Hera are a few examples using the Iris dataset again:

```
summary(iris)

boxplot(iris$Petal.Length)

boxplot(iris[, 1:4])

barplot(iris$Petal.Length) # for data #
exploration only

plot(iris$Petal.Width,iris$Petal.Length)
```

Not surprisingly, the scatterplot shows that irises with greater petal width also tend to have a larger sepal width too.

```
### Can we improve the plot a bit?

plot(iris$Petal.Width, iris$Petal.Length,
main="Iris Data")

### let’s add a legend...
```

Note that, in addition to scatterplots, the plot function is used to generate the default graphic for the specified object. Let’s give it a try…

Histograms

Histogram with a custom x-axis label and title

```
x <- rnorm(100, mean = 100, sd = 10)
hist(x, xlab = "x-values",
  main = "Histogram of 100 observations from
Normal(100, 10^2) distribution")
```

Boxplots

Single Boxplot

```
y <- rnorm(100, mean = 80, sd = 3)
boxplot(y)
```

Parallel Boxplot

```
grp <- factor(rep(c("Grp 1", "Grp 2"), each
= 100)) #make groups for x and y
dat <- c(x, y)
boxplot(dat ~ grp, xlab = "Group")
```

Scatterplots

Construct a scatterplot with x on the x-axis and y on the y-axis:

```
# generate vectors
x <- runif(20)
y <- 2 + 3 * x + rnorm(20)
plot(x, y)
```

Scatterplot with custom labels and title and a line for the deterministic part of the relationship between the variables:

```
plot(x, y, xlab="1st variable", ylab="2nd
variable", main="Title of the plot")
abline(2,3,col="blue")
```

Line plot of standard normal density

```
x <- seq(-4, 4, len = 1000)
y <- dnorm(x, mean = 0, sd = 1)
plot(x, y, xlab="x", ylab="density", type =
"l")
title("Density of Standard Normal")# same as
# adding main="Density of Standard Normal"
# in the plot function call
```

Plot of a probability mass function

```
#plot of Binomial(n = 20, p = .3) pmf
x <- 0:20
y <- dbinom(x, size = 20, prob = .3)
plot(x, y, xlab="# Successes", ylab="Prob",
type = "h")
title("pmf of Binomial(n = 20, p = .3)")
```

- Let us practice plotting with the `iris` data.
  - Plot a histogram with all sepal lengths.
    - Increase the number of bars for a better resolution.
  - Plot a histogram of the sepal lengths for the *Versicolor* irises only.
  - Add axis-labels and a title.
  - Save the resulting image as a PNG file (click the export button above the image).
  - Repeat the same for the *Virginica* irises.
  - Let's do parallel boxplots for *Versicolor* and *Virginica* irises to visually compare the two species on this feature.

10. Installing Packages

- Many useful R functions come in "packages", these are free libraries written by members of the R community.
- The CRAN online *package repository* currently features almost 20,000 packages. More packages are available outside of CRAN.
- If something is related to data manipulation and analysis and is seemingly not in R, it is more than likely available in a free library.
  - Perhaps outside CRAN on GitHub.
- Many packages come with built-in datasets that you can use to study and play.

- To install an R package, type:

```
# install.packages ("package_name"), e.g.:
install.packages ("faraway")
```

  - Use the lib.loc= argument if you don't have administrative permissions on your computer and you get errors.
- You can also use the menu `Packages->Install package(s)` directly in R or in `RStudio->Tools`.
- You need to install a package <u>only once</u> per R installation.
- R will download the package from Comprehensive R Archive Network (CRAN), so you must be connected to the internet. You might be prompted to select a location (mirror). Select a nearby location or <u>**Cloud**</u>.

- To access the functions and/or the data in a previously installed package in your current R session:

```
library("package_name"), e.g.:
library(faraway)
```

- You can see the list with already installed libraries with the command:

```
library()
```

- A short description of the library using

```
library(help = faraway)
```

- To check the names of all *functions* and *datasets* in a package, type:

```
ls("package:faraway")
```

- Note that prominently used packages are very well supported and documented.
- Some less prominently used packages might be slow and less refined, even buggy.
- As a rule of thumb, packages used in a published book are likely to be good.

- A very good starting point to consider for R packages is RStudio's *Quick list of useful R packages available*:

https://support.rstudio.com/hc/en-us/articles/201057987-Quick-list-of-useful-R-packages

  - The list is frequently updated.
  - Let's explore some…

- Now let us install the `ggplot2` library as we will need it soon.

11. IF Statements

- If you need to execute different code depending on a condition that you need to check, you can use an "if statement":

```
if ( condition1 ) {
   statement1
} else if ( condition2 ) { # optional
   statement2
} else {                   # optional
   statement3
}
```

- Recall we have two types of Boolean operators:

```
Comparison operators:  <, <=, >, >=, ==, !=
```

```
Logical operators:
!, &, |, &&, ||, xor()
```

  - What is the difference between `&` and `&&` ?

In recent lab, I assigned a chapter from DataCamp to study and practice the if statements:

https://www.datacamp.com/courses/intermediate-r

DataCamp:Intermediate R / Chapter 1. Conditionals and Control Flow

Let us practice if statements a bit more…

```
a=5;b=4 # will be changing these later to
test the code.


# compare a and b with if statement and
print "a is greater than b" or "b is
greater than a" or "a and b are equal"
```