Introduction to R

A. Di Liddo

6/13/2019

R is a programming language and statistical environment for the analysis and visualization of data. The R software runs on all common operating systems (Windows, UNIX platforms, MacOS) and can be downloaded from http://www.r-project.org.

A program written in R language is not directly executable but it requires a working environments like **RStudio**. RStudio can be downloaded from http://www.rstudio.org

Arithmetic with R

R can be used as a simple calculator using arithmetic operators:

Addition: +

```
20 + 7
```

```
## [1] 27
```

Subtraction: -

```
20 - 7
```

```
## [1] 13
```

Multiplication: *

```
20 * 7
```

```
## [1] 140
```

Division: /

```
20 / 7
```

```
## [1] 2.857143
```

Exponentiation: ^

```
2^3
```

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

Modulo: %%

```
41 %%  2
```

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

logical comparison operators:

< for less than
> for greater than
<= for less than or equal to
>= for greater than or equal to

== for equal to each other
!= not equal to each other

```
1014 < 2929
```

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

```
1014 == 1014 # numbers
```

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

```
"hello" == "hello" # strings
```

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

Basic data types in R

- Decimal values (5.3) are called *numerics*.

- Natural numbers (201) are called *integers*. Integers are also numerics.

- Boolean values (TRUE or FALSE) are called *logical*.

- Text (or string) values are called *characters*.
- **variable**: Variables are used to store data, whose value can be changed according to our need. Variables can contain numbers or characters The assignment operator <- is used to create new variables.

The name of a variable usually starts with a letter, can contain numbers, dots (.), underscores (_). Examples: tab, tab1, dna_elix …

```
a <- 1
print(a)
```

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

The value of a variable can be changed by assigning a new value.

```
a <- 5
a
```

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

```
a <- a + 3
a
```

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

- **vector**: A vector is a sequence of data elements of the same type. It allows to store multiple elements in a single variable.

The function class() can be use to define the data type of a variable

```
a  <- "home"
class(a)
```

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

```
a <- TRUE
class(a)
```

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

```r
a <- c(1,2,4,10)
class(a)
```

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

```r
numeric_vect <- c(1,2,3) # vector of numbers
numeric_vect
```

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

```r
dna <- c("A","C","T","G","T","G", "T","T") # vector of characters
dna
```

```
## [1] "A" "C" "T" "G" "T" "G" "T" "T"
```

The single components of a vector can be accessed by squared brackets. For instance, to retrieve the second element of the vector **a** ("C")

```r
a[2]
```

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

If the vector is composed of both character and numeric elements, numbers will be considered characters

```r
vect = c(1,4,"A","V")
vect
```

```
## [1] "1" "4" "A" "V"
```

```r
class(vect)
```

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

If the vector is composed of numeric elements, some basic operations can be done:

```r
numeric_vect <- c(1,2,5,20,10)

# compute the mean of the elements in the vector
mean(numeric_vect)
```

```
## [1] 7.6
```

```r
# compute the sum of the elements in the vector
sum(numeric_vect)
```

```
## [1] 38
```

```r
# add a value to all elements in the vector
numeric_vect + 1
```

```
## [1]  2  3  6 21 11
```

the size of a vector can be obtained with the function length()

```r
length(a)
```

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

```r
length(dna)
```

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

Further examples:

```r
a <- 1:20
print(a)
```

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

```r
length(a)
```

```
## [1] 20
```

```r
H <-  5 # height
W <- 3 # width
area <- H * W # area of the rectangle
area
```

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

```r
dna1 <- c("A","T","A","A","G","C","G","C")
dna2 <- c("G","G","C","T")
dna <- c(dna1,"T",dna2,"G")
dna
```

```
##  [1] "A" "T" "A" "A" "G" "C" "G" "C" "T" "G" "G" "C" "T" "G"
```

```r
# compare vectors
v <- c("A", "T", "hello", "bye")
l <- c("A", "hello")

v == l
```

```
## [1]  TRUE FALSE FALSE FALSE
```

```r
l %in% v
```

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

```r
v %in% l
```

```
## [1]  TRUE FALSE  TRUE FALSE
```

paste() function allows to concatenate characters vectors

```r
a <- "hello"
b <- c("world")

paste(a, b, sep = " ")
```

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

```r
a = c("hello", "goodbye")
b = "world"

paste(a, b, sep = " ")
```

```
## [1] "hello world"   "goodbye world"
```

```r
1:3 + 1 # vector vs. single number
```

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

```r
1:3 + 1:3 # vector vs. vector
```

```
## [1] 2 4 6
```

```r
a <- 1:3
a[2]
```

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

```r
seq(from = 1, to = 100, length.out = 10)
```

```
##  [1]   1  12  23  34  45  56  67  78  89 100
```

```r
rep("hello", times = 10)
```

```
##  [1] "hello" "hello" "hello" "hello" "hello" "hello" "hello" "hello" "hello"
## [10] "hello"
```

```r
rep(c("hello", "world"), times = 10)
```

```
##  [1] "hello" "world" "hello" "world" "hello" "world" "hello" "world" "hello"
## [10] "world" "hello" "world" "hello" "world" "hello" "world" "hello" "world"
## [19] "hello" "world"
```

```r
rep(c("hello", "world"), each = 3)
```

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

```r
v <- c("C","A", "F", "L", "E", "Z", "X")
order(v, decreasing = T) # index
```

```
## [1] 6 7 4 3 5 1 2
```

```r
v[order(v, decreasing = T)] # order the vector
```

```
## [1] "Z" "X" "L" "F" "E" "C" "A"
```

```r
order(v, decreasing = F) # index
```

```
## [1] 2 1 5 3 4 7 6
```

```r
v[order(v, decreasing = F)]  # order the vector
```

```
## [1] "A" "C" "E" "F" "L" "X" "Z"
```

```r
a <- 10

a < 11 & a > 5
```

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

```r
a < 11 & a < 5
```

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

```r
a > 11 | a < 5
```

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

```r
a
```

```
## [1] 10
```

```r
a <- c(7,2,11)
b <- a < 5

a[b]
```

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

Dataframes

To create a table, the function data.frame() can be used, as in the following example:

```r
df <- data.frame(Numbers = 1:20, Letters = sample(letters,20))
df <- data.frame(Numbers = 1:20, Letters = letters[1:20])

# retrieve the number of columns
ncol(df)
```

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

```r
# retrieve the number of rows
nrow(df)
```

```
## [1] 20
```

```r
# show the first 10 lines
head(df)
```

```
##   Numbers Letters
## 1       1       a
## 2       2       b
## 3       3       c
## 4       4       d
## 5       5       e
## 6       6       f
```

Tables can be also loaded in R using multiple functions, depending on the format of the file: read.table() read.csv() . . .

Elements of the table can be accessed with brackets [,].

```r
# show the element at the row 5 and column 2
df[5,2]
```

```
## [1] e
## Levels: a b c d e f g h i j k l m n o p q r s t
```

```r
# show the element at the row 5,6,7 and column 2
df[c(5,6,7),2]
```

```
## [1] e f g
## Levels: a b c d e f g h i j k l m n o p q r s t
```

Columns can be accessed with the symbol $ followed by the name of the column. This will give a vector with values included in the column.

```r
# show the column Numbers
df$Numbers
```

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

```r
# show the 4th element of the column Letters
df$Letters[4]
```

```
## [1] d
## Levels: a b c d e f g h i j k l m n o p q r s t
```

Tables can be subset according to specific criteria

```r
# select rows with Numbers > 7
df[df$Numbers > 15, ]
```

```
##    Numbers Letters
## 16      16       p
## 17      17       q
## 18      18       r
## 19      19       s
## 20      20       t
```

```r
# select rows with Numbers > 7 and letter = p
df[df$Numbers > 15 & df$Letters == "p", ]
```

```
##    Numbers Letters
## 16      16       p
```

```r
# using subset function
subset(x = df, Numbers > 15 & Letters == "p")
```

```
##    Numbers Letters
## 16      16       p
```

Visualizing data in R with ggplot2

ggplot2 is a package for advanced graphics in R. It allows to largely customize plots.

qplot() function is a basic function of the ggplot2 package, to create plots quicky, but not recommended for complex graphics.

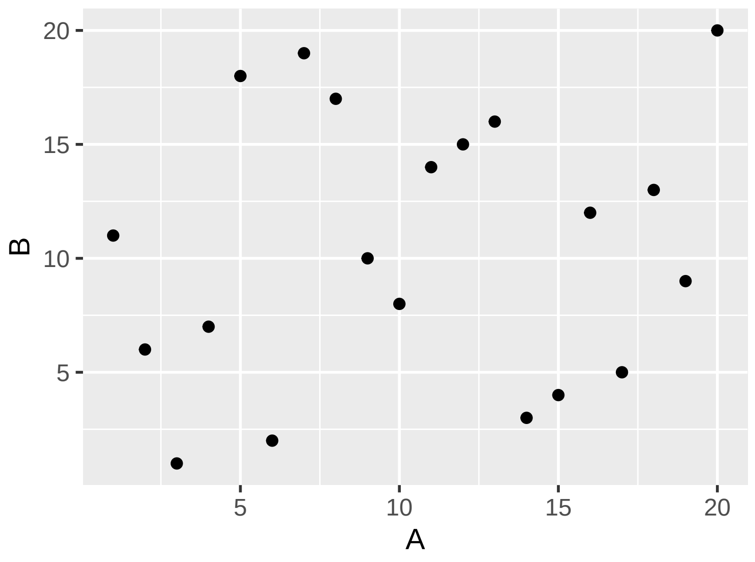
ggplot() function is more suitable to produce complex graphics and it requires two main arguments:
- data: the dataset to be plotted, usually a dataframe - mapping: aesthetic mappings provided by aes() function

Components like points, lines, bars etc. can be added to a plot with geom_*() functions following the symbol +. For a comprehensive list of geom_*() functions see https://ggplot2.tidyverse.org/reference/

```r
library(ggplot2)

df <- data.frame(A = sample(1:20, 20), B = sample(1:20,20))
```

scatter plot

```r
qplot(x = A, y = B, data = df, geom="point")
```



Let's use some public data included in R.

```r
data("iris")

class(iris)
```

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

```r
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
ncol(iris)
```

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

```r
nrow(iris)
```

```
## [1] 150
```

```r
# the function dim shows the number of rows and columns as a numeric vector
dim(iris)
```

```
## [1] 150   5
```

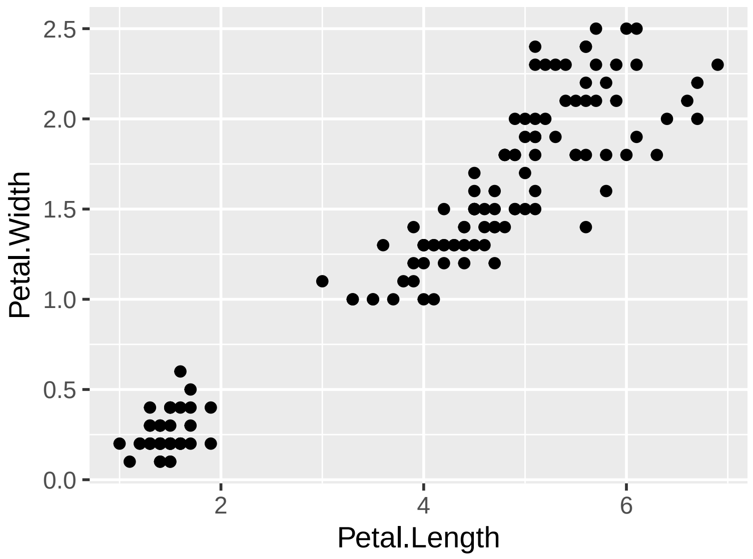
The function summary() allows to get a statistic summary of the data

```r
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300  
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

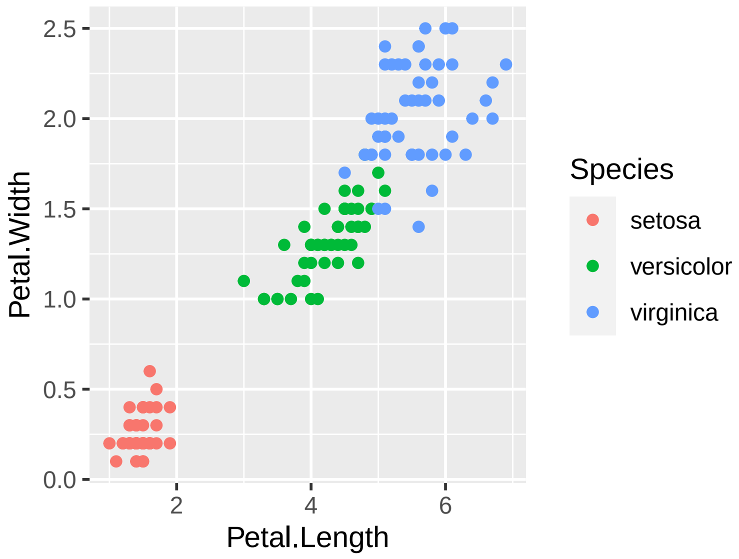
Lets have a look at the relationship between the petal length and the petal width

```
# scatterplot with ggplot

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



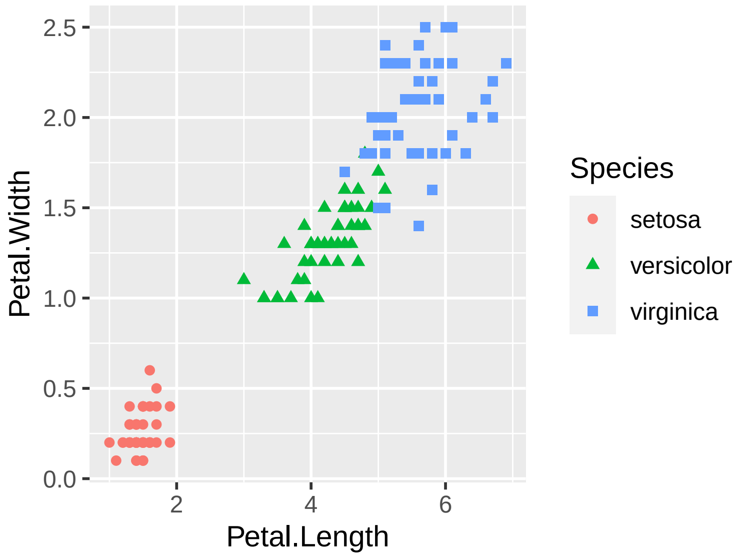
Colour by species

```
ggplot(data=iris, aes(x=Petal.Length, y=Petal.Width, colour = Species)) +
  geom_point()
```



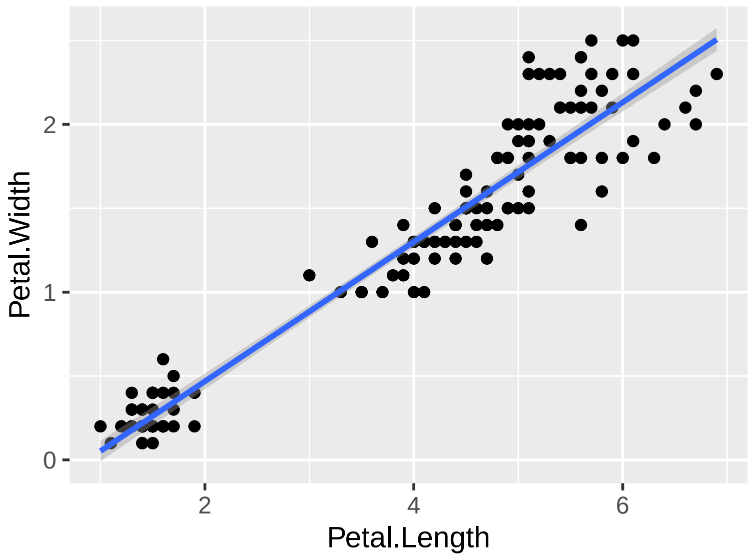
Shape by species

```r
ggplot(data=iris,aes(x=Petal.Length, y=Petal.Width, color = Species, shape = Species)) +
  geom_point()
```



Add regression line

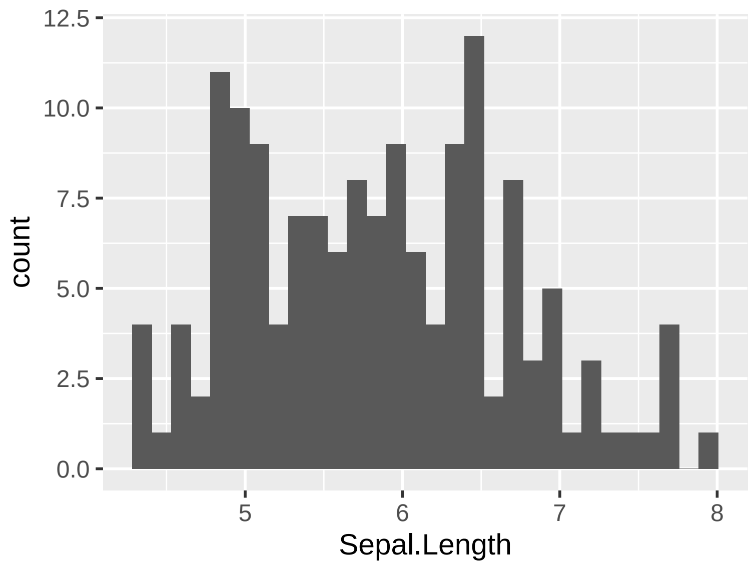
```r
ggplot(data=iris,aes(x=Petal.Length, y=Petal.Width)) + geom_point() +
  geom_smooth(method='lm',formula=y~x) # linear model
```



Histogram

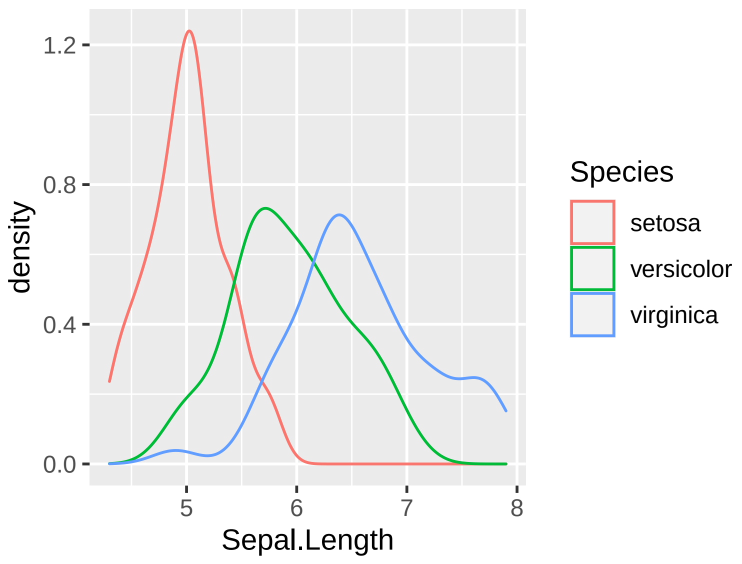
```r
ggplot(data=iris,aes(x=Sepal.Length)) + geom_histogram()
```

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



Density plot

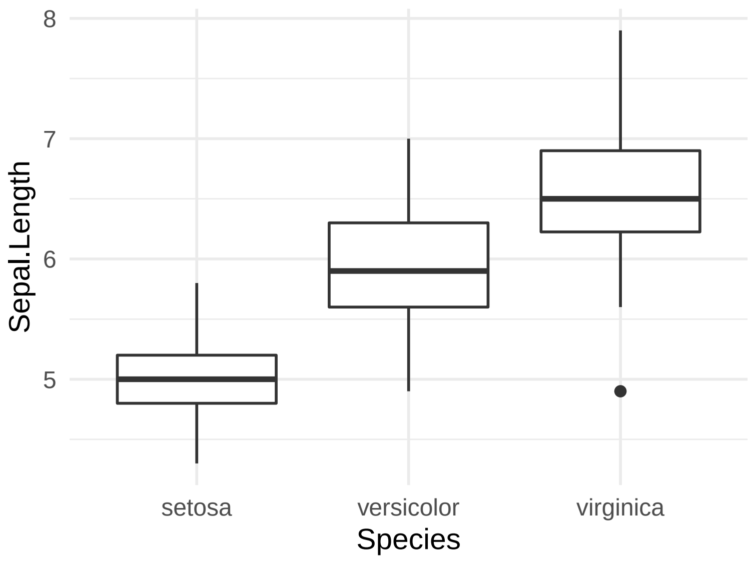
```r
ggplot(data=iris,aes(x=Sepal.Length, color = Species)) + geom_density()
```



Boxplot

The distribution of data can be shown with a boxplot. This a standardized way of displaying the distribution of data and it is based on a five number summary ("minimum", first quartile (Q1), median, third quartile (Q3), and "maximum").

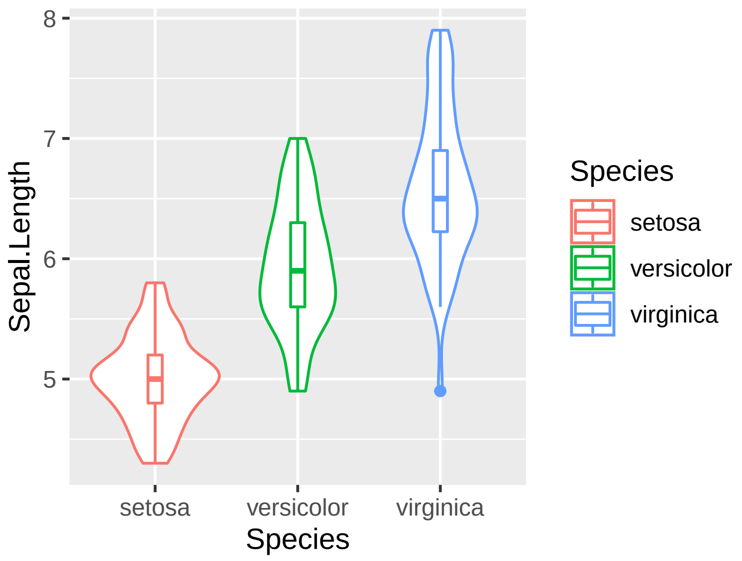
```r
gg1 <- ggplot(data=iris, mapping = aes(x = Species,y = Sepal.Length)) +
  geom_boxplot() +
  theme_minimal()
gg1
```



Violin plot

Another graphics useful to observe the distribution of data is a violin plot, a sort density plot that is rotated and placed on each side, to show the distribution shape of the data.

```
gg1 <- ggplot(data=iris, mapping = aes(x = Species,y = Sepal.Length, colour = Species)) +
  geom_violin()  +
  geom_boxplot(width = 0.1)
gg1
```



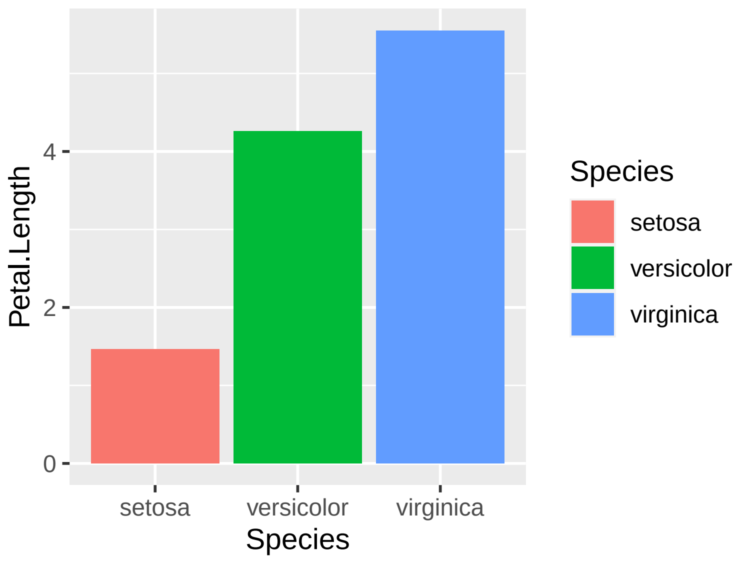
Barplot

Here we plot the average length of petals in the different species:

```
ggplot(data=iris,aes(x=Species,y=Petal.Length,fill=Species)) +
  geom_bar(stat = "summary", fun.y = "mean")
```

```
## Warning: Ignoring unknown parameters: fun.y
```

```
## No summary function supplied, defaulting to `mean_se()`
```

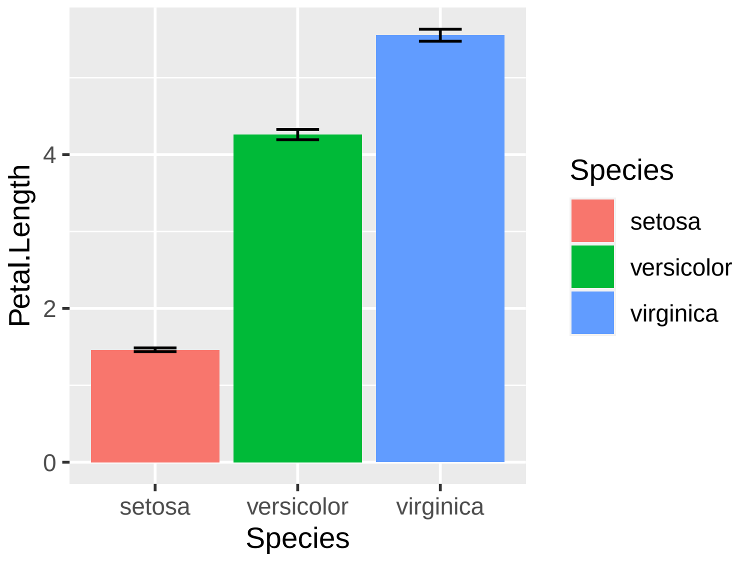


We can add error bars indicating the standard error of the mean

```
ggplot(data=iris,aes(x=Species,y=Petal.Length,fill=Species)) +
  geom_bar(stat = "summary", fun.y = "mean") +
  stat_summary(geom = "errorbar", fun.data = mean_se, width = 0.3)
```

```
## Warning: Ignoring unknown parameters: fun.y
```

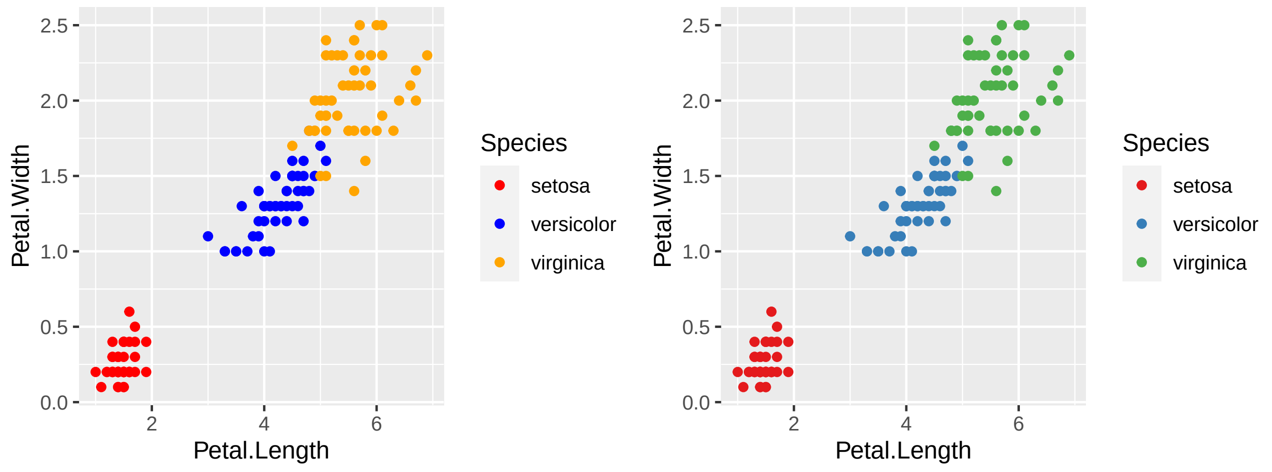
```
## No summary function supplied, defaulting to `mean_se()`
```



Customize plot

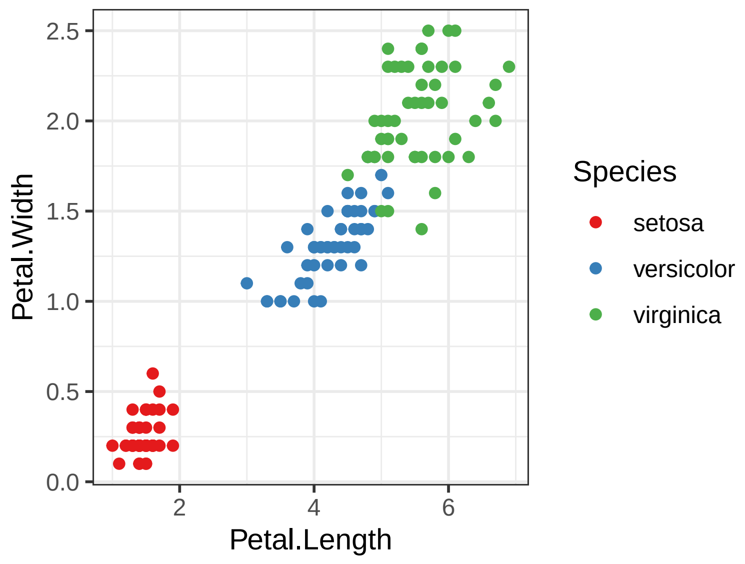
Lets go back to our scatter plot and try to customize it. - Change color: with custom colors (see http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf) or with RColorBrewer palettes.

```
gg1 <- ggplot(data=iris,aes(x=Petal.Length, y=Petal.Width, colour = Species)) +
  geom_point() +
  scale_colour_manual(values = c(setosa = "red", versicolor = "blue", virginica = "orange"))

gg2 <- ggplot(data=iris,aes(x=Petal.Length, y=Petal.Width, colour = Species)) +
  geom_point() +
  scale_colour_brewer(palette = "Set1")

# combine multiple plots (gg1 and gg2 in one row and two columns)
gridExtra::grid.arrange(gg1, gg2, nrow = 1, ncol = 2)
```

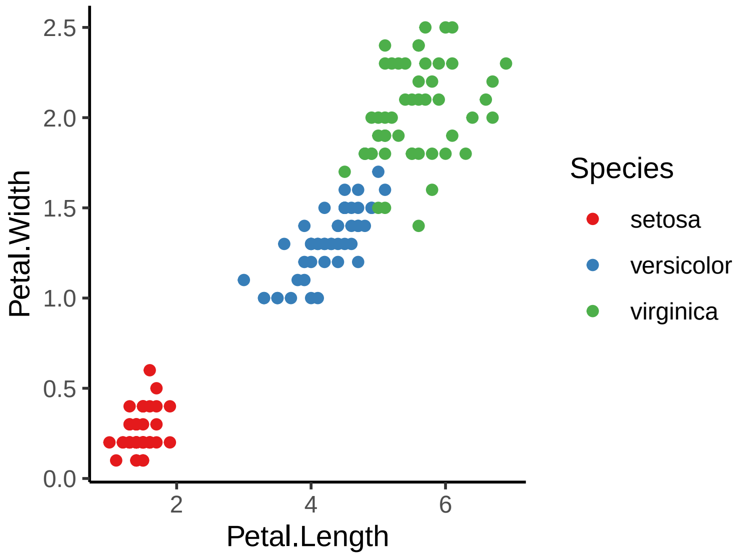


- Change theme

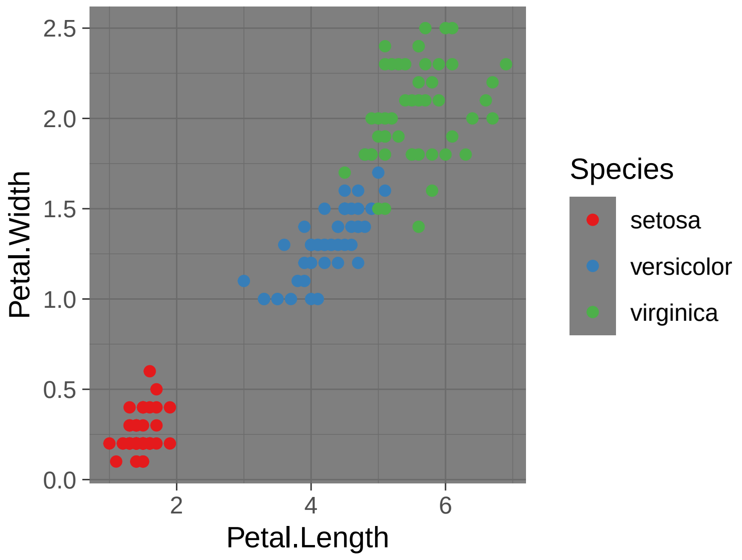
```
gg2 + theme_bw()
```



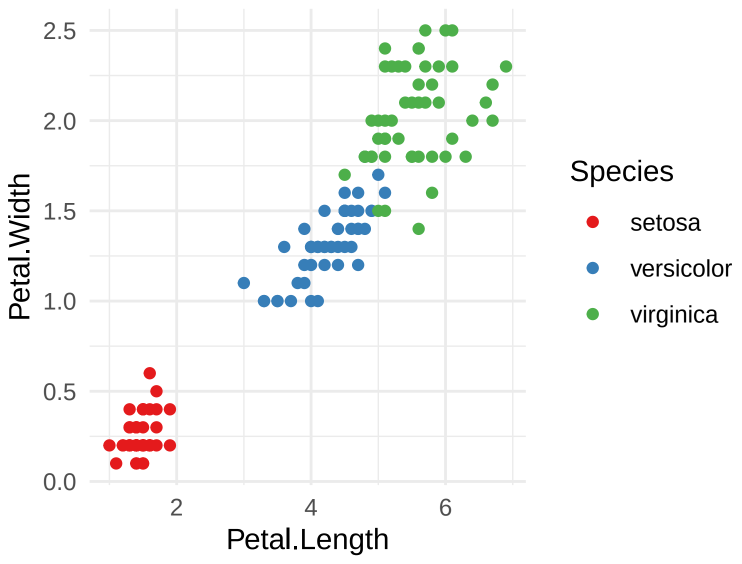
```
gg2 + theme_classic()
```



```r
gg2 + theme_dark()
```



```r
gg2 + theme_minimal()
```



GRanges to store Genomic Data