Session #2 - Exercises

Adrian C Lo and Ludovic Telley

12/4/2019

INSTRUCTIONS

There are 3 segments with around 10 questions each that increase in difficulty. Fill in the answer within the code chunk. When you wish to test the code chunk, press the *green play button* on the right side of the code chunk to see your output.

The hints will guide you what functions are required. You can access further information with *?x* where x is the function name, e.g. ?print().

Also check this page, specifically sections 8.3 and 10.2 for additional help.

Quick Reminder

For additional (free) learning, visit this webpage

DNF-UNIL Github page

Click here for the resources from the first DNF coding club session.

R is a console where you can type all the calculations you want and see the result

```r
1 + 1
```

```
## [1] 2
```

```r
10 / 5
```

```
## [1] 2
```

```r
2^5
```

```
## [1] 32
```

```r
1 * 5
```

```
## [1] 5
```

You can assign value or text to an object

```r
a <- 2
b <- 3

a + b
```

```
## [1] 5
```

```r
c <- "text"
c
```

```
## [1] "text"
```

You can call a function. E.g, to do a sum

```r
# ?sum

sum(1, 2, 3, 4, 5)
```

```
## [1] 15
```

If you need help about a function

```
# ?c()
```

Exercises

BASIC OPERATIONS

1. Create a vector x containing the elements 0,1,2,3,4,5,6. Create a vector y containing the elements 0,1,2,-3,4,-5,6,10,200,300.

HINT: c()

```
x <- c(0,1,2,3,4,5,6)
x
```

```
## [1] 0 1 2 3 4 5 6
```

```
# x is a "special" vector where there is logic in the sequence.
# This vector can also be constructed with seq(from = ..., to = ..., by = 1)
x <- seq(from = 0, to = 6, by = 1)
x
```

```
## [1] 0 1 2 3 4 5 6
```

```
y <- c(0,1,2,-3,4,-5,6,10,200,300)
y
```

```
##  [1]   0   1   2  -3   4  -5   6  10 200 300
```

2. Write an if-else statement to check the first element of x. If it is zero, output "This is good", else output "This is bad".

HINT: print() and logical operator ==

Also check here for extra help.

```
if( x[1] == 0) {
    print("This is good")
} else {
    print("This is bad")
}
```

```
## [1] "This is good"
```

3. Write a for-loop that iterates over each element of vector x using print()

Also check here for extra help.

```
# ii is a placeholder that goes through the sequence after "in"
for(ii in 1:length(x)) {
    print(x[ii])
}
```

```
## [1] 0
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
```

4. Write a for-loop that iterates over each element of vector x and print the cube of each value.

HINT: 2^3 = cube of 2 = 8

```
for(ii in 1:length(x)) {
    print(x[ii]^3)
}
```

```
## [1] 0
## [1] 1
## [1] 8
## [1] 27
## [1] 64
## [1] 125
## [1] 216
```

5. Write a for-loop that makes a right angle triangle pattern where each row is a number x repeating itself x times.

HINT: rep()

The pattern is:

1

22

333

4444

```
for(ii in 1:4) {
    print(rep(ii, times = ii))
}
```

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

6. Write a for-loop that iterates over the inbuilt iris dataset that evaluates the number of characters in the column name. The output should be like:

Sepal.Length (12)

Sepal.Width (11)

Petal.Length (12)

Petal.Width (11)

Species (7)

HINT: paste0() and nchar()

```
for(ii in 1:ncol(iris)) {
    print(paste0(colnames(iris)[ii], " (", nchar(colnames(iris)[ii]), ")"))
}
```

```
## [1] "Sepal.Length (12)"
## [1] "Sepal.Width (11)"
## [1] "Petal.Length (12)"
## [1] "Petal.Width (11)"
## [1] "Species (7)"
```

```r
# for code readability, you could create intermediate objects
column <- colnames(iris)

for(ii in 1:ncol(iris)) {
    phrase <- paste0(column[ii], " (", nchar(column[ii]), ")")
    print(phrase)
}
```

```
## [1] "Sepal.Length (12)"
## [1] "Sepal.Width (11)"
## [1] "Petal.Length (12)"
## [1] "Petal.Width (11)"
## [1] "Species (7)"
```

7. Write a for-loop that iterates over the inbuilt iris dataset ad outputs for each flower/row as output:

Flower **1** is a **setosa** and has a Sepal.Length of **5.1**

```r
for(ii in 1:nrow(iris)) {
    print(paste("Flower", ii, "is a", iris$Species[ii], "and has a Sepal.Length of", iris$Sepal.Length[
}
```

```
## [1] "Flower 1 is a setosa and has a Sepal.Length of 5.1"
## [1] "Flower 2 is a setosa and has a Sepal.Length of 4.9"
## [1] "Flower 3 is a setosa and has a Sepal.Length of 4.7"
## [1] "Flower 4 is a setosa and has a Sepal.Length of 4.6"
## [1] "Flower 5 is a setosa and has a Sepal.Length of 5"
## [1] "Flower 6 is a setosa and has a Sepal.Length of 5.4"
## [1] "Flower 7 is a setosa and has a Sepal.Length of 4.6"
## [1] "Flower 8 is a setosa and has a Sepal.Length of 5"
## [1] "Flower 9 is a setosa and has a Sepal.Length of 4.4"
## [1] "Flower 10 is a setosa and has a Sepal.Length of 4.9"
## [1] "Flower 11 is a setosa and has a Sepal.Length of 5.4"
## [1] "Flower 12 is a setosa and has a Sepal.Length of 4.8"
## [1] "Flower 13 is a setosa and has a Sepal.Length of 4.8"
## [1] "Flower 14 is a setosa and has a Sepal.Length of 4.3"
## [1] "Flower 15 is a setosa and has a Sepal.Length of 5.8"
## [1] "Flower 16 is a setosa and has a Sepal.Length of 5.7"
## [1] "Flower 17 is a setosa and has a Sepal.Length of 5.4"
## [1] "Flower 18 is a setosa and has a Sepal.Length of 5.1"
## [1] "Flower 19 is a setosa and has a Sepal.Length of 5.7"
## [1] "Flower 20 is a setosa and has a Sepal.Length of 5.1"
## [1] "Flower 21 is a setosa and has a Sepal.Length of 5.4"
## [1] "Flower 22 is a setosa and has a Sepal.Length of 5.1"
## [1] "Flower 23 is a setosa and has a Sepal.Length of 4.6"
## [1] "Flower 24 is a setosa and has a Sepal.Length of 5.1"
## [1] "Flower 25 is a setosa and has a Sepal.Length of 4.8"
## [1] "Flower 26 is a setosa and has a Sepal.Length of 5"
## [1] "Flower 27 is a setosa and has a Sepal.Length of 5"
## [1] "Flower 28 is a setosa and has a Sepal.Length of 5.2"
## [1] "Flower 29 is a setosa and has a Sepal.Length of 5.2"
## [1] "Flower 30 is a setosa and has a Sepal.Length of 4.7"
## [1] "Flower 31 is a setosa and has a Sepal.Length of 4.8"
## [1] "Flower 32 is a setosa and has a Sepal.Length of 5.4"
## [1] "Flower 33 is a setosa and has a Sepal.Length of 5.2"
## [1] "Flower 34 is a setosa and has a Sepal.Length of 5.5"
## [1] "Flower 35 is a setosa and has a Sepal.Length of 4.9"
## [1] "Flower 36 is a setosa and has a Sepal.Length of 5"
## [1] "Flower 37 is a setosa and has a Sepal.Length of 5.5"
## [1] "Flower 38 is a setosa and has a Sepal.Length of 4.9"
## [1] "Flower 39 is a setosa and has a Sepal.Length of 4.4"
## [1] "Flower 40 is a setosa and has a Sepal.Length of 5.1"
## [1] "Flower 41 is a setosa and has a Sepal.Length of 5"
## [1] "Flower 42 is a setosa and has a Sepal.Length of 4.5"
## [1] "Flower 43 is a setosa and has a Sepal.Length of 4.4"
## [1] "Flower 44 is a setosa and has a Sepal.Length of 5"
## [1] "Flower 45 is a setosa and has a Sepal.Length of 5.1"
## [1] "Flower 46 is a setosa and has a Sepal.Length of 4.8"
## [1] "Flower 47 is a setosa and has a Sepal.Length of 5.1"
## [1] "Flower 48 is a setosa and has a Sepal.Length of 4.6"
## [1] "Flower 49 is a setosa and has a Sepal.Length of 5.3"
## [1] "Flower 50 is a setosa and has a Sepal.Length of 5"
## [1] "Flower 51 is a versicolor and has a Sepal.Length of 7"
## [1] "Flower 52 is a versicolor and has a Sepal.Length of 6.4"
## [1] "Flower 53 is a versicolor and has a Sepal.Length of 6.9"
## [1] "Flower 54 is a versicolor and has a Sepal.Length of 5.5"
## [1] "Flower 55 is a versicolor and has a Sepal.Length of 6.5"
## [1] "Flower 56 is a versicolor and has a Sepal.Length of 5.7"
## [1] "Flower 57 is a versicolor and has a Sepal.Length of 6.3"
## [1] "Flower 58 is a versicolor and has a Sepal.Length of 4.9"
## [1] "Flower 59 is a versicolor and has a Sepal.Length of 6.6"
## [1] "Flower 60 is a versicolor and has a Sepal.Length of 5.2"
## [1] "Flower 61 is a versicolor and has a Sepal.Length of 5"
## [1] "Flower 62 is a versicolor and has a Sepal.Length of 5.9"
## [1] "Flower 63 is a versicolor and has a Sepal.Length of 6"
## [1] "Flower 64 is a versicolor and has a Sepal.Length of 6.1"
## [1] "Flower 65 is a versicolor and has a Sepal.Length of 5.6"
## [1] "Flower 66 is a versicolor and has a Sepal.Length of 6.7"
## [1] "Flower 67 is a versicolor and has a Sepal.Length of 5.6"
## [1] "Flower 68 is a versicolor and has a Sepal.Length of 5.8"
## [1] "Flower 69 is a versicolor and has a Sepal.Length of 6.2"
## [1] "Flower 70 is a versicolor and has a Sepal.Length of 5.6"
## [1] "Flower 71 is a versicolor and has a Sepal.Length of 5.9"
## [1] "Flower 72 is a versicolor and has a Sepal.Length of 6.1"
## [1] "Flower 73 is a versicolor and has a Sepal.Length of 6.3"
## [1] "Flower 74 is a versicolor and has a Sepal.Length of 6.1"
## [1] "Flower 75 is a versicolor and has a Sepal.Length of 6.4"
## [1] "Flower 76 is a versicolor and has a Sepal.Length of 6.6"
## [1] "Flower 77 is a versicolor and has a Sepal.Length of 6.8"
## [1] "Flower 78 is a versicolor and has a Sepal.Length of 6.7"
## [1] "Flower 79 is a versicolor and has a Sepal.Length of 6"
## [1] "Flower 80 is a versicolor and has a Sepal.Length of 5.7"
## [1] "Flower 81 is a versicolor and has a Sepal.Length of 5.5"
## [1] "Flower 82 is a versicolor and has a Sepal.Length of 5.5"
## [1] "Flower 83 is a versicolor and has a Sepal.Length of 5.8"
## [1] "Flower 84 is a versicolor and has a Sepal.Length of 6"
## [1] "Flower 85 is a versicolor and has a Sepal.Length of 5.4"
## [1] "Flower 86 is a versicolor and has a Sepal.Length of 6"
## [1] "Flower 87 is a versicolor and has a Sepal.Length of 6.7"
## [1] "Flower 88 is a versicolor and has a Sepal.Length of 6.3"
## [1] "Flower 89 is a versicolor and has a Sepal.Length of 5.6"
## [1] "Flower 90 is a versicolor and has a Sepal.Length of 5.5"
## [1] "Flower 91 is a versicolor and has a Sepal.Length of 5.5"
## [1] "Flower 92 is a versicolor and has a Sepal.Length of 6.1"
## [1] "Flower 93 is a versicolor and has a Sepal.Length of 5.8"
## [1] "Flower 94 is a versicolor and has a Sepal.Length of 5"
## [1] "Flower 95 is a versicolor and has a Sepal.Length of 5.6"
## [1] "Flower 96 is a versicolor and has a Sepal.Length of 5.7"
## [1] "Flower 97 is a versicolor and has a Sepal.Length of 5.7"
## [1] "Flower 98 is a versicolor and has a Sepal.Length of 6.2"
## [1] "Flower 99 is a versicolor and has a Sepal.Length of 5.1"
## [1] "Flower 100 is a versicolor and has a Sepal.Length of 5.7"
## [1] "Flower 101 is a virginica and has a Sepal.Length of 6.3"
## [1] "Flower 102 is a virginica and has a Sepal.Length of 5.8"
## [1] "Flower 103 is a virginica and has a Sepal.Length of 7.1"
## [1] "Flower 104 is a virginica and has a Sepal.Length of 6.3"
## [1] "Flower 105 is a virginica and has a Sepal.Length of 6.5"
## [1] "Flower 106 is a virginica and has a Sepal.Length of 7.6"
## [1] "Flower 107 is a virginica and has a Sepal.Length of 4.9"
## [1] "Flower 108 is a virginica and has a Sepal.Length of 7.3"
## [1] "Flower 109 is a virginica and has a Sepal.Length of 6.7"
## [1] "Flower 110 is a virginica and has a Sepal.Length of 7.2"
## [1] "Flower 111 is a virginica and has a Sepal.Length of 6.5"
## [1] "Flower 112 is a virginica and has a Sepal.Length of 6.4"
## [1] "Flower 113 is a virginica and has a Sepal.Length of 6.8"
## [1] "Flower 114 is a virginica and has a Sepal.Length of 5.7"
## [1] "Flower 115 is a virginica and has a Sepal.Length of 5.8"
## [1] "Flower 116 is a virginica and has a Sepal.Length of 6.4"
## [1] "Flower 117 is a virginica and has a Sepal.Length of 6.5"
## [1] "Flower 118 is a virginica and has a Sepal.Length of 7.7"
## [1] "Flower 119 is a virginica and has a Sepal.Length of 7.7"
## [1] "Flower 120 is a virginica and has a Sepal.Length of 6"
## [1] "Flower 121 is a virginica and has a Sepal.Length of 6.9"
## [1] "Flower 122 is a virginica and has a Sepal.Length of 5.6"
## [1] "Flower 123 is a virginica and has a Sepal.Length of 7.7"
## [1] "Flower 124 is a virginica and has a Sepal.Length of 6.3"
## [1] "Flower 125 is a virginica and has a Sepal.Length of 6.7"
## [1] "Flower 126 is a virginica and has a Sepal.Length of 7.2"
## [1] "Flower 127 is a virginica and has a Sepal.Length of 6.2"
## [1] "Flower 128 is a virginica and has a Sepal.Length of 6.1"
## [1] "Flower 129 is a virginica and has a Sepal.Length of 6.4"
## [1] "Flower 130 is a virginica and has a Sepal.Length of 7.2"
## [1] "Flower 131 is a virginica and has a Sepal.Length of 7.4"
## [1] "Flower 132 is a virginica and has a Sepal.Length of 7.9"
## [1] "Flower 133 is a virginica and has a Sepal.Length of 6.4"
## [1] "Flower 134 is a virginica and has a Sepal.Length of 6.3"
## [1] "Flower 135 is a virginica and has a Sepal.Length of 6.1"
## [1] "Flower 136 is a virginica and has a Sepal.Length of 7.7"
## [1] "Flower 137 is a virginica and has a Sepal.Length of 6.3"
## [1] "Flower 138 is a virginica and has a Sepal.Length of 6.4"
## [1] "Flower 139 is a virginica and has a Sepal.Length of 6"
## [1] "Flower 140 is a virginica and has a Sepal.Length of 6.9"
## [1] "Flower 141 is a virginica and has a Sepal.Length of 6.7"
## [1] "Flower 142 is a virginica and has a Sepal.Length of 6.9"
## [1] "Flower 143 is a virginica and has a Sepal.Length of 5.8"
## [1] "Flower 144 is a virginica and has a Sepal.Length of 6.8"
## [1] "Flower 145 is a virginica and has a Sepal.Length of 6.7"
## [1] "Flower 146 is a virginica and has a Sepal.Length of 6.7"
## [1] "Flower 147 is a virginica and has a Sepal.Length of 6.3"
## [1] "Flower 148 is a virginica and has a Sepal.Length of 6.5"
## [1] "Flower 149 is a virginica and has a Sepal.Length of 6.2"
## [1] "Flower 150 is a virginica and has a Sepal.Length of 5.9"
```

8. Write a for-loop that iterates over each element of vector y and only outputs the positive numbers, i.e. ignoring the negative numbers.

HINT: next()

```r
for(ii in 1:length(y)) {
    if(y[ii] < 0) { next() }
    print(y[ii])
}
```

```
## [1] 0
## [1] 1
## [1] 2
## [1] 4
## [1] 6
## [1] 10
## [1] 200
## [1] 300
```

```r
# this can also be solved without next()
for(ii in 1:length(y)) {
    if(y[ii] >= 0) { print(y[ii]) }
}
```

```
## [1] 0
## [1] 1
## [1] 2
## [1] 4
## [1] 6
## [1] 10
## [1] 200
## [1] 300
```

9. Write a for-loop that iterates over each element of vector y, till it reaches value 10. If it reaches value 10, stop the for-loop.

HINT: break()

```r
for(ii in 1:length(y)) {
    if(y[ii] == 10) { break() }
    print(y[ii])
}
```

```
## [1] 0
## [1] 1
## [1] 2
## [1] -3
## [1] 4
## [1] -5
## [1] 6
```

10. Use the inbuilt rivers dataset and write a for-loop that checks each value. If the value is lower than 500, output "short river". If the river is more than 2000, output "long river". Everything else, output the original value.

```r
for(ii in 1:length(rivers)) {
    if(rivers[ii] < 500) {
        print("short river")
    } else if(rivers[ii] > 2000) {
        print("long river")
    } else {
        print(rivers[ii])
    }
}
```

```
## [1] 735
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] 524
## [1] "short river"
## [1] 1459
## [1] "short river"
## [1] "short river"
## [1] 600
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] 870
## [1] 906
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] 1000
## [1] 600
## [1] 505
## [1] 1450
## [1] 840
## [1] 1243
## [1] 890
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] 525
## [1] 720
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] 850
## [1] "short river"
## [1] 630
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] 730
## [1] 600
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] 710
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] 680
## [1] 570
## [1] "short river"
## [1] "short river"
## [1] 560
## [1] 900
## [1] 625
## [1] "short river"
## [1] "long river"
## [1] 1171
## [1] "long river"
## [1] "long river"
## [1] "long river"
## [1] 780
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] 760
## [1] 618
## [1] "short river"
## [1] 981
## [1] 1306
## [1] 500
## [1] 696
## [1] 605
## [1] "short river"
## [1] "short river"
## [1] 1054
## [1] 735
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] 1270
## [1] 545
## [1] "short river"
## [1] 1885
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] 800
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] 538
## [1] 1100
## [1] 1205
## [1] "short river"
## [1] "short river"
## [1] 610
## [1] "short river"
## [1] 540
## [1] 1038
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] "short river"
## [1] 620
## [1] "short river"
## [1] 652
## [1] 900
## [1] 525
## [1] "short river"
## [1] "short river"
## [1] 529
## [1] 500
## [1] 720
## [1] "short river"
## [1] "short river"
## [1] 671
## [1] 1770
```

READR and DPLYR

1. Import ggplot2, readr and dplyr packages/libraries

```r
library(ggplot2)
library(readr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

2. Import dataset tidy.csv and store it in an object called "mydata"

HINT: read_csv()

```r
mydata <- read_csv("tidy.csv")
```

```
## Parsed with column specification:
## cols(
##   id = col_double(),
##   genotype = col_character(),
##   treatment = col_character(),
##   gender = col_character(),
##   var_x = col_double(),
##   var_y = col_double(),
##   var_z = col_double(),
##   norm = col_double()
## )
```

```r
mydata
```

```
## # A tibble: 240 x 8
##       id genotype treatment gender var_x var_y var_z  norm
##    <dbl> <chr>    <chr>     <chr>  <dbl> <dbl> <dbl> <dbl>
##  1     1 WT       CTR_X     M       98.6  97.6 104.      1
##  2     2 WT       CTR_X     M      105.   96.6 105.      1
##  3     3 WT       CTR_X     M       92.5 110.  113.      1
##  4     4 WT       CTR_X     M       89.2  99.2  93.5     1
##  5     5 WT       CTR_X     M       96.4 108.  112.      1
##  6     6 WT       CTR_X     M      109.   77.1  91.5     1
##  7     7 WT       CTR_X     M      108.  109.   83.5     1
##  8     8 WT       CTR_X     M       96.4 112.  103.      1
##  9     9 WT       CTR_X     M       94.3 107.  108.      1
## 10    10 WT       CTR_X     M       96.4  82.7 101.      1
## # ... with 230 more rows
```

3. Check out mydata, how many rows and columns are there?

```r
nrow(mydata)
```

```
## [1] 240
```

```r
ncol(mydata)
```

```
## [1] 8
```

```
# to have both nrow() and ncol() in one call
dim(mydata)
```

```
## [1] 240   8
```

4. Genotype, treatment and gender should be categorical variables with some order (instead of the default alphabetical order). Transform these into order "WT","GM" for genotype, "CTR_X","TRT_X","TRT_Y","TRT_Z" for treatment and "M","F" for gender. Store it again in object "mydata".

HINT: mutate() and factor(. . . , levels = c(. . .)).

Example: factor(class, levels = c("level_1","level_2","level_3"))

```
mydata <- mutate(mydata,
                 genotype = factor(genotype, levels = c("WT","GM")),
                 treatment = factor(treatment, levels = c("CTR_X","TRT_X","TRT_Y","TRT_Z")),
                 gender = factor(gender, levels = c("M","F")))
```

5. Continue from mydata from question 4. Variables are unnormalized. Create three new columns with the variables multiplied by the norm column. Name the new columns var_x_norm, var_y_norm and var_z_norm. Store it again in object "mydata".

HINT: mutate()

```
mydata <- mutate(mydata,
                 var_x_norm = var_x * norm,
                 var_y_norm = var_y * norm,
                 var_z_norm = var_z * norm)
```

6. Continue from mydata. Filter values that only contain male (gender == "M") samples, store this in an object called "mymales". Continue working with mymales untill question 10.

HINT: filter()

```
mymales <- filter(mydata,
                  gender == "M")
```

7. Continue from mymales. The summarise() function is very useful for collapsing a large dataset into a single observation. Summarise the mymales dataset to find the following columns / values: min_value_x_norm (minimum value for var_x_norm) and mean_value_x_norm (mean value for value_x_norm).

HINT: min() and mean()

```
summarise(mymales,
          min_value_x_norm = min(var_x_norm),
          mean_value_x_norm = mean(var_x_norm))
```

```
## # A tibble: 1 x 2
##   min_value_x_norm mean_value_x_norm
##              <dbl>             <dbl>
## 1             86.4              142.
```

8. In combination with group_by() function, you can summarise the value of interest by each group. For example, grouping according to genotype, you can find the values of question 7 for each genotype.

HINT: %>% makes the line of execution more readable.

Example 1:

function2(function1(x,y))

is more readable as (and equivalent to)

x %>% function1(y) %>% function2()

Example 2:

data %>% group_by(x) %>% summarise(. . .)

is equivalent to (but more readable than)

summarise(group_by(data, x), . . .)

```
mymales %>%
    group_by(genotype) %>%
    summarise(
        min_value_x_norm = min(var_x_norm),
        mean_value_x_norm = mean(var_x_norm))
```

```
## # A tibble: 2 x 3
##   genotype min_value_x_norm mean_value_x_norm
##   <fct>               <dbl>             <dbl>
## 1 WT                   88.2              151.
## 2 GM                   86.4              132.
```

9. Grouping can be performed on multiple levels. Try to find values of question 7 when grouping for genotype and then treatment. Summaries can also be stored in an object. Store this in an object called “mysummary”

```
mysummary <-
    mymales %>%
    group_by(genotype) %>%
    summarise(
        min_value_x_norm = min(var_x_norm),
        mean_value_x_norm = mean(var_x_norm)
    )
```

10. Export your result with write_csv(). Call it “myfirstroutput.csv”

```
write_csv(mysummary, path = "myfirstroutput.csv")
```

11. So far you have checked the values of question 7 for only “var_x_norm”. Repeat this for each variable (“var_x_norm”,“var_y_norm”,“var_z_norm”) the values of question 17 when grouping for genotype and then treatment.

```
mymales %>%
    group_by(genotype) %>%
    summarise(
        min_value_x_norm = min(var_x_norm),
        mean_value_x_norm = mean(var_x_norm)
    )
```

```
## # A tibble: 2 x 3
##   genotype min_value_x_norm mean_value_x_norm
##   <fct>               <dbl>             <dbl>
## 1 WT                   88.2              151.
## 2 GM                   86.4              132.
```

```
mymales %>%
    group_by(genotype) %>%
    summarise(
        min_value_y_norm = min(var_y_norm),
        mean_value_y_norm = mean(var_y_norm)
    )
```

```
## # A tibble: 2 x 3
##   genotype min_value_y_norm mean_value_y_norm
##   <fct>               <dbl>             <dbl>
## 1 WT                   77.1              135.
## 2 GM                   77.2              129.
```

```
mymales %>%
    group_by(genotype) %>%
    summarise(
        min_value_z_norm = min(var_z_norm),
        mean_value_z_norm = mean(var_z_norm)
    )
```

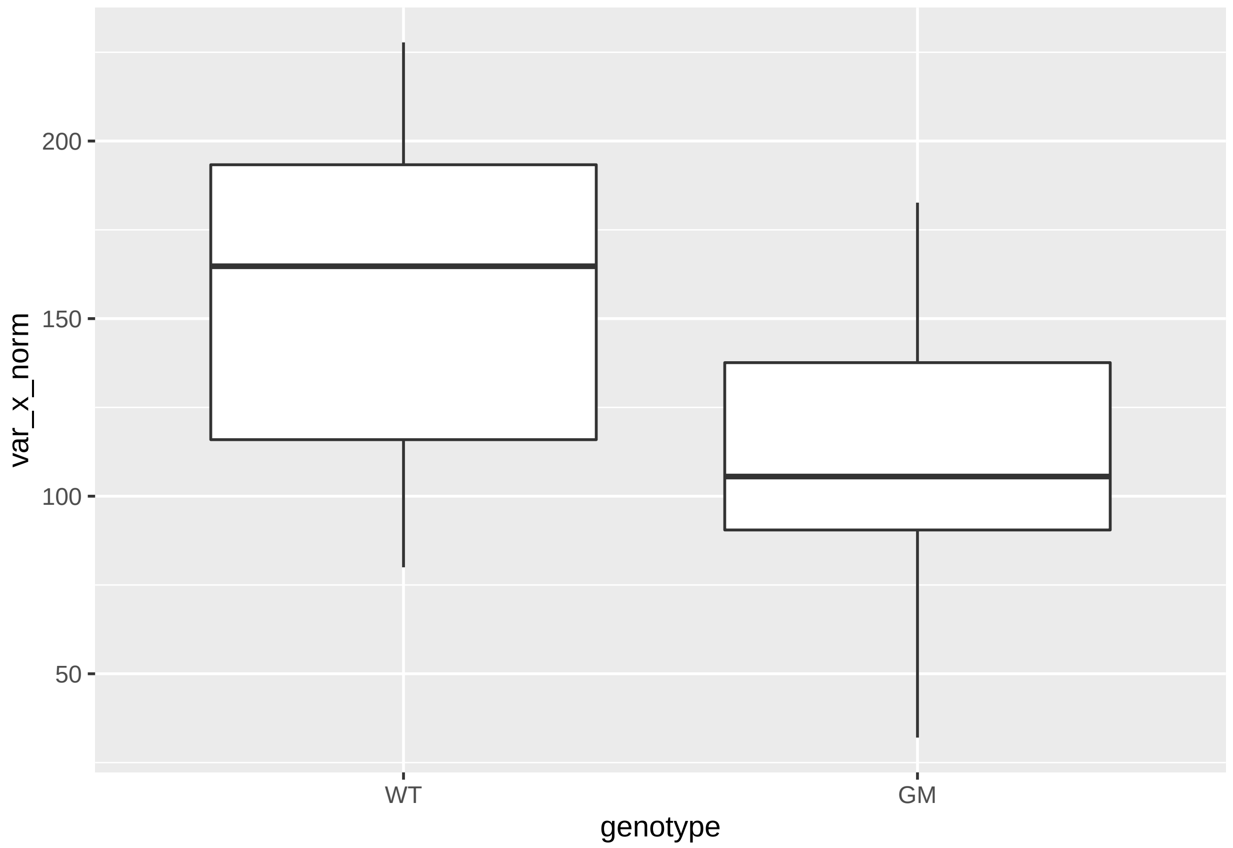
```
## # A tibble: 2 x 3
##   genotype min_value_z_norm mean_value_z_norm
##   <fct>               <dbl>             <dbl>
## 1 WT                   83.5              116.
## 2 GM                   77.7              116.
```

GGPLOT2

1. Work with mydata_t from the previous segment. Create a boxplot with on the y-axis var_x_norm and on the x-axis genotype. How does your plot look like? What can you conclude from this?

HINT: ggplot(data, aes(x = x, y = y) + geom_boxplot()

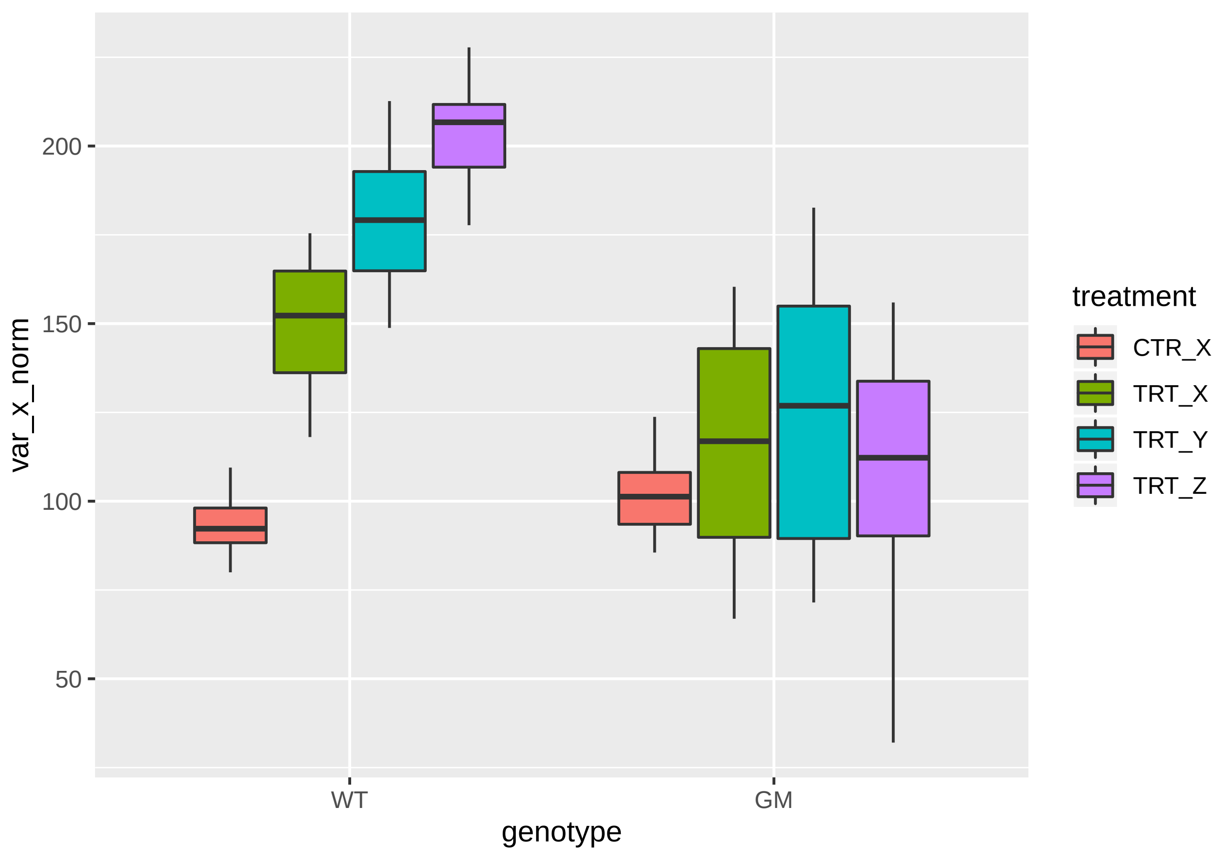
```
ggplot(mydata, aes(x = genotype, y = var_x_norm)) + geom_boxplot()
```



2. The previous graph takes a general view of var_x_norm according to genotype. Add within aesthetics (aes) that treatment is defined by fill.

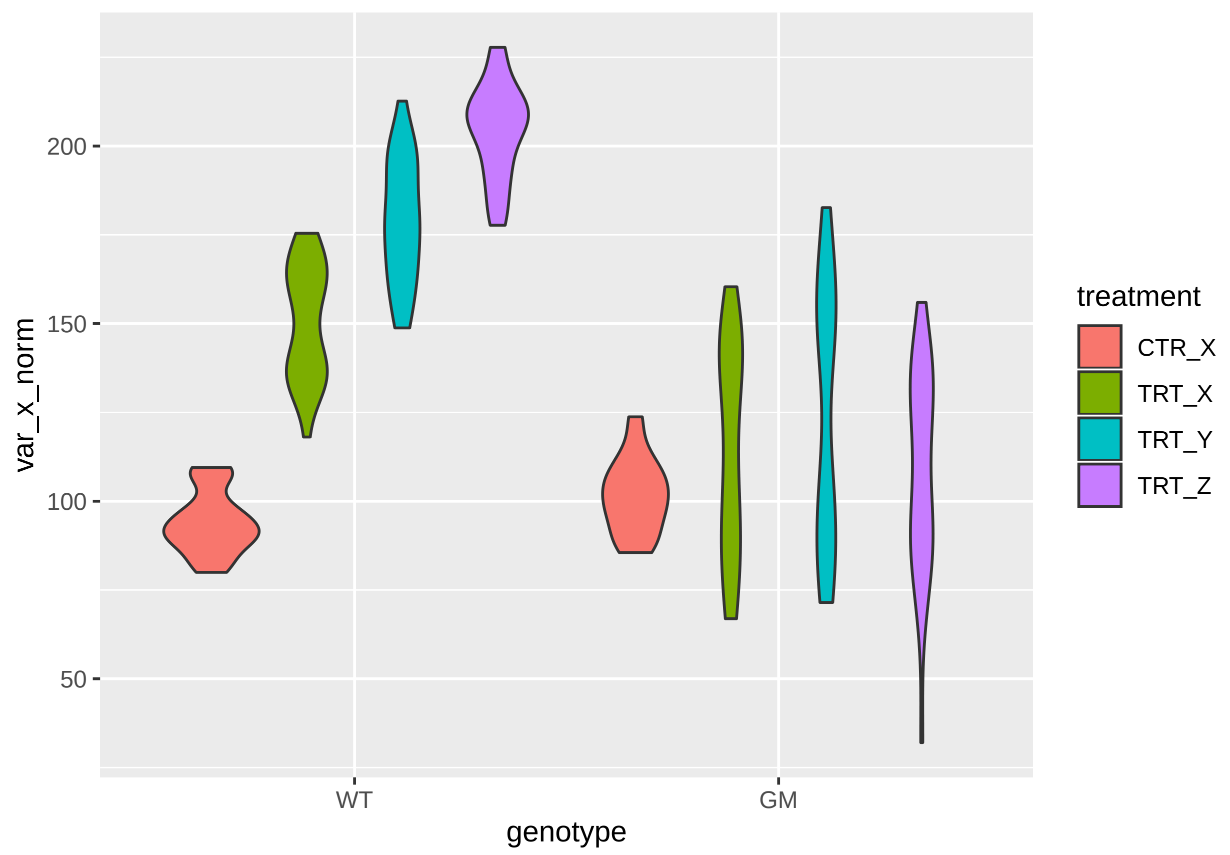
HINT: fill = . . .

```
ggplot(mydata, aes(x = genotype, y = var_x_norm, fill = treatment)) + geom_boxplot()
```



3. Re-use the code from question 2 and adjust it slightly so you have violinplots instead of boxplots.

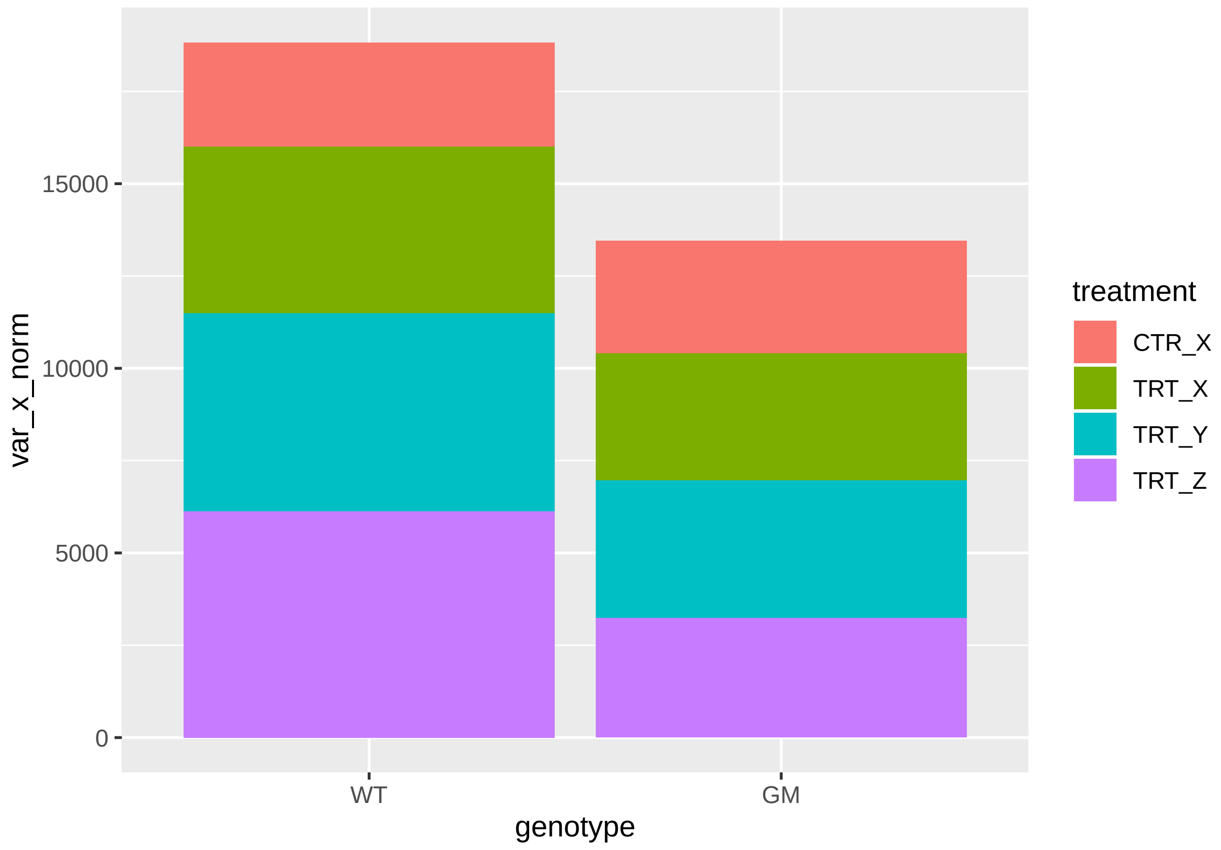
```
ggplot(mydata, aes(x = genotype, y = var_x_norm, fill = treatment)) + geom_violin()
```



4. Re-use the code from question 2 and adjust it slightly so you have stacked bars instead of boxplots.

HINT: geom_col()

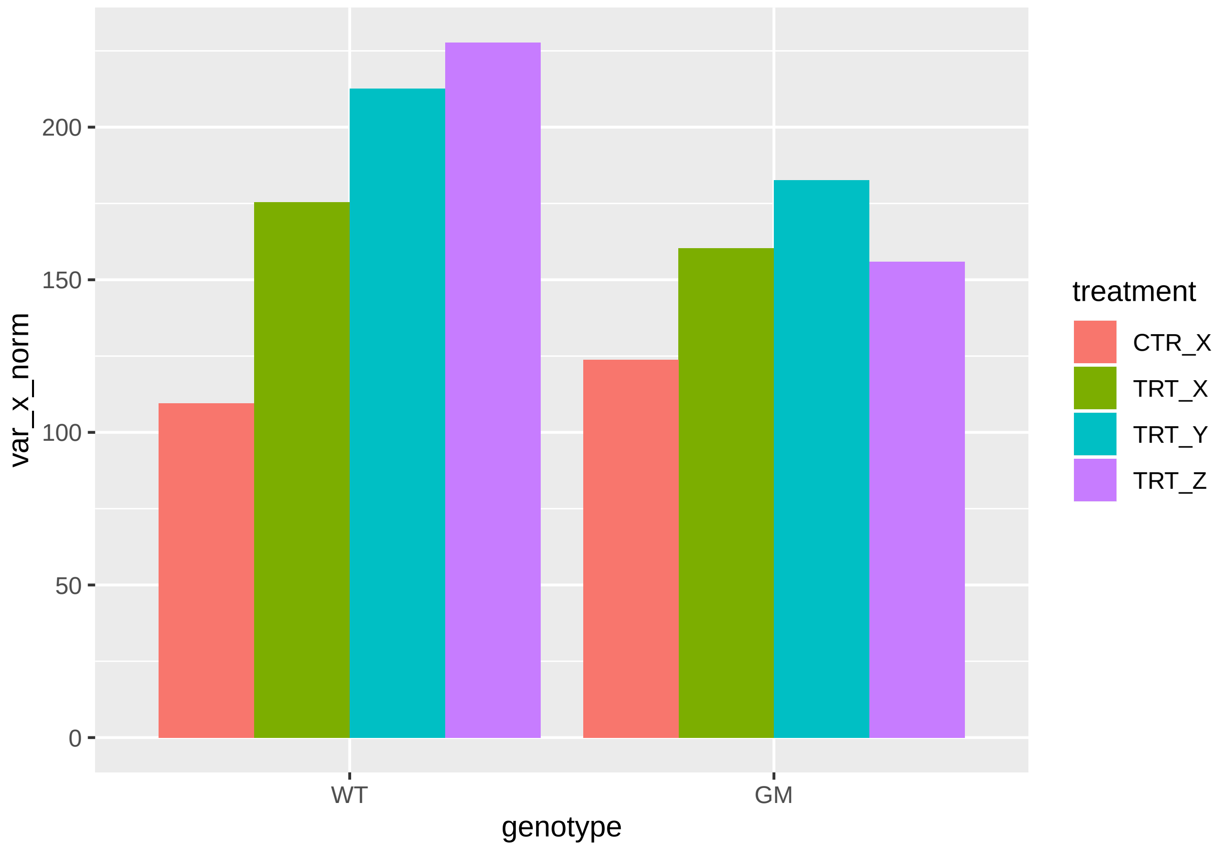
```
ggplot(mydata, aes(x = genotype, y = var_x_norm, fill = treatment)) + geom_col()
```



5. Re-use the code from question 4 and adjust it slightly to have side-by-side bars instead of stacked bars.

HINT: position = “dodge” inside geom_col()

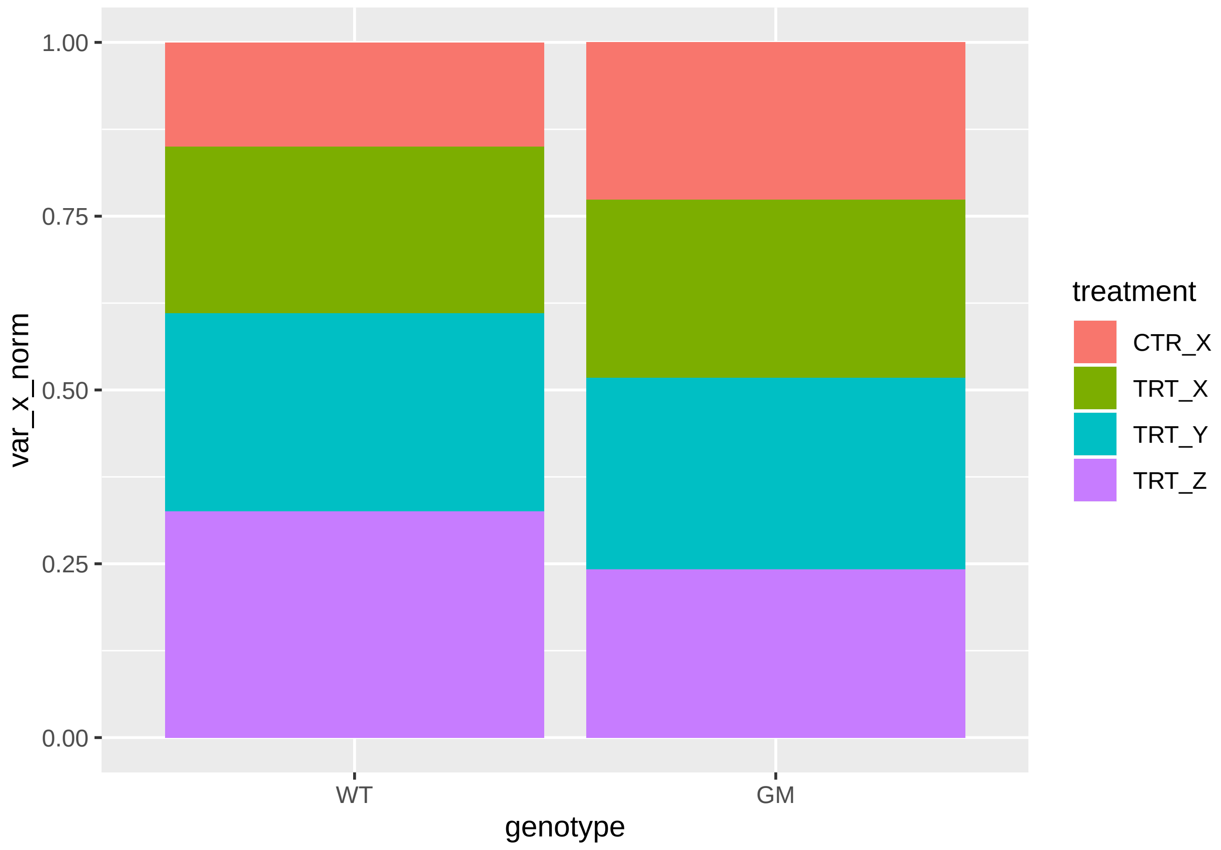
```
ggplot(mydata, aes(x = genotype, y = var_x_norm, fill = treatment)) + geom_col(position = "dodge")
```



6. Re-use the code from question 4 and adjust it slightly to have stacked bars of equal height, instead of stacked bars with different height.

HINT: position = "fill"

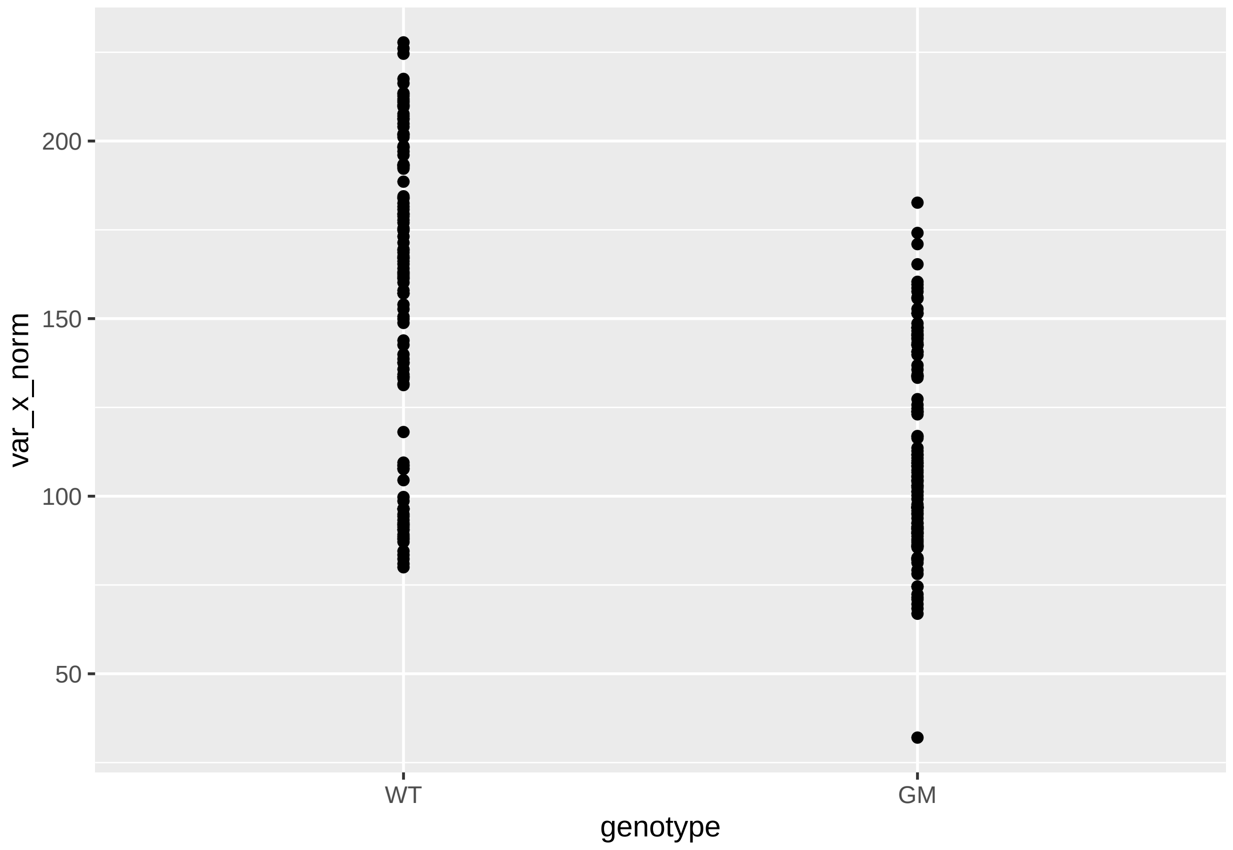
```
ggplot(mydata, aes(x = genotype, y = var_x_norm, fill = treatment)) + geom_col(position = "fill")
```



7. Create a scatter plot with on the y-axis var_x_norm and on the x-axis genotype. How does your plot look like?

HINT: geom_point()

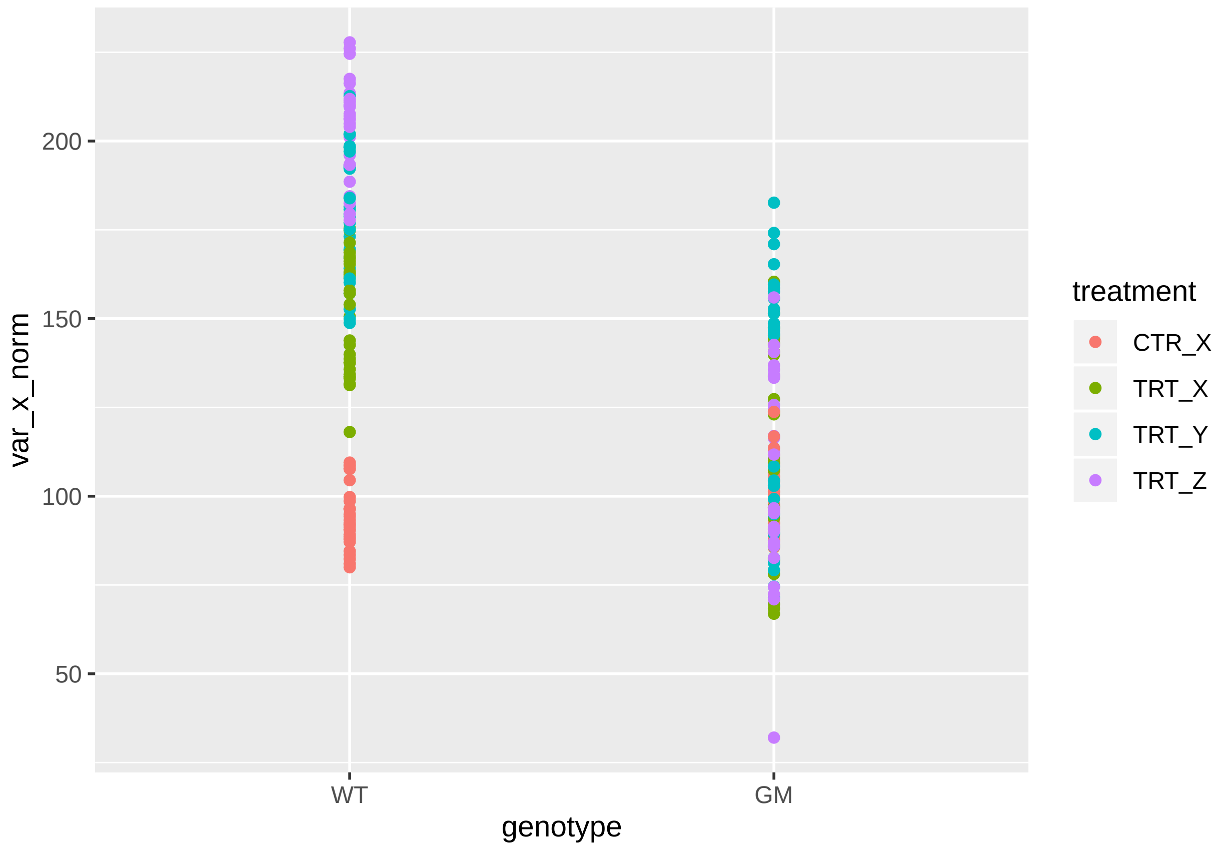
```
ggplot(mydata, aes(genotype, var_x_norm)) + geom_point()
```



8. All points are only segmented according to genotype, but no difference is made between treatments. Add within aesthetics (aes) that treatment is defined by color.

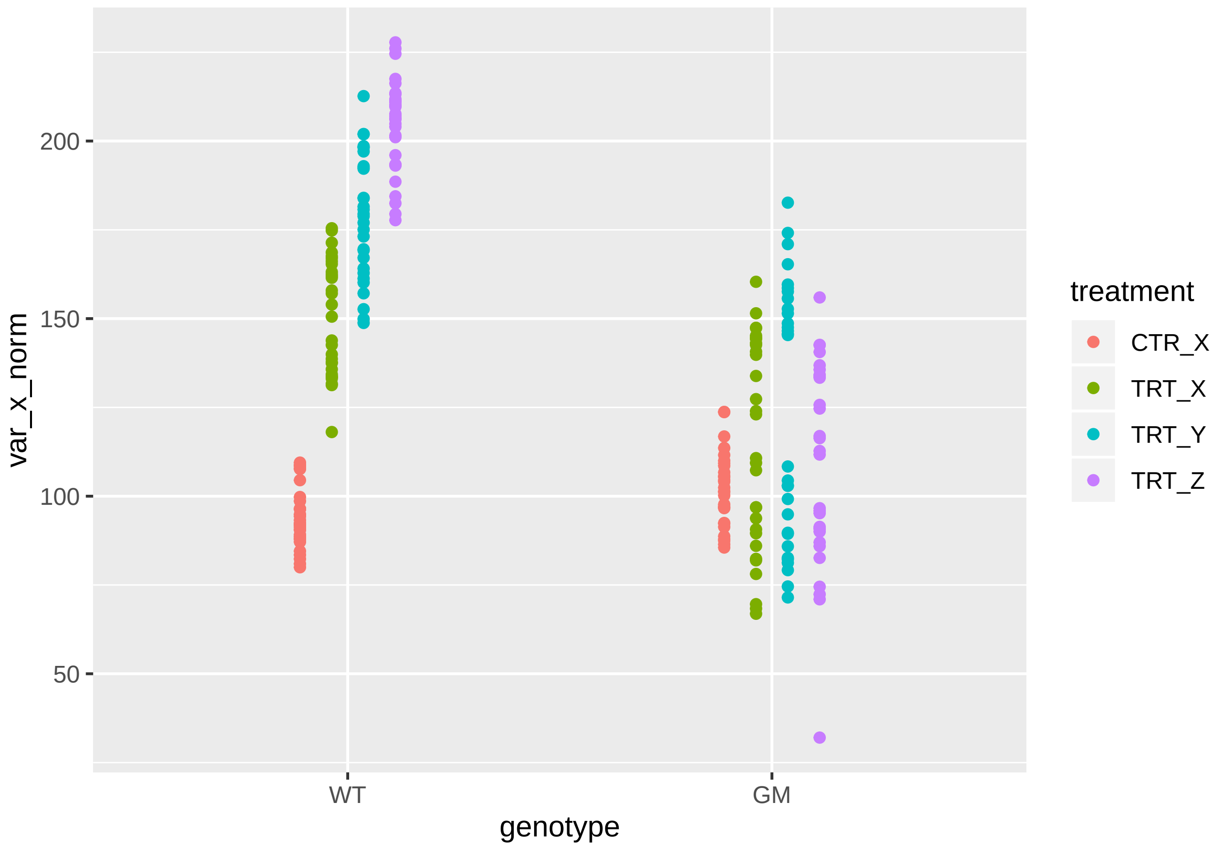
HINT: color = . . .

```
ggplot(mydata, aes(genotype, var_x_norm, color = treatment)) + geom_point()
```



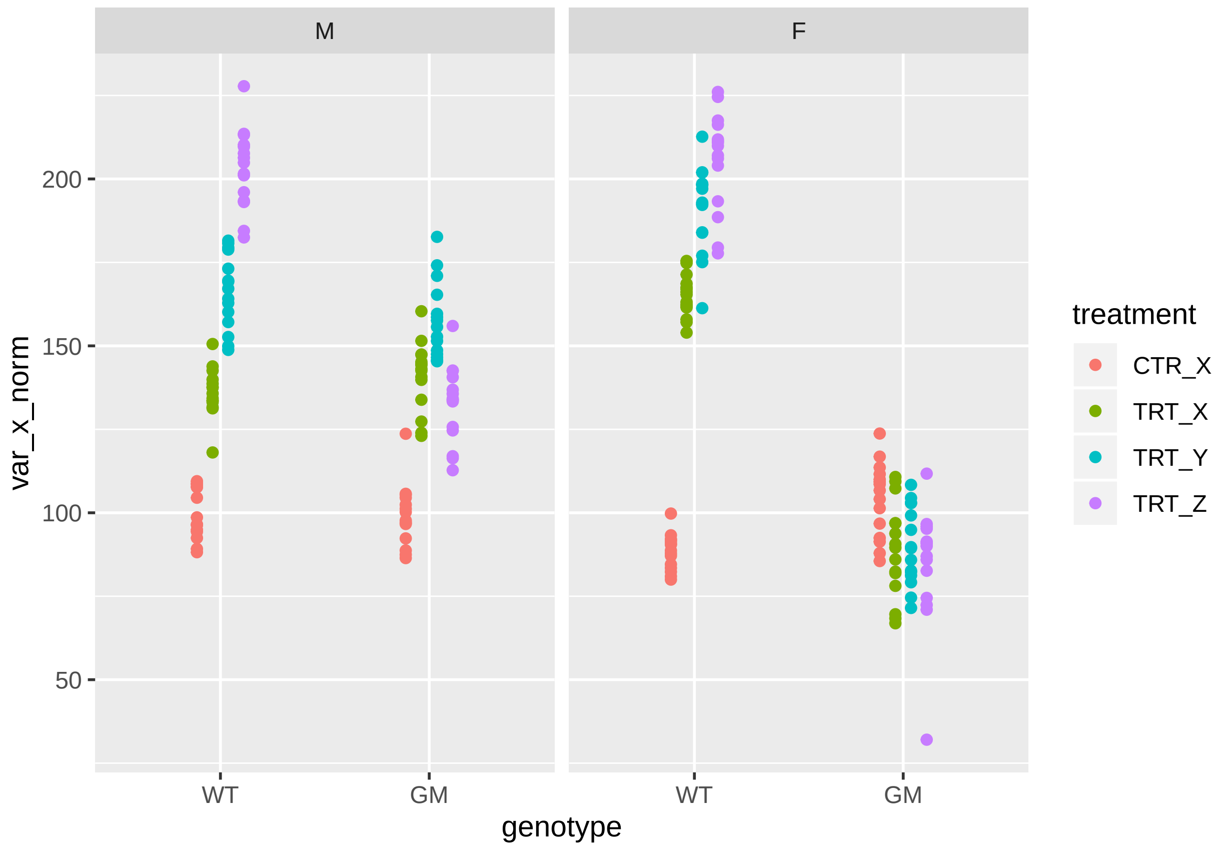
9. Treatments are defined with different colors but are all on top of each other. To make more clear. Add inside geom_point() that position has to be changed. This can be done with the statement position = position_dodge(1). The number (0-1) determines how much space there is between the respective columns within genotype. Try out different values to see how it makes the graph more aesthetic/readable.

```
ggplot(mydata, aes(genotype, var_x_norm, color = treatment)) + geom_point(position = position_dodge(0.3
```

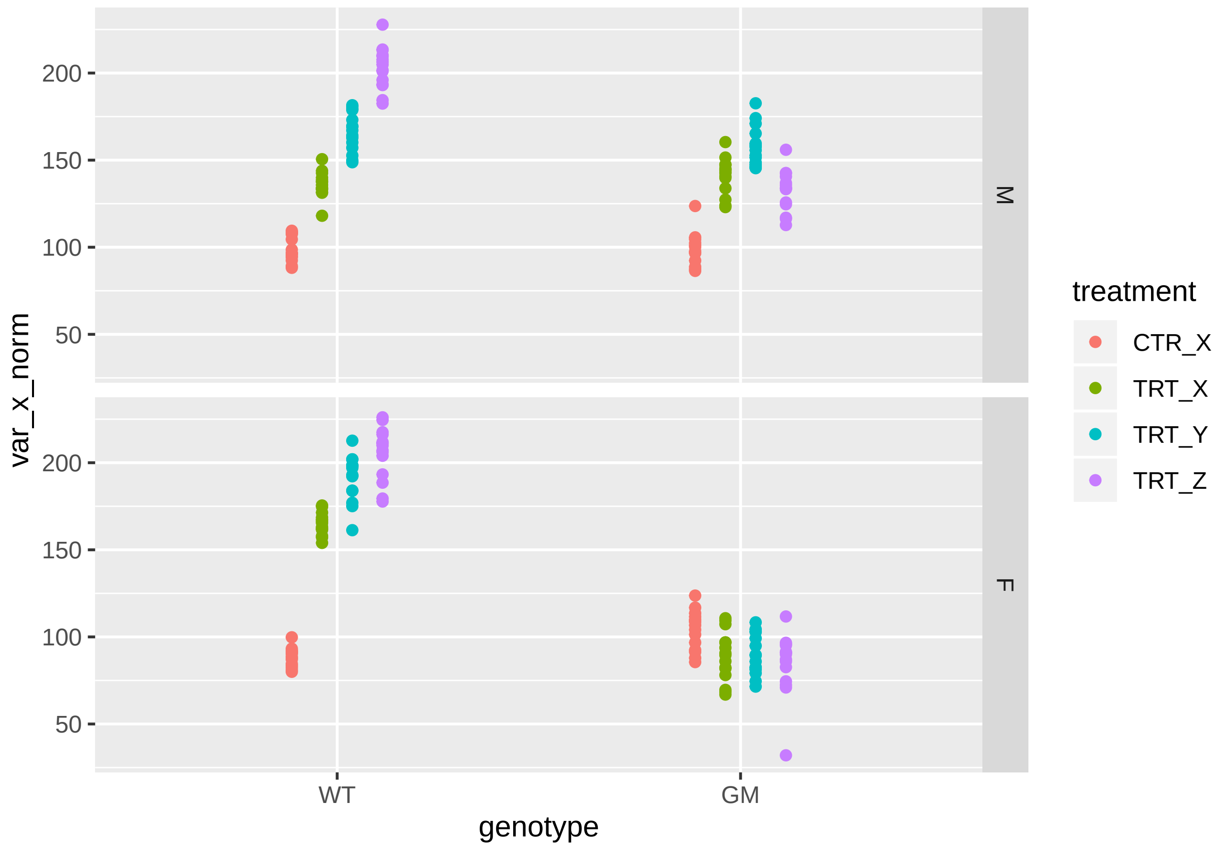


10. Besides colors and fills, shapes can also be used as an aesthetic to define categorical groups. Add shape = gender within aesthetics to see what happens. What can you see? How can the visibility be improved? See facet_wrap() or facet_grid() and try out.

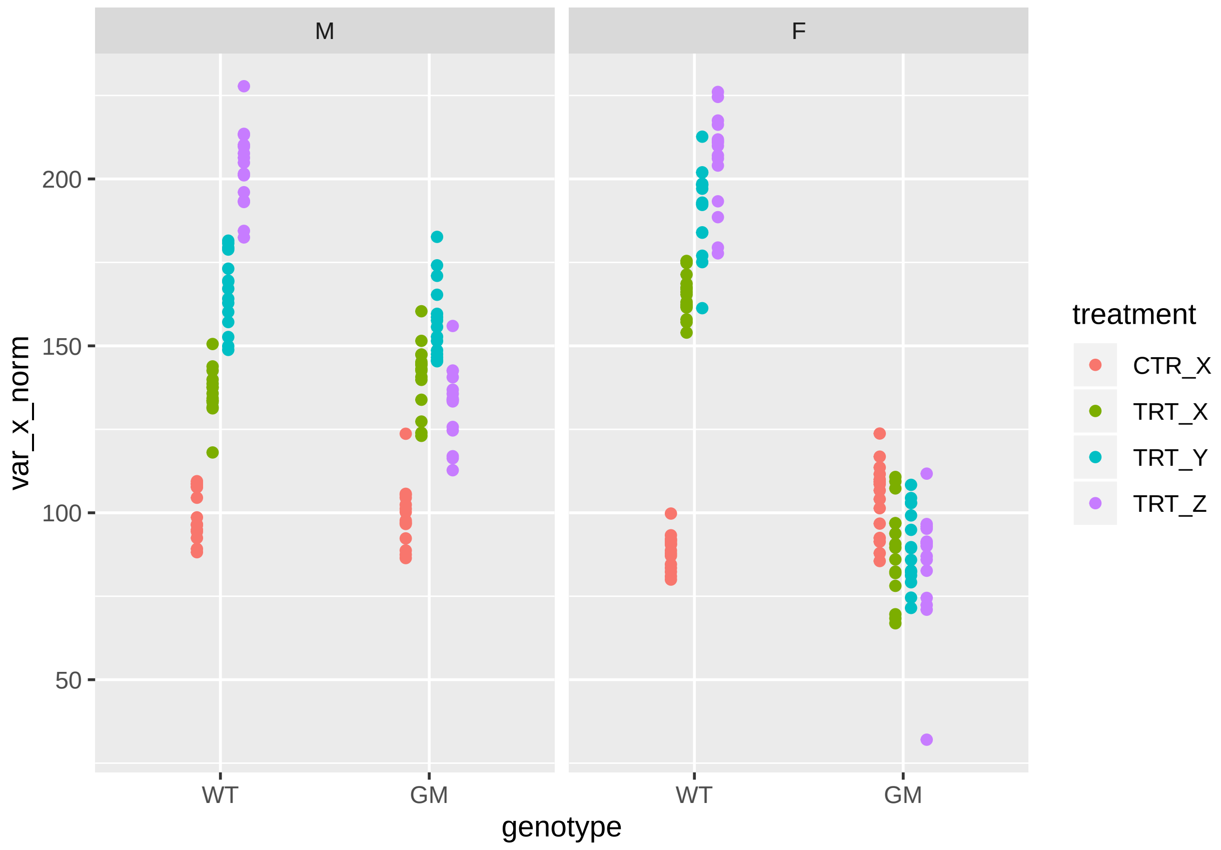
```
ggplot(mydata, aes(genotype, var_x_norm, color = treatment)) +
    geom_point(position = position_dodge(0.30)) +
    facet_wrap(~ gender)
```



```r
ggplot(mydata, aes(genotype, var_x_norm, color = treatment)) +
    geom_point(position = position_dodge(0.30)) +
    facet_grid(gender ~ .)
```



```r
ggplot(mydata, aes(genotype, var_x_norm, color = treatment)) +
    geom_point(position = position_dodge(0.30)) +
    facet_grid(. ~ gender)
```


M
F
var_x_norm
200
150
100
50
WT
GM
WT
GM
genotype
treatment
CTR_X
TRT_X
TRT_Y
TRT_Z