Introduction to R and Data Visualisation

Viswanathan Satheesh
Genome Informatics Facility, Iowa State University

2023-05-10

Installation of R and RStudio
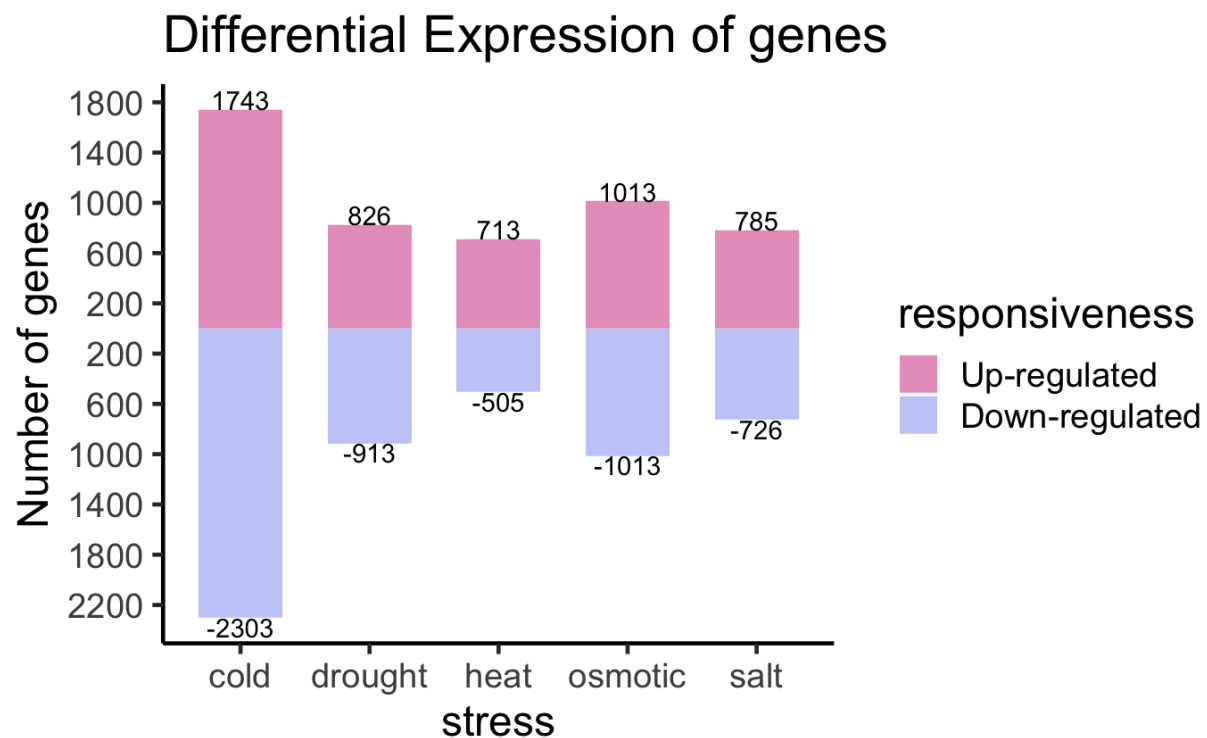
Please go these websites to download and install R and RStudio.

R - https://www.r-project.org

RStudio - https://posit.co/downloads/

For Rstudio, download the free version available on the downloads page.

Goal of our workshop is to create this plot that we see below:

Differential Expression of genes

Basic operations

Some very basic operations you can can carry out in R.

```r
1 + 1 # addition
```

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

```r
2 - 1 # subtraction
```

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

```r
2 * 2 # multiplication
```

```
## [1] 4
```

```r
6 / 2 # division
```

```
## [1] 3
```

```r
3 ** 2 # exponential
```

```
## [1] 9
```

```r
3 ^ 2 # exponential
```

```
## [1] 9
```

Here R works like a calculator. Apart from numbers, R can also help us print letters or a string of letters.

```r
"a"
```

```
## [1] "a"
```

```r
'a'
```

```
## [1] "a"
```

```r
"language"
```

```
## [1] "language"
```

```r
"R is my favourite programming language"
```

```
## [1] "R is my favourite programming language"
```

When working with large numbers such as **1934929292** and **23992343**, we cannot keep them in mind, or for that matter, remember complex computations. So, we have the concept of object or variable.

```r
a <- 1934929292
b <- 23992343
a
```

```
## [1] 1934929292
```

```r
b
```

```
## [1] 23992343
```

Here, we assign "<-" the first number to "a" and the second to "b". The "<-" is called the "assignment operator". "a" and "b" are called objects or variables. This now enables us to actually use the variables for doing further operations as seen below.

```r
a + b
```

```
## [1] 1958921635
```

```r
a - b
```

```
## [1] 1910936949
```

```r
a * b
```

```
## [1] 4.642349e+16
```

```
a ^ b
```

```
## [1] Inf
```

We can do something similar with strings too.

```
x <- "language"
y <- "R is my favourite programming language"
x
```

```
## [1] "language"
```

```
y
```

```
## [1] "R is my favourite programming language"
```

```
print(x)
```

```
## [1] "language"
```

```
print(y)
```

```
## [1] "R is my favourite programming language"
```

Creating a vector

In R, vectors are the most basic data objects. Let us create vectors **x** and **y**. We will do that in two ways. One, using the *c()* function, and the other using the *seq()* function. 'c' combines values into a vector. 'seq' is a sequence generator.

```
x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
y <- seq(11, 20)
x
```

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

```
y
```

```
##  [1] 11 12 13 14 15 16 17 18 19 20
```

The code above creates two vectors, **x** and **y**. Adding the two vectors gives:

```
x + y
```

```
##  [1] 12 14 16 18 20 22 24 26 28 30
```

```
x * y
```

```
##  [1]  11  24  39  56  75  96 119 144 171 200
```

The elements are added element-wise. The operations in R are element-wise. As an exercise you can try doing the other mathematical operations on the two vectors.

Indexing

The elements in the vectors are indexed. So, to extract an element you need only know its position. To extract the first element in x and in y:

```
x[1]
```

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

This returns **1**. Try the following and see what you get.

```r
y[1:4]
```

```
## [1] 11 12 13 14
```

```r
y[c(1, 3, 5)]
```

```
## [1] 11 13 15
```

```r
y[c(-1, -3, -5)]
```

```
## [1] 12 14 16 17 18 19 20
```

```r
y[-c(1, 3, 5)]
```

```
## [1] 12 14 16 17 18 19 20
```

Data types

What are the important data types? They can be listed as:

- **integer**
- **numeric**
- **logical**
- **character** Let us first create some vectors

```r
n <- 1 # numeric
i <- 1L # integer
l <- TRUE # logical
c <- "Some string" #character
```

Here we have four vectors created. x is a numeric vector, y an integer vector, t a logical vector, and c is a character vector. Remember, in R everything is a vector. There are no scalars. Therefore, all these vectors that we have created are all vectors of length one. To check the length of the vector, use the length() function:

```r
length(n)
```

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

```r
length(i)
```

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

```r
length(l)
```

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

```r
length(c)
```

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

You can see that all objects created are of length one.

Now let us check these vectors using the *class()* function:

```r
class(n)
```

```
## [1] "numeric"
```

```r
class(i)
```

```
## [1] "integer"
```

4

```r
class(l)
```

```
## [1] "logical"
```

```r
class(c)
```

```
## [1] "character"
```

Now let us create vectors of length > 1

```r
num_vr <- c(1, 3.0, 5.0) # numeric vector
int_vr <- c(1L, 3L, 5L) # integer vector
log_vr <- c(TRUE, FALSE, TRUE) # logical vector
char_vr <- c("I am", "a", "string.") # character vector
```

Now get their class.

```r
class(num_vr); length(num_vr)
```

```
## [1] "numeric"
```

```
## [1] 3
```

```r
class(int_vr); length(int_vr)
```

```
## [1] "integer"
```

```
## [1] 3
```

```r
class(log_vr); length(log_vr)
```

```
## [1] "logical"
```

```
## [1] 3
```

```r
class(char_vr); length(char_vr)
```

```
## [1] "character"
```

```
## [1] 3
```

We learnt to make vectors before, and now we have learnt to understand them a bit more. We now move on to matrices. First, let us create some vectors.

```r
v1 <- 1:5
v2 <- 6:10
v3 <- 11:15
```

We have three vectors v1, v2, and v3 and we are going to bind them column-wise.

```r
cbind(v1, v2, v3)
```

```
##      v1 v2 v3
## [1,]  1  6 11
## [2,]  2  7 12
## [3,]  3  8 13
## [4,]  4  9 14
## [5,]  5 10 15
```

The output just spews out to the console, which is not helpful. Let us create a variable, my_mat, and store the output

```r
my_mat <- cbind(v1, v2, v3)
my_mat
```

```
##      v1 v2 v3
## [1,]  1  6 11
## [2,]  2  7 12
## [3,]  3  8 13
## [4,]  4  9 14
## [5,]  5 10 15
```

Here, we used a function, cbind(), to bind three vectors into three columns. Now let us use the class function on the my_mat variable.

```
class(my_mat)
```

```
## [1] "matrix" "array"
```

my_matrix is a matrix. It has three columns, v1, v2, and v3. And, as it should be clear now, we used three vectors to create a matrix. Let us now see an alternate method for creating a matrix.

```
trial_mat <- matrix(1:20, nrow=5, ncol=4, byrow = TRUE)
trial_mat
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]   13   14   15   16
## [5,]   17   18   19   20
```

This creates a matrix with 4 rows and 5 columns. The [1,] refers to the first row. The [,1] refers to the first column.

Let us now talk about another kind of data structure, data frame. So, a data frame is similar to a matrix, but it can hold vectors of different classes. Let us create the same vectors we created previously even though they are still in memory.

```
num_vr <- c(1, 3.0, 5.0) # numeric vector
int_vr <- c(1L, 3L, 5L) # integer vector
log_vr <- c(TRUE, FALSE, TRUE) # logical vector
char_vr <- c("I am", "a", "string.") # character vector

# Let us use the cbind() function to put them together.

new_mat <- cbind(num_vr, int_vr, log_vr, char_vr)
new_mat
```

```
##      num_vr int_vr log_vr  char_vr
## [1,] "1"    "1"    "TRUE"  "I am"
## [2,] "3"    "3"    "FALSE" "a"
## [3,] "5"    "5"    "TRUE"  "string."
```

Looking at the output, we know that it is something we do not want. What is the class of the new variable?

```
class(new_mat)
```

```
## [1] "matrix" "array"
```

The class of the object new_mat is "matrix". A matrix can hold data belonging to a particular class. In this case, every data point is converted into a character. This is called coercion. Here we need a different kind of data structure that can hold different classes of data. To demostrate this point, let us create some vectors that we will make use of in creating this structure.

```r
set.seed(1234) # since the numbers are random, this will make sure we always
               # get the same set of random numbers
plant_height <- rnorm(100, 110, 10)
head(plant_height)
```

```
## [1]  97.92934 112.77429 120.84441  86.54302 114.29125 115.06056
```

Too many decimals. Let us round it off to two.

```r
plant_height <- round(plant_height, 2)
head(plant_height)
```

```
## [1]  97.93 112.77 120.84  86.54 114.29 115.06
```

```r
set.seed(237)
flowering_50 <- round(rnorm(100, 100, 10))
head(flowering_50)
```

```
## [1] 102 100 101 100 107 101
```

```r
set.seed(6438)
spikelet_fertility <- round(rnorm(100, 90, 3), 2)
head(spikelet_fertility)
```

```
## [1] 89.05 89.18 93.49 92.28 88.04 93.61
```

```r
max(spikelet_fertility)
```

```
## [1] 95.27
```

```r
set.seed(345)
thousand_seed_weight <- round(rnorm(100, 22, 3), 2)
head(thousand_seed_weight)
```

```
## [1] 19.65 21.16 21.52 21.13 21.80 20.10
```

Now let us combine the four vectors into a single data structure.

```r
my_data <- cbind(plant_height, flowering_50, spikelet_fertility, thousand_seed_weight)
head(my_data)
```

```
##      plant_height flowering_50 spikelet_fertility thousand_seed_weight
## [1,]        97.93          102              89.05                19.65
## [2,]       112.77          100              89.18                21.16
## [3,]       120.84          101              93.49                21.52
## [4,]        86.54          100              92.28                21.13
## [5,]       114.29          107              88.04                21.80
## [6,]       115.06          101              93.61                20.10
```

```r
class(my_data)
```

```
## [1] "matrix" "array"
```

Let us now create some numbers that we will use as genotype ids. We have 100 observations and that makes it 100 genotypes. We will name the genotypes from "001" to "100". Let us use the paste() function to create these ids.

```r
a1 <- paste("00", 1:9, sep = "")
a2 <- paste0("0", 10:99)
genotypes <- c(a1, a2, 100)
genotypes
```

```
##   [1] "001" "002" "003" "004" "005" "006" "007" "008" "009" "010" "011" "012"
##  [13] "013" "014" "015" "016" "017" "018" "019" "020" "021" "022" "023" "024"
##  [25] "025" "026" "027" "028" "029" "030" "031" "032" "033" "034" "035" "036"
##  [37] "037" "038" "039" "040" "041" "042" "043" "044" "045" "046" "047" "048"
##  [49] "049" "050" "051" "052" "053" "054" "055" "056" "057" "058" "059" "060"
##  [61] "061" "062" "063" "064" "065" "066" "067" "068" "069" "070" "071" "072"
##  [73] "073" "074" "075" "076" "077" "078" "079" "080" "081" "082" "083" "084"
##  [85] "085" "086" "087" "088" "089" "090" "091" "092" "093" "094" "095" "096"
##  [97] "097" "098" "099" "100"
```

Let us add this vector to our my_data object.

```
newdat <- cbind(genotypes, my_data)
head(newdat)
```

```
##      genotypes plant_height flowering_50 spikelet_fertility
## [1,] "001"     "97.93"      "102"        "89.05"
## [2,] "002"     "112.77"     "100"        "89.18"
## [3,] "003"     "120.84"     "101"        "93.49"
## [4,] "004"     "86.54"      "100"        "92.28"
## [5,] "005"     "114.29"     "107"        "88.04"
## [6,] "006"     "115.06"     "101"        "93.61"
##      thousand_seed_weight
## [1,] "19.65"
## [2,] "21.16"
## [3,] "21.52"
## [4,] "21.13"
## [5,] "21.8"
## [6,] "20.1"
```

We have seen this problem before; the entire data getting converted into a character class. To overcome this problem we use the data.frame() function.

```
field_data <- data.frame(genotypes, my_data)
head(field_data)
```

```
##   genotypes plant_height flowering_50 spikelet_fertility thousand_seed_weight
## 1       001        97.93          102              89.05                19.65
## 2       002       112.77          100              89.18                21.16
## 3       003       120.84          101              93.49                21.52
## 4       004        86.54          100              92.28                21.13
## 5       005       114.29          107              88.04                21.80
## 6       006       115.06          101              93.61                20.10
```

This output is more like it. Let us check the class of the df object.

```
class(field_data)
```

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

It is a dataframe. A dataframe, unlike a matrix, can hold vectors of different classes. Using the most important function in R, str(), we get a glimpse of what the field object contains.

```
str(field_data)
```

```
## 'data.frame':    100 obs. of  5 variables:
##  $ genotypes          : chr  "001" "002" "003" "004" ...
##  $ plant_height       : num  97.9 112.8 120.8 86.5 114.3 ...
```

```
## $ flowering_50         : num   102 100 101 100 107 101 92 95 93 78 ...
## $ spikelet_fertility  : num   89 89.2 93.5 92.3 88 ...
## $ thousand_seed_weight: num   19.6 21.2 21.5 21.1 21.8 ...
```

The "field" object is a data frame with 100 observations and 4 variables. Except for a, which is a factor, plant_height, flowering_50, and spikelet_fertility are numeric. Remember that 'a' is a vector containing the genotype ids. Therefore, "a" is recognised as a factor here.

Writing data to file

```
write.csv(field_data, "field_data.csv", quote = F, row.names = F)
```

Plotting with ggplot2

Introduction

ggplot2 is "A system for 'declaratively' creating graphics, based on `The Grammar of Graphics`. You provide the data, tell 'ggplot2' how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details."

`The Grammar of Graphics`, written by Leland Wilkinson, presents a theoretical foundation for producing quantitative graphics. (https://www.amazon.com/Grammar-Graphics-Statistics-Computing/dp/0387245448/ ref=as_li_ss_tl)

This book is the foundation for ggplot2 created by Hadley Wickham.

We are going to use the `faithful` and `iris` data sets to explore ggplot2. The data sets are part of the R package.

The following are important while using ggplot2.

1) Data

- Most important aspect
- Data representation holds the key to what can be done with the data

2) Mapping

- Aesthetic mapping Variables in the data linked to graphical properties
- Facet mapping Variables are linked to panels

3) Geometries

- geom_*()

4) Themes
5) Scale

Installing and loading the package.

```
# install.packages("ggplot2")
library(ggplot2)
```

The `faithful` data set contains information on the eruption pattern of the Old Faithful geyser in Yellowstone National Park.
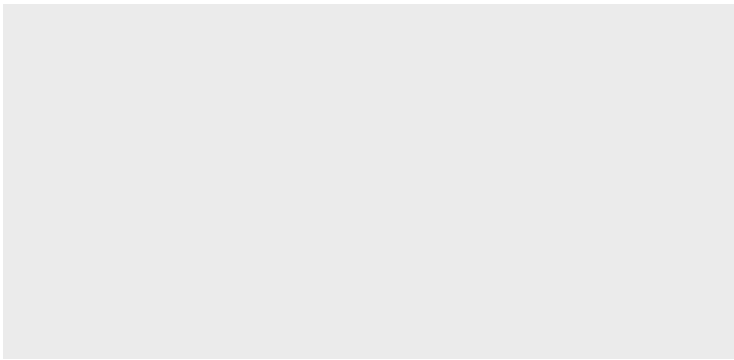
```
# Look at the data
str(faithful)
```

```
## 'data.frame':    272 obs. of  2 variables:
## $ eruptions: num   3.6 1.8 3.33 2.28 4.53 ...
## $ waiting  : num   79 54 74 62 85 55 88 85 51 85 ...
```

```
head(faithful)
```

```
##   eruptions waiting
## 1     3.600      79
## 2     1.800      54
## 3     3.333      74
## 4     2.283      62
## 5     4.533      85
## 6     2.883      55
```
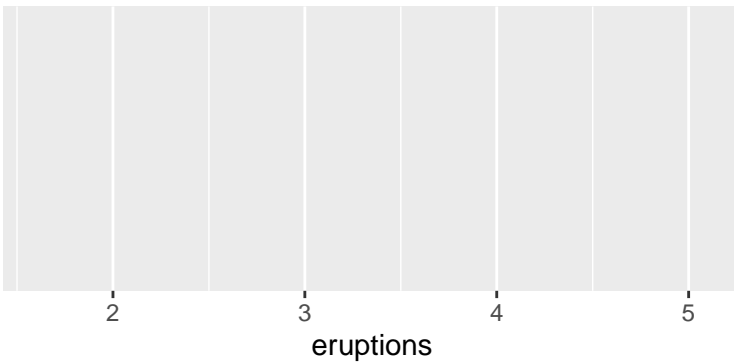
1) Data

```
#data("faithful")
ggplot(data = faithful)
```
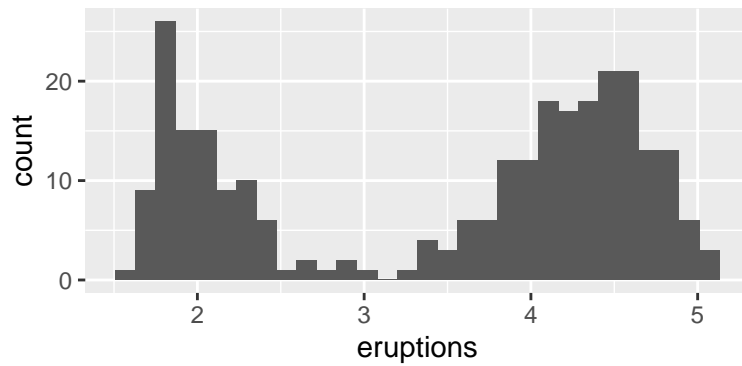

2) Mapping

```
# Adding the mapping
ggplot(data = faithful, mapping = aes(x = eruptions))
```


3) Geometry

```
# Basic histogram
ggplot(data = faithful, mapping = aes(x = eruptions)) +
  geom_histogram()
```
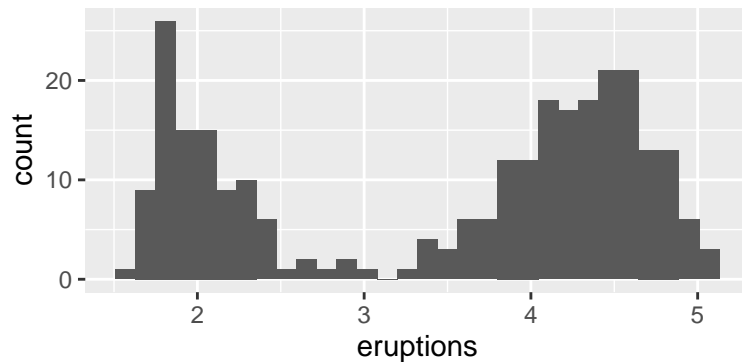
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

10

The data and the aesthetics can be specified within the layer as well.

```
ggplot() +
  geom_histogram(data = faithful, aes(x = eruptions))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```


4) Theme

```
ggplot(faithful, aes(x = eruptions)) +
  geom_histogram(colour = "black", fill = "white") +
  # theme_classic()
  # theme_bw()
  theme_minimal()
```
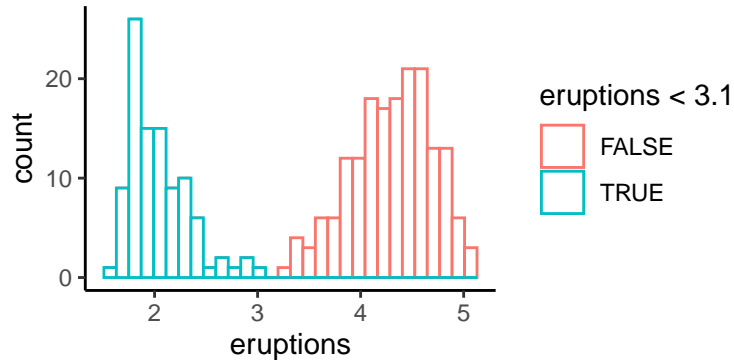
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```
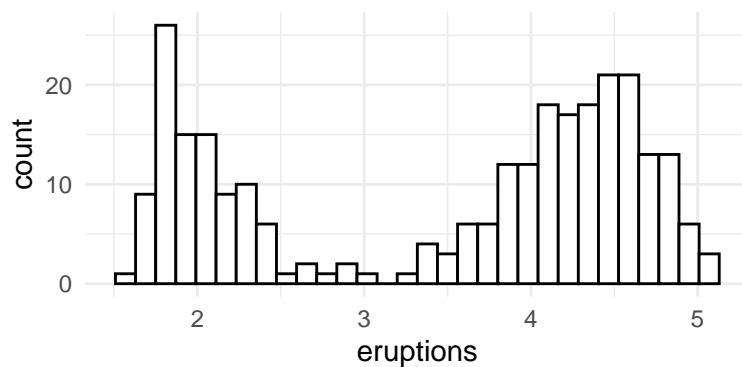

Colour based on mapping

```
ggplot(faithful, aes(x = eruptions)) +
  geom_histogram(aes(colour = eruptions < 3.1), fill = "white") +
  theme_classic()
```

`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

Fill based on mapping

```
ggplot(faithful, aes(x = eruptions)) +
  geom_histogram(aes(fill = eruptions < 3.1), colour = "black") +
  theme_classic()
```

`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
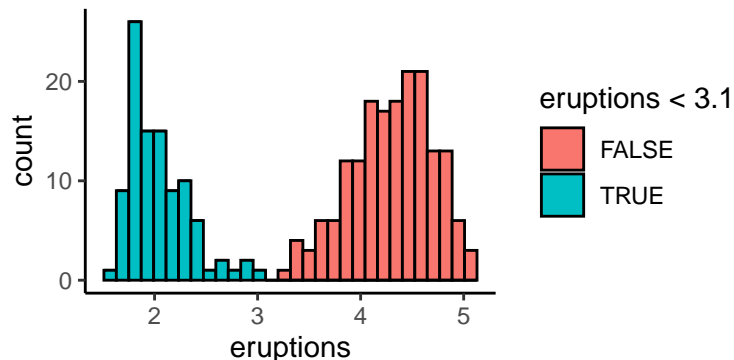
Let us now use the `iris` data set for further exploration of the ggplot2 package.

```
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 ...
```

```
head(iris)
```

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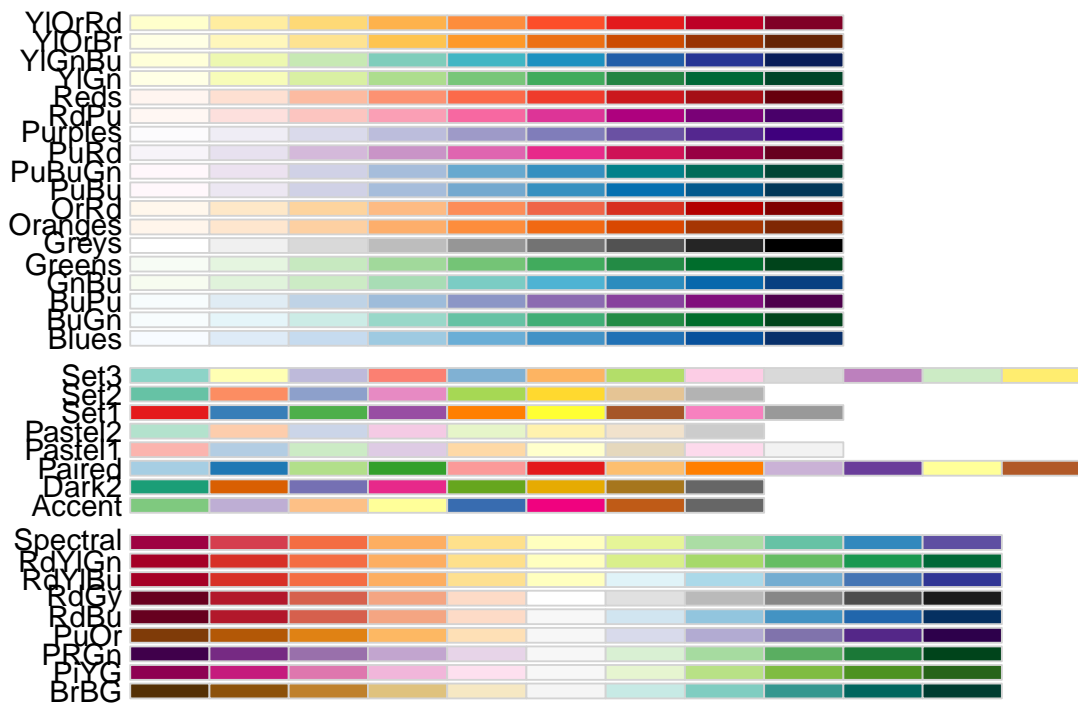
```
## 5          5.0         3.6           1.4           0.2   setosa
## 6          5.4         3.9           1.7           0.4   setosa
```

```r
# Basic scatterplot
ggplot(data = iris, mapping = aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(aes(colour = Species))+
  theme_classic()
```
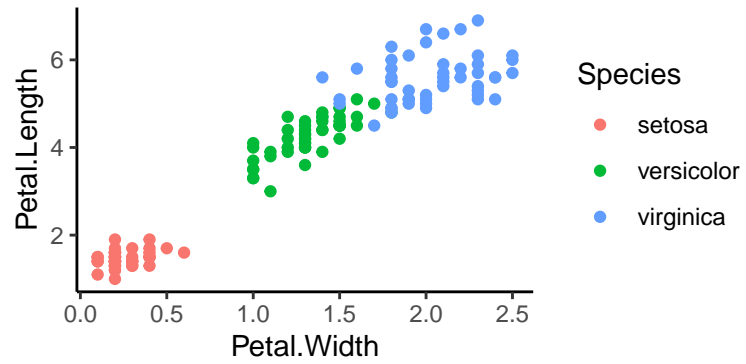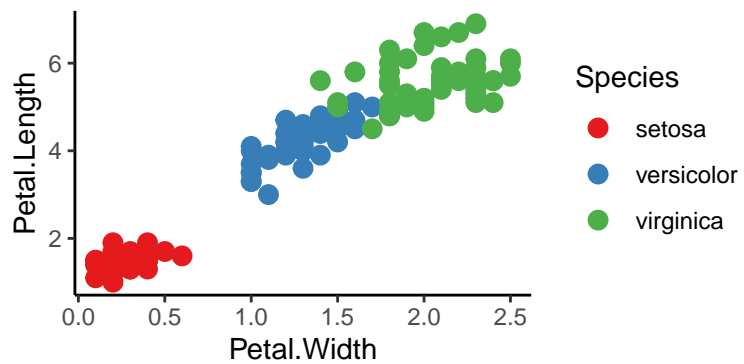

5) Scale Adding a different colour scheme

```r
RColorBrewer::display.brewer.all()
```
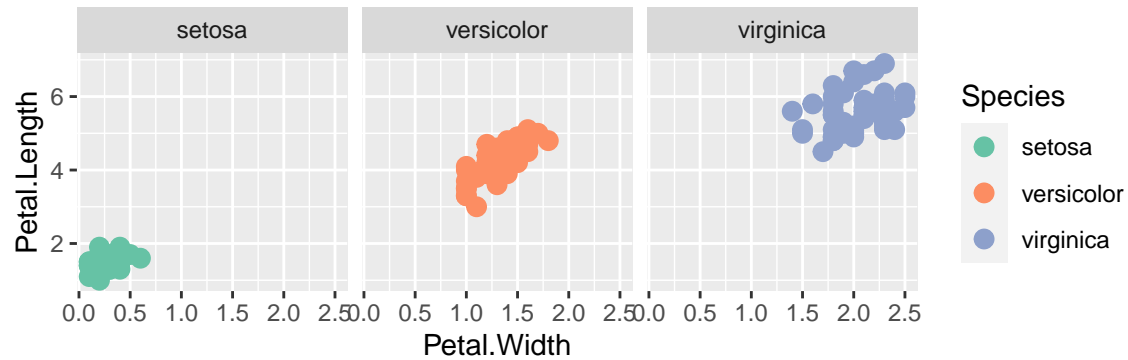
```
ggplot(data = iris, mapping = aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(aes(colour = Species), size = 3) +
  theme_classic() +
  scale_colour_brewer(palette = "Set1")
```
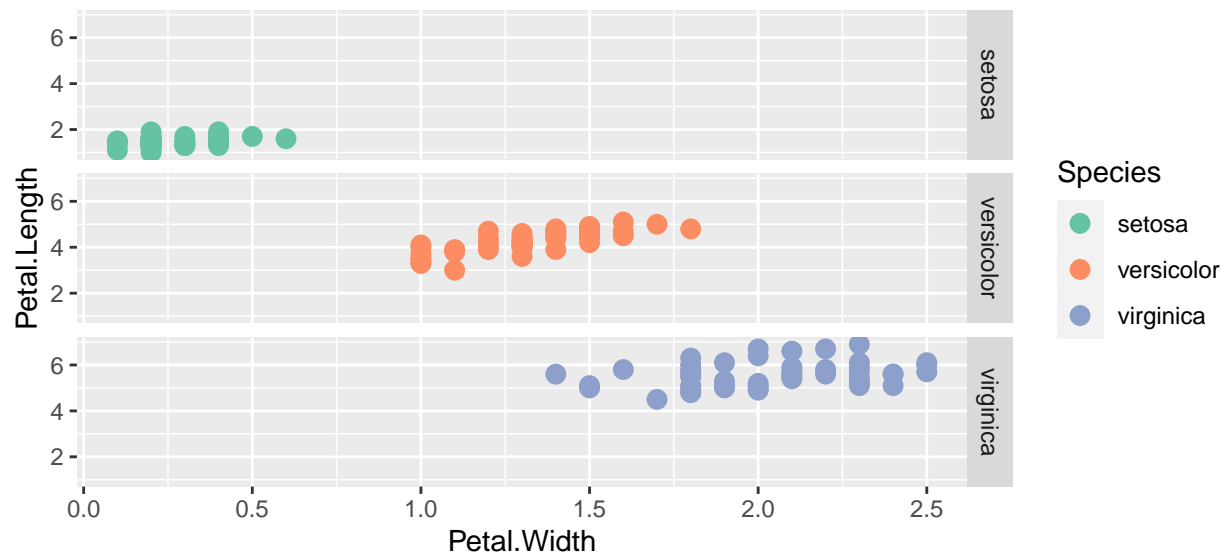

6) Facets

```
ggplot(data = iris, mapping = aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(aes(colour = Species), size = 3) +
  facet_wrap(~ Species) +
  scale_colour_brewer(palette = "Set2")
```
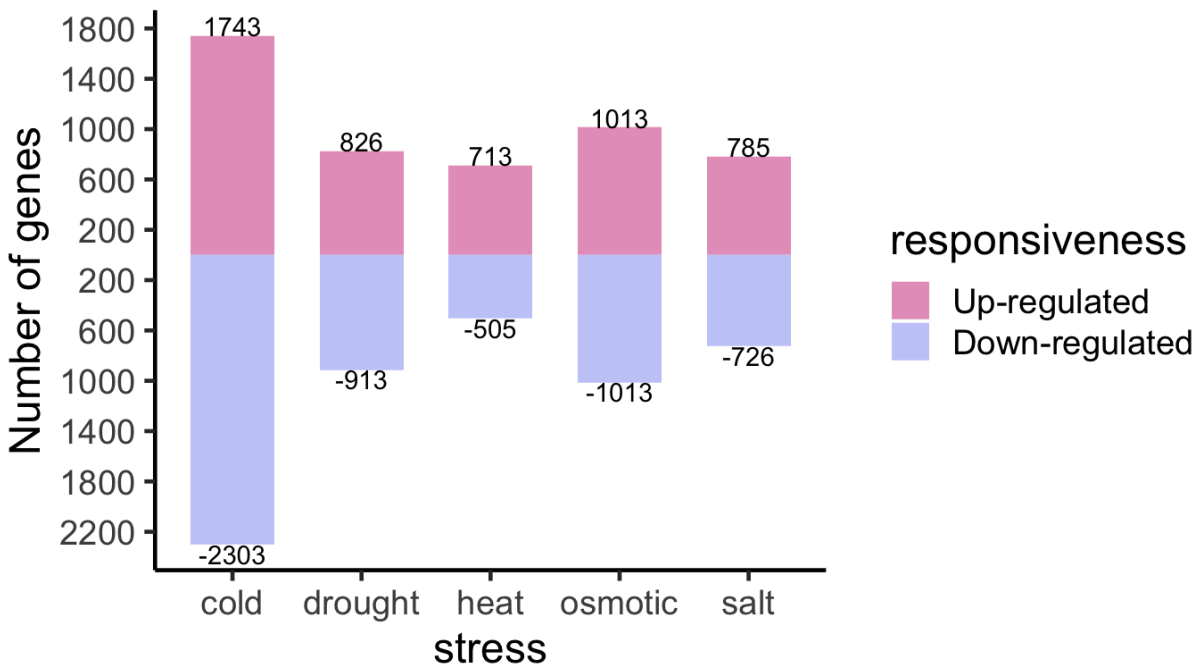


```
ggplot(data = iris, mapping = aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(aes(colour = Species), size = 3) +
  facet_grid(Species ~ .) +
  scale_colour_brewer(palette = "Set2")
```


Creating a mirror plot

In this section we will see how to make the plot shown below.

Reading data in:

```
gene_nums_mirror <- read.csv( "up_down_gene_numbers.csv" )
gene_nums_mirror
```

```
##       stress responsiveness num_genes
## 1      cold             up      1743
## 2      cold           down      2303
## 3   osmotic             up      1013
## 4   osmotic           down      1013
## 5      heat             up       713
## 6      heat           down       505
## 7      salt             up       785
## 8      salt           down       726
## 9   drought             up       826
## 10  drought           down       913
```

This data set has three columns and 10 rows. It is about differentially expressed genes under different stress conditions. So, let us start plotting with *ggplot2*.

```
#install.packages("ggplot2")
library( ggplot2 )
ggplot(data = gene_nums_mirror, aes(x = stress, y = num_genes)) +
  # geom_bar(stat = "identity")
  geom_bar( stat = "identity", position = "stack", width = 0.65 )
```
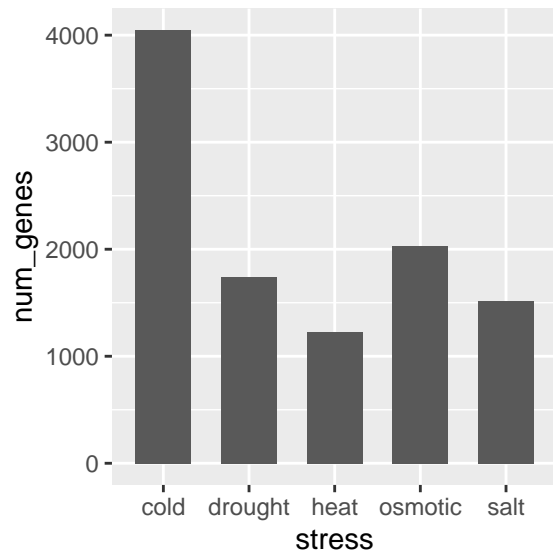
```
gene_nums_mirror$num_genes[gene_nums_mirror$responsiveness=="down"] <-
  -gene_nums_mirror$num_genes[gene_nums_mirror$responsiveness=="down"]
gene_nums_mirror
```

```
##      stress responsiveness num_genes
## 1      cold             up      1743
## 2      cold           down     -2303
## 3   osmotic             up      1013
## 4   osmotic           down     -1013
## 5      heat             up       713
## 6      heat           down      -505
## 7      salt             up       785
## 8      salt           down      -726
## 9   drought             up       826
## 10  drought           down      -913
```
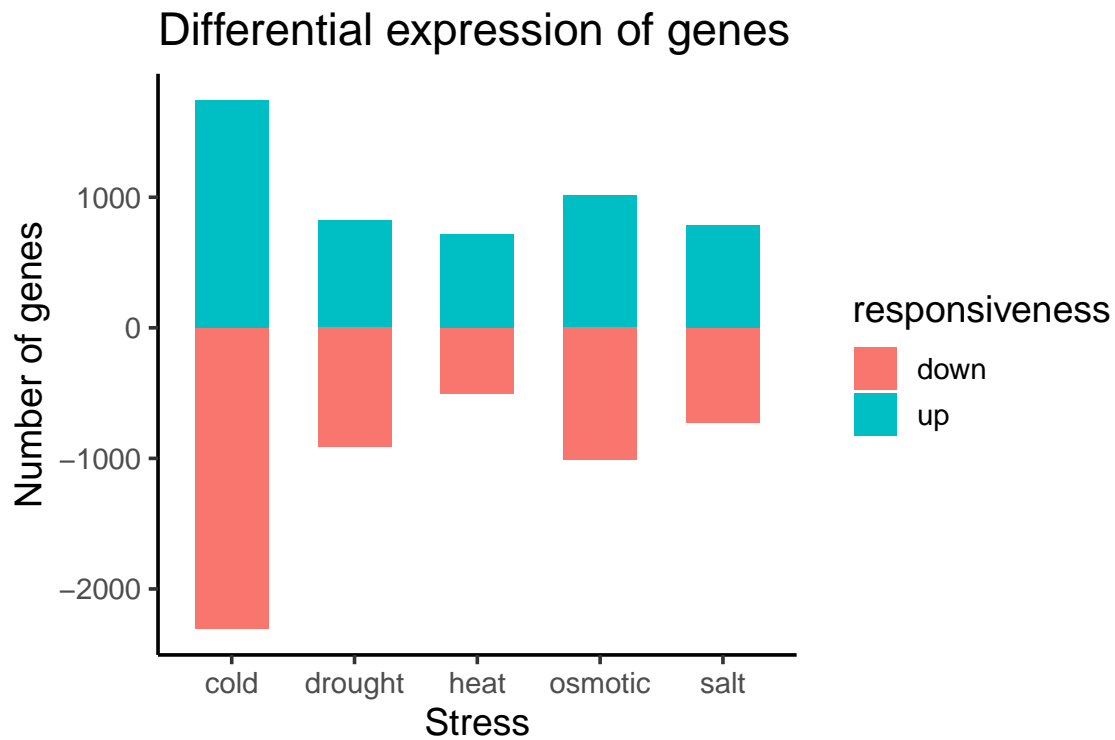
```
ggplot(data = gene_nums_mirror, aes(x = stress, y = num_genes)) +
  geom_bar(stat = "identity", width = 0.65)
```
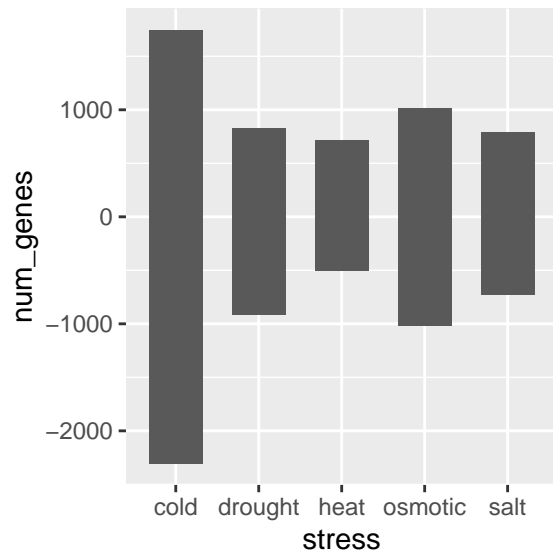
Adding colour to the bars, a theme, x- and y-axis labels, and a main title.

```
ggplot(data = gene_nums_mirror, aes(x = stress, y = num_genes, fill = responsiveness)) +
  geom_bar(stat = "identity", width = 0.6) +
  theme_classic(base_size = 14) +
  ylab("Number of genes") +
  xlab("Stress") +
  ggtitle("Differential expression of genes")
```


The figure legend has "down" on the top and "up" below. To change that default behavoiur, we need to re-order the levels: "up" first followed by "down", in the data set.

```
# Define the new level order
new_levels <- c("up", "down", setdiff(levels(gene_nums_mirror$responsiveness), c("up", "down")))

# Modify the "responsiveness" column
gene_nums_mirror$responsiveness <- factor(gene_nums_mirror$responsiveness, levels = new_levels)
```

Apart from changing the levels, we provide labels for the legend in the code below.

```
ggplot(data = gene_nums_mirror,
       aes(x = stress, y = num_genes,
           fill = factor(responsiveness, labels = c("Up-regulated", "Down-regulated")))) +
  labs(fill = "responsiveness") +
  geom_bar( stat = "identity", position = "identity", width = 0.6 ) +
  ylab("Number of genes") +
  xlab("Stress") +
  theme_classic(base_size = 14) +
  ggtitle("Differential Expression of genes")
```
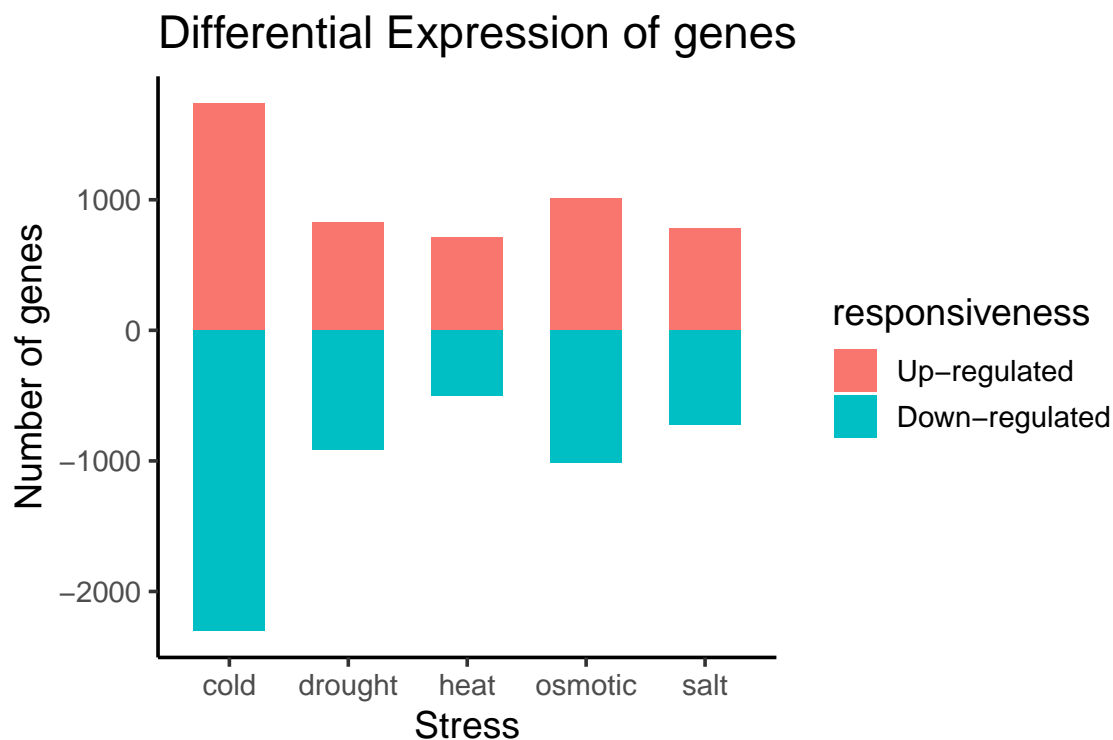

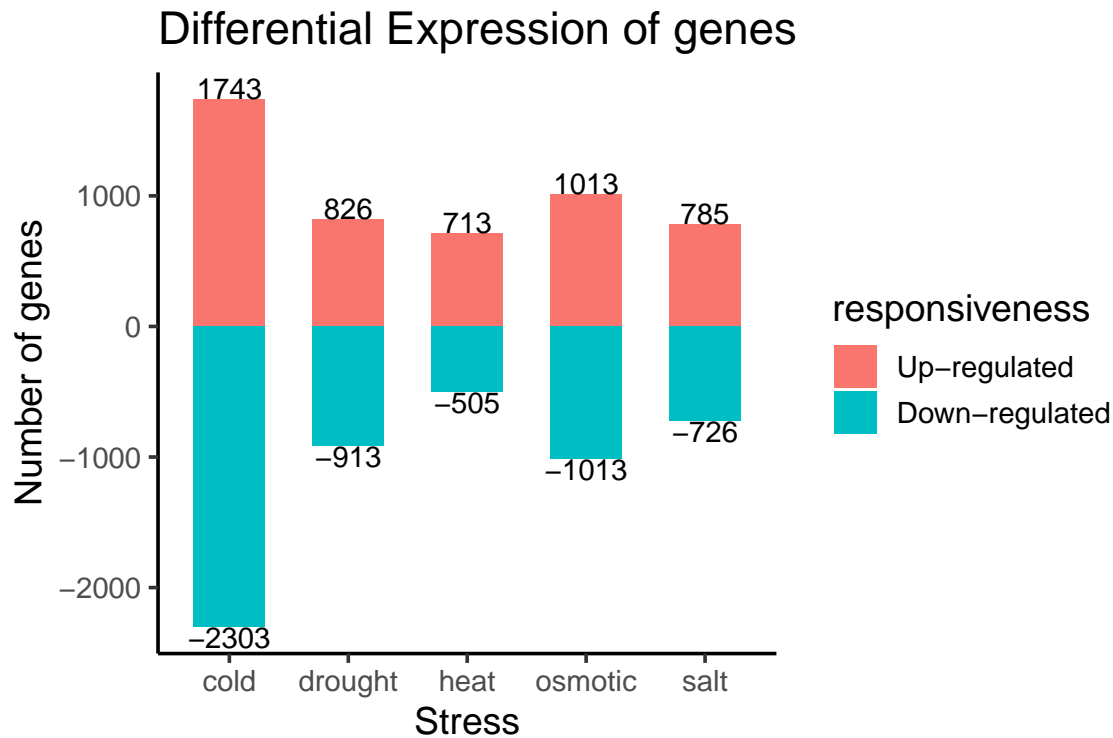Adding the number of genes on the bars.

```
ggplot(data = gene_nums_mirror,
       aes(x = stress, y = num_genes,
           fill = factor(responsiveness, labels = c("Up-regulated", "Down-regulated")))) +
  labs(fill = "responsiveness") +
  geom_bar( stat = "identity", position = "identity", width = 0.6 ) +
  ylab("Number of genes") +
  xlab("Stress") +
  theme_classic(base_size = 14) +
  ggtitle("Differential Expression of genes") +
  # geom_text(aes(label = num_genes))
```

```
geom_text(aes(label = num_genes), vjust = ifelse(gene_nums_mirror$num_genes>0, 0,1),
          colour = "black")
```

Differential Expression of genes

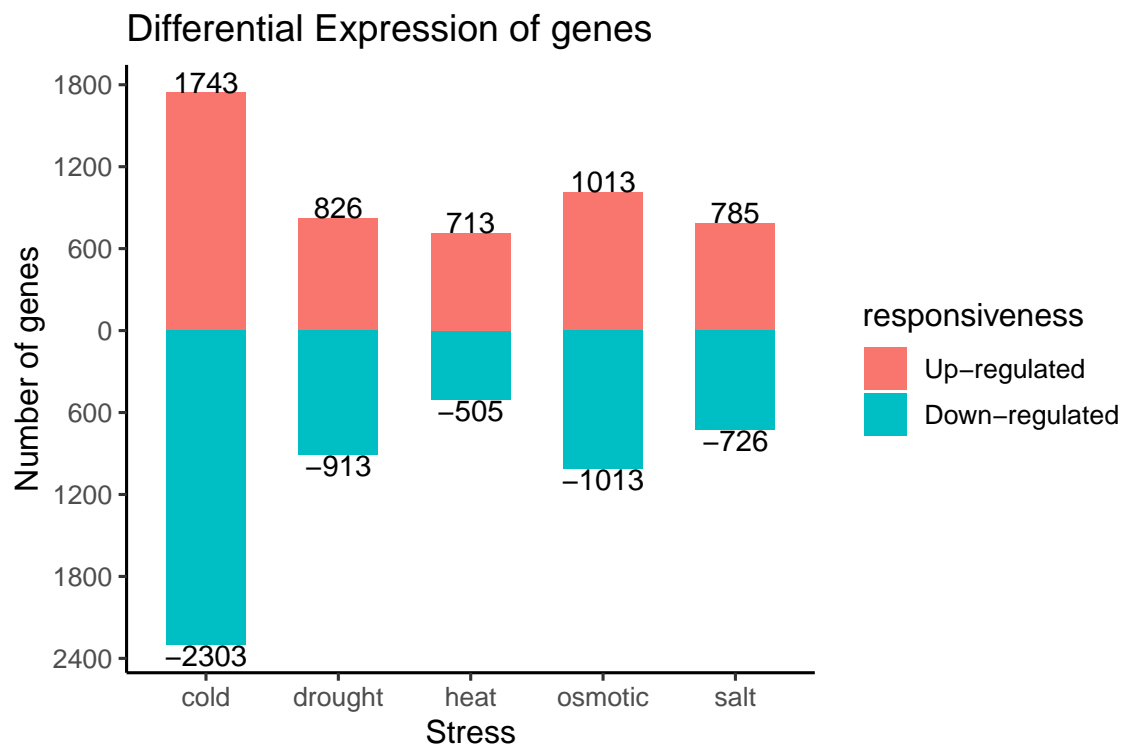Setting the y-axis ticks:

```
ggplot(data = gene_nums_mirror,
       aes(x = stress, y = num_genes,
           fill = factor(responsiveness, labels = c("Up-regulated", "Down-regulated")))) +
  labs(fill = "responsiveness") +
  geom_bar( stat = "identity", position = "identity", width = 0.6 ) +
  ylab("Number of genes") +
  xlab("Stress") +
  theme_classic(base_size = 12) +
  ggtitle("Differential Expression of genes") +
  geom_text(aes(label = num_genes), vjust = ifelse(gene_nums_mirror$num_genes>0, 0,1),
            colour = "black") +
  # scale_y_continuous(breaks=seq(-3000, 1800, by = 600))
  scale_y_continuous(breaks=seq(-3000,1800,by=600),labels=abs(seq(-3000,1800,by=600)))
```

Differential Expression of genes

The final code snippet with a different colour scheme.

```r
# install.packages( "wesanderson" )
library( wesanderson )

ggplot(data = gene_nums_mirror,
       aes(x = stress, y = num_genes,
           fill = factor(responsiveness, labels = c("Up-regulated", "Down-regulated")))) +
  labs(fill = "responsiveness") +
  geom_bar( stat = "identity", position = "identity", width = 0.6 ) +
  ylab("Number of genes") +
  xlab("Stress") +
  theme_classic(base_size = 14) +
  ggtitle("Differential Expression of genes") +
  geom_text(aes(label = num_genes), vjust = ifelse(gene_nums_mirror$num_genes>0, 0,1),
            colour = "black") +
  scale_y_continuous(breaks=seq(-3000,1800,by=400),labels=abs(seq(-3000,1800,by=400))) +
  scale_fill_manual(values = wes_palette(n=2, name = "GrandBudapest2"))
```

Differential Expression of genes


```
sessionInfo()
```

```
## R version 4.2.1 (2022-06-23)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur ... 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] wesanderson_0.3.6 ggplot2_3.4.1
##
## loaded via a namespace (and not attached):
##  [1] rstudioapi_0.14    knitr_1.42         magrittr_2.0.3     tidyselect_1.2.0
##  [5] munsell_0.5.0      colorspace_2.1-0   R6_2.5.1           rlang_1.0.6
##  [9] fastmap_1.1.0      fansi_1.0.4        highr_0.10         dplyr_1.1.0
## [13] tools_4.2.1        grid_4.2.1         gtable_0.3.1       xfun_0.38
## [17] utf8_1.2.3         cli_3.6.0          withr_2.5.0        htmltools_0.5.4
## [21] yaml_2.3.7         digest_0.6.31      tibble_3.1.8       lifecycle_1.0.3
## [25] RColorBrewer_1.1-3 farver_2.1.1       vctrs_0.5.2        glue_1.6.2
## [29] evaluate_0.20      rmarkdown_2.21     labeling_0.4.2     compiler_4.2.1
## [33] pillar_1.8.1       generics_0.1.3     scales_1.2.1       pkgconfig_2.0.3
```