(subset = XenaHostNames=="tcgaHub") %>%
    XenaFilter(filterDatasets = "GBM.*HiSeqV2_PANCAN|HiSeqV2_PANCAN.*GBM")
file_op2@datasets
```

option 3: use datasetID directly

In [ ]:
```
datasetID = 'TCGA.GBM.sampleMap/HiSeqV2_PANCAN'
file_op3 =
    XenaGenerate() %>%
    XenaFilter(filterDatasets = datasetID)
file_op3@datasets
```

6.2.2. step2~4: query, download, prepare

In [ ]:
```
df_gbm_op3 =
    file_op3 %>%
    XenaQuery() %>%
    XenaDownload() %>%
    XenaPrepare()
```

In [ ]:
```
dim(df_gbm_op3)
```

In [ ]:
```
head(df_gbm_op3)
```

6.3 direct download from Xena

demo webpage

In [ ]:
```
#file = 'https://raw.githubusercontent.com/yenhsieh/LSL_2022/main/data/TCGA_CHOL_RNAseq'
file_url = 'https://tcga-xena-hub.s3.us-east-1.amazonaws.com/download/TCGA.CHOL.sampleMap%2FHiSeqV2.gz'
data = read_delim(file_url, delim = '\t')
head(data)
```

7. Differential gene expression and geneset analysis

Over-representation analysis (ORA) is a statistical method to determine if a geneset (e.g. pathway from Gene Ontology or KEGG) is over-representated within user **gene list**, e.g. differentially expressed genes (DEGs) between two groups selected by the significance cutoff. Geneset enrichment analysis (GSEA) is a extented version of ORA whereas DEGs are **ranked by the fold change** as input, unlike ORA using DEGs list only, is much feasible for situations where all genes in a predefined gene set change in a small but coordinated way. As metioned above, either ORA or GSEA are requiring DEGs as input component, which can be identified through various methods/packages (e.g. *DESeq2*, *edgeR*, etc) and here we will start from *limma-voom* of *edgeR* to discover DEGs for downstream ORA and GSEA analysis.

> *clusterProfiler* is a R package which provide a various accession to a list of well-known annotation databases, e.g. KEGG, Reactome, Wiki Pathways. Apart from these annotation collected, it is also capable for both semantic similarity and functional enrichment analysis with a friendly and tidy fashion for biologist without command line tool experience to access, analyze and visualize their results. According to official document, genomic coordinates, gene and gene cluster are supported as input format. (source)

7.1. data retrieval and pre-process

To demonstrate the workflow of DEG, here we start from downloding bladder cancer's gene expression data from Xena as example. With the input format (a *n*m numeric matrix* object whereas columns are samples and rows are genomic features) as suggested by *edgeR* document, you may prepare the input beyond this cancer data example. (data source)

7.1.1. data retrieval

In [ ]:
```
dataID = 'TCGA.BLCA.sampleMap/HiSeqV2$'
df_exp =
    XenaGenerate() %>%
    XenaFilter(filterDatasets = dataID) %>%
    XenaQuery() %>%
    #XenaDownload(force = F, destdir = "C:/Users/YHC/Documents") %>% # Windows System
    XenaDownload(force = F) %>%
    XenaPrepare() %>%
    dplyr::rename('gene'='sample')
```

7.1.2. data preprocess

data conversion (data-dependent)

Considering the unit from Xena is **log2(norm_count+1)**, here we reverse expected_count to original count and take the nearest integer and you may find the difference below.

In [ ]:
```
df_exp_new =
    df_exp %>%
    mutate_if(is.numeric, function(x){round(2^x-1, digits = 0)})
```

In [ ]:
```
head(df_exp)
```

In [ ]:
```
head(df_exp_new)
```

In order to confirm the input having any missing data, genes having all zero count, and correct format (i.e. *matrix*) for followed-up analysis, data wrangling are required.

checking NA values

In [ ]:
```
# Check if missing value exist and convert dataframe into matrix format
df_exp_new %>%
    select_if(is.numeric) %>% # select numeric columns
    as.matrix() %>% {.->>m_exp} %>% # convert to matrix
    as.vector() %>% # convert 2D matrix as 1D array
    is.na() %>% # check if NA
    sum() # NA count
```

design matrix annotation

In [ ]:
```
# convert input as matrix and add row names
row.names(m_exp) = df_exp_new$gene
m_exp[1:6, 1:5]
```

removal of rows with all zeros

In [ ]:
```
# identify rows with all zero
all_zero = map_lgl(1:nrow(m_exp), function(i){ sum(m_exp[i,] == 0) == ncol(m_exp)})
table(all_zero)
```

In [ ]:
```
# retain rows not all zero
m_exp_new = m_exp[-which(all_zero == TRUE),]

dim(m_exp)
dim(m_exp_new)
```

As indicated, some rows (genes) are having all patients with no expression count hence will be removed.

sample group annotation

Sample group annotation is required since the main object for this analysis is to distinguish the difference between two groups and here TCGA samples will be classified as tumor or adjacent by the barcode accordingly.

In [22]:
```
sample_class = map_chr(colnames(m_exp_new), function(x){
        ifelse(str_sub(x, 14,14)=='0', yes = 'tumor', no = 'adjNormal')
    }) %>% factor()

table(sample_class)
```

```
sample_class
adjNormal     tumor
       19       407
```

Patient data collected by TCGA are labeled with its own barcode system, which a dataset may composed of tumor and adjacent normal samples. Whether if the sample is tumor(*0*) or adjacent normal(*1*) could be defined by the 14th position of the barcode. With all these process, our data is ready for differential gene expression analysis.

(barcode table))

7.2. differential expression gene analysis

7.2.1. normalization

In [23]:
```
library(edgeR)
# quantile normalization
y = normalizeQuantiles(m_exp_new)
```

```
Loading required package: limma
```

7.2.2. count/sample filter

With the setting of desired cutoff, genes with low-expressed count and expressed in small size samples would be neglected. The difference of before/after this procedure could be visualized via Voom variance plot, whereas each dot in the figure represents a gene and also the mean-variance relationship of included genes in the dataset. The main purpose of this plot is to identify if low counts genes have been filtered and if lots of variation in the data exist. The cutoff setting is data-dependent.

In [24]:
```
# cutoff setting
cutoff_count = 10
cutoff_sample = 20

genes_keep = rowSums(y > cutoff_count) >= cutoff_sample
table(genes_keep)
```

```
genes_keep
FALSE  TRUE 
 3011 17203 
```

7.2.3. intepretation via voom variance plot

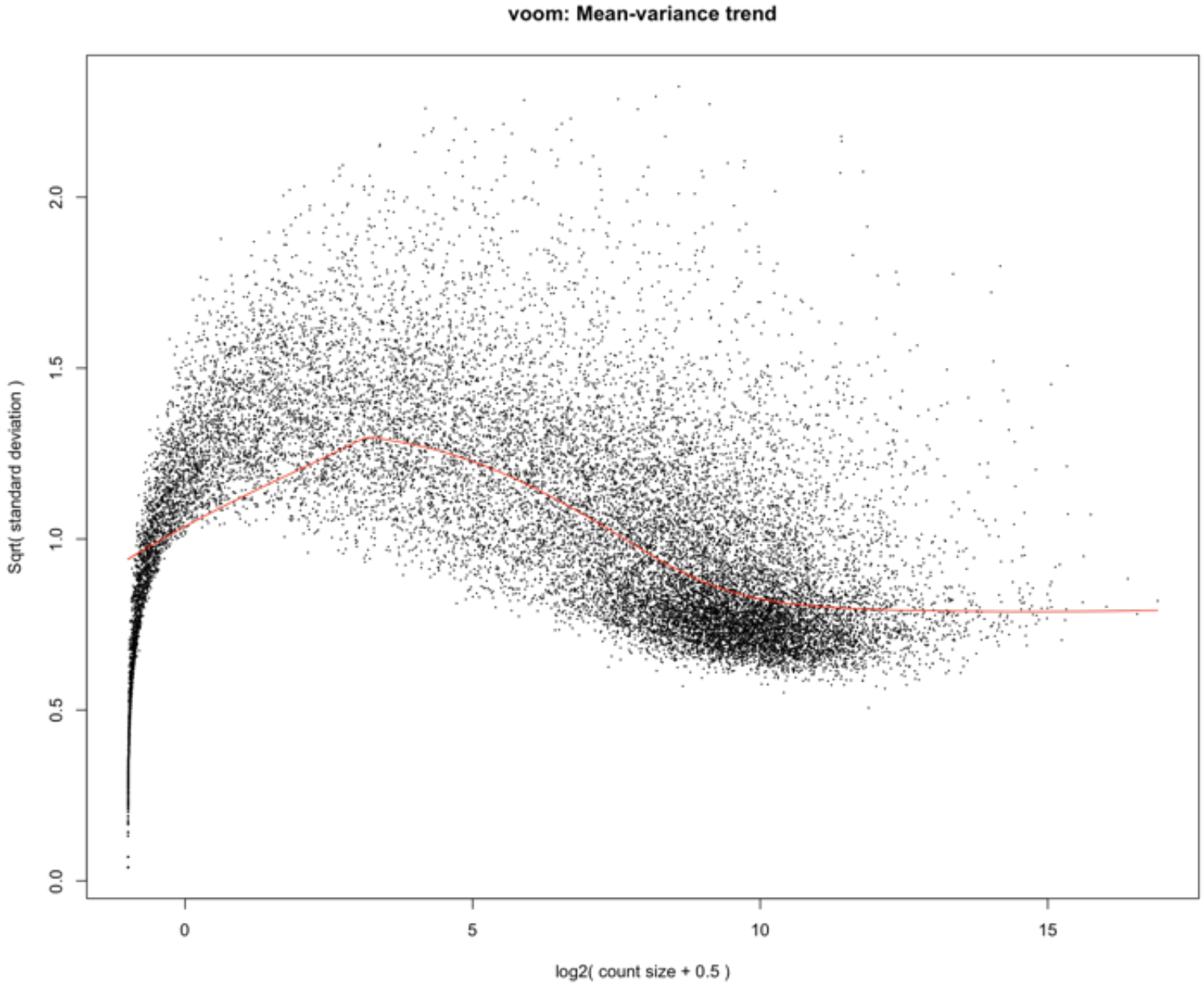
In [25]:
```
design = model.matrix(~0 + sample_class)
colnames(design) = c("adjNormal","tumor")
head(design)
```

A matrix: 6 × 2 of type dbl

	adjNormal	tumor
1	1	0
2	0	1
3	0	1
4	0	1
5	0	1
6	0	1

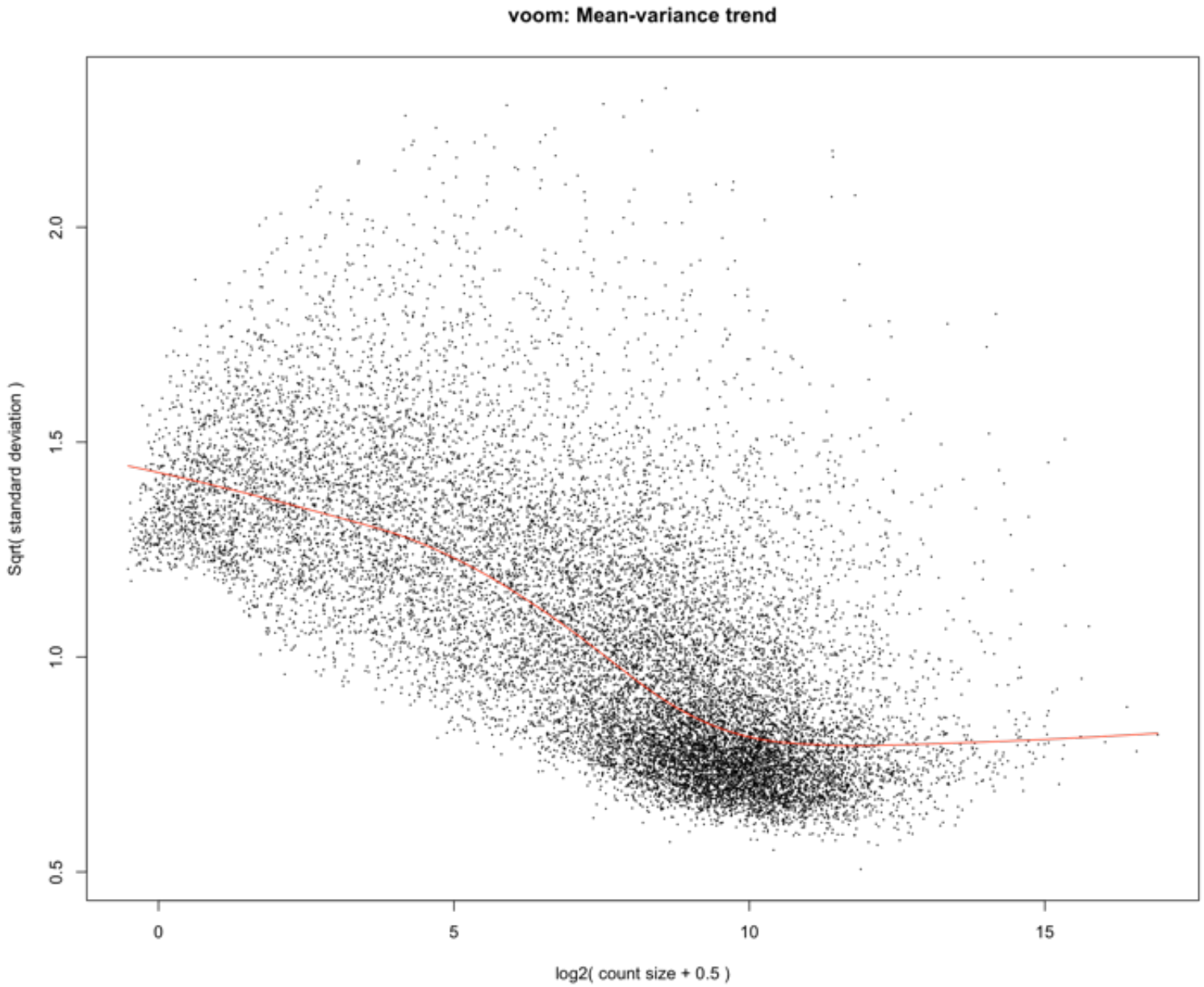
In [26]:

```
bef = voom(y, design, plot = T)
```


voom: Mean-variance trend
Sqrt(standard deviation)
0.0
0.5
1.0
1.5
2.0
0
5
10
15
log2(count size + 0.5)

In [27]:

```
y_after = y[genes_keep,]
aft = voom(y_after, design, plot = T)
```



7.2.4. model fitting

In [28]:

```
contrast = makeContrasts(tumor-adjNormal, levels=design)
contrast
```

A matrix: 2 × 1 of type dbl

	tumor - adjNormal
adjNormal	-1
tumor	1

In [29]:

```
fit = lmFit(y_after, design)
fitC = contrasts.fit(fit, contrast)
fitC = eBayes(fitC, robust=TRUE, trend=TRUE)
summary(decideTests(fitC))
```

```
       tumor - adjNormal
Down                3081
NotSig             10614
Up                  3508
```

In [30]:

```
# fitting result table
tab_deg =
    topTable(fitC, adjust="BH", n=Inf) %>%
    na.omit() %>%
    rownames_to_column('gene') %>%
    as_tibble() %>%
    dplyr::select(gene, everything())

dim(tab_deg)
head(tab_deg)
```

17203 · 7

A tibble: 6 × 7

gene	logFC	AveExpr	t	P.Value	adj.P.Val	B
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
C16orf89	-620.0308	50.57542	-28.23560	1.408132e-99	2.422410e-95	-1.788981
F10	-352.6064	37.23251	-26.80088	2.084259e-93	1.792775e-89	-1.893120
CLEC3B	-2284.7473	219.54431	-25.25750	1.158656e-86	6.644122e-83	-2.015517
SCARA5	-2271.2513	157.10191	-24.93476	3.050763e-85	1.312057e-81	-2.042396
PI16	-5118.6775	333.11552	-24.31372	1.688589e-82	5.809759e-79	-2.095473
C1QTNF7	-464.7453	57.62090	-23.03347	8.268577e-77	2.370739e-73	-2.210654

In [31]:

```
# retain significant DEGs
sig_deg = filter(tab_deg, abs(logFC)>0.5 & tab_deg$adj.P.Val < 0.05)
```

In [32]:

```
# classify as up-/down-regulated DEGs
deg_up = filter(sig_deg, logFC > 1) %>% arrange(desc(logFC))
deg_dn = filter(sig_deg, logFC < -1) %>% arrange(desc(logFC))
```

7.3. geneset analysis

The main object of geneset/pathway analysis is to determines if a pre-defined sets, pathways collected from KEGG or Gene Ontology, are over-represented in a subset of user data beyond expected. For instance, given a geneset composed of 100 genes (e.g. liver cancer-specific pathway) and a knockout experiment with 500 DEGs while 90 of them are in the geneset, then the geneset is highly over/under-represented in the experiment. Above mentioned is the straightforward concept of conventional over-representation analysis (ORA) and followed by statistical tests such as the cumulative hyper-geometric test, significance of the overlapping could be calculated and filtered by cutoff (e.g. p-value 0.05) to select the annotated genesets.

Unlike ORA, Gene Set Enrichment Analysis (GSEA) ranks all genes in the genome to eliminate the need of cutoff as ORA (e.g. fold change) to define the input gene set. With such ranking, e.g. level of differential expression, genesets are expected as high or low via running-sum statistic.

In [ ]:
```
library(org.Hs.eg.db)
library(clusterProfiler)
library(enrichplot)
```

7.3.1. ORA analysis and visualization

Here we take down-regulated DEGs to determine the enrichment result in the Biological Process category of Gene Ontology and also several visualization presentation.

In [34]:
```
geneList_dn = deg_dn$logFC %>% setNames(deg_dn$gene)

DEG_ora_dn = enrichGO(
    gene = names(geneList_dn), # only gene list is required
    OrgDb = org.Hs.eg.db,
    keyType="SYMBOL",
    ont = "BP",
    pAdjustMethod = "BH")
```

In [35]:
```
DEG_ora_dn
```

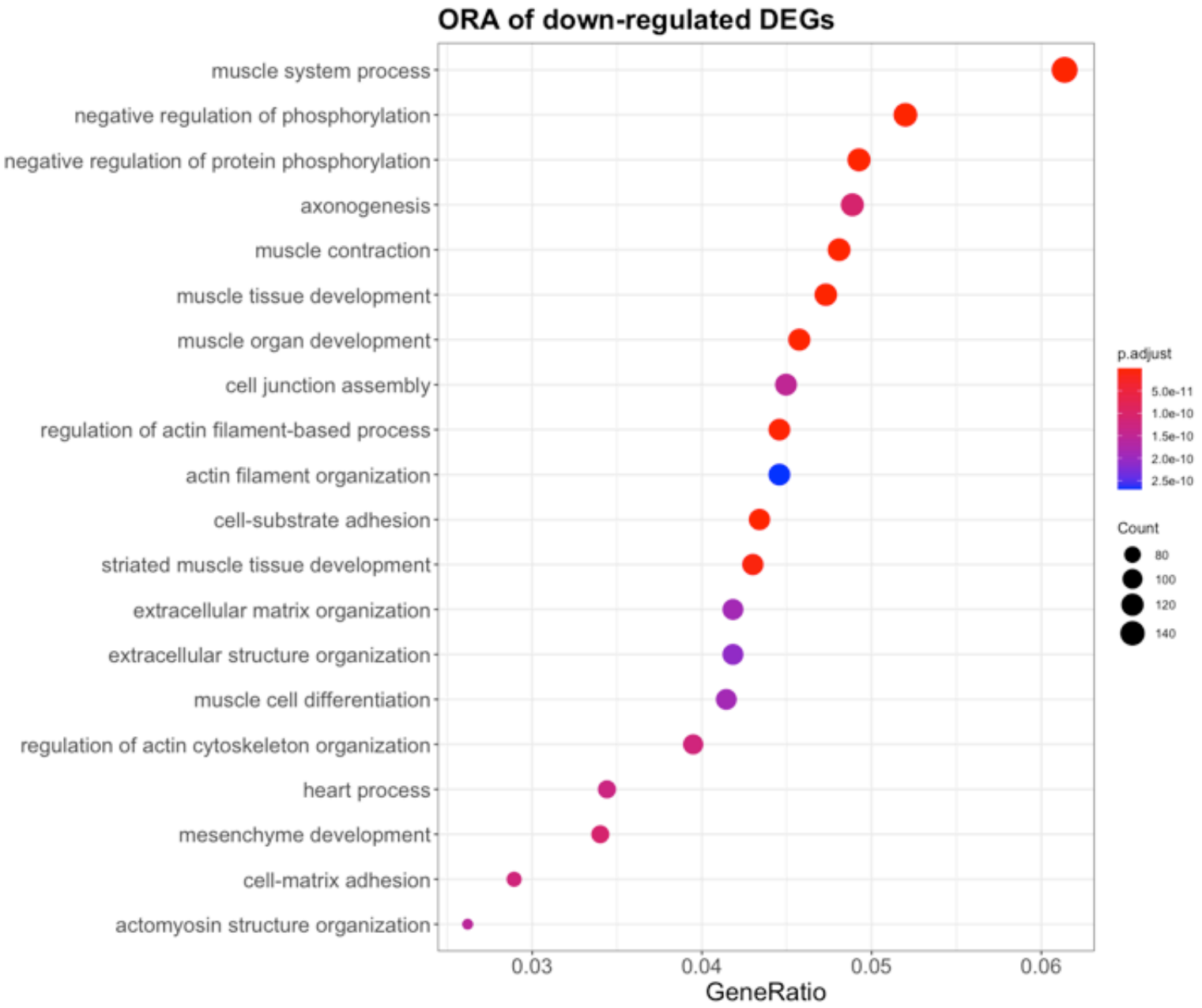
```
#
# over-representation test
#
#...@organism    Homo sapiens
#...@ontology    BP
#...@keytype     SYMBOL
#...@gene        chr [1:3081] "DMRTC1B" "SEC1" "FAM7A3" "C22orf26" "CCDC158" "CC2D2B" ...
#...pvalues adjusted by 'BH' with cutoff <0.05
#...1314 enriched terms found
'data.frame':   1314 obs. of  9 variables:
 $ ID         : chr  "GO:0003012" "GO:0006936" "GO:0031589" "GO:0060537" ...
 $ Description: chr  "muscle system process" "muscle contraction" "cell-substrate adhesion" "muscle tissue develop
ment" ...
 $ GeneRatio  : chr  "157/2558" "123/2558" "111/2558" "121/2558" ...
 $ BgRatio    : chr  "467/18866" "362/18866" "359/18866" "409/18866" ...
 $ pvalue     : num  3.14e-29 1.41e-23 7.24e-18 1.12e-17 9.06e-17 ...
 $ p.adjust   : num  1.95e-25 4.39e-20 1.50e-14 1.74e-14 1.13e-13 ...
 $ qvalue     : num  1.40e-25 3.14e-20 1.08e-14 1.25e-14 8.08e-14 ...
 $ geneID     : chr  "HTR2A/SLC8A3/KCNJ3/HTR2B/NMUR1/SCN3B/CTNNA3/TNNI3K/SCN2B/TMOD4/MYOT/ACTA1/LMOD3/KCNN2/TRPC3/
KCNA5/SMPX/CKMT2/TA"| __truncated__ "HTR2A/SLC8A3/KCNJ3/HTR2B/NMUR1/SCN3B/CTNNA3/TNNI3K/SCN2B/TMOD4/MYOT/ACTA1/LMO
D3/KCNN2/KCNA5/SMPX/CKMT2/TACR1/KC"| __truncated__ "TECTA/MADCAM1/PPFIA2/FER/RADIL/EDA/COL13A1/CORO2B/EPB41L5/PRKC
E/EDIL3/ANGPT1/ATP1B2/NTNG1/FREM1/TEK/CLASP2/ECM2"| __truncated__ "KCNK2/KCNAB1/MTM1/GLI1/SGCG/ACTA1/LMOD3/FGF9/ME
OX2/S100B/TWIST1/MYOM2/PPARGC1A/MYO18B/SAV1/TBX20/MAP2K4/CREB1/B"| __truncated__ ...
 $ Count      : int  157 123 111 121 126 133 117 114 110 125 ...
#...Citation
  Guangchuang Yu, Li-Gen Wang, Yanyan Han and Qing-Yu He.
  clusterProfiler: an R package for comparing biological themes among
  gene clusters. OMICS: A Journal of Integrative Biology
  2012, 16(5):284-287
```

dot plot display

In [36]:

```
dotplot(DEG_ora_dn, showCategory = 20) +
    labs(title = "ORA of down-regulated DEGs") +
    theme_bw() +
    theme(
        plot.title = element_text(face = 'bold', size = 20),
        axis.text = element_text(size = 15),
        axis.title = element_text(size = 18)
    )
```

```
wrong orderBy parameter; set to default `orderBy = "x"`
```

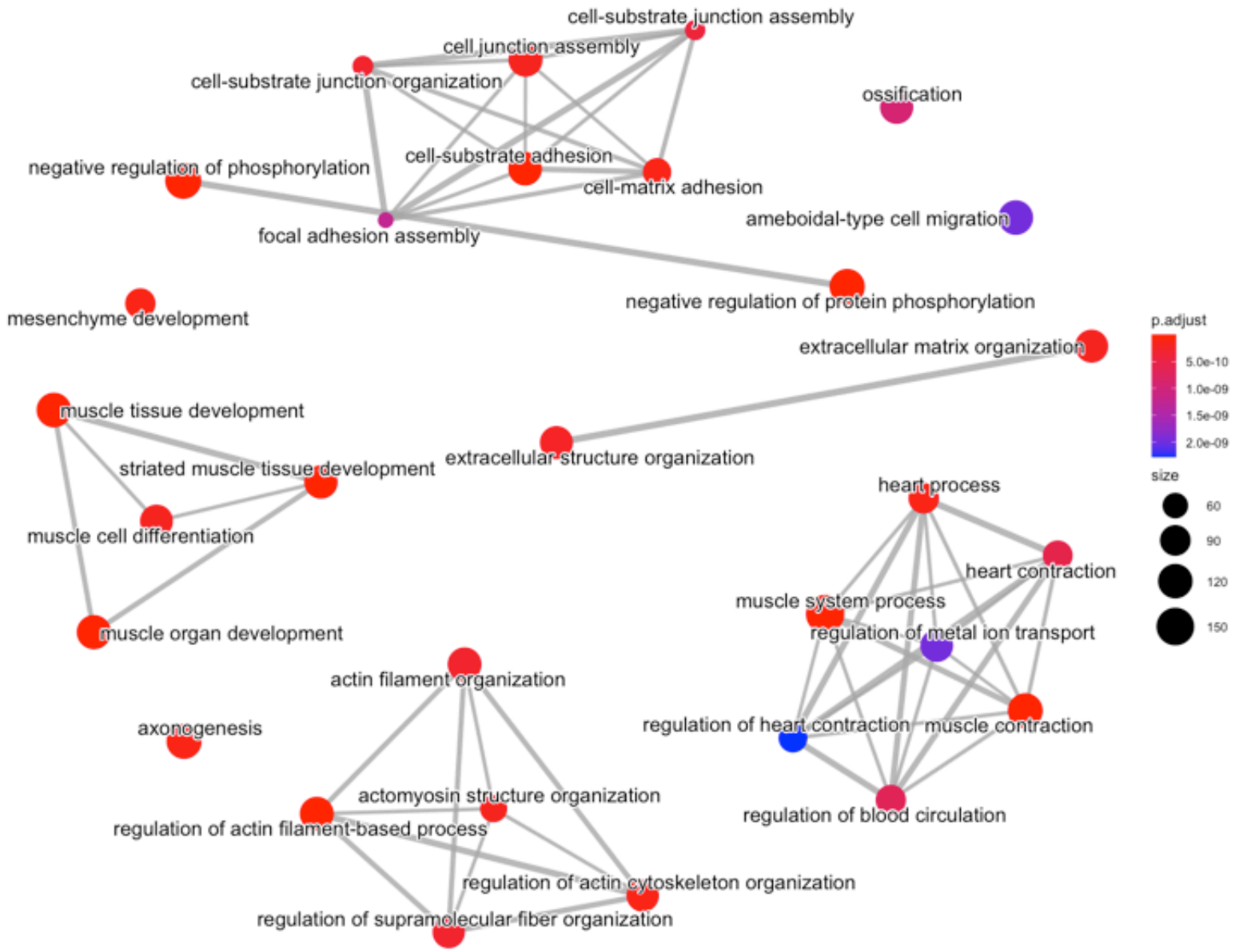


enrichment map display

In [37]:

```
DEG_ora_dn %>%
    pairwise_termsim() %>%
    emapplot(layout = 'kk', cex_category = 1.5) +
    labs(title = 'Enrichment map of down-regulated DEGs') +
    theme(plot.title = element_text(size = 18, face = 'bold'))
```

Enrichment map of down-regulated DEGs



7.3.2. GSEA analysis and visualization

Here again we take up-regulated DEGs to determine the GSEA result in the Biological Process category of Gene Ontology and also several visualization presentation. The visualization methods are interchangable with previous ORA section.

In [38]:
```
DEG_gsea_dn = gseGO(
    gene = geneList_dn, # fold change value required
    OrgDb = org.Hs.eg.db,
    keyType="SYMBOL",
    ont = "BP",
    pAdjustMethod = "BH")
```

```
preparing geneSet collections...

GSEA analysis...

Warning message in fgseaMultilevel(...):
“There were 3 pathways for which P-values were not calculated properly due to unbalanced (positive and negative) g
ene-level statistic values. For such pathways pval, padj, NES, log2err are set to NA. You can try to increase the
value of the argument nPermSimple (for example set it nPermSimple = 10000)”
leading edge analysis...

done...
```

In [39]:
```
DEG_gsea_dn
```

```
#
# Gene Set Enrichment Analysis
#
#...@organism    Homo sapiens
#...@setType     BP
#...@keytype     SYMBOL
#...@geneList    Named num [1:3081] -3.67 -3.99 -4.11 -4.13 -4.3 ...
 - attr(*, "names")= chr [1:3081] "DMRTC1B" "SEC1" "FAM7A3" "C22orf26" ...
#...nPerm
#...pvalues adjusted by 'BH' with cutoff <0.05
#...6 enriched terms found
'data.frame':   6 obs. of  11 variables:
 $ ID             : chr  "GO:0003008" "GO:0009888" "GO:0003012" "GO:0006936" ...
 $ Description    : chr  "system process" "tissue development" "muscle system process" "muscle contraction" ...
 $ setSize        : int  416 456 157 123 331 200
 $ enrichmentScore: num  -0.769 -0.76 -0.837 -0.861 -0.765 ...
 $ NES            : num  -1.29 -1.27 -1.4 -1.43 -1.28 ...
 $ pvalue         : num  1.25e-08 4.62e-08 6.00e-07 6.77e-07 4.94e-06 ...
 $ p.adjust       : num  3.62e-05 6.71e-05 4.92e-04 4.92e-04 2.87e-03 ...
 $ qvalues        : num  3.41e-05 6.31e-05 4.63e-04 4.63e-04 2.70e-03 ...
 $ rank           : num  345 464 130 130 412 432
 $ leading_edge   : chr  "tags=21%, list=11%, signal=21%" "tags=29%, list=15%, signal=29%" "tags=28%, list=4%, sig
nal=28%" "tags=22%, list=4%, signal=22%" ...
 $ core_enrichment: chr  "AKAP13/EFEMP1/KCNMB1/FOSL1/PLN/P2RX1/JAM3/CYB5R3/MAP1A/CXCL12/ITPR1/LDLR/MGLL/ADARB1/CD3
4/HSPB7/DMD/ATP1A2/KCNM"| __truncated__ "STAT5B/PDCD4/TGM1/DCHS1/FZD7/CNFN/ANO6/LAMA2/SVEP1/LTBP3/STC1/PDLIM5/OVOL
1/MYOCD/KDM6B/MEF2D/SBDS/DKK1/SPRY1/KL"| __truncated__ "DMD/ATP1A2/KCNMA1/ERRFI1/CASQ2/SRF/NR4A3/CALM1/DMPK/SORBS2
/CRYAB/CACNA1H/KLF4/RGS2/PTGS2/PI16/SLMAP/CAV1/ANXA6/"| __truncated__ "RGS2/PTGS2/SLMAP/CAV1/ANXA6/ATP2B4/VCL/MYL6
/SORBS1/SMTN/TLN1/HSPB6/PPP1R12B/LMOD1/ACTC1/CALD1/TPM2/MYLK/TPM1/CN"| __truncated__ ...
#...Citation
  Guangchuang Yu, Li-Gen Wang, Yanyan Han and Qing-Yu He.
  clusterProfiler: an R package for comparing biological themes among
  gene clusters. OMICS: A Journal of Integrative Biology
  2012, 16(5):284-287
```

concept map display

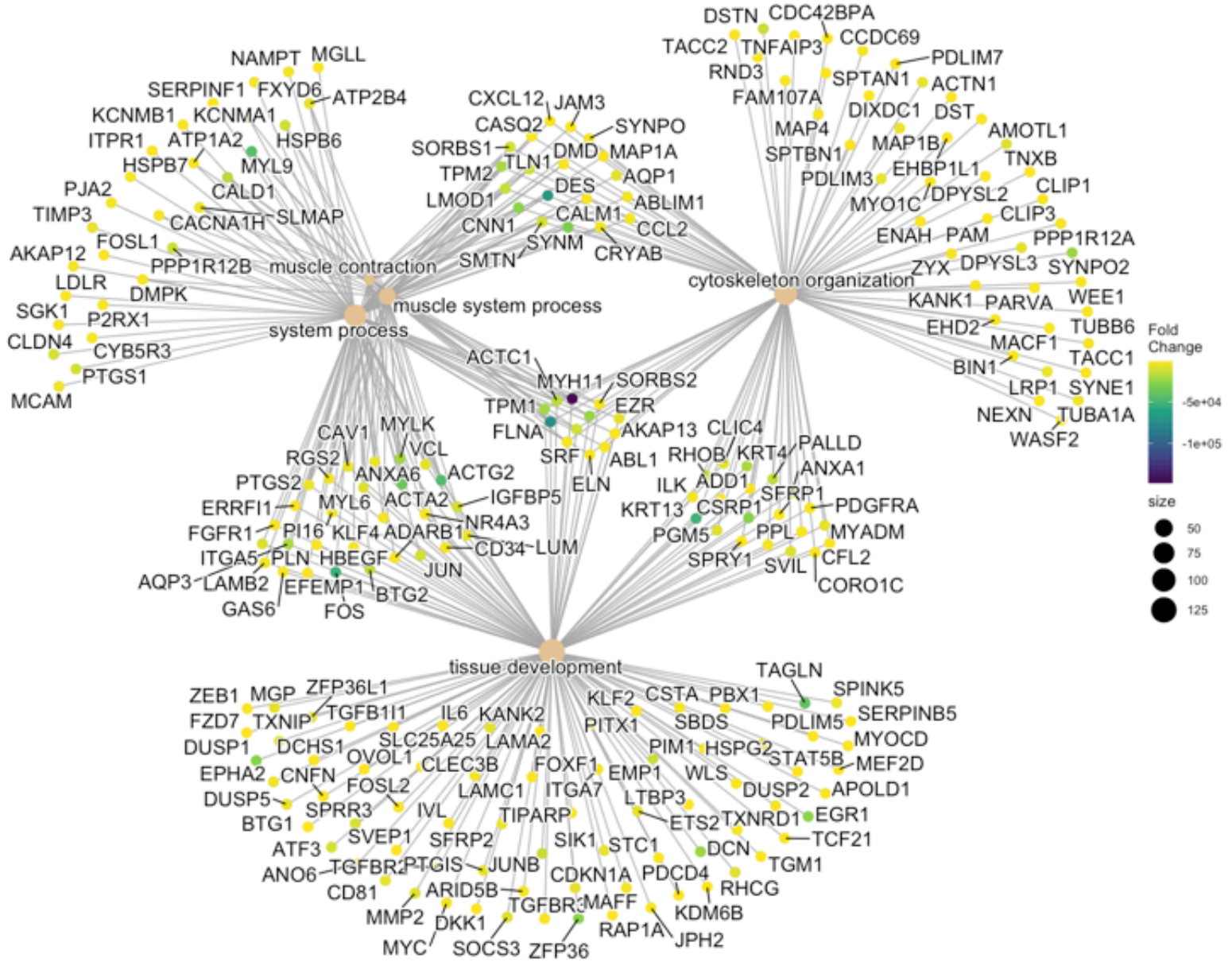
In [40]:

```
enrichplot::cnetplot(DEG_gsea_dn, foldChange = geneList_dn) +
    scale_color_viridis_c() +
    labs(color = 'Fold\nChange', title = 'Concept network of down-regulated DEGs') +
    theme_void() +
    theme(plot.title = element_text(face = 'bold', size = 24))
```

```
Scale for 'colour' is already present. Adding another scale for 'colour',
which will replace the existing scale.

Warning message:
“ggrepel: 114 unlabeled data points (too many overlaps). Consider increasing max.overlaps”
```

Concept network of down-regulated DEGs



single/multiple geneset visualization

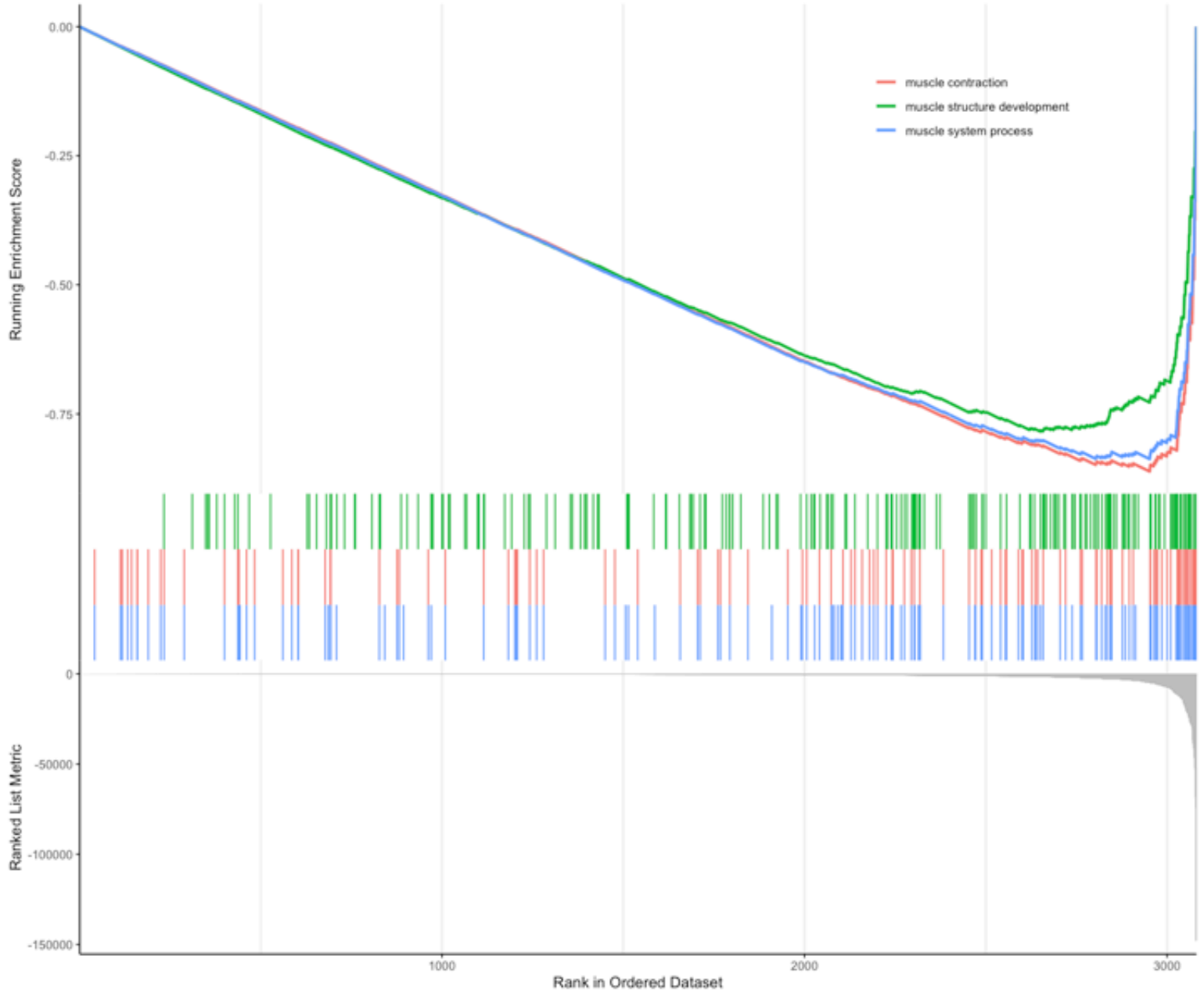
In [41]:
```
as_tibble(DEG_gsea_dn@result)
```

ID	Description	setSize	enrichmentScore	NES	pvalue	p.adjust	qvalues	rank	leading_edge	
<chr>	<chr>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>	
GO:0003008	system process	416	-0.7691471	-1.287183	1.246404e-08	3.622051e-05	3.407276e-05	345	tags=21%, list=11%, signal=21%	
GO:0009888	tissue development	456	-0.7595425	-1.271279	4.617575e-08	6.709337e-05	6.311496e-05	464	tags=29%, list=15%, signal=29%	STAT5B/PDCD4/T
GO:0003012	muscle system process	157	-0.8373947	-1.398320	6.002587e-07	4.917579e-04	4.625983e-04	130	tags=28%, list=4%, signal=28%	
GO:0006936	muscle contraction	123	-0.8614637	-1.434024	6.768862e-07	4.917579e-04	4.625983e-04	130	tags=22%, list=4%, signal=22%	
GO:0007010	cytoskeleton organization	331	-0.7653192	-1.280751	4.942689e-06	2.872691e-03	2.702350e-03	412	tags=28%, list=13%, signal=27%	
GO:0061061	muscle structure development	200	-0.7840662	-1.309561	5.303902e-05	2.568856e-02	2.416532e-02	432	tags=34%, list=14%, signal=32%	

In [42]:
```
term_of_interest = 'muscle'
idx = grep(DEG_gsea_dn@result$Description, pattern = term_of_interest, value = F, ignore.case = T)
print(idx)
```

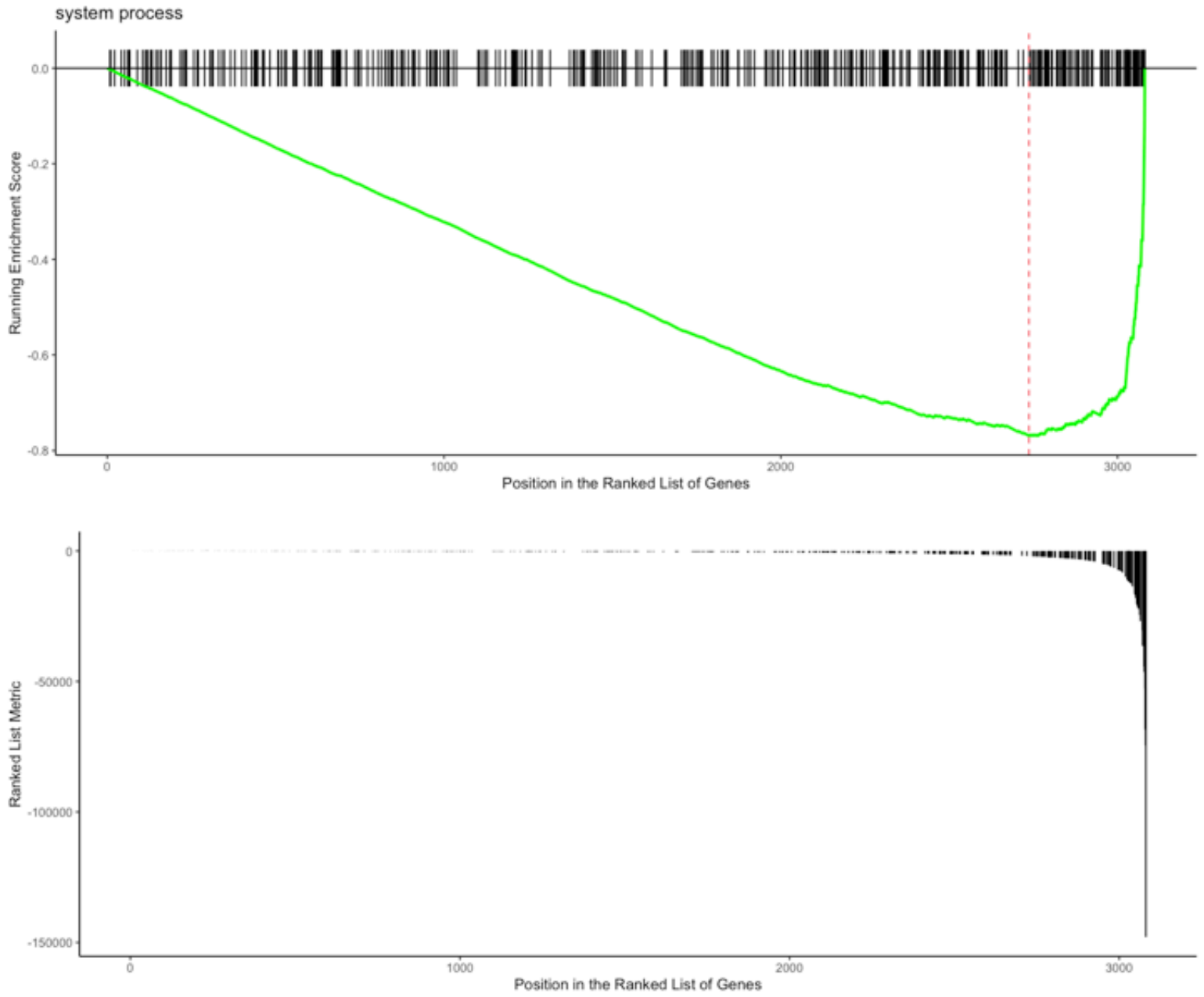
```
[1] 3 4 6
```

In [43]:
```
gseaplot2(DEG_gsea_dn, geneSetID = idx)
```


muscle contraction
muscle structure development
muscle system process
Running Enrichment Score
0.00
-0.25
-0.50
-0.75
0
Ranked List Metric
-50000
-100000
-150000
1000
2000
3000
Rank in Ordered Dataset

In [44]:

```
ggpubr::ggarrange(
    gseaplot(DEG_gsea_dn, geneSetID = 1, by = 'runningScore', title = DEG_gsea_dn@result$Description[1]) +
        theme_classic(),
    gseaplot(DEG_gsea_dn, geneSetID = 1, by = 'preranked') +
        theme_classic(),
    ncol =1, nrow = 2)
```



8. References

- Horst AM, Hill AP, Gorman KB (2020). palmerpenguins: Palmer Archipelago (Antarctica) penguin data. R package version 0.1.0. https://allisonhorst.github.io/palmerpenguins/.
- Wickham H, Averick M, Bryan J, Chang W, McGowan LD, François R, Grolemund G, Hayes A, Henry L, Hester J, Kuhn M, Pedersen TL, Miller E, Bache SM, Müller K, Ooms J, Robinson D, Seidel DP, Spinu V, Takahashi K, Vaughan D, Wilke C, Woo K, Yutani H (2019). "Welcome to the tidyverse." Journal of Open Source Software, 4(43), 1686. doi: 10.21105/joss.01686.
- Alboukadel Kassambara (2020). ggpubr: 'ggplot2' Based Publication Ready Plots. R package version 0.4.0. https://CRAN.R-project.org/package=ggpubr
- Wang et al., (2019). The UCSCXenaTools R package: a toolkit for accessing genomics data from UCSC Xena platform, from cancer multi-omics to single-cell RNA-seq. Journal of Open Source Software, 4(40), 1627, doi: 10.21105/joss.01627.
- Robinson MD, McCarthy DJ, Smyth GK (2010). "edgeR: a Bioconductor package for differential expression analysis of digital gene expression data." Bioinformatics, 26(1), 139-140. doi: 10.1093/bioinformatics/btp616.
- Yu G, Wang L, Han Y, He Q (2012). "clusterProfiler: an R package for comparing biological themes among gene clusters." OMICS: A Journal of Integrative Biology, 16(5), 284-287. doi: 10.1089/omi.2011.0118.
- Carlson M (2019). org.Hs.eg.db: Genome wide annotation for Human. R package version 3.8.2.
- Yu G (2022). enrichplot: Visualization of Functional Enrichment Result. R package version 1.14.2, https://yulab-smu.top/biomedical-knowledge-mining-book/.

In [ ]: