Lab

Introduction to Data Analysis

Lab objectives:

- To become familiar with the basic summary statistics functions in R.
- To learn to use tables to learn about a dataset
- To learn about probability distribution functions
- To visualise univariate data
- To manipulate data

Getting started

Download the following 2 files from CampusMoodle:

- consumptionTransposed.csv (see previous weeks data files)
- browserdata.csv

Read the file "consumptionTransposed.csv" (into an object called *Consumption*. There are several useful functions for examining objects. Apply the following functions to *Consumption* and try to work out what they do:

```
summary, str, class, head, tail.
```

Now apply these functions to a single column of *Consumption*, e.g. to obtain the summary for the column, take the dataset, select the Oil column and feed it to the summary function.

Feed Consumption to the select function

```
Consumption |> select(Oil) |> summary()
```

Feed the selected column, i.e. Oil , to the summary function.

Check the help pages for more information. For example,

```
?summary()
```

Summary statistics

There are several functions that return one or more summary statistics when applied to a vector of data values. `summary` also works with a dataframe as above, but many functions don't. To obtain a column as a vector, we use function `pull()`.

An example of extracting column Gas from Consumtion in the form of a vector is as follows:

```
Consumption |> pull(Gas)
```

The names of the following functions are a good indicator of what they do. We have functions `mean, median, sd, var, min, max, range, IQR, quantile, fivenum`. Apply these functions to the Gas column of the Consumption data and see if they do what you think they should. For example, to obtain the mean

```
Consumption |> pull(Gas) |> mean()
```

Again, check the help files for more information. What is the difference between `quantile` and `fivenum`? Which standard deviation (population or sample) is returned by `sd` – what if you want the other one?

Exercise

1. Read file browserdata.csv into a variable called `browserdata`. It contains details of users, including browser used (Browser), their type of membership (Membership), average daily time online (MinsPerDay) and use level (Use).
   Use summary statistic functions to explore the data.

Frequency counts

One-way tables are a useful way of checking the distribution of variable values. To get a one-way table for the Browser column in the browserdata dataset

```
counts <- browserdata |> pull(Browser) |> table()
counts
```

And to get proportions

```
prop.table(counts)
```

Two-way tables can be used to check the distribution of values for 2 variables. To get a two-way table of `Browser` and `Membership`.

```
freqBrowser <- xtabs(~ Browser+Membership, data=browserdata)
freqBrowser
```

And to get probabilities instead

```
prop.table(freqBrowser,1) # each row row adds up to 1

prop.table(freqBrowser,2)  # each column adds up to 1
```

Converting to data frame

We will be using the data in the `freqBrowser` table below, but it needs to be converted to a data frame. [this is needed to use the ggplot function in the bar plot section below].

```
# convert data to dataframe first
e <- as.data.frame(freqBrowser)
```

Use summary statistics to explore the new data frame that you have just created. What columns does it have?

Plots when analysing single variables

Plots are often used to analyse data using a visualisation. You have seen most of these plots already. Below you can see examples of their use.

Bar chart plots

The following code produces a stacked bar plot of browser use per membership type.

For a simple bar plot

```
p <- ggplot(e, aes(x=Membership,fill=Browser))
p <- p + geom_bar(aes(y=Freq),  stat="identity")
p
```

We can add a title and labels with the labs() function

```
p <- ggplot(e, aes(x=Membership, y=Freq, fill=Browser))
p <- p +  geom_bar(stat="identity")
p <- p +  labs(title = "Browser use by membership type",
                x = "Membership type", y = "Frequency count")
p
```

We can change the colour scheme and the background

```
p <- ggplot(e, aes(x=Membership, y=Freq, fill=Browser))
p <- p +  geom_bar(stat="identity")
p <- p +  labs(title = "Browser use by membership type",
                x = "Membership type", y = "Frequency count")
p <- p +  scale_fill_brewer(palette="Set1")
p <- p +  theme_classic()

p
```

Similarly, the bar plot could show each browser data in a different column.

```
p <-  ggplot(e, aes(x=Membership, y=Freq, fill=Browser) )
p <- p +   geom_bar(stat="identity", position=position_dodge())
p <- p +   labs(title = "Browser use by membership type", x =
"membership type", y="Frequency count")
p <- p +   scale_fill_brewer(palette="Dark2")
p <- p + theme_classic()
p
```

Histograms

Create a histogram of the daily time spent online as follows:

```
p <- ggplot(browserdata, aes(x= MinsPerDay))
p <- p +   geom_histogram(aes(y=..density..), binwidth=20 )
p
```

Adding colour, titles, labels, and a line

```
p <- ggplot(browserdata, aes(x= MinsPerDay))
p <- p +     geom_histogram(aes(y=..density..),col="red", fill="red"
, binwidth=20)
p <- p +     labs(x="Average time spent online (mins/day)",
y="Density")
p <- p +     geom_density(col="blue")
p <- p +     theme_classic()
p
```

Experiment with the binwidth parameter.

Dot plots

The following code produces a dot plot for column Use, which shows the distribution of values.

```
p <- ggplot(browserdata, aes(x= Use))
p <- p +     geom_dotplot(binwidth=0.05)
p
```

Experiment changing the binwidth.

The following produces a nicer-looking dot plot.

```
p <- ggplot(browserdata, aes(x= Use))
p <- p +     geom_dotplot(col="red", fill="red" , binwidth=0.05)
p <- p +     labs(x="Use level", y="")
p <- p +     theme_classic()
p
```

Box plots

The following code produces a boxplot for the average daily time online.

```
p <- ggplot(browserdata, aes(y= MinsPerDay))
p <- p +     geom_boxplot()
p
```

Adding colour, titles, etc.

```
p <- ggplot(browserdata, aes(y= MinsPerDay))
p <- p +     geom_boxplot(col="blue", fill="lightblue")
p <- p +     labs(title="User average minutes/day online",
                      x="", y="")
p <- p +     theme_classic()
p
```

You may want to have a boxplot per Browser type, by adding a variable to the x axis. For example

```
p <- ggplot(browserdata, aes(x=Browser, y= MinsPerDay))
p <- p +     geom_boxplot(aes(col=factor(Browser)) )
p <- p +    labs(title="User average minutes/day online by browser",
                x="",y="Average minutes per day")
p <- p +    theme_classic()
p
```

Exercises

2. Download a couple of datasets of interest to you. Convert the data to csv format if needed.
3. Use summary statistics to explore the data.
4. Use plots to further explore the data.

Note: R does have some datasets already in it. You may check what's available using
```
data()
```

You can use any of the datasets available. For example, to explore the Orange dataset use
```
head(Orange)
str(Orange)
summary(Orange)
```

Plotting functions

It's not immediately obvious how to plot a specified function using **ggplot**. The trick is to set the required limits in the call to **ggplot** and then use **stat_function**. Which function does the following return? (Note the use of pi.)

```
p <- ggplot(data.frame(x=c(-pi,pi)),aes(x=x))
p + stat_function(fun=cos)
```

Try to find out the names of other common functions available in R.

You can also define your own functions, as below.

```
mysquare <- function(xval) {xval*xval}
```

Make use of mysquare to plot the function x^2 between x = 0 and x = 1.

You can annotate and modify the graph in the manner shown in previous labs. Try to add a chart title, modify axis labels and change the colour of the curve.

Probability distributions

Common probability distributions are also available in R. For the normal distribution, we have the following:

`dnorm(x)` returns the density function at x.

`pnorm(x)` returns the cumulative density function at x. This corresponds to the probability that a random number drawn from the distribution is less than x.

`qnorm(q)` returns the value of x for which pnorm(x)=q. For example, qnorm(0.25) returns the first quartile of the distribution.

`rnorm(n)` returns n random numbers drawn from the normal distribution.

Other distributions have similar functions, named on the same pattern. For example, **dpois, ppois, qpois, rpois** for the Poisson distribution.

To see the normal distribution:

```
p <- ggplot(data.frame(x=c(-3,3)),aes(x=x))
p <- p + stat_function(fun=dnorm)
p
```

Now graph the cumulative density function over the same range.

We can change the parameters of the normal distribution using the arguments ‘mean’ and ‘sd’; the defaults are mean =0 and sd =1 (the standard normal distribution). Note how we get these arguments into **stat_function**.

```
p + stat_function(fun=dnorm,args = list(mean=1,sd=2))
```

Experiment with the values of `mean` and `sd` until you understand how they affect the shape of the distribution.

Try typing the following commands in the sequence shown. Do you understand what is happening? What does **set.seed** do? How might this be useful?

```
rnorm(10)
rnorm(10)
set.seed(123)
rnorm(10)
rnorm(10)
set.seed(123)
rnorm(10)
```

Visualising univariate data using a function to obtain the dataset (skip if getting confused).

Draw 1000 numbers from the standard normal distribution and assign the output to a variable *normal.sampled*

To draw a histogram of this data:

```
ggplot(NULL,aes(x=normal.sampled,y=..density..)) + geom_histogram()
```

(We set data to be NULL to use *normal.sampled* directly, instead of putting it into a dataframe.)

What happens if you omit the instruction '`y=..density..`'?

Notice the warning about the 'binwidth' parameter. Experiment with different binwidths to see what effect this has.

Compare the histogram with the graph of the normal density function. Now increase the sample size to 10000 and repeat. What do you notice? Do you understand what is happening?

Try replacing **geom_histogram** with **geom_freqpoly** and **geom_density**. These functions may give similar results but are very different in reality. The frequency polygon is just a different representation of the histogram, while the density function is a smoothed estimate of the underlying population density. Read the help on these functions for more details. Experiment with different values of 'adjust' (the bandwidth) for **geom_density**.

Now let's look at the Poisson distribution, which is a discrete distribution. We tak e 50 samples from a Poisson distribution of mean 3.
[Use this instruction: `poisson50 <- rpois(n=50,lambda=3)`]
This time we will wrap the result in a dataframe, although we could have used the NULL trick as above.

```
p50.df <- data.frame(poisson50)
p <- ggplot(p50.df,aes(x=poisson50))
p <- p  + geom_histogram(binwidth=1)
```

Since the random numbers are all integers, we have set the bin width to be 1. We have a fairly small amount of data, so let's show them in a dot plot.

```
plot.pois + geom_dotplot(binwidth=0.5)
```

As a final example, we shall sample two different normal distributions and compare them using a boxplot. Do you understand what the picture is showing? Can you work out what **stack** does?

```
normal.m1.s1 <- rnorm(1000,mean=1,sd=1)
normal.m2.s3 <- rnorm(1000,mean=2,sd=3)
norm.compare <- data.frame(normal.m1.s1,normal.m2.s3)
bplot.data  <- stack(norm.compare)
p <- ggplot(bplot.data,aes(x=ind,y=values,fill=ind))
p <- p + geom_boxplot()
p
```