Lab

Some Basic Plots

Lab objectives:

* To use an editor
* To create vectors and data frames (or tibbles)
* To input data from files
* To become familiar with some basic plots: histograms, bar plots (and column plots) as well as boxplots.

Other basic plots will be covered in future weeks. Similarly the addition of elements such as titles, labels, etc, will be covered in future weeks.

## Libraries

You will use library tidyverse. Load it

library(tidyverse)

## Remider about the syntax

> Is the screen prompt

+ is the continuation prompt (when your command is unfinished).

Two or more expressions can be placed on a single line so long as they are separated by semi-colons. For example

a <- 7; b<- 10

# Vectors

In the previous lab you learnt the c() command to create a vector. For example,

|  |
| --- |
| primeNums <- c(2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97)  primeNums  [1] 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67  [20] 71 73 79 83 89 97 |
|  |
|  |

# Data frames and tibbles

Data frames and tibbles are structures which allow the holding of data of different types (like a table). Thus, each column has its own data type. You can construct:

* A data frame using the data.frame function, indicating the values of each column using vectors.
* A tibble using the tibble function, indicating the values of each column using vectors.

Most of the functions which can be used with data frames can be used with tibbles. Below, we will learn functions which apply to both.

If data frames and tibbles are very similar, why are you learning both? Both types of data structure are widely used, so it is important that you learn how similar they are. Tibbles are said to be the modern version of data frames (and are a type of data frame).

Assume you want to create a dataset which holds information about your studies, including module codes, module names, and credits. You can first create a vector for each of the 3 components (columns) in your data frame.

# define the module codes

m <- c("CMM004", "CMM007", "CMM020", "CMM022", "CMM024",

"CMM028", "CMM012","CMM513")

# define the module names

mn <- c("SPE", "Intranet Systems Dev", "Data Vis and Analysis",

"Databases", "OOP", "ITI and Administration",

"Project Investigation", "MSc Project")

# define the credits

cr <- c(15,15,15,15,15,15, 15, 45)

You can then put the 3 vectors into a data frame called whatToStudy.

# put everything into a data frame

whatToStudy <- data.frame(module = m, moduleName=mn,

credits = cr, stringsAsFactors=T)

Note that at we have put stringsAsFactors=T in our data frame code. This is important, as it tells R to treat strings as factors.

You can view the content of whatToStudy by clicking on it in the environment area (top right).

### Accessing columns

To access individual columns use the name of the data frame, a $ and the name of the column (no spaces). For example:

whatToStudy$moduleName

The same can be achieved by putting the name of the dataframe and a comma followed by the column position in square brackets. For example,

whatToStudy[,2]

Note that the first element in the square brackets is empty and refers to the row number. If it is empty, we want all rows.

### Accessing rows

To access the 4rd row, use the data frame name followed by the required row followed by a comma in square brackets.

whatToStudy[4,]

### Accessing elements

To access individual elements, you can put its position in square brackets after the name of the column. For example, the 3rd module name

whatToStudy$moduleName[3]

or you can use an index for both the row and the column

### Data frame dimensions

You can use dim() to get the dimensions (rows and columns) of your data frame. For example,

dim(whatToStudy)

You can also use nrow() and ncols() to get just the number of rows or the number of columns.

nrow(whatToStudy)

ncol(whatToStudy)

### Summary of data frame

To get a summary of what the data frame contains use

summary(whatToStudy)

In this case the summary is not very interesting, as most values only occur once, but in the exercise section you will see more interesting ones.

To create a tibble table similar to the dataframe above you can use

whatToStudyT <- tibble(module = m, moduleName=mn,

credits = cr)

To ensure strings are factors you use

whatToStudyT <- whatToStudyT |>

mutate(across(where(is.character),as\_factor))

## Exercise

1. Check that the commands that you have learnt above for data frames work with a tibble too. Use tibble whatToStudyT, which you created earlier.

# Obtaining objects names and removing objects

To obtain the names of all the objects defined you can use

ls()

To remove an object use rm. For example, to remove object power3nums

rm(power3nums)

To remove everything use

rm(list=ls())

# Working on a file

The file menu (top left of RStudio) allows you to create a new file, open an existing file and save a file. This can be useful for keeping your code together.

Download the solutions to lab 1 from CampusMoodle (in Week 1). You will notice that the file has extension “.Rmd”. Open this file.

You will see the file is divided into sections. You can select one section, and click on *run* (green triangle on the top right of the section) to run it. The results will appear below the selected block.

R markdown allows you to combine R code with text in a single document. The file menu (top left of RStudio) allows you to create a new R markdown file (extension .rmd), open an existing file and save a file. Within an R markdown file, you can execute sections or you can “knit” the whole file onto a document (word, pdf or html) which contains code, text and the results of running the code. The lecturer gave a demo on R markdown during the lecture time and documentation is available on CampusMoodle. **It is recommended that you use R markdown from now on.**

To create a new R Markdown file go to

**File ⭢ New File ⭢ R Markdown**

A window will appear which will ask you for details such as title, author, etc. Complete the details (these can be changed later on) and press OK. A new file named Untitled1 will appear.

**Save your file with a new name on your H: drive**. If your file is not saved, you will not be able to knit it onto a document.

Delete everything from line 12 onwards, as it is just an example and has no use in your R program.

You can now start coding by inserting a new R chunk (small green square on the top right of the left panel).

**Note that for an R Mardown file to work all the commands need to be included in it, in the right order.** This means that:

* The environment is wiped and then the code is run from the top of the file towards the bottom. You may want to include

rm(list=ls())

at the top of your file (first new R chunk) so that you test your code with a clear environment.

* You should load all the libraries using code. You are advised to do this in your second new code chunk. For example

library(tidyverse)

* You must load your data files using code (i.e. do **not** use ticks on the packages pane) – see later section entitled “Data input”.

# Colours and colour palettes

R has several colour palettes that you can use for your colour selection:



* rainbow()
* heat.colors()
* terrain.colors()
* topo.colors()
* cm.colors()
* grey()

To get all the available colours type

colors()

You will use some of the colours in a later section.

# Data input

If you are to use big sets of data, it is useful to be able to input data from a file.

Download file *pressureTemp.csv* from CampusMoodle and save it on your H: drive.

Inspect this simple dataset.

On the console window (left), type the following

pressureTemp <-read.csv("pressureTemp.csv", header=T,

stringsAsFactors = T)

StringsAsFactors=T ensures all strings are taken as factors (values from a specific set of values) and not as characters.

Alternative, if you would like to store your data in a tibble use

pressureTempTib <- read\_csv("pressureTemp.csv" )

pressureTempTib <- pressureTempTib |>

mutate(across(where(is.character),as\_factor))

The second part is equivalent to loading onto a data frame with the setting stringsAsFactors=T

Remember to add the path to the data file if the file is not in your working directory.

This reads the pressureTemp dataset in comma separated form and assigns it to variable *pressureTemp*. Note that the assign operator ( “<-“) has assigned this dataset to *pressureTemp*. This data is of type data.frame. Also note that header=T means that the first line of the file contains the names for the columns. The inclusion of stringsAsFactors = T in the call tells R to treat strings as factors, not characters.

You will see that *pressureTemp* appears on the Environment tab. If you click on it, it will appear on the top left.

Now that we have some data, we can plot it.

# Plots

Library ggplot2 contains a number of functions which can be used to create sophisticated plots. We will learn the details of this library next week, but we can start using some of the functions now.

The first thing to do is load the library

library(ggplot2)

We are going to use a variable, which we are going to call p, to put in the code for various elements which we need in our plot. Once we have everything in p, we will call it so that the plot is generated.

# Histogram

Let’s see how to build a simple histogram plot using an example. Let’s assume that we want to plot a histogram of the pressure attribute in the pressureTemp dataset. First, we define the dataset and the variable to be used using function ggplot() and we put them in p.

p <- ggplot(data = pressureTemp, aes(x=Pressure))

Now we add the fact that we want a histogram.

p <- p + geom\_histogram()

OK, we are now ready to plot the histogram

p

Note that the variable to be used can sometimes be defined in different ways. For example, above we did the following

p <- ggplot(data = pressureTemp, **aes(x=Pressure)**)

p <- p + geom\_histogram()

p

The code below is equivalent, where the variable included in the geom\_histogram instea of in the ggplot definition.

p <- ggplot(data = pressureTemp)

p <- p + geom\_histogram(**aes(x=Pressure)**)

p

Our histogram is a little dull, and has lots of bars. We can change the colour of the bars by using setting “fill” in the geom\_histogram function.

For example,

p <- ggplot(data = pressureTemp, aes(x=Pressure))

p <- p + geom\_histogram(fill="blue")

p

Similarly, we can set the binwidth to make the bars wider. For example,

p <- ggplot(data = pressureTemp, aes(x=Pressure))

p <- p + geom\_histogram(binwidth=4, fill="blue")

p

An alternative to changing the binwidth is to specify the number of bins (columns) we want. For example , to have 10 columns

p <- ggplot(data = pressureTemp, aes(x=Pressure))

p <- p + geom\_histogram(bins=10, fill="blue")

p

Note that when 2 columns have the same height, they appear as one wider column.

## Exercises

1. Plot a histogram of the temperature in pressureTemp.
2. Modify your temperature histogram to add colour to it. Experiment with colours not used earlier.
3. Modify your temperature histogram to have 7 bins.
4. Modify your temperature histogram to have bins of width 2.

# Bar plot

Bar plots are built in a similar way to histograms, but instead of using a geom\_ histogram, a geom\_bar is needed and binwidth and bins cannot be used.

## Exercise

1. Build a bar plot to display the counts for the Readby column in the pressureTemp dataset.
2. Change the colour of your bars to a colour you have not used before.

# Bar vs column plots

The file consumption.csv contains some data regarding energy consumption taken from one of BP’s reports. The data is measured in million tons of oil equivalent.

Download the consumption dataset from CampusMoodle and load it onto RStudio, e.g.

consumption <- read.csv("consumption.csv", header=T, stringsAsFactors = T)

Inspect the consumption data frame. It is a very small dataset. Imagine that you would like to plot some of the amounts. You do not want to count the number of times each energy type appears (the all appear once!) instead. You want to know the energy consumption for each type in a specific year, say X2035. In this case, we are wanting two variables, Energy and X2035, so we will need to indicate this.

p <- ggplot(consumption, aes(x=Energy, y=X2035))

We cannot use a geom\_bar, as it is used to count occurrences of a value and uses only one variable. Instead, we will use geom\_col

p <- p + geom\_col()

We are now ready to display the plot

p

The results may not be very useful because of the way the data was presented.

## Exercises

1. Use the consumption dataset to display the actual energy consumption per energy source in 2030, using pink bars.
2. File BPConsumptionTransposed.csv contains the same data as the consumption.csv file, only transposed (columns and rows swapped). Load this file

consumptionTransposed <- read.csv(“consumptionTransposed.csv",

header=T,

stringsAsFactors = T)

1. Plot the Oil per Year. You want the years on the x axis and the amount of oil consumed on the y axis. Comment on the plot.
2. Change the Year column to be a factor instead of a number. You can do this by typing

consumptionTransposed$Year <- as.factor(consumptionTransposed$Year)

1. Plot Oil per Year again. Can you spot the difference with your previous plot?
2. Plot Coal per year in a different colour.

# Boxplots

Look at the pressureTemp dataset. You suspect there is something wrong with the device that one of the readers (in column readby) uses for one of the measures as their values seem to be quite different. You want to check Temperature first.

You can obtain a boxplot per reader for the Temperatures as follows

p <- ggplot(pressureTemp, aes(x=Readby, y=Temperature))

p <- p+ geom\_boxplot()

p

You will see that the 3 boxes look quite similar and that they all have some outliers.

Remember – boxplots sort the data and present them in a way which indicates e

* minimum value – bottom of the vertical line below the box.
* q1 – the point with the lowest 25% of data below it – bottom of the box
* median – value in the middle of the data if all the values are sorted in ascending order. A horizontal line in the box.
* q3 - the point with 75% of the lowest data below it – top of the box
* maximum – top of the line on top of the box.

If there are outliers, the maximum and minimum may be outliers instead of top (bottom) of the highest (lowest) vertical line.

## Exercises

1. Change the colour of the boxes in the temperature boxplot to red.
2. Get each box to be a different colour. Tip: instead of using just one colour, use a vector of colours.
3. Get each box to be a different colour using a colour palette (see earlier section). Tip: to use 3 terrain.colors use terrain.colors(3)
4. Obtain a boxplot of the Pressure for each reader (i.e. Readerby). Can you spot a problem? Who may have a faulty pressure reader?

If you have time: go to <http://www.datavis.ca/gallery>/ [accessed 02 February 2022] for examples of good and bad visualisations. Explore the examples fully.