R manual for CSC 1706 Probability and Statistics

Afidalina Tumian
10-27-2017

Contents

1 Preface

R is a statistical package widely used for data analytics and visualization. It is freely available with a lot of resources accessible online.

The purpose of using R in this subject is to complement the theory learnt in each chapter and apply relevant tools for practical purposes. The instructions and R commands provided in this manual have been tested on my Windows 10 machine. Therefore, things may be a bit different, but not that difficult to figure out, for those using Mac or Linux machine.

The chapters in this manual are arranged according to the topics covered in CSC 1706. This manual aims to provide you with the relevant R codes and hands-on exercises. Answers to the exercises will be provided by the instructor.

It is important to highlight that this manual is not intended to furnish all possible details on R programming. Therefore, extra references will be given at the end of each topic or suggested by the instructor.

Students are encouraged to give suggestions on how to further improve this manual for the benefit of subsequent classes.

Dr. Afidalina Tumian

2 Introduction to R software

2.1 Installing R software

Before starting to use R, you need to download and install the software. R is freely available from the Comprehensive R Archive Network (CRAN) at https://wbc.upm.edu.my/cran/. You need to choose the right version for your machine: either Windows, Linus, or MacOS.

After downloading the installation file, click on it and follow the instructions given with default settings.

For the more adventurous ones, you can also install RStudio from the following website: https://www.rstudio.com/. However, for this subject I will be using only the basic R software throughout the text.

2.2 Getting started

In order to run R, just click on the R icon or shortcut on your machine. It will look like Figure 1 on a Windows machine:

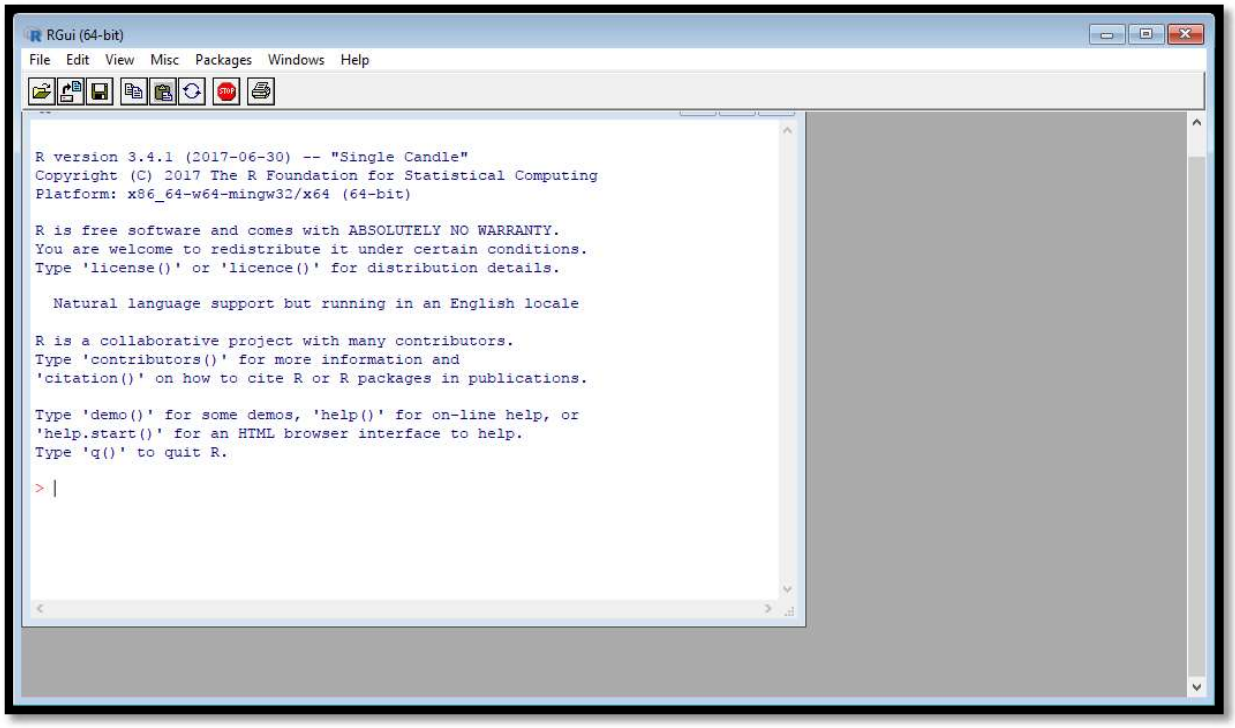

Figure 1:A screenshot of R software on Windows

You will see this sign **>** on your R main window and this is where you will type your R commands. To execute each line, you need to press **enter**. If there are some syntax error in your R commands, some messages will be given. Otherwise, it means that your R commands are well executed (YAY!).

It is important to highlight here that before starting any analysis on R, it is highly recommended to perform **changing directory** to the folder that contains all your data.

For example, you may have a folder on the Desktop named 'Data' which contains all your data to be analysed. To change the directory in R, click **File -> Change Dir…** and choose the Data folder that you have on your Desktop.

To check whether you have changed directory successfully, you can type the following R command, and the path to the directory you have chosen will appear subsequently:

```
> getwd()
```

Besides that, you can also type your R commands on a script window. For this method, you need to click **File -> New Script**, and a window as in Figure 2 will appear.

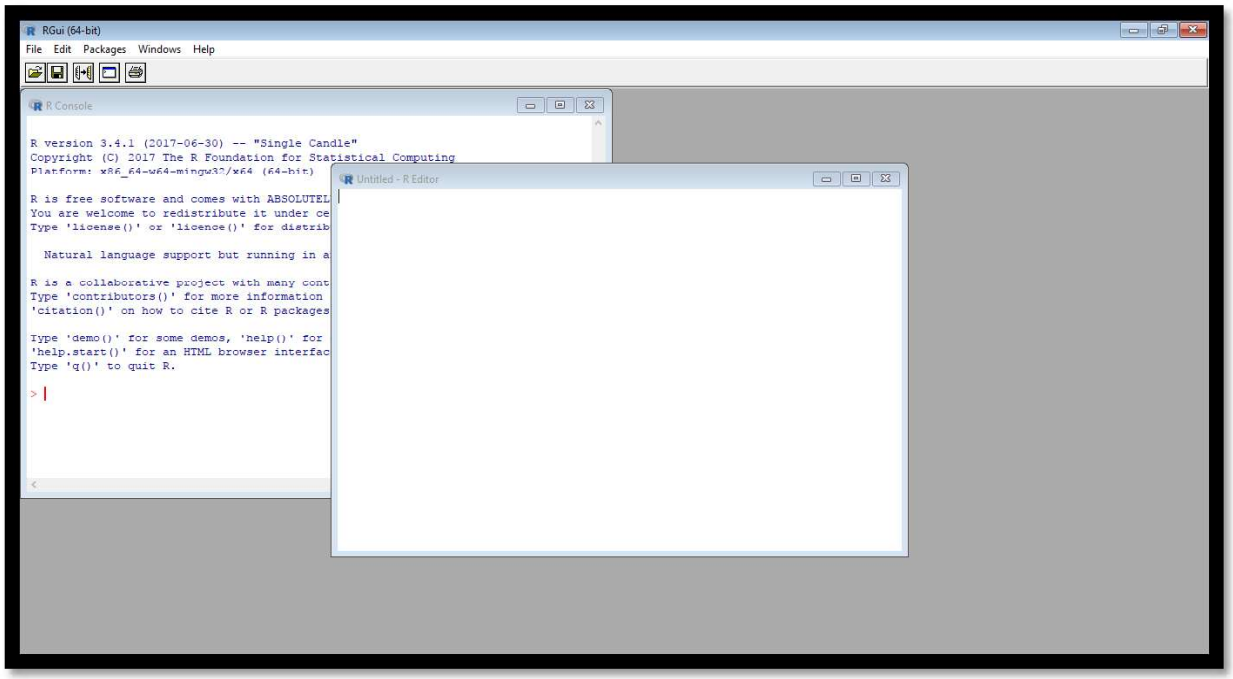

Figure 2: R environment with a new script window

Save the file with the following name: **myfirstscript.R**. Type all your R commands in the file. To run the lines, highlight and right click on the lines that you want to execute. Choose **Run line or selection**.

In summary, to run R commands, you can type R commands on either:

1. the main interface window as in Figure 1.
2. a script window as in Figure 2.

2.3 Installing R packages

The best thing in R is that there are many specific tools or packages developed for specific needs across different application areas. You can either *youtube* or *Google* what you want to do in R and I can almost guarantee there will be references on it. Many R experts have written these packages with detailed documentation on it and made them freely available.

Therefore, before we can use the packages in our R software, we need to install them first.

To install a package, run the following command:

```
#let X is the name of the package you want to install
> install.packages("X")
```

For example, one of the famous package for data visualization is **ggplot2**, thus you need to run the following command to install ggplot2 package:

```
> install.packages("ggplot2")       #to install ggplot2 package
```

In order to use the package, run the following command:

```
> library(ggplot2)
```

2.4 Basics of R

R can work in various data types, such as vector, matrix, list, etc. R can be your mathematics calculator.

You can create your own functions in R; in fact you can do almost everything that any other programming language has offered.

To start with, let's create a vector named **x** ranging from 1 to 10.

```
>  x = 1:10
```

To know what is the value in x, type x and press enter. You will get to see the following:

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

Let's create another vector named y with the following R code:

```
>  y = c(2, 4, 7, 9, 13, 15, 14, 18, 21, 24)
```

You notice that you have created two vectors using two different ways. To learn more about the objects/variables you created on R, you can use **`str()`** command.

```
> str(x)
> str(y)
```

To know more about what a function does in R, you can utilise the help manual. For example, you want to know what **`c()`** is doing, type any of the following commansd:

```
> help(c)
> help("c")
> ?c
```

You will be prompted to a web page explaining the details of **`c()`** function.

So far you have created x and y. Let's create a plot with x vector as the x-axis and y vector as the y axis.

```
> plot(x, y)
```

Do you get something like this (see Figure 3)?

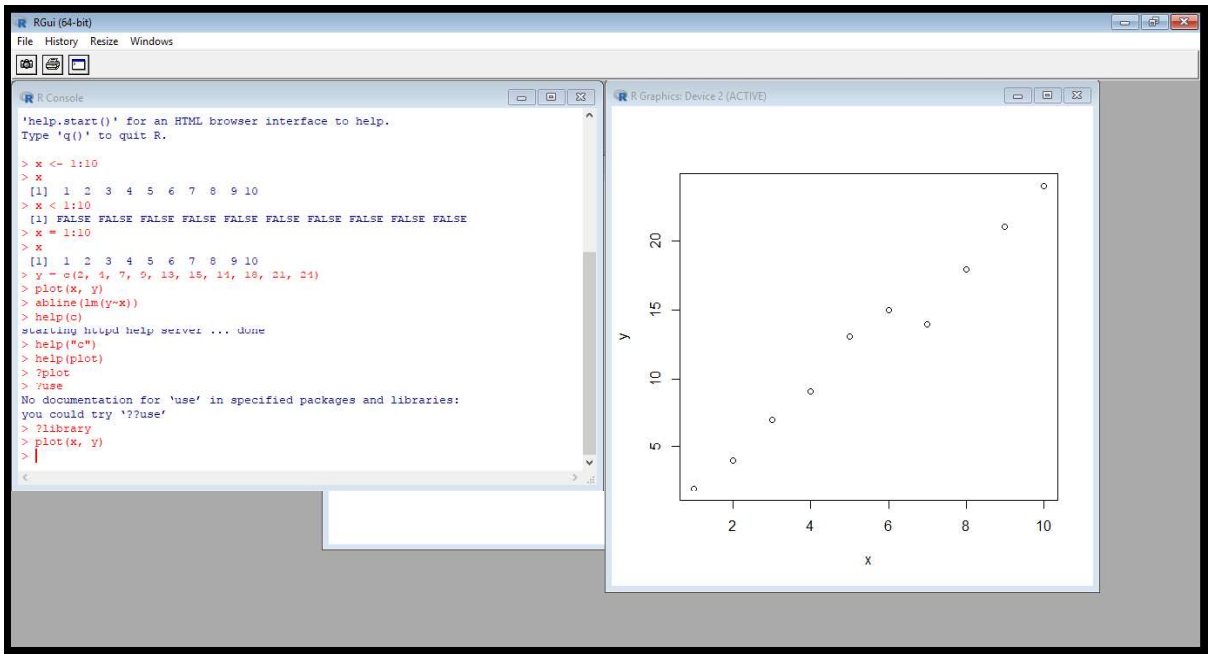

Figure 3: Plotting the x and y vectors

Let's fit a straight line through the points using the following command:

```
> abline(lm(y~x))
```

The command should result in the following plot as shown in Figure 4.

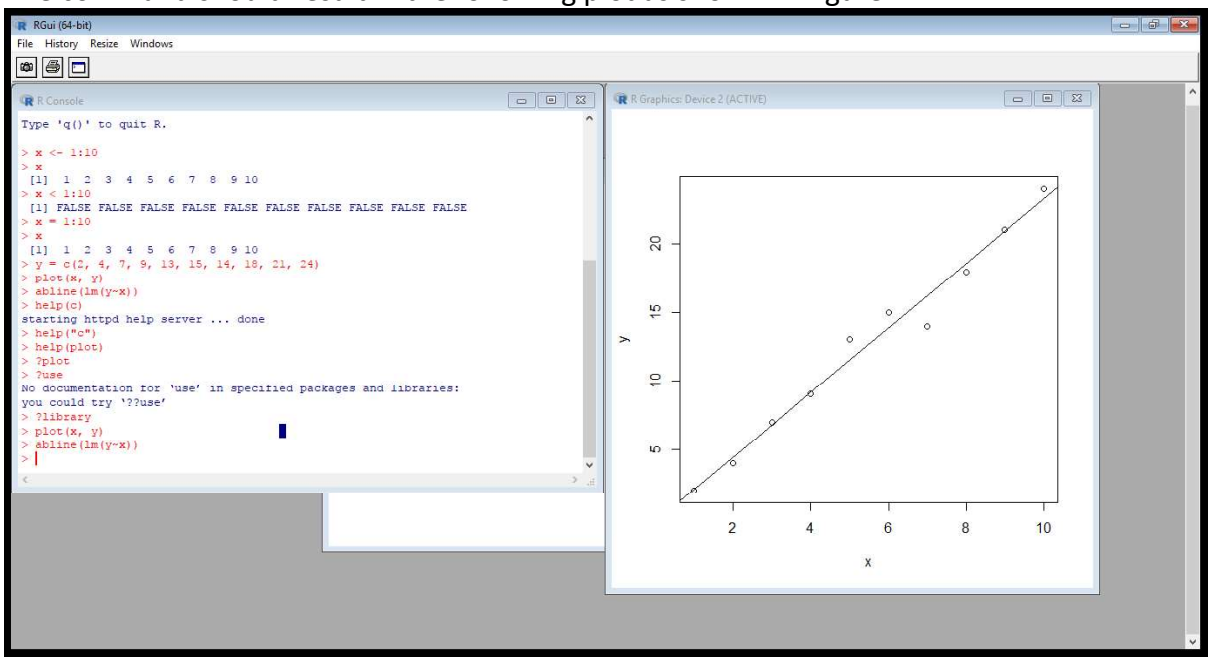

Figure 4: Fitting a straight line though the points

You can further customize your plot by providing more details to your **`plot()`** function. You might want to create smaller points or change the color of your plots. To know more about **`plot()`** function, type **`?plot`** and click on the **par** word.

For example, to create blue solid circle representing the points and blue line passing through the points (see Figure 5), we run these commands:

```
> plot(x, y, pch = 16, col = "blue")
> abline(lm(y~x), col = "red")
```

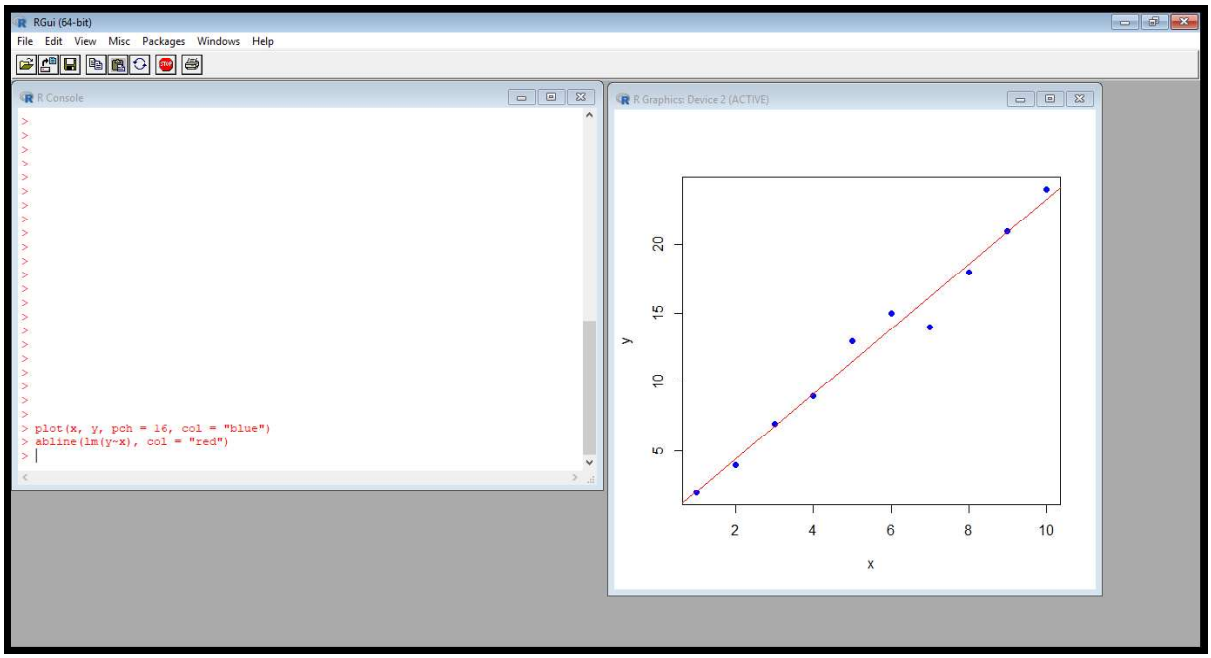

Figure 5: Update the plot with more parameters defined in the plot() and abline() functions.

To save the figure, you can click to File -> Save as, and choose whichever file extension you would like to. Let's say we save the file as **myfirstplot.png**.

2.5 Importing data into R

The first step in doing any statistics study is to collect data. After data collection, you need to organize them properly for further analysis.

A lot of functions have been developed in R to deal with different kinds of data files. Some data are stored in text files, and some other data are kept in excel files. Therefore, depending on where you store your data, different functions will be used to import different kind of data files.

For our first experiment here, download **temperature.txt** from your italeem page and store into the folder you're working on. The file contains 50 data points representing temperatures collected on 50 consecutive days.

To import the data, run the following command:

```
data <- as.numeric(scan("temperature.txt", what = "char"))
```

To see what has been stored in **data** variable, type **`data`** and press **enter**.

You can get the summary statistics on the temperature data by executing **`summary(data)`**. You might want to create a simple histogram on the data by executing **`hist(data)`**.

Some data have multiple columns representing different information and data types. For example, this link (http://www.data.gov.my/data/en_US/dataset/vital-statistics-malaysia-1051) leads you to information on the birth rate in Malaysia from 1911 to 2014. Download the file into your default folder. Click on to the downloaded file and have a glance on the data. Note that the file has .csv as its file extension.

To read the csv file, run the following command:

```
birthrate = read.csv("bppd_Crude_Birth_Rate_Malaysia_1911-2014.csv", header= TRUE, stringsAsFactors = FALSE)
```

To read data from an excel files, you need to install a package named **openxlsx**. Assuming that you're reading the first sheet from **data.xlsx** file, you can run the following command:

```
data <- read.xlsx("data.xlsx", sheet = 1)
```

2.6 Simple random sampling

We have learnt about random sampling briefly in the previous class. Let's say you want to generate 100 random numbers ranging from 1 to 100.

```
x <- 1: 100 #generate a vector from 1 to 100
sample(x, 100, replace = FALSE) #try with replace = TRUE
```

[Task] Try to create sample of 10 numbers ranging from 50 to 2000.

2.7 Ending your R session

Before ending the current R session, you can save the commands history to a file.

```
> savehistory("sept21class")
```

This command will save the commands into sept21class.Rhistory into the folder you have been working on.

To load the command history from the sept21class.Rhistory file in the next session, run the following command:

```
> loadhistory("sept21class")
```

With that, you can use the variables that you have created in the earlier session.

Do not forget to save your script file and any generated figures before you end your R session. To end your session, you can either:

1. Run `q()` command
2. Click **File -> Exit**

The next prompted command will ask whether you want to save the workspace image or not. Totally up to you!

3 Frequency distributions and graphs

In this chapter, we are going to explore more on our data by organizing them into frequency distribution and perform some data visualization.

3.1 Data exploration

We are going to use the temperature data as previously mentioned in Section 2.5. Run the following lines of codes and observe the outcomes.

```
data <- as.numeric(scan("temperature.txt", what = "char"))

str(data)

range(data)

min(data)

max(data)

data.freq <- table(data)

percent.freq <- data.freq*100/sum(data.freq)

data2 <- cbind(as.numeric(names(data.freq)), data.freq, percent.freq )

colnames(data2)[1] <- "temperature"

write.table(data2, file = "updatedtemperature.txt", row.names = FALSE,
quote = FALSE)
```

[Question] What are these lines of codes doing? Write your notes beneath every line.

Assume that we want to create construct a frequency distribution using 7 classes. We need to first break the range into non-overlapping intervals by defining a sequence of equal distance break points.

Determine the width of the class. Substitute the value you calculate to width variable below. Then run the following code.

```
#remove the question mark and substitute the value of width here
width = ?

breaks = seq(100, 135, by  = width)

class.limits = cut(data, breaks, right=FALSE)

class.freq = table(class.limits)

data3 <- cbind(row.names(class.freq), class.freq)

colnames(data3) <- c("class.limits", "class.freq")

write.table(data3, file = "freq.dist.temperature.txt", row.names = FALSE,
quote = FALSE, col.names = TRUE)
```

3.2 Plotting histograms

One of the most common data visualization technique is to plot histogram, that helps visualize the frequency distribution.

You need to first determine the class boundaries from the class limits we have constructed in Section 3.1. See **`class.bound`** variable below.

```
class.bound <- c(99.5, 104.5, 109.5, 114.5, 119.5, 124.5, 129.5, 134.5)

hist(data, breaks = class.bound, xlim = c(99.5, 134.5),  ylim = c(0, 18),
axes = F)

axis(side=1, at =seq(99.5,134.5, width), labels=seq(99.5,134.5, width))

axis(side=2, at =seq(0, 18, 3), labels=seq(0, 18, 3))
```

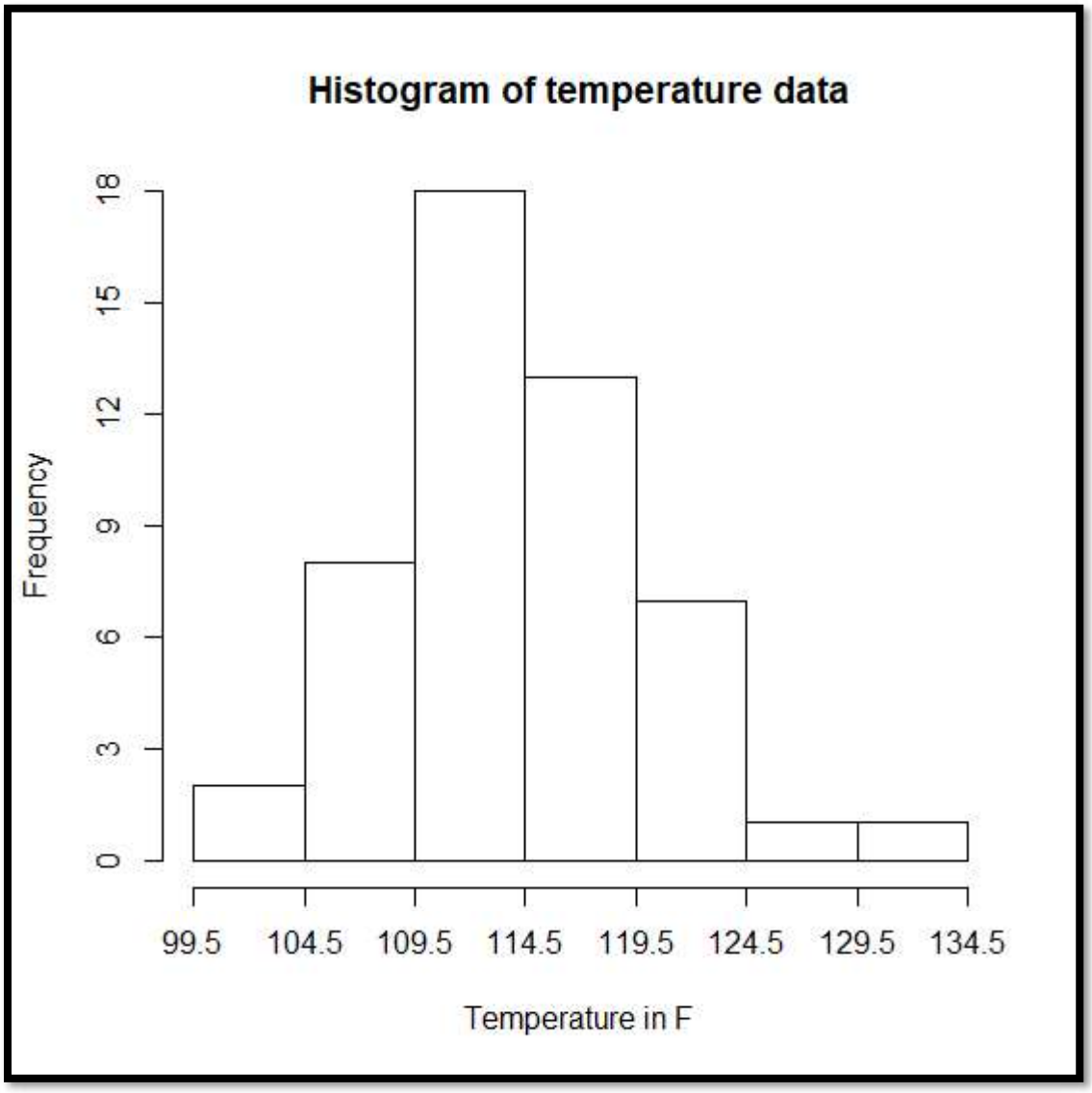

Figure 6: Histogram plot of temperature data

3.3 Plotting frequency polygon

Run the codes below to plot frequency polygon. Explore each command by typing the name of the variables.

```
h <- hist(data, breaks = class.bound, xlim = c( class.bound[1]-
width,class.bound[length(class.bound)]+width), ylim = c(0, 18), xlab =
"Temperature in F", ylab= "Frequency", main = "Histogram of temperature
data", axes = F)

mp = c(min(h$mids) - (h$mids[2] - h$mids[1]), h$mids, max(h$mids) +
(h$mids[2] - h$mids[1]))

freq = c(0, h$counts, 0)

lines(mp, freq, type = "b", pch = 20, col = "red", lwd = 3)

axis(side=1, at =seq(class.bound[1]-width,
class.bound[length(class.bound)]+width, width),labels=seq(class.bound[1]-
width,class.bound[length(class.bound)]+width, width))

axis(side=2, at =seq(0, 18, 3), labels=seq(0, 18, 3))
```

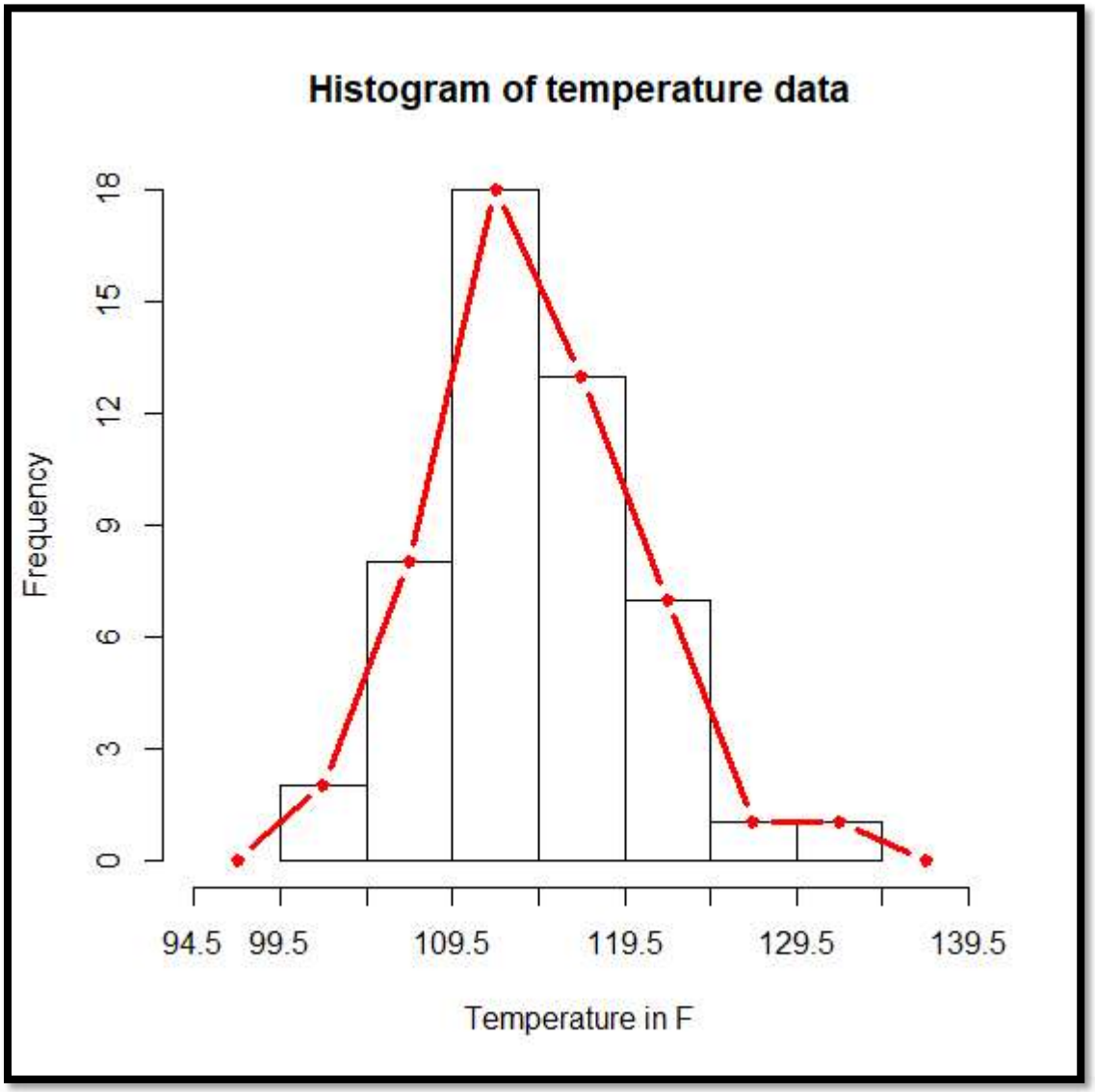

Figure 7: Frequency polygon with histogram

3.4 Plotting ogive

Ogive provides the visualization for cumulative frequencies for the classes in a frequency distribution. Run the codes below to produce ogive for the temperature data.

```
interval.cut <- cut(data, class.bound, right=FALSE, dig.lab = 4)

temp.freq <- table(interval.cut)

cumfreq <- c(0, cumsum(temp.freq))

plot(breaks, cumfreq, main="Temperatures recorded", xlab="Temperature in
Fahrenheit", ylab="Cumulative frequency", axes = F)

lines(breaks, cumfreq)  # join the points

axis(side=1, at =seq(99.5,134.5, width), labels=seq(99.5,134.5, width))

axis(side=2, at =seq(0, 50, width), labels=seq(0, 50, width))
```

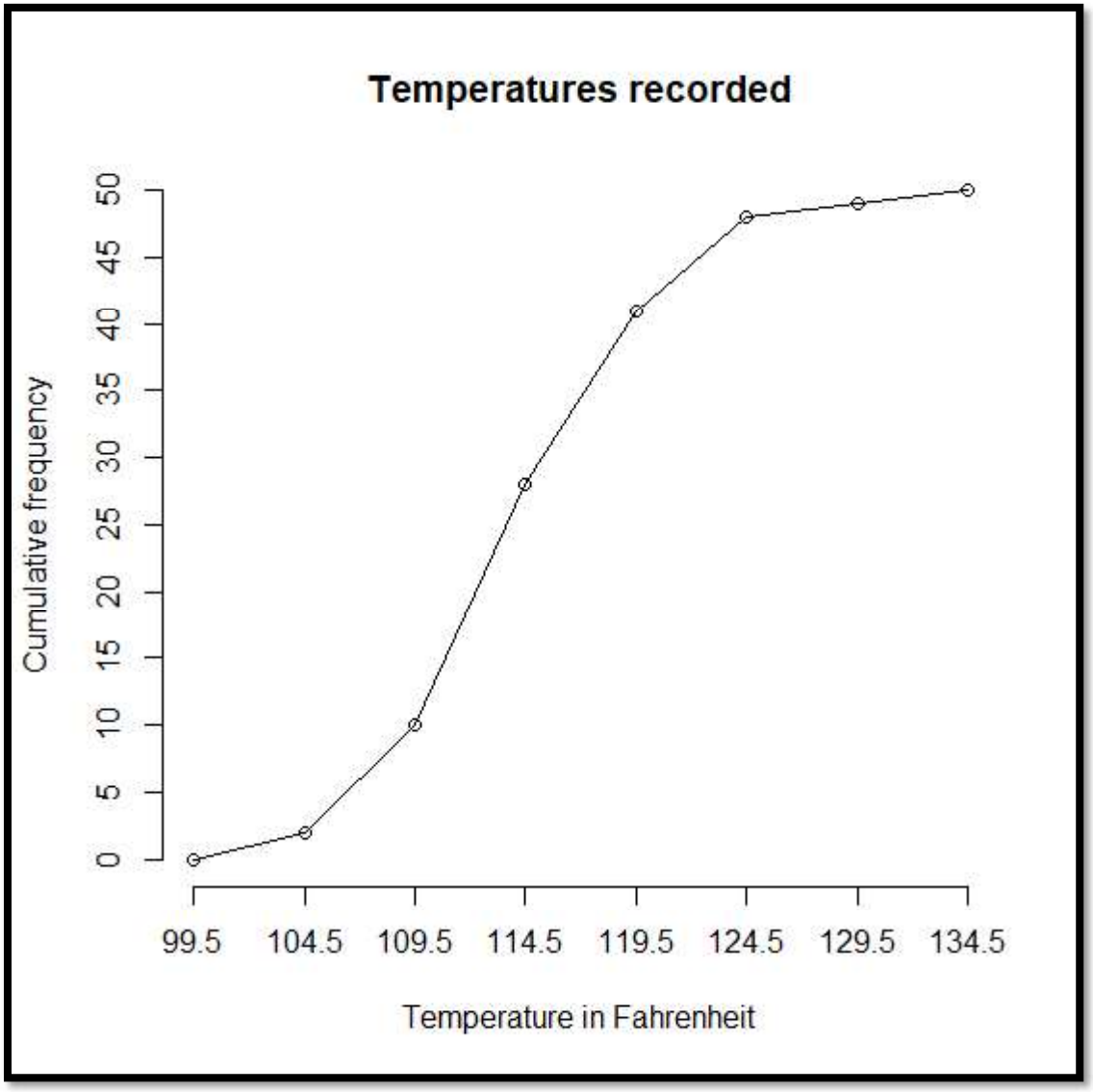

Figure 8: Ogive for temperature data

3.5 Relative frequency graphs

So far, the graphs we plot gives the frequency information. What if we would like to see its relative frequency? The changes required are highlighted in red. Compare with the ones we see earlier.

```
class.bound <- c(99.5, 104.5, 109.5, 114.5, 119.5, 124.5, 129.5, 134.5)

hist(data, breaks = class.bound, xlim = c(99.5, 134.5), ylim = c(0, 0.1),
xlab = "Temperature in F", ylab= "Relative Frequency", main = "Histogram of
temperature data", freq = FALSE, axes = F)

axis(side=1, at =seq(99.5,134.5, width), labels=seq(99.5,134.5, width))

axis(side=2, at =seq(0, 0.1, 0.01), labels=seq(0, 0.1, 0.01))
```

```
interval.cut <- cut(data, class.bound, right=FALSE, dig.lab = 4)

temp.freq <- table(interval.cut)

cumfreq <- c(0, cumsum(temp.freq)/sum(temp.freq))

plot(breaks, cumfreq, main="Temperatures recorded",xlab="Temperature in
Fahrenheit", ylab="Cumulative relative frequency", axes = F)

lines(breaks, cumfreq)  # join the points

axis(side=1, at =seq(99.5,134.5, width), labels=seq(99.5,134.5, width))

axis(side=2, at =seq(0, 1, 0.2), labels=seq(0, 1, 0.2))
```

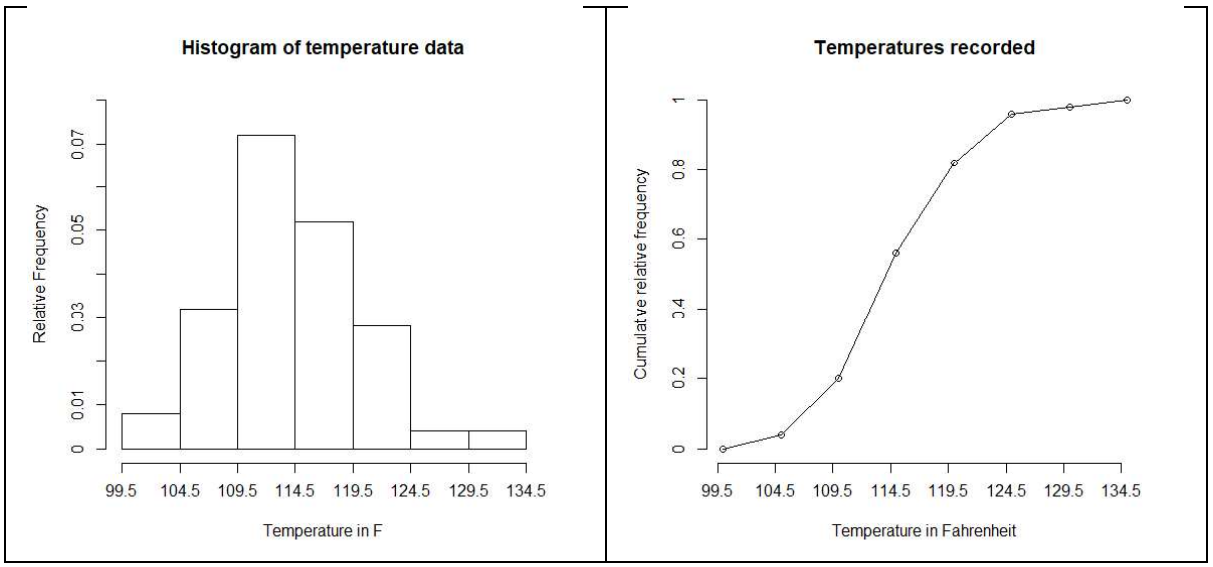

Figure 9: Histogram and ogive for temperature data in terms of its relative frequency

4 Data description

This chapter will provide more understanding on any given data, by applying various measures of statistical analysis.

We will use temperature data as in previous chapters. To start with the analysis in this chapter, make sure that you have already imported the temperature data (assuming the data is stored in **`data`** variable) into your R environment.

4.1 Measures of central tendency

Type	R codes
Mean	`mean(data)`
Median	`median(data)`

Mode	`table.data <- table(data)`
	
	`table.data[which(table.data == max(table.data))]`

Midrange	`(min(x) + max(x))/2`

Weighted mean	`x <- c(3.7,3.3,3.5,2.8)`
	`weight <- c(5, 5, 4, 1)/15`
	`mean.weight <- weighted.mean(x, weight)`

Table 1: Various measures of central tendency in R

Apply the functions in Table 1 to your temperature data and evaluate the values you get.

[Question] Are mean, median and mode values the same? What is the conclusion?

4.2 Measures of variation

Type	R codes
Range	`range(data)`
Sample variance	`var(data)`
Sample standard deviation	`sd(data)`
Population variance	`sum((data-mean(data))^2)/length(data)`
Population standard deviation	`sqrt(sum((data-mean(data))^2)/length(data))`

Coefficient of variance	For samples:
	`sd(data)*100/mean(data)`
	
	For populations:
	`sqrt(sum((data-mean(data))^2)/length(data))*100/mean(data)`

Table 2: Various measures of central tendency in R

Apply the functions in Table 2to your temperature data and evaluate the values you get.

4.3 Measures of position

Type	R codes
Data sorting in increasing manner	`sort(data, decreasing = FALSE)`
Z-score	`(data – mean(data))/ sd(data)`
Quartiles	`quantile(data)`
Deciles	`quantile(data, prob = seq(0, 1, length = 11))`
Percentiles	`quantile(data, prob = seq(0, 1, length = 101))`

Table 3:Various measures of position in R

4.4 Exploratory data analysis

Type	R codes
Five number summary	`fivenum(data)`
Box plot	`boxplot(data)`