R training - Session 3

Rebecca G. Smith

05/09/2022

---

1. *Previously on R Workshops*: topics covered in last session

In R Workshop 2, we...

- Learned about datatypes
    - What datatypes exist

  - Examining datatypes
  - Converting datatypes
- Data structures
  - Vectors and lists
  - Matrices and dataframes
  - Indexing of data structures
  - Data structure exploration
- Factors
- Naming conventions

---

2. Learning Objectives

In this session, you will...

- Learn how to explore your data
  - How to subset data parts
  - Matching elements
  - Merging and binding your data
- Data manipulations
  - How to use the paste() function
  - How to rename your elements
  - How to use the seq() function
- Basic visualisation
  - Know what type of plots there are and how to make them
  - How to plot on top of plots
  - How to add to your plot
  - How to manipulate margins
  - how to create composite plots
  - How to export your plots

---

3. Set up

Set your working directory, either by navigating to "Session" -> "Set Working Directory" -> "Choose Directory..." and navigating to the directory. Alternatively you can set your working directory using the function "setwd" as below, and putting the path in quotation or apostrophes.

```
setwd("C:/Users/rgs212/OneDrive - University of Exeter/R Training/R workshop UoE")
```

Create and save a new script and name it "R training session 3".

---

4. Data

Read or load in the data for this session. We will be using the R data called "iris".

```
data(iris)
```

---

5. Topics

3.1 Data exploration

When we have our data loaded in, we may need to explore and manipulate it to get it into the format we require, remove sample, subset, match etc. There are several easy to use tools within R which can help us with these tasks.

Subsetting

Depending on the type of data you have (vector, data frame, matrix etc.), you can use tools to select parts of it using square bracket i.e. [ ]. These brackets have a lot of utility and allow us to select based on position and name. In order to know how to use them, we need to know how many dimensions the data has. If it is a vector, it has one dimension, if it is a data frame or matrix, it has two. This also informs how many arguments (separated by a comma) the brackets require for selection.

```
# Using a vector, we only need to give one selection and no comma
test <- 1:10
# Select the first value in the vector "test"
test[1]
```

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

```
# Using a matrix or data frame, we need to give two selections with a comma separating
# The number before the column is the row, the number after the comma is the column
# To select the value in the first row and second column
iris[1, 2]
```

```
## [1] 3.5
```

We can also leave one of the values blank, to select all values which meet the criteria.

```
# By leaving the column argument empty, we will select all columns which are in the first row
iris[1, ]
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
```

```
# By leaving the row argument empty, we will select all rows which are in the first columns
iris[, 3]
```

```
##   [1] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 1.5 1.6 1.4 1.1 1.2 1.5 1.3 1.4
##  [19] 1.7 1.5 1.7 1.5 1.0 1.7 1.9 1.6 1.6 1.5 1.4 1.6 1.6 1.5 1.5 1.4 1.5 1.2
##  [37] 1.3 1.4 1.3 1.5 1.3 1.3 1.3 1.6 1.9 1.4 1.6 1.4 1.5 1.4 4.7 4.5 4.9 4.0
##  [55] 4.6 4.5 4.7 3.3 4.6 3.9 3.5 4.2 4.0 4.7 3.6 4.4 4.5 4.1 4.5 3.9 4.8 4.0
##  [73] 4.9 4.7 4.3 4.4 4.8 5.0 4.5 3.5 3.8 3.7 3.9 5.1 4.5 4.5 4.7 4.4 4.1 4.0
##  [91] 4.4 4.6 4.0 3.3 4.2 4.2 4.2 4.3 3.0 4.1 6.0 5.1 5.9 5.6 5.8 6.6 4.5 6.3
## [109] 5.8 6.1 5.1 5.3 5.5 5.0 5.1 5.3 5.5 6.7 6.9 5.0 5.7 4.9 6.7 4.9 5.7 6.0
## [127] 4.8 4.9 5.6 5.8 6.1 6.4 5.6 5.1 5.6 6.1 5.6 5.5 4.8 5.4 5.6 5.1 5.1 5.9
## [145] 5.7 5.2 5.0 5.2 5.4 5.1
```

We can also use square brackets to select more than one value. Using "c()" and ":", we can select more than one value.

```
# Using ":" to select several values
# Below we select the values in the 5th, 6th, 7th and 8th row and the first three columns
iris[5:8, 1:3]
```

```
##   Sepal.Length Sepal.Width Petal.Length
## 5          5.0         3.6          1.4
## 6          5.4         3.9          1.7
## 7          4.6         3.4          1.4
## 8          5.0         3.4          1.5
```

```
# Using "c()" to select several values
# Below we select the values in the 3th, 8th and 10th row and the 2nd and 4th
iris[c(3, 8, 10), c(2, 4)]
```

```
##    Sepal.Width Petal.Width
## 3          3.2         0.2
## 8          3.4         0.2
## 10         3.1         0.1
```

We can also use names to select rows or columns, although this is most useful using column names.

```
# Using "c()" to select several values
# Using column name "Sepal.Width" to select values in this column, in the first three rows
iris[1:3, "Sepal.Width"]
```

```
## [1] 3.5 3.0 3.2
```

In data frames, we can use the "$" to select values in individual columns. important to note that this will not work in matrices.

```
# Using "$" and the column name "Petal.Length", we can select the values in this column
iris$Petal.Length
```

```
##   [1] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 1.5 1.6 1.4 1.1 1.2 1.5 1.3 1.4
##  [19] 1.7 1.5 1.7 1.5 1.0 1.7 1.9 1.6 1.6 1.5 1.4 1.6 1.6 1.5 1.5 1.4 1.5 1.2
##  [37] 1.3 1.4 1.3 1.5 1.3 1.3 1.3 1.6 1.9 1.4 1.6 1.4 1.5 1.4 4.7 4.5 4.9 4.0
##  [55] 4.6 4.5 4.7 3.3 4.6 3.9 3.5 4.2 4.0 4.7 3.6 4.4 4.5 4.1 4.5 3.9 4.8 4.0
##  [73] 4.9 4.7 4.3 4.4 4.8 5.0 4.5 3.5 3.8 3.7 3.9 5.1 4.5 4.5 4.7 4.4 4.1 4.0
##  [91] 4.4 4.6 4.0 3.3 4.2 4.2 4.2 4.3 3.0 4.1 6.0 5.1 5.9 5.6 5.8 6.6 4.5 6.3
## [109] 5.8 6.1 5.1 5.3 5.5 5.0 5.1 5.3 5.5 6.7 6.9 5.0 5.7 4.9 6.7 4.9 5.7 6.0
## [127] 4.8 4.9 5.6 5.8 6.1 6.4 5.6 5.1 5.6 6.1 5.6 5.5 4.8 5.4 5.6 5.1 5.1 5.9
## [145] 5.7 5.2 5.0 5.2 5.4 5.1
```

Matching

There are several circumstances we will need to check for matching and use that information. There are several ways we can do this using R depending on what we need.

Using identical(), we can check if values or collections of values are identical.

```
# Checking if the first and second row values in the "Species" column of iris are identical
identical(iris$Species[1], iris$Species[2])
```

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

```
# Checking if the first and 51st row values in the "Species" column of iris are identical
identical(iris$Species[1], iris$Species[51])
```

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

Using all.equal() is similar to identical(), but allows for some tolerance in how similar values can be. For example, we may want to check two numbers with lots of decimal places, but only need them to be within 0.01 of each other. Therefore we can give a tolerance of 0.01

```
x1 <- 1.232529
x2 <- 1.23366
all.equal(x1, x2, tolerance=0.01)
```

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

```
all.equal(x1, x2, tolerance=0.0001)
```

```
## [1] "Mean relative difference: 0.0009176255"
```

We can use "==" as a selector to pull all matching entries. We can give a numeric value or a character in quotations.

```
iris[iris$Species == "setosa", ]
```

```
##    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
## 7           4.6         3.4          1.4         0.3  setosa
## 8           5.0         3.4          1.5         0.2  setosa
## 9           4.4         2.9          1.4         0.2  setosa
## 10          4.9         3.1          1.5         0.1  setosa
## 11          5.4         3.7          1.5         0.2  setosa
## 12          4.8         3.4          1.6         0.2  setosa
## 13          4.8         3.0          1.4         0.1  setosa
## 14          4.3         3.0          1.1         0.1  setosa
## 15          5.8         4.0          1.2         0.2  setosa
## 16          5.7         4.4          1.5         0.4  setosa
## 17          5.4         3.9          1.3         0.4  setosa
## 18          5.1         3.5          1.4         0.3  setosa
## 19          5.7         3.8          1.7         0.3  setosa
## 20          5.1         3.8          1.5         0.3  setosa
## 21          5.4         3.4          1.7         0.2  setosa
## 22          5.1         3.7          1.5         0.4  setosa
## 23          4.6         3.6          1.0         0.2  setosa
## 24          5.1         3.3          1.7         0.5  setosa
## 25          4.8         3.4          1.9         0.2  setosa
## 26          5.0         3.0          1.6         0.2  setosa
## 27          5.0         3.4          1.6         0.4  setosa
## 28          5.2         3.5          1.5         0.2  setosa
## 29          5.2         3.4          1.4         0.2  setosa
## 30          4.7         3.2          1.6         0.2  setosa
## 31          4.8         3.1          1.6         0.2  setosa
## 32          5.4         3.4          1.5         0.4  setosa
## 33          5.2         4.1          1.5         0.1  setosa
## 34          5.5         4.2          1.4         0.2  setosa
## 35          4.9         3.1          1.5         0.2  setosa
## 36          5.0         3.2          1.2         0.2  setosa
## 37          5.5         3.5          1.3         0.2  setosa
## 38          4.9         3.6          1.4         0.1  setosa
## 39          4.4         3.0          1.3         0.2  setosa
## 40          5.1         3.4          1.5         0.2  setosa
## 41          5.0         3.5          1.3         0.3  setosa
## 42          4.5         2.3          1.3         0.3  setosa
## 43          4.4         3.2          1.3         0.2  setosa
## 44          5.0         3.5          1.6         0.6  setosa
## 45          5.1         3.8          1.9         0.4  setosa
## 46          4.8         3.0          1.4         0.3  setosa
## 47          5.1         3.8          1.6         0.2  setosa
## 48          4.6         3.2          1.4         0.2  setosa
## 49          5.3         3.7          1.5         0.2  setosa
## 50          5.0         3.3          1.4         0.2  setosa
```

We can use objects or parts of objects to select rows and columns within [ ] using the "%in%".

```r
select <- "versicolor"
iris[iris$Species %in% select, ]
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 51           7.0         3.2          4.7         1.4 versicolor
## 52           6.4         3.2          4.5         1.5 versicolor
## 53           6.9         3.1          4.9         1.5 versicolor
## 54           5.5         2.3          4.0         1.3 versicolor
## 55           6.5         2.8          4.6         1.5 versicolor
## 56           5.7         2.8          4.5         1.3 versicolor
## 57           6.3         3.3          4.7         1.6 versicolor
## 58           4.9         2.4          3.3         1.0 versicolor
## 59           6.6         2.9          4.6         1.3 versicolor
## 60           5.2         2.7          3.9         1.4 versicolor
## 61           5.0         2.0          3.5         1.0 versicolor
## 62           5.9         3.0          4.2         1.5 versicolor
## 63           6.0         2.2          4.0         1.0 versicolor
## 64           6.1         2.9          4.7         1.4 versicolor
## 65           5.6         2.9          3.6         1.3 versicolor
## 66           6.7         3.1          4.4         1.4 versicolor
## 67           5.6         3.0          4.5         1.5 versicolor
## 68           5.8         2.7          4.1         1.0 versicolor
## 69           6.2         2.2          4.5         1.5 versicolor
## 70           5.6         2.5          3.9         1.1 versicolor
## 71           5.9         3.2          4.8         1.8 versicolor
## 72           6.1         2.8          4.0         1.3 versicolor
## 73           6.3         2.5          4.9         1.5 versicolor
## 74           6.1         2.8          4.7         1.2 versicolor
## 75           6.4         2.9          4.3         1.3 versicolor
## 76           6.6         3.0          4.4         1.4 versicolor
## 77           6.8         2.8          4.8         1.4 versicolor
## 78           6.7         3.0          5.0         1.7 versicolor
## 79           6.0         2.9          4.5         1.5 versicolor
## 80           5.7         2.6          3.5         1.0 versicolor
## 81           5.5         2.4          3.8         1.1 versicolor
## 82           5.5         2.4          3.7         1.0 versicolor
## 83           5.8         2.7          3.9         1.2 versicolor
## 84           6.0         2.7          5.1         1.6 versicolor
## 85           5.4         3.0          4.5         1.5 versicolor
## 86           6.0         3.4          4.5         1.6 versicolor
## 87           6.7         3.1          4.7         1.5 versicolor
## 88           6.3         2.3          4.4         1.3 versicolor
## 89           5.6         3.0          4.1         1.3 versicolor
## 90           5.5         2.5          4.0         1.3 versicolor
## 91           5.5         2.6          4.4         1.2 versicolor
## 92           6.1         3.0          4.6         1.4 versicolor
## 93           5.8         2.6          4.0         1.2 versicolor
## 94           5.0         2.3          3.3         1.0 versicolor
## 95           5.6         2.7          4.2         1.3 versicolor
## 96           5.7         3.0          4.2         1.2 versicolor
## 97           5.7         2.9          4.2         1.3 versicolor
## 98           6.2         2.9          4.3         1.3 versicolor
## 99           5.1         2.5          3.0         1.1 versicolor
```

```
## 100          5.7         2.8          4.1         1.3 versicolor
```

Merging and binding

We will often need to bring multiple two-dimensional objects together. We can do this multiple ways. Using rbind() and cbind(), we can combine objects together. rbind() allows us to bind together rows.

```
# First we look at the dimension of "iris"

dim(iris)
```

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

```
# Using rbind() to put together two copies of  causing double rows
rbind(iris, iris) -> iris.r
dim(iris.r)
```

```
## [1] 300   5
```

```
iris.r[ ,1]
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
##  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
##  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
##  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
##  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8
## [163] 4.8 4.3 5.8 5.7 5.4 5.1 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7
## [181] 4.8 5.4 5.2 5.5 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6
## [199] 5.3 5.0 7.0 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7
## [217] 5.6 5.8 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0
## [235] 5.4 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8
## [253] 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0
## [271] 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4
## [289] 6.0 6.9 6.7 6.9 5.8 6.8 6.7 6.7 6.3 6.5 6.2 5.9
```

cbind() allows us to bind together columns.

```
# First we look at the dimension of "iris"

dim(iris)
```

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

```r
# Using cbind() to put together two copies of iris causing double columns
cbind(iris, iris) -> iris.c
dim(iris.c)
```

```
## [1] 150  10
```

```r
iris.c[1,]
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal.Length
## 1          5.1         3.5          1.4         0.2  setosa          5.1
##   Sepal.Width Petal.Length Petal.Width Species
## 1         3.5          1.4         0.2  setosa
```

Using merge(), we can merge objects together assigning what we bind by using "by =". For example, we can bind using the rownames of our objects using "by = row.names", we can merge by a specific column present in both objects (e.g. by = "Name"), or different columns in each object (by.x = "Species", by.y = "Name").

```r
# First we look at the dimension of "iris"

dim(iris)
```

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

```r
# Merging iris by row names
merge(iris, iris, by = "row.names") -> iris.double
dim(iris.double)
```

```
## [1] 150  11
```

```r
iris.double[1, ]
```

```
##   Row.names Sepal.Length.x Sepal.Width.x Petal.Length.x Petal.Width.x Species.x
## 1         1            5.1           3.5            1.4           0.2    setosa
##   Sepal.Length.y Sepal.Width.y Petal.Length.y Petal.Width.y Species.y
## 1            5.1           3.5            1.4           0.2    setosa
```

Tasks

1. Display the following from "iris"
    a. 14th row
    b. 4th column
    c. 120th row of 2nd column
    d. 4th, 6th, 10th and 12th row
    e. 111th, 112th and 113th row of "Species" column

2. Work out if the following match
    a. 9th and 13th row of "Petal.Length"
    b. 145th and 146th row of "Sepal.Length"
    c. 150th row of "Sepal.Length" and 103rd row of "Petal.Length"
    d. 13th and 108th row of "Sepal.Width"
    e. 50 divided by 55 and 10 divided by 12
    f. 50 divided by 55 and 10 divided by 12 to a tolerance of 0.1

3. Create two objects, one containing numbers 1-10, one containing numbers 11-20
    a. Bind them together to make an object of two rows, row 1 being 1:10, row 2 being 11-20
    b. Bind them together to make an object of two columns, columns 1 being 11-20, column 2 being 1-10

5.2 Data manipulation

Paste

The function paste() is a way to concatenating together vectors. It can be applied to a characters or numbers, vector and column(s) of a data frame or matrix. You can define what you want the separator is (sep =), or use paste0() or paste() with sep = " " for no space. You can also give text to add to a character or vector.

```
# Adding text to a value in iris
paste("Species", iris$Species[1])
```

```
## [1] "Species setosa"
```

```
# Pasting together two columns of iris
paste(iris$Species[1:10], iris$Sepal.Length[1:10], sep = ":")
```

```
##  [1] "setosa:5.1" "setosa:4.9" "setosa:4.7" "setosa:4.6" "setosa:5"
##  [6] "setosa:5.4" "setosa:4.6" "setosa:5"   "setosa:4.4" "setosa:4.9"
```

Renaming columns and rows

By using rownames() and colnames(), we can look at what the rownames and colnames of an object are. We can also use this to replace the rownames and colnames of the object by assigning using "<-".

```
# Renaming the colnames in iris
iris -> iris2
colnames(iris2)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```
colnames(iris2) <- c("S.Length", "S.Width", "P.Length", "P.Width", "Type")
colnames(iris2)
```

```
## [1] "S.Length" "S.Width"  "P.Length" "P.Width"  "Type"
```

Adding and removing variables

Adding data to your objects can be very useful. Adding an extra column is very easy by assigning what you want to a new column.

```
# Adding a new column
iris -> iris2
iris2$new.column <- 1:nrow(iris2)
```

Removing a column can be done by using the "NULL" value. on the column of interest.

```
# Removing a column
iris2$new.column <- NULL
```

---

seq()

To generate regular sequences, we can use seq(). We provide it a value to start from (from =), and where to end (to =) abd then a value to increase by (by =).

```
# Create a sequence from 0 to 100 increasing by 5 each time
seq(from = 0, to = 100, by = 5)
```

```
##  [1]   0   5  10  15  20  25  30  35  40  45  50  55  60  65  70  75  80  85  90
## [20]  95 100
```

Tasks

1. Create a copy of "iris"
    a. Rename the columns of "iris" by prefixing with the word "flower" and the separator "_"
    b. In your copy, duplicate the "Species" column
    c. Add a column to your copy which contains numbers between 4 and 600 increased by 4 each time

---

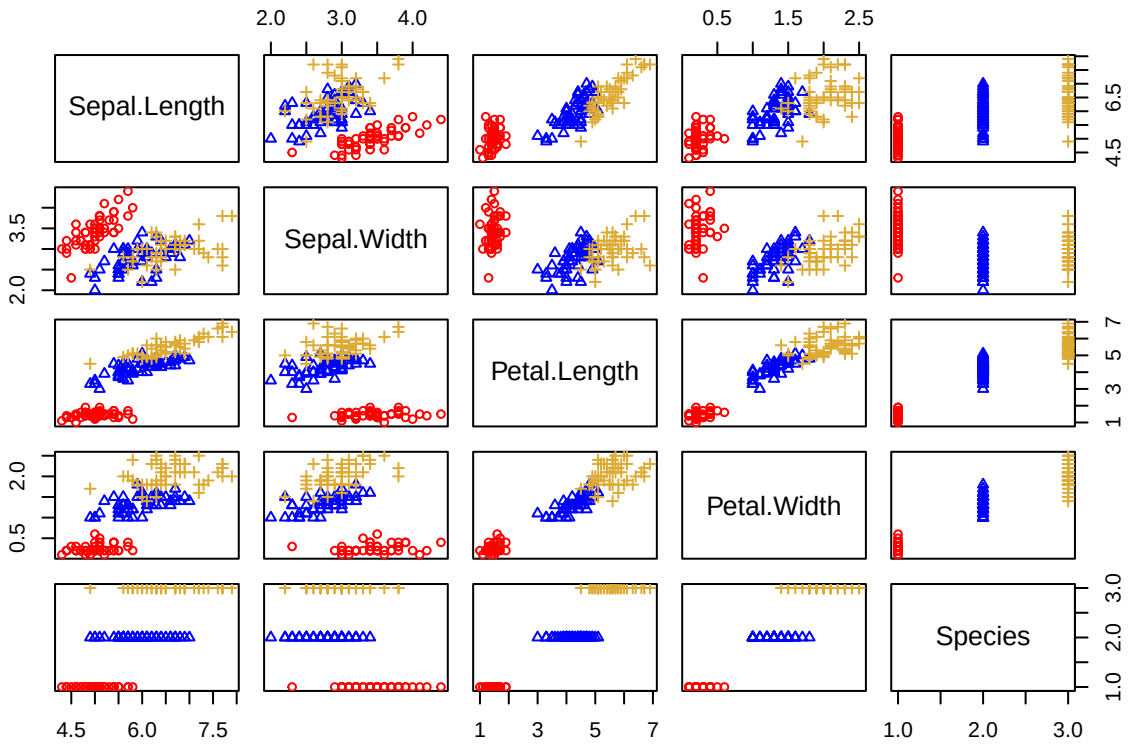
5.3 Basic visualisations

Types of plots

R is capable of plotting many different types of plots and many are possible through simple plotting techniques. Depending on the data we are attempting to visualize, the type of plot we may want to use will vary. R will automatically try and plot the most appropriate plot for the data provided. You can also define to make box plots (boxplot()), histograms (hist()) and bar plots (barplot()).

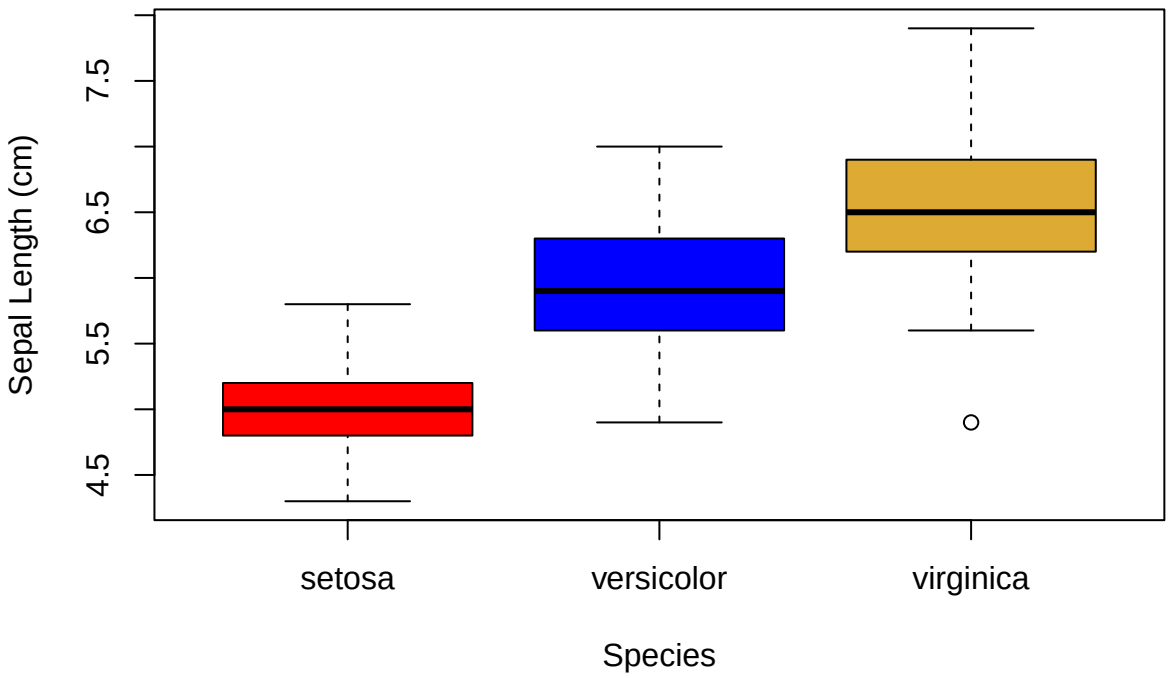
There are also many arguments we can give to R for the plots we want to make.

```
# Plotting all the iris data
plot(iris, col = c("red", "blue", "#ddaa33")[as.numeric(iris$Species)],
     pch = c(1, 2, 3)[as.numeric(iris$Species)], cex = 0.8)
```



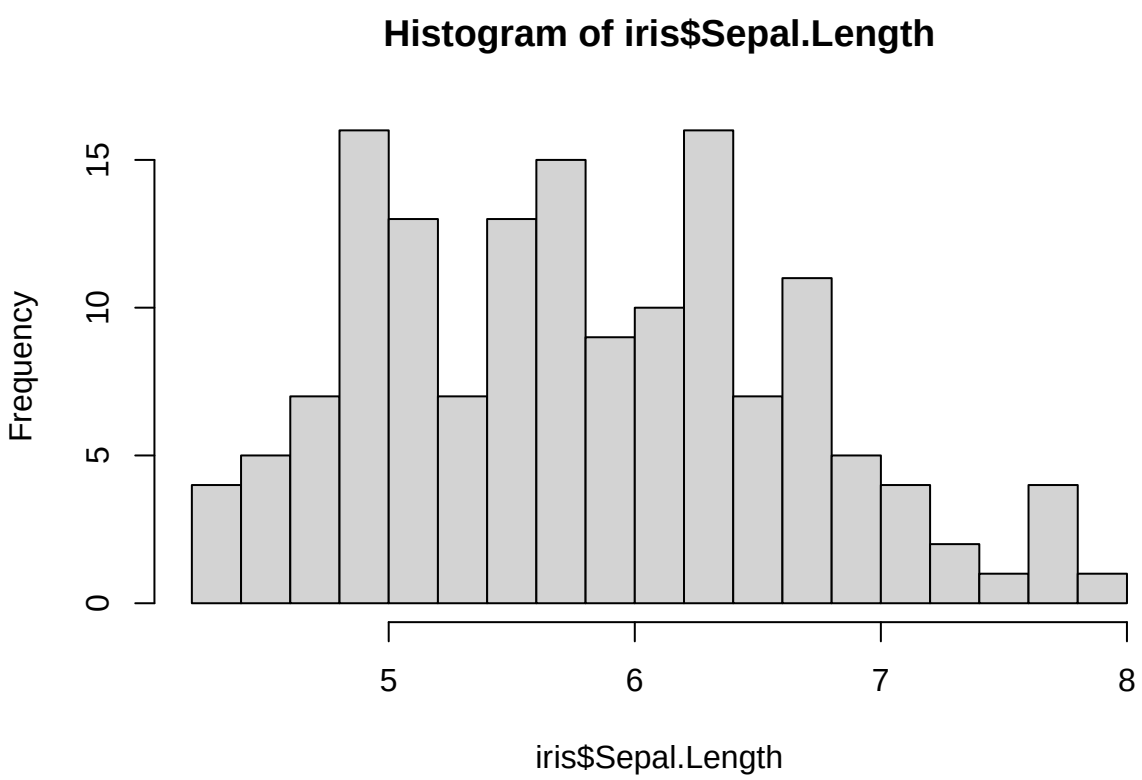
Box plots can be made using the boxplot() function. It uses a lot of the standard plotting functions.

```
plot(iris$Sepal.Length ~ iris$Species, ylab = "Sepal Length (cm)", xlab = "Species",
     col =  c("red", "blue", "#ddaa33"))
```



Histograms are plotted using the function hist(), which allows us to plot the frequency of a vector. You can use standard plotting arguments such as col, but also used the argument breaks to adjust the amount of bins.

```
# Plotting "Sepal.Length" with increased bins
hist(iris$Sepal.Length, breaks = 16)
```



Below you can see a table containing a lot of basic plotting arguments. Also, for colour selection, making colour themes and looking for colour blind options, you can use https://www.colorhexa.com/ which will give you R friendly colour codes.

Argument	Description	Example
bg	The color to be used for the background	bg = “blue”
cex	Character size and expansion	cex = 1.5, cex = 0.8
cex.axis	The magnification to be used for axis annotation	cex.axis = 1.2
cex.lab	he magnification to be used for x and y label	cex.lab = 0.8
cex.main	The magnification to be used for main titles	cex.main = 1.3
cex.sub	The magnification to be used for sub-titles	cex.sub = 0.9
col	Colour	col = “red”, col = “#ff0000”
family	Font on the plot	family = “Arial”
fg	The colour to be used for the foreground of plots	fg = “orange”
font	An integer which specifies which font to use for text. Italic, bold etc.	font = 3
font.axis	The font to be used for axis annotation	font.axis = 2
font.lab	The font to be used for x and y labels	font.lab = 3
font.main	The font to be used for plot main titles	font.main = 2
font.sub	The font to be used for plot sub-titles	font.sub = 2
lty	Line type	lty = 2
lwd	Line width	lwd = 3
main	Plot primary title	main = “Iris”
pch	Scatter plot symbol for points	pch = 1, pch= “p”
srt	The string rotation in degrees	srt = 90

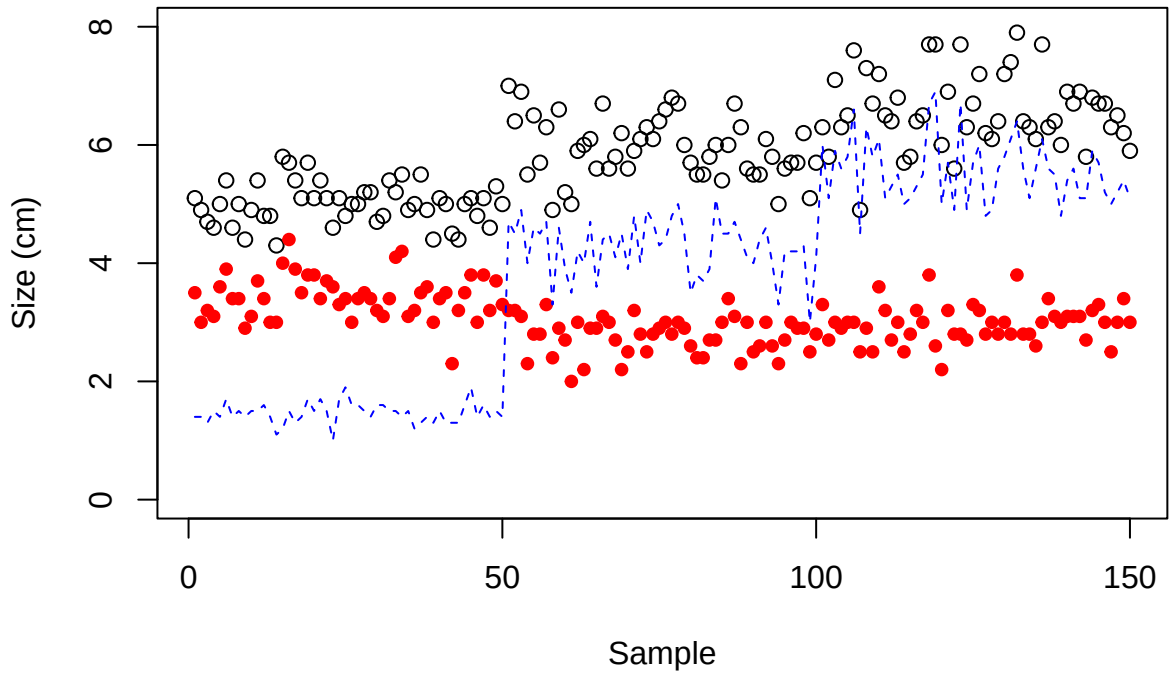
Argument	Description	Example
sub	Subtitle of plot	sub = “All data”
xlab, ylab	Label of the x or y axis	xlab = “Distance (Miles)”
xlim, ylim	Min/max x or y axis values	xlim = c(0, 10)
xpd	If true, allows plotting outside the plot	xpd = TRUE

Plots on plots

Once you have created a plot, there are various methods which allow us to add more data on top. For example, we may want to add a individual data points on top of a box plot, or extra points to a plot. To do this we can use functions such as points(), lines() or abline().

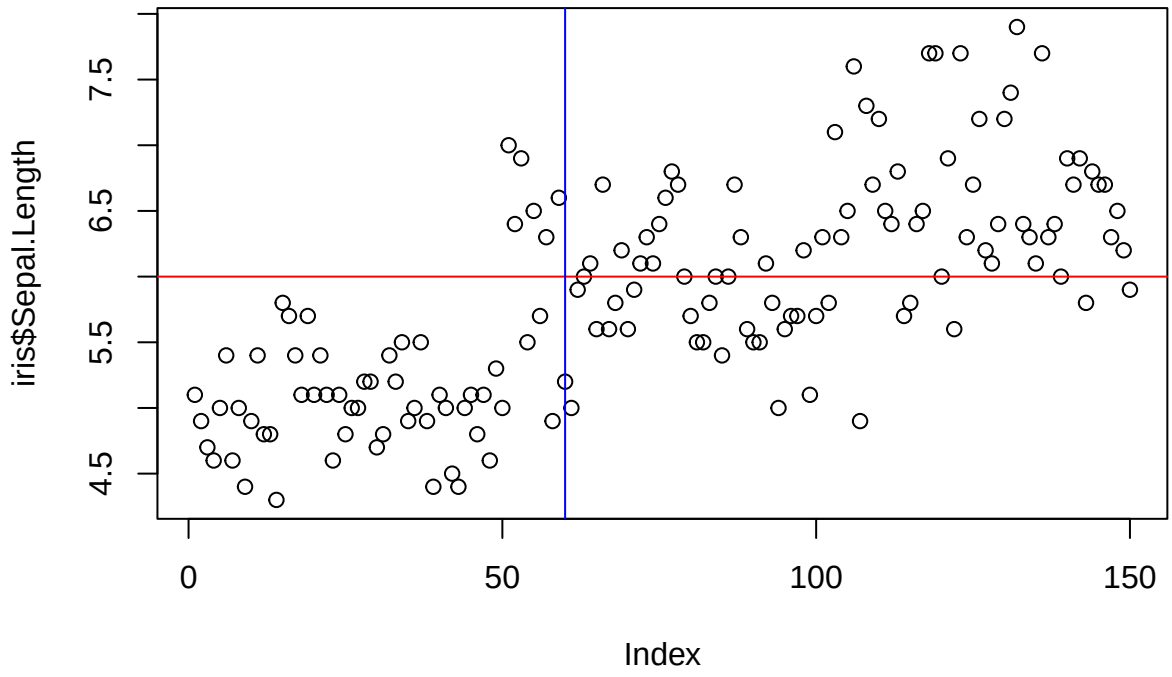
Using points() and lines(), you can add more data to your plot. They use similar arguments to plot(), such as col, lty, pch, cex etc.

```
# Plot "Sepal.Length" on a plot, then add "Sepal.Width" data as points and "Petal.Length" as lines
plot(iris$Sepal.Length, ylim = c(0, 8), ylab = "Size (cm)", xlab = "Sample")
points(iris$Sepal.Width, col = "red", pch = 19, cex = 0.8)
lines(iris$Petal.Length, lty = 2, col = "blue")
```

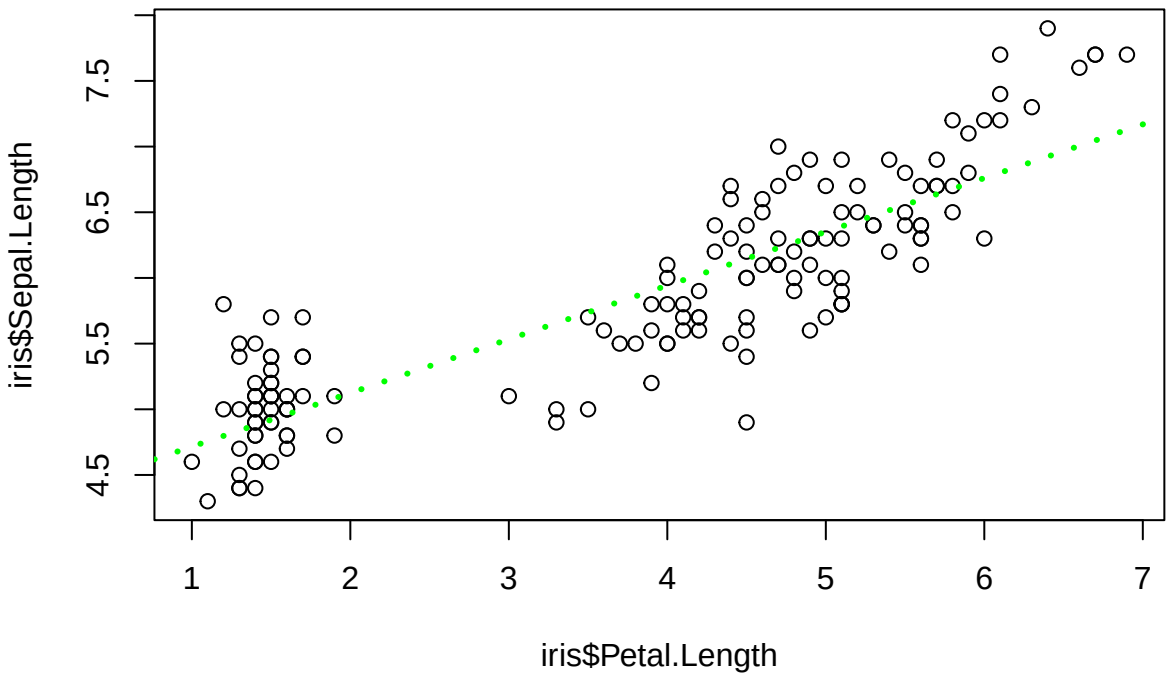


using abline(), you can add horizontal lines (h=), vertical lines (v=), diagonal or correlations (x, y). It can also be wrapped around a linear regression (lm()) to add a line of best fit.

```r
# Adding a horizontal line at 6 and a vertical line at 60
plot(iris$Sepal.Length)
abline(h = 6, col = "red")
abline(v = 60, col = "blue")
```




```r
# Plotting "Sepal.Length" against "Petal.Length" and adding a line of best fit
plot(iris$Sepal.Length ~ iris$Petal.Length)
abline(lm(iris$Sepal.Length ~ iris$Petal.Length), lty = 3, col = "green", lwd = 3)
```

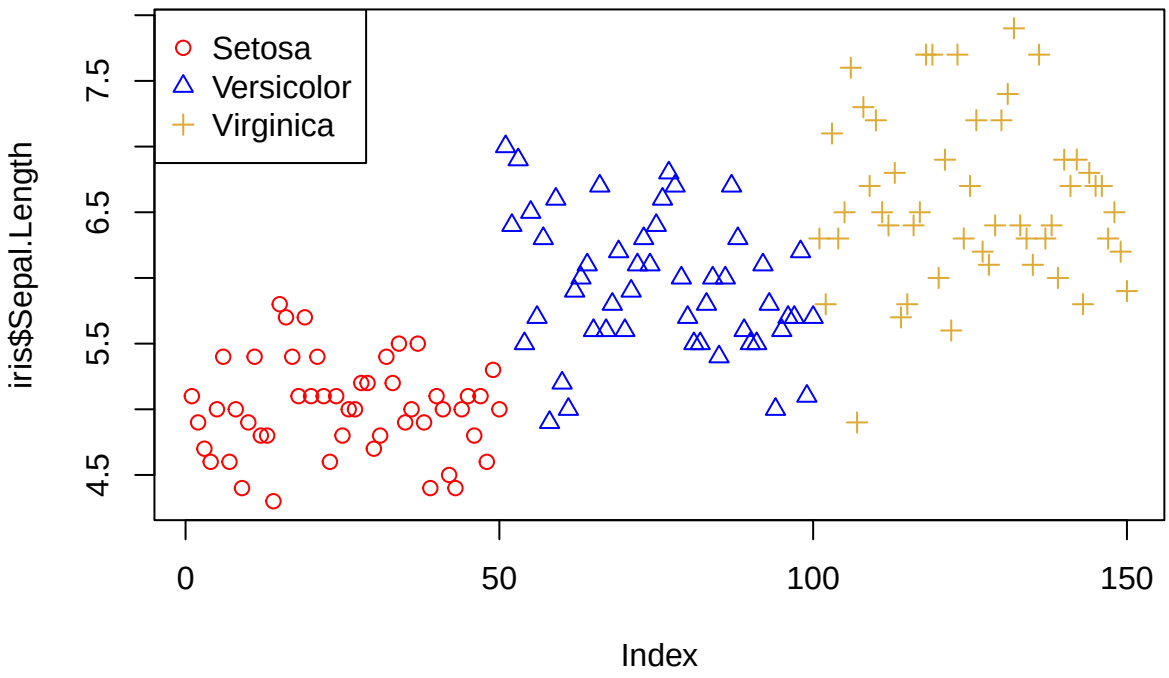


Plot additions

We can make many additions to our plots. As there are so many, we will only explore the most common ones here but will list additional ones which may be useful in the future.

We often need to add a legend to our plot, and can do this using legend(). The legend() function allows us to define position, either by using x, y coordinates or position such as "topleft". We also provide the text, colours and background.

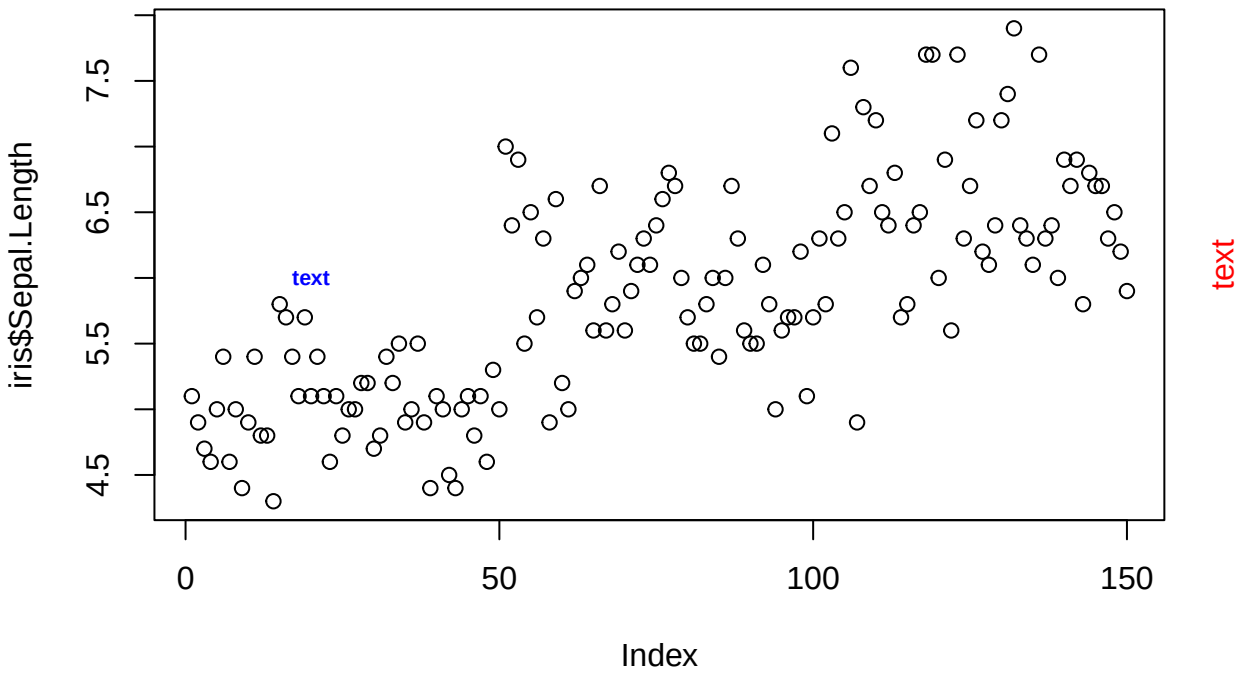
```
# Plotting "Sepal.Length" and colouring and using different points by "Species", then adding a legend
plot(iris$Sepal.Length, col = c("red", "blue", "#ddaa33")[as.numeric(iris$Species)],
     pch = c(1, 2, 3)[as.numeric(iris$Species)])
legend("topleft", legend = c("Setosa", "Versicolor", "Virginica"),
       pch = c(1,2,3), col = c("red", "blue", "#ddaa33"))
```



We can add additional text to a plot by using text() or mtext() for putting text in the margin. To use text(), we provide x, y coordinates, the text to be written (labels =), size (cex =), and colour (col =) and font (font =).

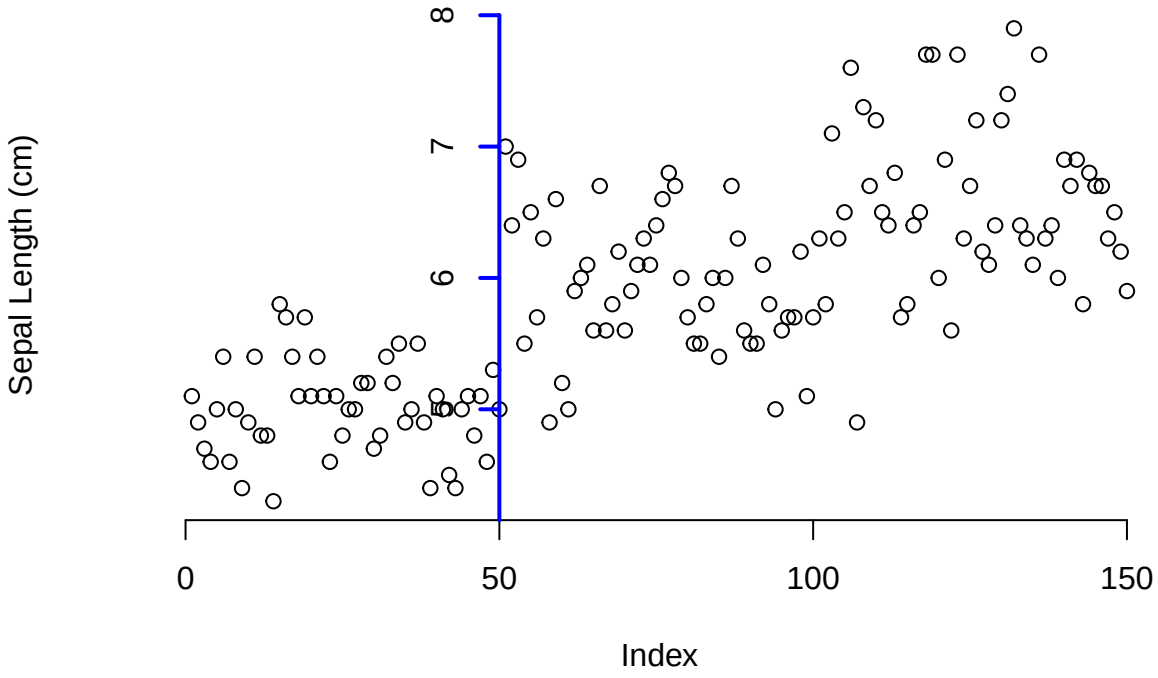
Using mtext() requires different arguments as it is relation to the margin side we put the text in. It requires the text (text =), the side of the plot the text will go (side =) with 1 = bottom, 2 = left, 3 = top, 4 = right, the margin line to put the text on (line =).

```
# Adding text to a plot
plot(iris$Sepal.Length)
text(20, 6, "text", cex = 0.7, col = "blue", font = 2)
mtext("text", side = 4, line = 1, col = "red")
```



There are incidences when we may want to add additional axes or move and adjust the axes of a plot. For this, we use axis(), and may add the argument to plot() to remove axes from our plot (axes = F) so we can draw them. Arguments for axis() are which side you want the axis (side =) using 1 = below, 2 = left, 3 = above and 4 = right, the point at which tick-marks are drawn (at =), what the labels are (labels =), how far from the axis the ticks should extend (line =), the position of the axis (pos =) and if tick marks should be drawn (tick =). You can also change the line width (lwd =), colour (col =) and type of line (lty =).

```
# Plotting "Sepal.Length" without axes and adding them with adjustment
plot(iris$Sepal.Length, axes = F, ylab="Sepal Length (cm)")
axis(1)
axis(2, pos = 50, at = 1:8, lwd = 2, col = "blue")
```



Additional functions to add to your plots are segments(), arrows(), curve(), rect(), polygon() and grid().

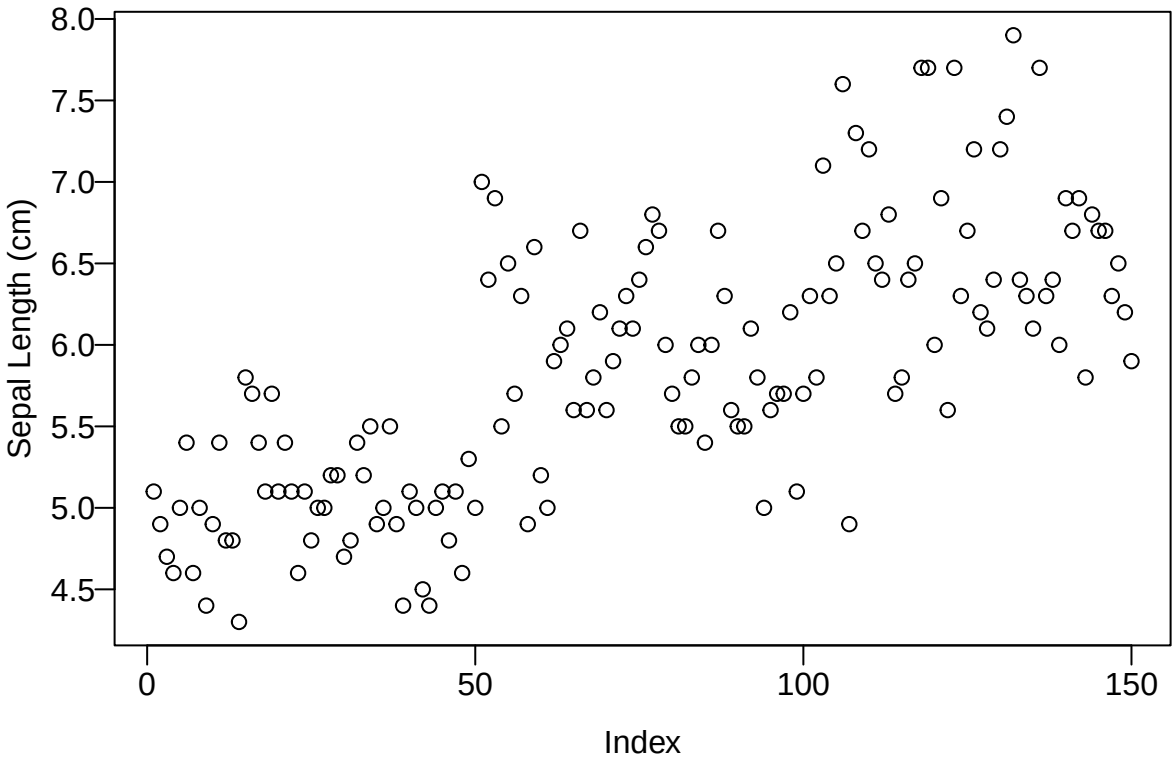
Margins

Using par(), mar() and mgp() before the plot, we can adjust the margins of our plot. Using mar() allows us to adjust the margins giving the number of lines of margin, with the default being c(5, 4, 4, 2) + 0.1 relating to bottom, left, top and right respectively.

Using mgp() sets the axis label locations relative to the edge of the inner plot window. The first value represents the location the axis label, the 2nd the tick-mark labels, and 3rd the tick marks. The default is c(3, 1, 0).

Using las() allows us to define the orientation of the tick mark labels and any other text added to a plot. The options are parallel to the axis (the default, 0), always horizontal (1), always perpendicular to the axis (2), and always vertical (3).

```
par(mar = c(4, 4, 2, 1), mgp = c(2, 0.5, 0), las = 1)
plot(iris$Sepal.Length, ylab="Sepal Length (cm)")
```

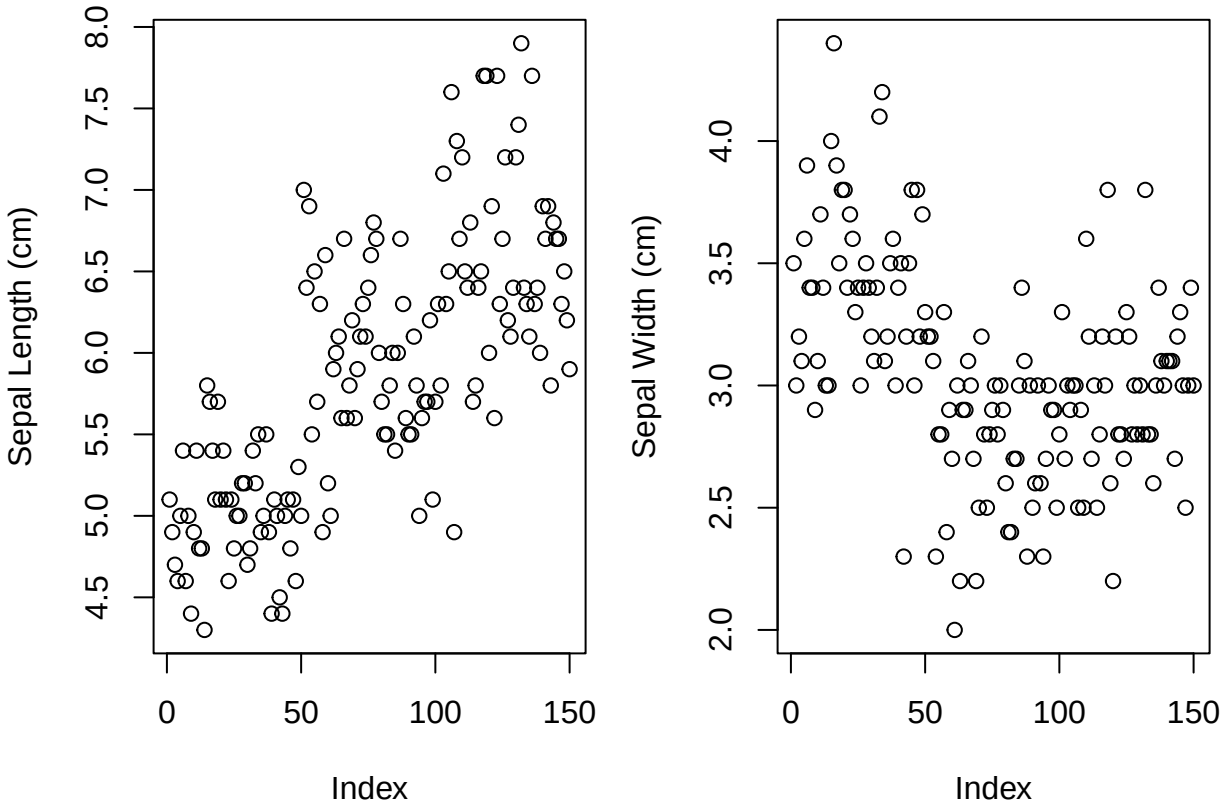


Composites

There are two options for making composite plots, or plots with multiple plots as panels. These are par(mfrow/mfcol =) and layout().

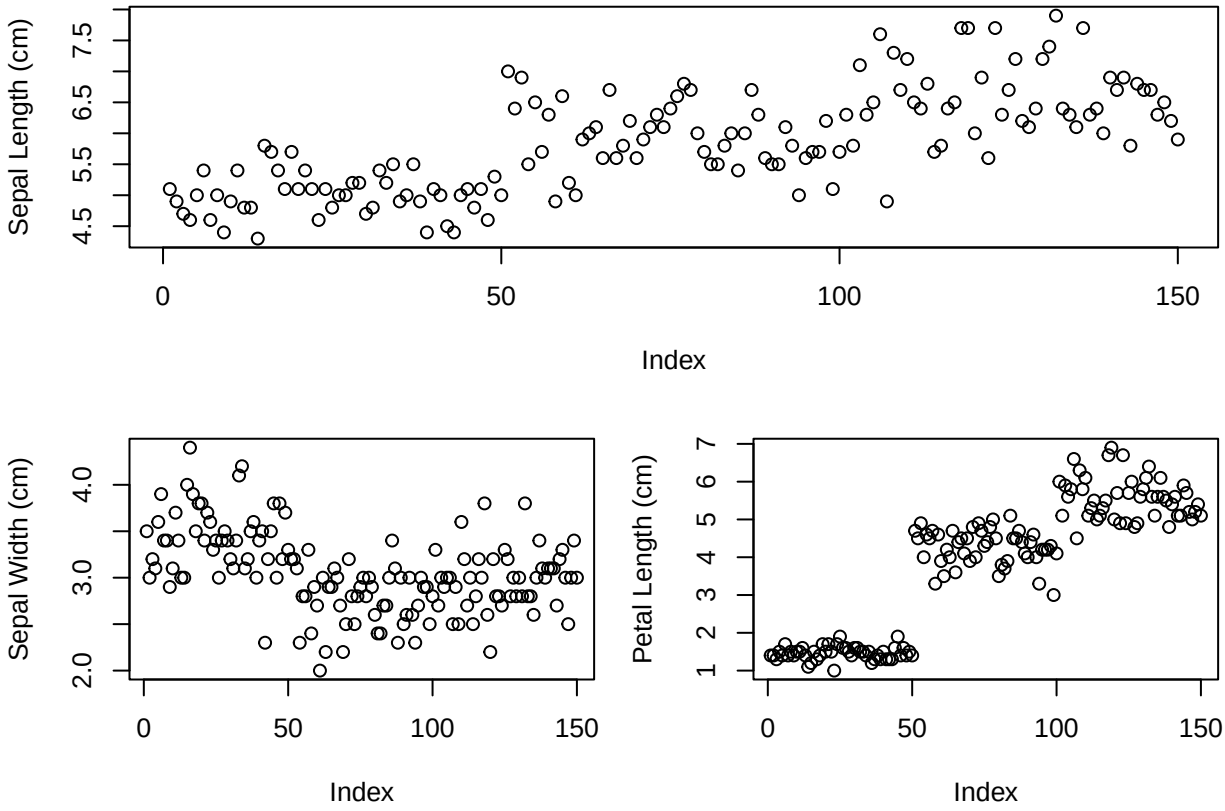
The arguments for par(mfrow/mfcol =), allows us to define how many rows and columns to make the composite of. Using mfcol draws by columns and mfrow draws by rows.

```
par(mar = c(4, 4, 2, 1))
par(mfrow = c(1, 2))
plot(iris$Sepal.Length, ylab="Sepal Length (cm)")
plot(iris$Sepal.Width, ylab="Sepal Width (cm)")
```



The arguments for layout() are more complicated but allows for unequal sizes of composite sections. The matrix argument allows you to define what plots (in the order of plotting under the layout function). The following two numbers are the number of rows and columns in our composite, respectively. The byrow argument, if true, will add plots by row, if false, it will add by column.

```
par(mar = c(4, 4, 2, 1))
layout(matrix(c(1, 1, 2, 3), 2, 2, byrow = TRUE))
plot(iris$Sepal.Length, ylab = "Sepal Length (cm)")
plot(iris$Sepal.Width, ylab = "Sepal Width (cm)")
plot(iris$Petal.Length, ylab = "Petal Length (cm)")
```



Saving and exporting plot

We can save our plots as various formats using the appropriate function. These include pdf (pdf()), jpeg (jpeg()), png (png()) and tiff (tiff()). These functions, with their arguments, will appear before the plot information, and will be succeeded by dev.off(). The first argument for any of these functions should be the path and file name to save the file under. You can also define the size of the image (width =, height =, units =), background colour (bg =) and resolution (res =).

```
# Saving a plot as a jpeg file
jpeg("exampleplot.jpg", width = 300, height = 300, units = "mm", bg = "white", res = 200)
plot(iris$Sepal.Length, ylab = "Sepal Length (cm)")
dev.off()
```

```
## pdf
##   2
```

Tasks

1. Make two objects, one object containing values 1-20, and another object containing values 40-21
    a. Using your objects, create a plot with the object containing 1:20 on the x-axis and the object containing 40-21 on y-axis

b. Change the x-axis label to "Independent Variable" and y-axis label to "Dependent Variable"
c. Expand both axes to show values 1-40
d. Increase the data point size, change their style and make them repeat between five colours
e. Add a legend top the top right of the plot showing the five colours you have chosen and labelling them A, B, C, D, and E
f. Add a horizontal line at 30, choose a colour, weight and style
g. Add a vertical line at 10, choose a different colour, weight and style
h. Add text saying "Cross Point" to the top right of the intersection of the two lines. Adjust the colour and size

2. Using "iris", create a scatter plot with "Sepal.Length" on x-axis, labelled "Sepal length (cm)", and the other three variables plotted on y-axis, with the label being "Size (cm)"

   a. Colour the three species differently and make the three measures different style of points
   b. Add a legend to show all the groups, and make sure it doesn't cover any points
   c. Make sure the x-axis limits are 0-8 and y-axis limit is 4-8
   d. Adjust the margins to give a larger space around the edge of the plot and move the axis labels a little away from the axes
   e. Export the image as a pdf

3. Create a composite plot with the following panels using "iris". Make the plots colourful and variable, that all points are visible in plotting window, axes have labels and measurement units. Export the plot as a high resolution (200) jpeg. Make sure the points and text are readable, and all info is visible. You may need to adjust margins

   a. Box plot of "Petal.Length" by species coloured by species
   b. Histogram of "Petal.Length" with 6 breaks, each one coloured differently, with a line added for the mean (hint: mean), coloured and labelled "Mean = X" where "X" is the mean value. Make this panel take more space
   c. "Petal.Length" against "Petal.Width" for just "virginica" species, with a line of best fit (hint: subsetting)

---

6. *Next on R Workshops*: topics covered in next workshop

In the next R Workshop, you will learn...

- Summary statistics
- Normal and other distributions
- Confidence Intervals
- Random Sampling
- Basic analysis
  - Correlation
  - T-test
  - ANOVA

---

7. Resources

- Basic R cheat sheet
- Data Carpentry Introduction to R
- Software Carpentry: Programming with R.
- R for Data Science
- ColorHexa
- R Base Graphics Cheatsheet

---

8. Acknowledgements

Inspired by and adopted from:

- John Blischak, Daniel Chen, Harriet Dashnow, and Denis Haine (eds): "Software Carpentry: Programming with R." Version 2016.06, June 2016, https://github.com/swcarpentry/r-novice-inflammation, 10.5281/zenodo.57541.
- Jenny Bryan: https://jennybc.github.io/purrr-tutorial
- Manuel Gimond: https://mgimond.github.io/ES218/Week02a.html
- Damaris Zurell (Ecology & Macroecology Lab, Univ. Potsdam 2020-2022): https://damariszurell.github.io/EEC-R-prep/index.html

---

9. License