F20SA / F21SA Labs

Contents

F20SA / F21SA Labs

Labs 1: Starting with R, Summary Statistics and Plots

Contents

1.1 Introduction

Over the course of these computer labs we will give you a brief introduction to R and basic statistical analysis using R.

R is in many application areas the statistical software of choice and is the language within which many cutting edge statistical techniques are first implemented. Here we will only scratch the surface of what you can achieve with R. If you are interested is exploring beyond this there are many books and online resources which can help you go further.

1.1.1 Basics

R is an interpreted programming language which has been developed to carry out statistical analysis. So like most programming languages we can:

Basic Calculations

Like most programming languages we can use R just as a basic calculator, for example adding two numbers together, try:

```
3+5
[1] 8
```

We can carry out any of the standard operations $+, -, \times, \div$, etc. For example we carry out complicated calculations for example we can evaluate:

$$\frac{2^{10} - 3^5}{5 \times 6}$$

by

```
(2^10-3^5)/(5*6)
[1] 26.03333
```

Storing information

As well we can store the results of our calculations as variables, for example:

```
x=3+4
x
[1] 7
```

We can then carry out calculations using that variables for example:

```
x=3+4
y=2
x*y
[1] 14
```

In addition to storing individual numbers we can also store arrays of numbers. For example:

```
x=c(3,4,5)
y=1:10
x
[1] 3 4 5
y
[1]  1  2  3  4  5  6  7  8  9 10
```

Here a:b is a shorthand array to obtain an array going from a to b by increments of 1. There are a number of other functions to create other standard arrays, eg. seq for a sequence and rep for a repetition. (See R reference card)
When we carry out basic operations on arrays R carries out the operations coordinate wise, for example:

```
x=c(3,6,5)
y=3:5
x+2
[1] 5 8 7
y*3
[1]  9 12 15
x+y
[1]  6 10 10
x*y
[1]  9 24 25
```

Inbuilt functions

As with most languages R has a number of inbuilt mathematical functions which can be applied to either numbers directly or variables. For example if we are interested in the sin of values:

```
sin(2)
[1] 0.9092974
x=3
sin(x)
[1] 0.14112
```

In general if you apply these basic mathematical functions to arrays it will carry out the function on each coordinate in turn but care has to be taken as this is not always the case.

```
sin(1:5)
[1]  0.8414710  0.9092974  0.1411200 -0.7568025 -0.9589243
```

Couple of warnings

Like all languages R has a number of features which might easily trip up first time users including the following two:

1. The first issue is that unlike most computer languages R indexes it arrays starting at 1 rather than 0.

   ```
   x=c(1,3,4)
   x[1]
   [1] 1
   x[3]
   [1] 4
   ```

2. R has a number of assignment operators unlike most languages. So far we have always used '=' but '`<-`' can also be used to assign the value to a variable for example:

   ```
   x<-2
   x
   [1] 2
   ```

   There are differences between the two operators but for our purposes we will treat them as interchangeable. In the wider community there is a general preference for '`<-`' to allow compatibility with older versions of R and it's predecessor S-Plus.

1.1.2 Finding Help

R has a well developed help systems which can be accessed straight from the prompt. If you want to find out further information about a function and often example usages we use the command `help`. For example if we wanted further information about the sine function we would type `help(sin)` and this would provide us with the information we wanted.

1.1.3 Reading Data

In most situations we will be interested in analysing a data set that we have obtain from elsewhere or been given by someone else. Therefore we will need to load this data into R to allow us to analysis it. R has a number of inbuilt functions for doing this depending on what format the data is in, for example read.xls for Excel spreadsheets or read.spss for SPSS datasets. Through out the rest of this sheet we will use read.table which allows us to read in either comma separated data or space separated data.

(For this and the remainder of the worksheet you will need to download abalone.data from Vision and place it in your working directory. Note you can use the function `setwd` to change your working directory if needed.)

An example of such a command is:

```
abalone<- read.table('abalone.data', sep=',', header=FALSE)
```

The first argument here is the name of the file from which we want to load the data. The second `sep=','` tells R that the separator used in this file is a comma. Finally the third argument `header=FALSE` tells R that the first row of the file is data rather than variable names. This command loads the data in and stores the information as a data frame.

1.1.4 Data Frames

In most situations our data will be within a data frame in R. This is a data structure which stores all the data for our experiment as a table with each row representing and observation and each column a variable that has been measured. In addition each column can be thought as a vector, all of equal length.

For example previously we used the read.table command to load in an abalone data set [a]. Here each row of the table represents an abalone we have collected data about. The columns are:

Column 1	Length of the sample	mm
Column 2	Diameter	mm
Column 3	Height	mm
Column 4	Whole weight	g
Column 5	Shell weight	g
Column 6	Rings	

Note that the first 5 variables are numerical continuous and the 6th is numerical discrete.

Since we loaded the data in without headers R has used its default labelling for each of the column, V1, . . . , V6. If we had loaded the data with headers then the header information would have been used to label the columns. R allows us to access each using a `$` suffix followed by the variable name. For example to access the information in the first column of the abalone data we loaded previously we would type:

```
abalone$V1
```

For example if we wanted to the number of entries we could use the lenght function:

```
length(abalone$V1)
[1] 1527
```

1.2 Basic Plots

We are now going to introduce a number of common plots you might use to explore a dataset. Through out we will use the abalone data set which should be loaded with the command:

```
abalone<- read.table('abalone.data', sep=',', header=FALSE)
```

Stem and Leaf Plot

The first plot we will produce is a stem and leaf plot. This is useful for any numerical data and relatively small data sets.

Here we will produce a stem and leaf plot for the length of the first 80 data samples from our data set. the first thing we will do is pull out the data we want to plot:

```
abalone_lengths<-abalone$V1[1:80]
```

[a] More information on this data set can be found at https://archive.ics.uci.edu/ml/datasets/abalone. The data we are using only contains the male samples

Now we can use the function `stem` to produce the plot.

```
stem(abalone_lengths)

  The decimal point is 1 digit(s) to the left of the |

  3 |
  3 | 5566778
  4 | 01133344
  4 | 555677777788899
  5 | 0001223334
  5 | 566666777888899
  6 | 000011111222344
  6 | 5577
  7 | 000113
```

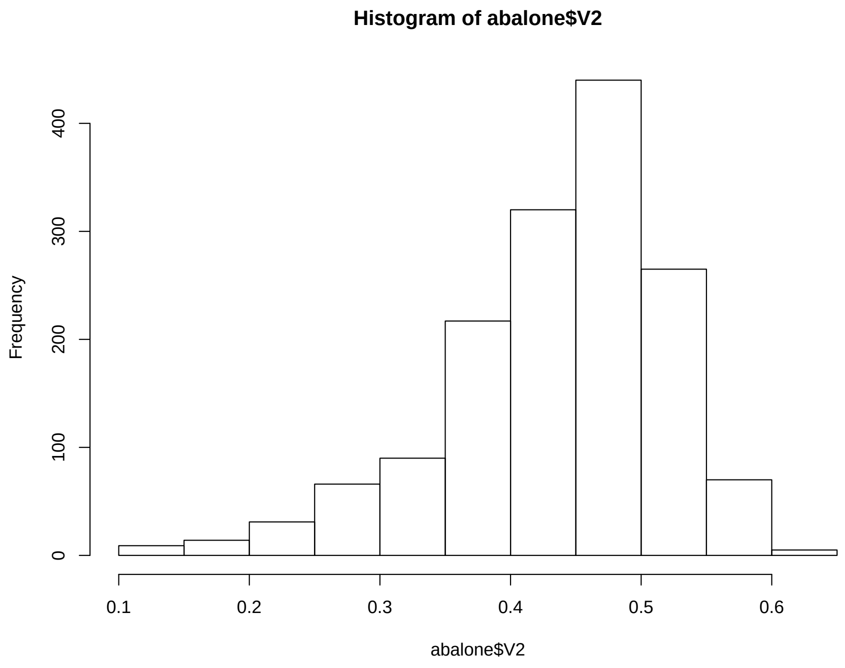
Here the first decimal place is shown as the stem and the leaves represent the second decimal place of the data.

Histogram

The second plot we will consider is the histogram. Remember this is useful for exploring continuous numerical data. In R we use the function `hist` to plot a histogram.
Here we will produce a histogram for the diameter of the abalone samples:

```
hist(abalone$V2)
```

This produces the following plot:



In producing this plot we have used the default settings and in most circumstances these will be adequate apart from the title and axis labs (see next section) but R does allow us to change a number of parameters. For example, in this plot we have allowed R to use its default method for deciding the number and width of the bins but for various reasons we might prefer a different number of bins. This can be set for example:

```
hist(abalone$V2, breaks= 50)
```

which produces a histogram with 50 bins.

Titles and Axis labels

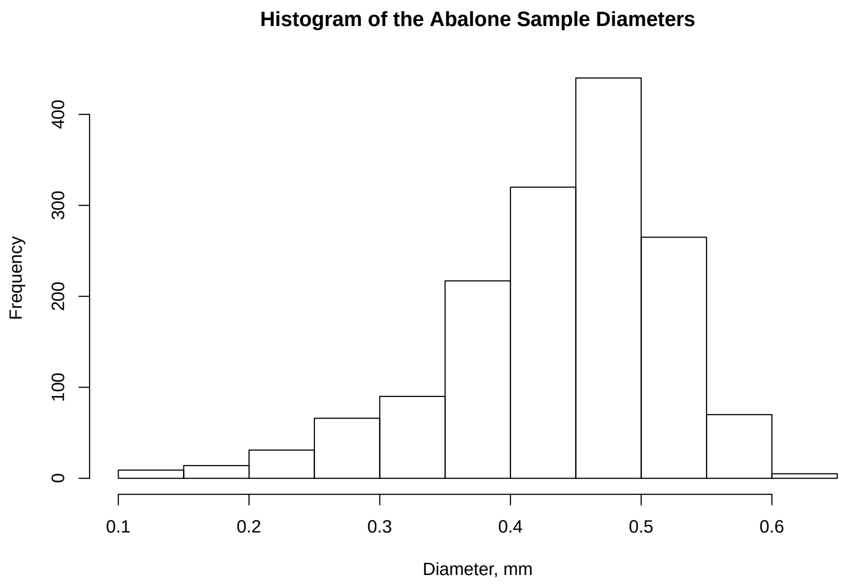
The most common parameters we want to change from their default value is the labels and title used for our plots. For most of the plots these are set by function parameters:

- main - for the main title of the plot
- xlab - for the x axis label
- ylab - for the y axis label

For example going to back the previous histogram we might want to change the title and x axis label:

```
hist(abalone$V2, main='Histogram of the Abalone Sample Diameters', xlab='Diameter, mm')
```

This produces the following plot:



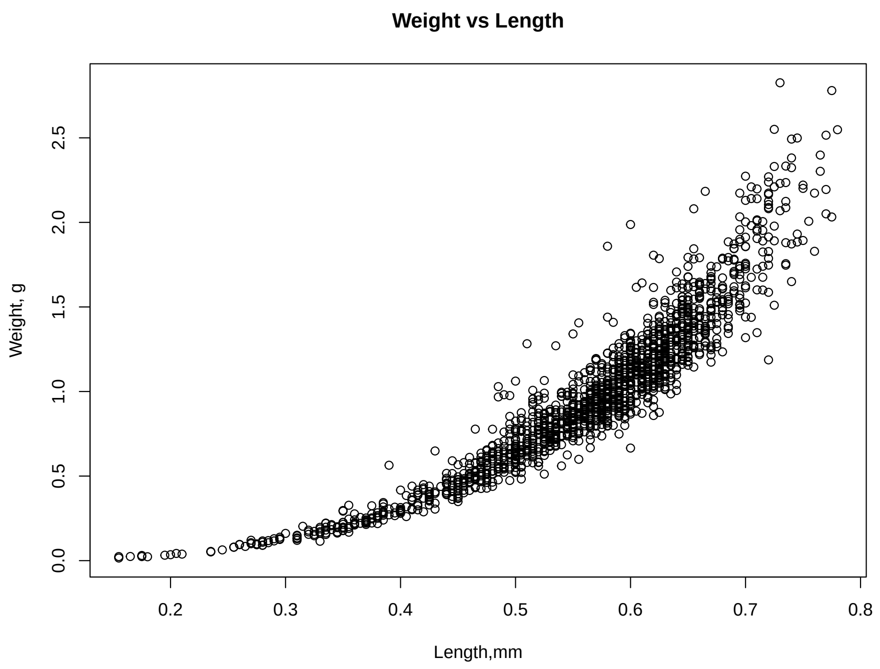
There are a number of other cosmetic parameters which can be set, information can be found by using R's help documentation.

Scatter plot

Often we are interested in exploring the relationship between two parameters, for example for the abalone data we might be interested in the relationship between the weight of the abalone and the length of abalone. The natural plot in this case is a scatter plot where each sample is represented by a point.
In R we use the plot function to produce a scatter plot, for example here:

```
plot(abalone$V1, abalone$V4, xlab='Length,mm', ylab='Weight, g', main='Weight vs Length')
```

This produces the following plot:



Note that the first argument of the function is the variable used for the x axis and the second is the variable used for the y axis. As before there are a wide range of parameters which can be changed from the type of marks representing each point to the colour of the points, again see the help documentation for more information.

Bar Plot

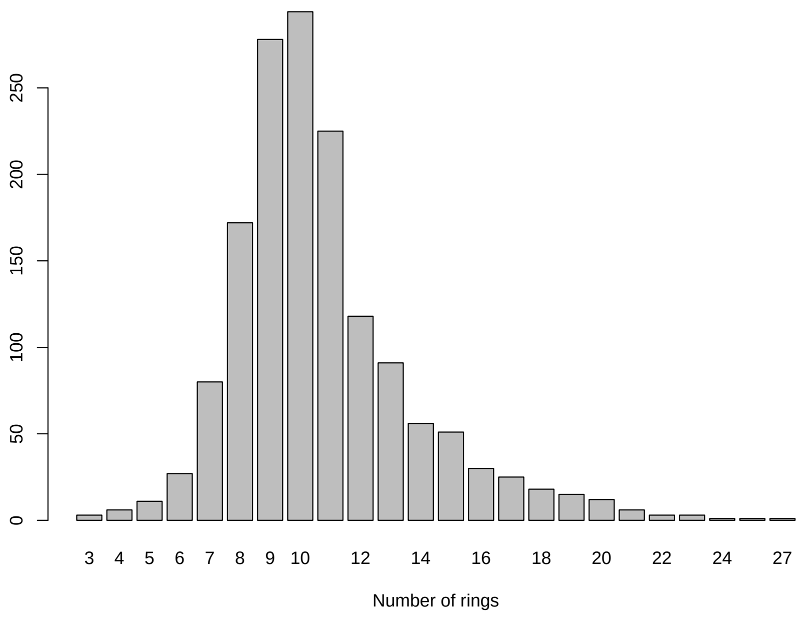
The final plot we consider here is the bar plot which is useful either discrete data or categorical data. Here we consider plotting a bar plot of the ages of the abalone samples. To produce a bar plot there are two steps, initially we have to produce a frequency count from our data of each value. We use the function `table` to do this:

```
table(abalone$V6)

  3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22
  3   6  11  27  80 172 278 294 225 118  91  56  51  30  25  18  15  12   6   3

 23  24  26  27
  3   1   1   1
```

So we see there are 3 samples with 3 rings, 6 with 4 rings etc. This information is then used by the function `barplot` to produce the chart.

```
freq<-table(abalone$V6)
barplot(freq, xlab='Number of rings')
```

This produces the following plot:



1.3 Summary Statistics

R has inbuilt functions to calculate all the summary statistics of our data:

Statistic	R Command
Mean	`mean`
Standard deviation	`sd`
Median	`median`
Quartiles	`quantile`
Interquartile range	`IQR`

For example we can calculate each of these for the shell weights of the abalone data:

```
mean(abalone$V5)
[1] 0.2820557

 sd(abablone$V5)
[1] 0.1308332

 median(abalone$V5)
[1] 0.276

 quantile(abalone$V5)
    0%    25%    50%    75%   100%
0.0050 0.1900 0.2760 0.3555 0.8970

 IQR(abalone$V5)
[1] 0.1655
```

Often we want to obtain all this information on a set of data, in these circumstances we can use the function `summary` which produces a summary of our data. For example with we apply this to the heights we obtain:

```
summary(abablone$V3)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0250  0.1300  0.1550  0.1514  0.1750  0.5150
```

Note we can also use summary on the whole data frame, in that case it produces a summary for each column:

```
summary(abablone)
```

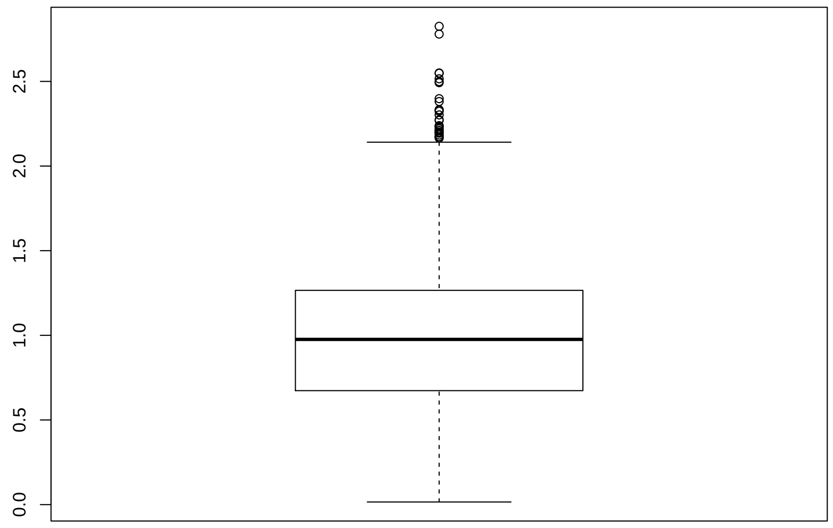
```
         V1                V2                V3                V4
 Min.   :0.1550   Min.   :0.1100   Min.   :0.0250   Min.   :0.0155
 1st Qu.:0.5050   1st Qu.:0.3950   1st Qu.:0.1300   1st Qu.:0.6733
 Median :0.5800   Median :0.4550   Median :0.1550   Median :0.9760
 Mean   :0.5615   Mean   :0.4393   Mean   :0.1514   Mean   :0.9918
 3rd Qu.:0.6300   3rd Qu.:0.5000   3rd Qu.:0.1750   3rd Qu.:1.2657
 Max.   :0.7800   Max.   :0.6300   Max.   :0.5150   Max.   :2.8255

 V5                V6
 Min.   :0.0050   Min.   : 3.0
 1st Qu.:0.1900   1st Qu.: 9.0
 Median :0.2760   Median :10.0
 Mean   :0.2821   Mean   :10.7
 3rd Qu.:0.3555   3rd Qu.:12.0
 Max.   :0.8970   Max.   :27.0
```

Boxplot

Finally we can produce a box plot for our data which graphically displays the summary statistics we have calculated. To do this we use the command `boxplot`. For example producing a box plot of the whole weight we would run the following command:

```
boxplot(abalone$V4)
```

This produces the following plot:



1.4 Exercises

1. On Canvas you will find a second data set called abalone f.data. This is the data from the original study of the female samples. Use the various plots and summary statistics you have learnt to compare the male and female populations.

2. On Canvas you will find a data set in the file birthweight.data. This data has the following columns:

Name	Variable
ID	Baby number
length	Length of baby (inches)
Birthweight	Weight of baby (lbs)
headcirumference	Head Circumference
Gestation	Gestation (weeks) Scale
smoker	Mother smokes 1 = smoker 0 = non-smoker
motherage	Maternal age
mnocig	Number of cigarettes smoked per day by mother
mheight	Mothers height (inches)
mppwt	Mothers pre-pregnancy weight (lbs)
fage	Father's age
fedyrs	Father's years in education
fnocig	Number of cigarettes smoked per day by father
fheight	Father's height (inches)
lowbwt	Low birth weight, 0 = No and 1 = yes
mage35	Mother over 35, 0 = No and 1 = yes

Use the various methods described in this sheet to explore this data set.

Note: this dataset does have headers for the first row so should be loaded into R with the following command:

```
birthweight<-read.table('birthweight.data', header=TRUE, sep=',')
```

and the columns now have the correct names. So to obtain the mean of the lengths the command should be:

```
mean(birthweight$length)
```