CS5701/2 Joint W1 Worksheet Notebook

Martin Shepperd and Isabel Sassoon

15/09/2022

This Worksheet is organised as an RMarkdown file. You can **read** it. You can **run** the embedded R and you can **add** your own R. We suggest you save it as another file so, if necessary, you can revert to the original.

Whenever you click **Preview** in the RStudio menu it will render this markdown into nicely formatted html which you can look at it in the Viewing Window in RStudio or via any web browser. You may find this easier to read, however, you must edit the .rmd file, i.e., the RMarkdown in the Edit Pane if you want to make any changes. You are encouraged to *explore* and *experiment.* Change something and see what happens!

Note, that as this is the first week, this Lab will be run as a joint session between Quantitative Data Analysis and Modern Data. This is because many of the topics and challenges are the same or closely interwoven. In subsequent weeks, the Labs will be distinct and follow the needs of each module although never be afraid to carry over lessons and ideas from one module to the other.

Pre-requisites

1. You should have completed the Week 0 "Getting Ready" and minimally must Install and test R and RStudio material. Alternatively, follow the guided worksheet and video on installing R and an example in the Week 0 material "QDA Week 0"
2. You should have attended/read the Week 1 Lectures:

- Modern Data Introduction
- QDA

0. Worksheet Topics

1. Create, save and edit R Notebooks using RStudio
2. Exploring the use of R using R
3. Some Basic R practice
4. Describing Data
5. Exploring Data
6. The diamonds dataset casestudy

A. *Extension*: Appendix - notes on YAML

1. Lab: create, save and edit R Notebooks using R~Studio

This worksheet is written using RMarkdown and saved as an R Notebook.

It is an extremely convenient way to combine explanatory text written in a very simple markdown language and interactive R. We will use them regularly for all non-trivial R code in both modules, including for labs and also for your coursework (CS5801) and exam (CS5802). So it's important you become comfortable with using RMarkdown documents.

If you've not already viewed my video "Introducing RStudio" or you lack confidence in using this important tool I suggest you should view it here.

1. You need a YAML header (see this file for an example, or click here for more information).
2. You can enter plain text as-is. Any simple editor will do or write them directly in RStudio. *Avoid* word processors such as Word because the text is not stored as plain text and includes lots of hidden non-printing characters which means you will got lots of horrible side effects!
3. You can use `#`, `##`, etc for different heading levels. Notice how RStudio automatically generates a nested Outline (navigation) Pane for you.

4. If you start a line with `1.` etc this generates an enumerated list (as per this example). There must be a space between the full stop and the text. If you just want bullets you can use `-` instead.
    - You can nest lists by indenting by 4 spaces.

5. You can insert a code chunk by clicking the green *Insert (Code Chunk)* button on the Edit Pane toolbar or by pressing *Cmd+Option+I*. Or you could, if you really want, type the three backticks followed by the language in braces, then the code and close the chunk with another three backticks.
6. You can also view and edit your RMarkdown in 'Visual' mode which shows the result of your formatting. It's your choice but I suggest you stick with 'Source' mode to start with until you are confident editing the RMarkdown.

There's also a very helpful guide on RMarkdown from RStudio here.

Below is an example of an R code chunk which can be executed from within the R Notebook. You can do this by clicking the green triangle to the top right of the chunk area within the Notebook. If you wanted to run all the code up until, and including, this code chunk you can click the grey triangle with the green bar [1].

```
# This is an R code chunk
print("Hello cruel world")
```

```
## [1] "Hello cruel world"
```

Exercise 1.1: Edit the R code to output "Goodbye cruel world" or some other message. Re-run the code chunk.

Exercise 1.2: Save the edited RMarkdown notebook When you save the notebook, an HTML file containing the code and output will also be saved (click the *Knit* button to preview the HTML file). The raw R markdown has a suffix .Rmd and the rendered HTML will have a suffix .html. Note that you can Knit to other formats if you rather such as pdf [2].

Check you can see the html version. The preview shows you a rendered HTML copy of the contents of the editor. The output from the R code chunks is whatever was generated when it was last run in the RStudio. You can also view the file in any web browser.

Exercise 1.3: Add a new 3rd level heading named "My RMarkdown Exercise" here. Add an anchor label `{#<anchor-name>}`. Now at another point in the document you can link to your anchor, as follows: `[text to display]` immediately followed by the '(). Lines 25-31 of this RMarkdown file are examples.

[1] In this case there is no previous code so it would make no difference but often RMarkdown contains multiple code chunks.

[2] The way the file is rendered into pdf relies upon you having an installation of LaTeX on your machine. If this doesn't work for you just focus on the html or reading the RMarkdown directly.

Exercise 1.4: Create a new RMarkdown file using the `File > New File > R Notebook` menu option. RStudio provides you with a basic template which you can modify as needed. Add a meaningful title and your name as an author in the YAML header. Render the file using the `Preview` option.

Click on my video here if you want further guidance on creating and manipulating RMarkdown documents.

To take it further there's a nice blog of R Markdown Tips.

So now you can handle the RMarkdown basics. Let's move onto R.

2. Exploring the use of R using R

2.1 Demand for R programmers

How useful are R skills and tools? An analysis of 2681 data scientist job postings from 8 North American cities on January 25th, 2020 indicated the top 10 tools required by the employers are:

```
Python (62%)
SQL (40%)
R (39%)
Spark (21%)
Cloud (20%)
Amazon Web Services (20%)
Java (19%)
Tensorflow (16%)
Hadoop (15%)
SAS (13%)
```

The data was collected by analysing online job adverts for data scientists on the Indeed website in North America. Note that Tools is a categorical and multi-valued attribute of a job listing, i.e., one listing could have zero or more tool categories associated, thus the percentages do not sum to 100%. NB 14% of the sampled job listings are excluded (since it is stated that only 86% of adverts identified skills or tools).

Exercise 2.1: It's important to look at the data critically, though we must accept nothing is certain or perfect! What questions or concerns might you have?

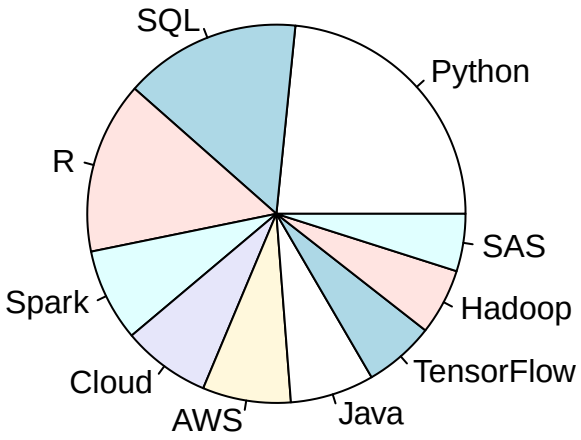
Since we're thinking about data and data analysis, it would be helpful to visualise the data. Unfortunately, we don't have access to the raw data so we will need to reproduce it. Therefore I have 'scraped' some of the data from their article and plugged this into R (see below). Don't worry about the coding details for the present although be aware that for reasons of simplicity I'm just using Base R for the graphics (functions like `pie()` and `barplot()`). Later we will introduce the {ggplot2}[3] package which will enable almost complete control and flexibility in producing beautiful data visualisations, however, if you are already a confident R programmer feel free to jump ahead.

If you want to know more about installing or using packages in the R ecosystem see my video here.

```
# Reproduce the data from the January 2020 required tools for Data Scientists

percent <- c(62,40,39,21,20,20,19,16,15,13)
skill <- c("Python","SQL","R","Spark","Cloud","AWS","Java","TensorFlow","Hadoop","SAS")
# Generate the pie chart
pie(percent, main="Pie chart of in demand data scientist tools", labels=skill)
```

[3] I, and many authors, use the convention of curly braces to denote an R package, so {ggplot2} means the package named ggplot2 which you could install with `install.package("ggplot2")` and then read into memory with `library(ggplot2)`. If you're already an expert you can look at Appendix 2 to see how to do a pie chart with {ggplot2}, however, be warned it's a bit fiddly because you're not really encouraged to make pie charts due to their many disadvantages.

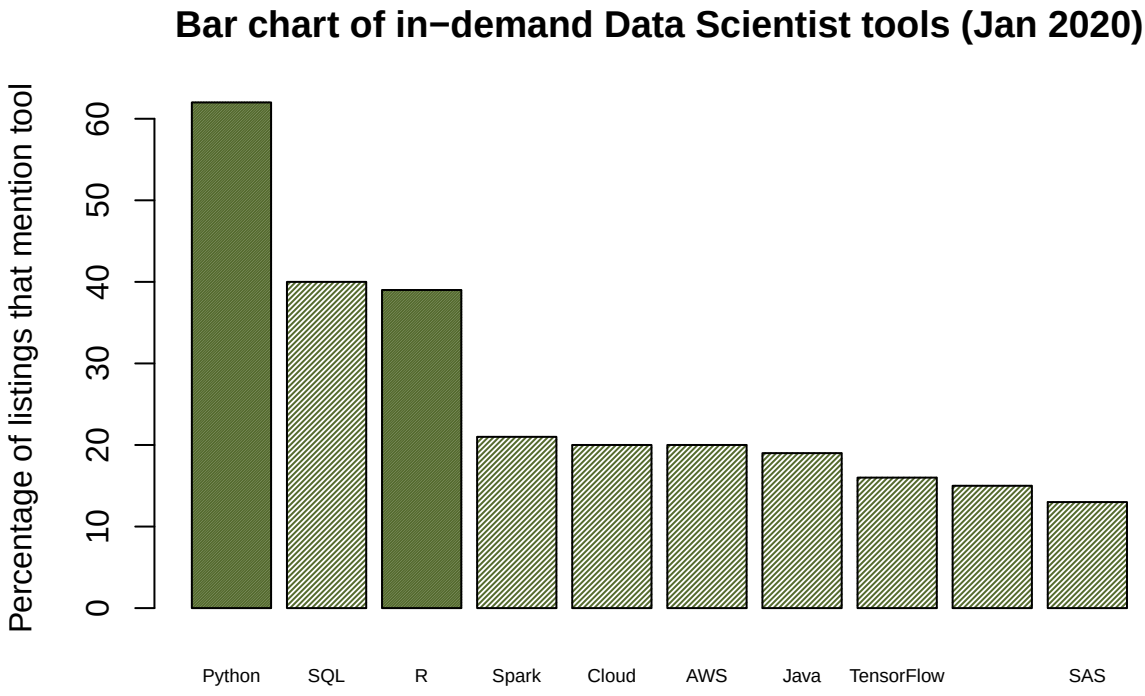
Pie chart of in demand data scientist tools



Exercise 2.2: Using the `clockwise` parameter change the order of display. Type `?pie` for help if required.

Exercise 2.3: If you look at the help for the `pie()` function you will read "Pie charts are a very bad way of displaying information." Why do you think this is so? [Prompt: what might be the impact of multi-valued data, e.g., many adverts will ask for more than one skill?]

```
# NB we have already reproduced the data: percent and skill come from the
# previous piechart chunk

# Generate a bar chart
barplot(percent,
main="Bar chart of in-demand Data Scientist tools (Jan 2020)",
sub = "Source: https://j.mp/2Ga8CD1",
ylab="Percentage of listings that mention tool",
names.arg=skill,
cex.names = 0.6,                                    # shrinks text size so fits on x-axis
col="darkolivegreen",                               # colour of the bars
density=c(100, 50, 100, 50, 50, 50, 50, 50, 50, 50)) # Shades Python and R in a darker colour
```



Source: https://j.mp/2Ga8CD1

Exercise 2.4: Hopefully you consider this bar chart to be an improvement, but it's not perfect. Can you think of any further improvements or clarifications? (Hint: some of the bars are quite hard to read off the exact proportion.) Remember that if the goal of Data Science is to *communicate* findings, then how we present information is of vital importance. For this reason we will return to this topic (Chapter 4) to dig deeper on data visualisation.

Exercise 2.5: The same article also lists the top 50 skills required e.g., machine learning, statistics, ... Scrape (estimate the percentages for the top 10) and produce equivalent charts to those for these skills.

Extension exercise 2.6: The above two plots are performed using base R (which is perfectly adequate for simple charts). A powerful and flexible alternative is the {ggplot2} package. How might you reproduce these charts using {ggplot2}? NB This exercise is primarily intended for those who already have some knowledge of R or who consider themselves proficient programmers.

3. Some Basic R practice

Essential tips for R:

- R is very sensitive to cases and ***syntax***.
- Objects are everything in R. Objects with parenthesis are ***functions*** and without are ***data types***.
- `#` in front of a sentence or a word is treated as a comment and is not executed.
- For help in R, `?<function or command>` e.g., `?head`

3.1 A simple variable assignment example

In the previous task you may well have largely relied on the provided R code. Now we move onto learning about basic R syntax and some coding exercises for you to undertake.

Suppose I'm trying to calculate how much money I'm going to make from my *Modern Data* book. There are several different things I will need to store:

- `sales` how many copies I'll sell `sales`

- `price` the price per copy
- `numberStudents` number of students in the class (assume I sell one copy per student)
- `recommend` the proportion of students so impressed that they recommend the book to friends and family
- `income` the income generated

Let's turn this into R.

```
# Compute sales income for Martin's book

price <- 10                                     # assume price is 10 UKP per book copy -
                                                # actually it's free but never mind!
numberStudents <- 100                           # number of students in the class
recommend <- 0.5                                # proportion of students who recommend the book

sales <- numberStudents * (1 + recommend)
income <- sales * price
paste("Martin will earn:", income)              # display a readable string with the paste function
```

```
## [1] "Martin will earn: 1500"
```

Although we could just output `income`, I have added some text to make the output of the code easier to interpret with the `paste()` function to combine a character string literal delimited by quotation marks and the calculated income.

Exercise 3.1: Change the number of students to 200 and the recommendation rate to 0.2. What's my new estimated income?

Exercise 3.2: Update the sales income R code to handle tax. Assume that income is taxed at a flat rate of 20%. Modify the above R code to output gross income and net (after tax) income. Using RStudio you can edit this Week 1 Worksheet Notebook.

```
# Your extended code needs to go here
```

HINT: I recommend you define a new numeric variable to store the tax rate and another for net income. For clarity you might want to rename `income` to make it obvious that it stores gross income.

Extension Exercise 3.3: How would you answer the question: it's possible to write a much shorter R program without any extra variables, so why bother?

It's very *important* to get into the habit of writing easy to follow R code. For this reason I have added comments and tried to select meaningful variable names.

3.2 Using vectors

Recall that at a Vector is a repeated group of the **same** data type, e.g., a vector of character strings.

```
# Define a new vector
greetings <- c("Hi!", "Hello", "Good morning", "Saludo", "Hej")
greetings
```

```
## [1] "Hi!"          "Hello"        "Good morning" "Saludo"       "Hej"
```

We can also check the type of `greetings`

```
# Check the type of greetings
is.vector(greetings)
```

```
## [1] TRUE
```

Exercise 3.4: Check the type of the variable `price` (from Section 3.1). What output do you expect?

To fully exploit the power of vectors we can access subsets of elements. To do this we use the square bracket notation `[<n>]` where n is the position of the required vector element.

```
# Different ways to index vector elements
greetings[2]        # Access the 2nd element
```

```
## [1] "Hello"
```

```
n <- 3
greetings[n]        # You can use a variable as an index
```

```
## [1] "Good morning"
```

```
greetings[1:3]      # Or you can specify a range vector elements
```

```
## [1] "Hi!"          "Hello"        "Good morning"
```

Notice in the above R chunk, that we can access more than one element at a time by using the `:` operator to return a range of elements. The range can be specified by literals as in `[1:3]` or integer[4] variables e.g., `[m:n]`.

3.3 Using data frames

Data frames are amongst the most useful of the different data structures. These are two dimensional and unlike 2-dimensional matrices can comprise different constituent data types.

If you're not very confident about data frames I suggest you view my short(ish) video here.

Let's start by examining a built-in data set mt-cars that stores information about motor cars from the 1970s that is stored as a data frame.

```
# Assign the built in data object mtcars to a new data frame df
df <- mtcars

# View the data frame in a new window in RStudio
View(df)
```

Exercise 3.5: How many variables and observations does `df` have? HINT: There are functions `nrow()` and `ncol()` if you can't be bothered to count.

You can use the `$` operator to specify a column/variable from a data frame, for instance:

[4] If the index variables are not integers R will do its best to coerce the values to integers.

```
# Display the observations for cylinder numbers from df
df$cyl
```

```
##  [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
```

```
# Apply the median function
median(df$cyl)
```

```
## [1] 6
```

Exercise 3.6: Using the `$` operator determine the mean and minimum mpg for cars in df.

Exercise 3.7: Update the number of gears for the Mazda RX4 from 4 to 6. HINT: `mt$gear` is a vector so you can use an index to reference a particular element.

So we can use the dollar operator to identify columns/variables. What about rows/observations? There are several ways to achieve this. Using indices via the `[ ]` notation notice that since a data frame has two dimensions two arguments are needed, but since we want the entire row we can leave the argument after the comma blank.

```
# To retrieve the 2nd row we could have.
df[2,]
```

```
##               mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

Exercise 3.8: Modify the above code to retrieve columns 1-3. HINT: remember the `:` operators from when we were manipulating vectors.

The other way to retrieve specific observations is when the rows have names e.g., "Mazda RX4", as in this case.

```
# Retrieve the observation named "Valiant"
df["Valiant",]
```

```
##          mpg cyl disp  hp drat   wt  qsec vs am gear carb
## Valiant 18.1   6  225 105 2.76 3.46 20.22  1  0    3    1
```

Exercise 3.9: Change the number of cylinders for the Valiant to 12.

Quiz: Please check your understanding of this chapter via this quick quiz (10 simple multiple choice questions with answers).

4. Describing Data

Both during QDA Lecture 1 and in the pre-reads material we introduced measures of spread and location. Now we can practice computing them using R!

4.1 Measures of location

Let's see how to use R to compute the measures of location (mean, median, and mode) and spread (range, inter quartile range, standard deviation and variance):

We are using a small vector of numeric data

```
data_vector <-c(4,7,2,5,6,2)
```

Mean

```
mean(data_vector)
```

```
## [1] 4.333333
```

Median

```
median(data_vector)
```

```
## [1] 4.5
```

Mode

Unlike the mean and the median there is *no* built in R function to compute the mode. There are many ways of implementing our own mode function. As a hint consider turning the vector into a table of frequencies that can then be sorted in decreasing order. You can see our solution here in the Appendix.

Extension exercise 4.1: Again for those with more programming background we can wrap this code into a user-defined function (see Chapter 3 of the Modern Data book). See below.

```
# Define a new function called mode
mode <- function(v) {
# calculate mode
# Just some dummy code you need to update as the answer is probably
# not 42 (just saying!)
return(42)
}

# Apply the function to our vector
mode(data_vector)
```

```
## [1] 42
```

Alternatively, more easily, we can install an R package {modeest} that contains a mode function.

```
# Remove the comment for the line below if you've never installed this package.
# Remember you only need do this once, but you need to run library() every new session.
# #install.packages("modeest")
library(modeest)
mfv(data_vector, method = "mfv")  # mfv=most frequent value aka the mode
```

```
## [1] 2
```

Weighted Mean

You will have seen during the induction talks and if you look at the Brunel Senate Regulations 3 and 4 that in order to compute your **grade point average** you will need to take into account the relative weights (in credits) of each module.

You can see the code below that uses the weighted mean function.

```
grade.point<-c(13,16,12)
credit<-c(15,15,30)
weighted.mean(grade.point, credit)
```

```
## [1] 13.25
```

This may not be something you are interested in now but you may find it handy later on in the academic year!

4.2 Measures of variation

Range

Range –> Minimum and the maximum value

```
range(data_vector)
```

```
## [1] 2 7
```

Interquartile range (IQR)

IQR: 25th and 75th percentile values

```
IQR(data_vector)
```

```
## [1] 3.25
```

This is based on the 25th and 75th percentile, which can also be computed in R using the quantile function:

```
quantile(data_vector, c(0.25,0.75))
```

```
##  25%  75%
## 2.50 5.75
```

Standard Deviation

```
sd(data_vector)
```

```
## [1] 2.065591
```

Variance

```
var(data_vector)
```

```
## [1] 4.266667
```

Is this is the easiest/fastest/R'st way to get everything in one go?

Glad you asked!!! You are tuning into R with the right approach...

```
summary(data_vector)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   2.500   4.500   4.333   5.750   7.000
```

Can we make it even simpler? Think about it!!!!

5. Exploring Data

In this part of the lab session the key objectives are to use R through RStudio to read in a small set of data and to explore the data. In R, it is possible to read data from different file formats:

- manual input
- plain text
- delimited text - such as .csv/.tsv files
- spreadsheets
- SAS formats

Reding in a csv file

So far we have made use of data that is already in R or input our own. In practice often data will be provided in .csv fomat and R makes it easy to read it in.

- Make sure you download the worms.csv data from Brightspace.
- Save a copy of that data in the same directory where you have saved this .Rmd file you are currently going through
- Run the code below to read data in:

```
worms <- read.csv("worms.csv")
```

If you want to check what the data looks like:

```
head(worms,n=10)
```

```
##         Field.Name Area Slope Vegetation Soil.pH  Damp Worm.density
## 1      Nashs.Field  3.6    11  Grassland     4.1 FALSE            4
## 2   Silwood.Bottom  5.1     2     Arable     5.2 FALSE            7
## 3    Nursery.Field  2.8     3  Grassland     4.3 FALSE            2
## 4      Rush.Meadow  2.4     5     Meadow     4.9  TRUE            5
## 5  Gunness.Thicket  3.8     0      Scrub     4.2 FALSE            6
## 6         Oak.Mead  3.1     2  Grassland     3.9 FALSE            2
## 7     Church.Field  3.5     3  Grassland     4.2 FALSE            3
## 8          Ashurst  2.1     0     Arable     4.8 FALSE            4
## 9      The.Orchard  1.9     0    Orchard     5.7 FALSE            9
## 10   Rookery.Slope  1.5     4  Grassland     5.0  TRUE            7
```

Now lets see how to do some descriptive data analysis.

Numerical summaries of categorical variables

If you recall from the QDA lecture the numerical summary for a categorical variable is a frequency table. These are very easy to produce with R functions.

```
table(worms$Damp, worms$Vegetation)
```

```
##
##         Arable Grassland Meadow Orchard Scrub
##   FALSE      3         8      0       1     2
##   TRUE       0         1      3       0     2
```

Play around with table - you can also do a table summary for one variable at a time.

There are many additional packages that offer more sophisticated options. We will explore relevant ones later in the module. It is also possible to do frequency tables for more than one column at the same time (in effect creating a contingency table):

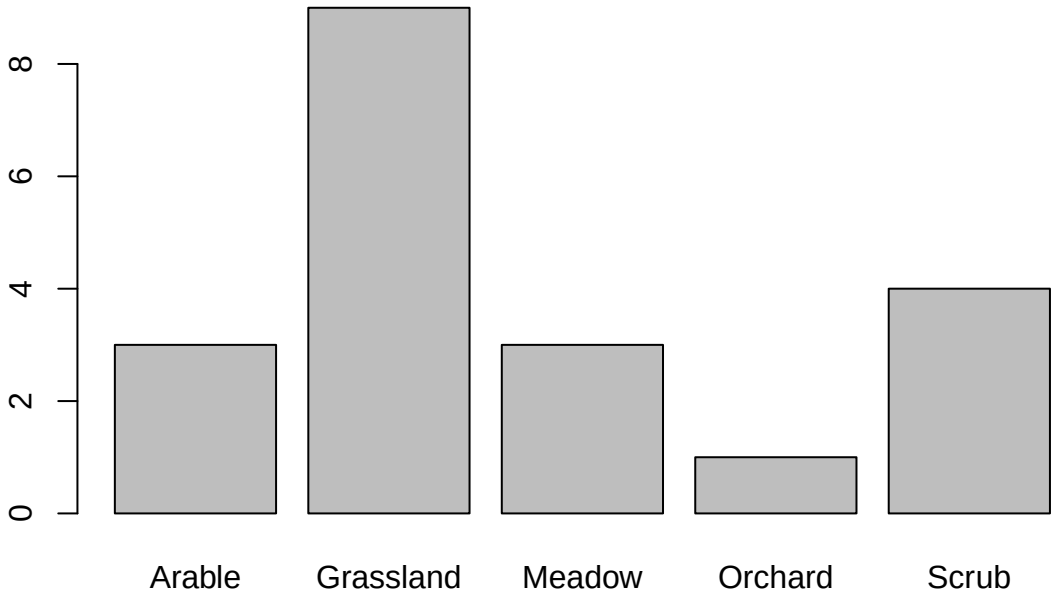
Graphical summaries for categorical variables

Now let's look at ways of graphically summarising a column of categorical data from our data frame. Let's start with the vegetation, using the basic R library

```
table(worms$Vegetation)
```

```
##
##    Arable Grassland    Meadow   Orchard     Scrub
##         3         9         3         1         4
```

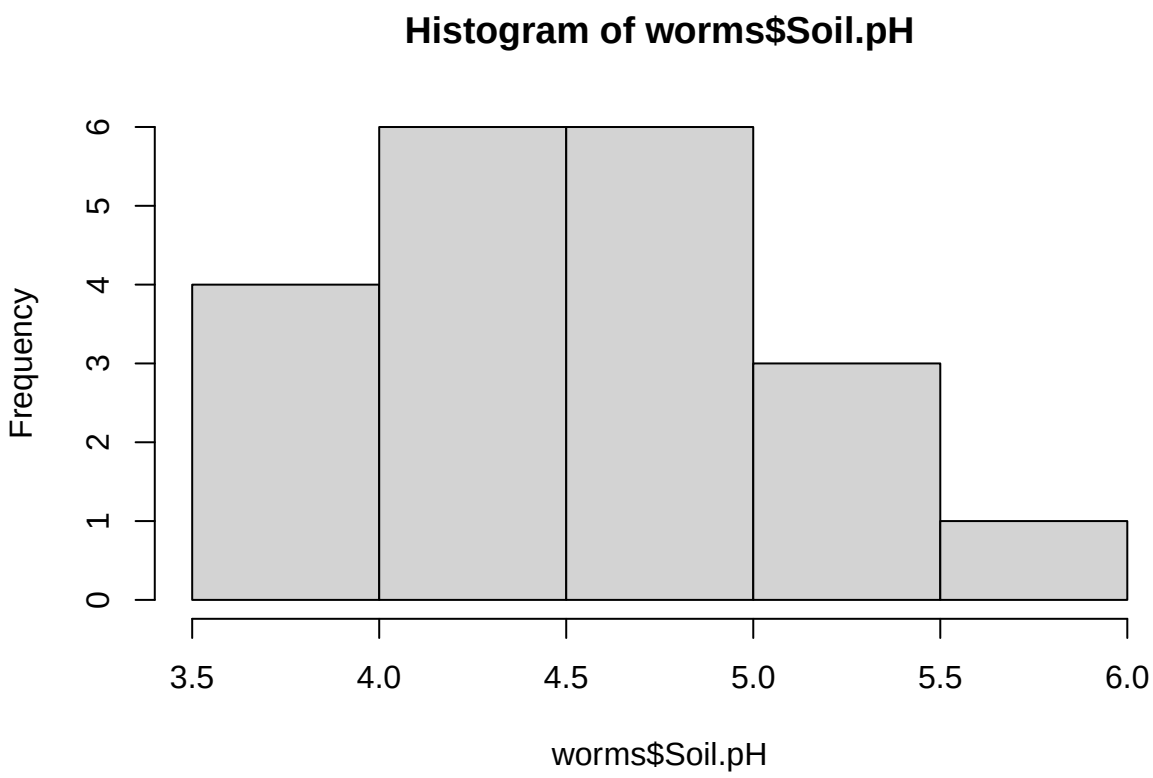
```
barplot(table(worms$Vegetation))
```



(You can use the Help pane to search how you can add a title to the plot . . .)

It is also possible to use the same syntax for the two way table (contingency table)

```
hist(worms$Soil.pH)
```



Using ggplot2 (Optional)

One library that is very handy for graphs is the {ggplot2}.

Before using a library you need to install it (first time) and load it. As the ggplot2 library is installed then you only need to load it:

```
library(ggplot2)
```

You can also use the menus to do this: * go to Tools -> install packages * or in the pane that shows Files, Plots etc use the Packages tab

There are a huge number of options in {ggplot2}. In most cases you start with `ggplot()`, supply a dataset and aesthetic mapping (with `aes()`). You then add on layers (like `geom_point()` or `geom_histogram()`), scales (like `scale_colour_brewer()`), faceting specifications (like `facet_wrap()`) and coordinate systems (like `coord_flip()`).

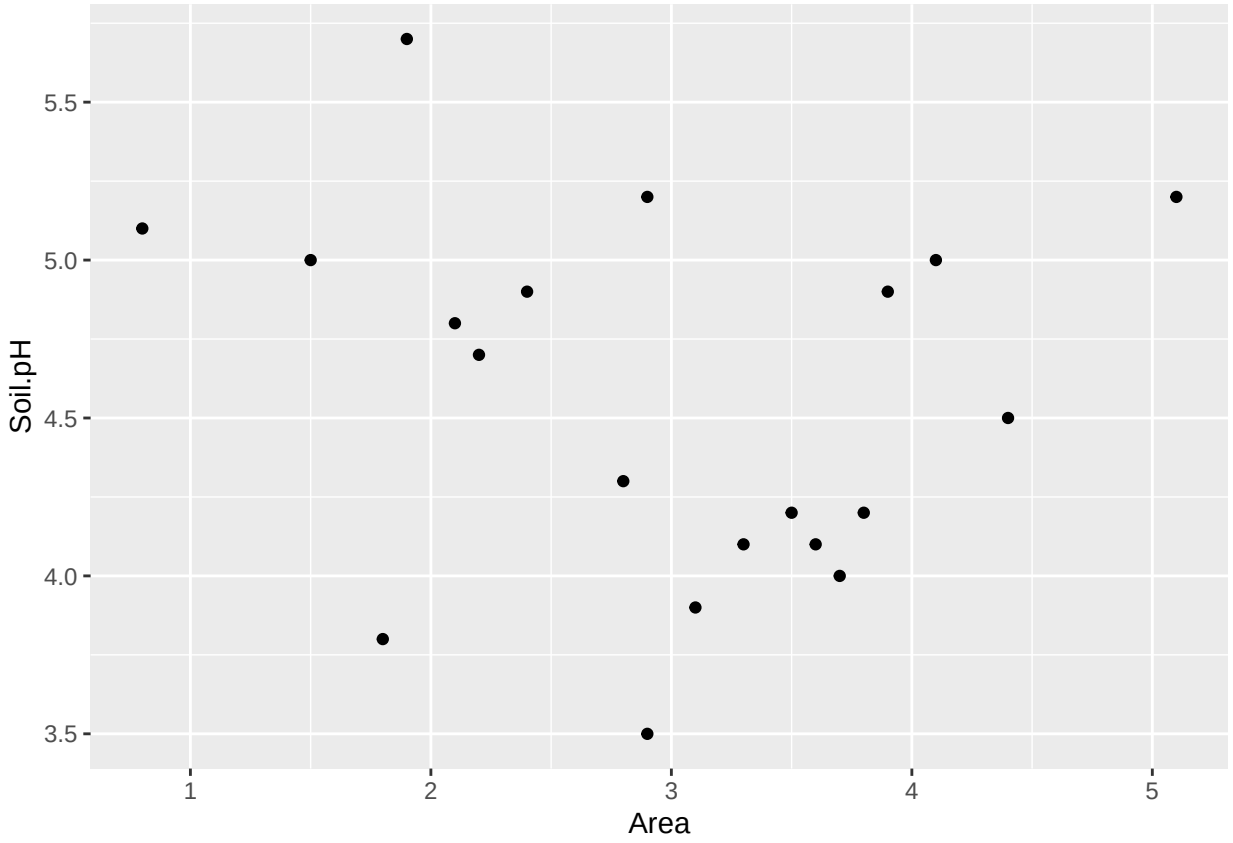
A good overview is available in the cheatsheet, you can access from the link

Lets look at replicating the plots above and more using ggplot.

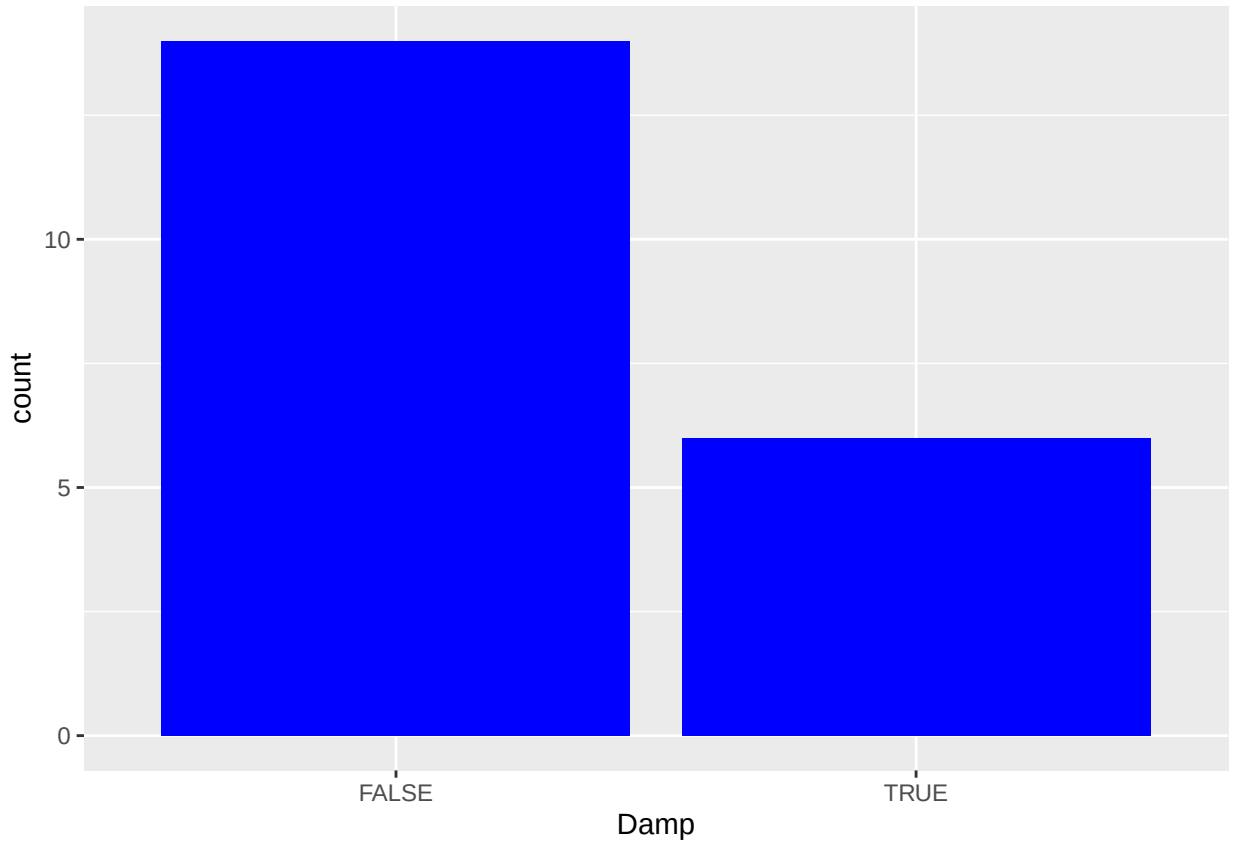
```
head(worms)
```

```
##        Field.Name Area Slope Vegetation Soil.pH  Damp Worm.density
## 1     Nashs.Field  3.6    11  Grassland     4.1 FALSE            4
## 2  Silwood.Bottom  5.1     2     Arable     5.2 FALSE            7
## 3   Nursery.Field  2.8     3  Grassland     4.3 FALSE            2
## 4     Rush.Meadow  2.4     5     Meadow     4.9  TRUE            5
## 5 Gunness.Thicket  3.8     0      Scrub     4.2 FALSE            6
## 6        Oak.Mead  3.1     2  Grassland     3.9 FALSE            2
```

```
ggplot(data=worms)+geom_point(aes(x=Area,y=Soil.pH))
```

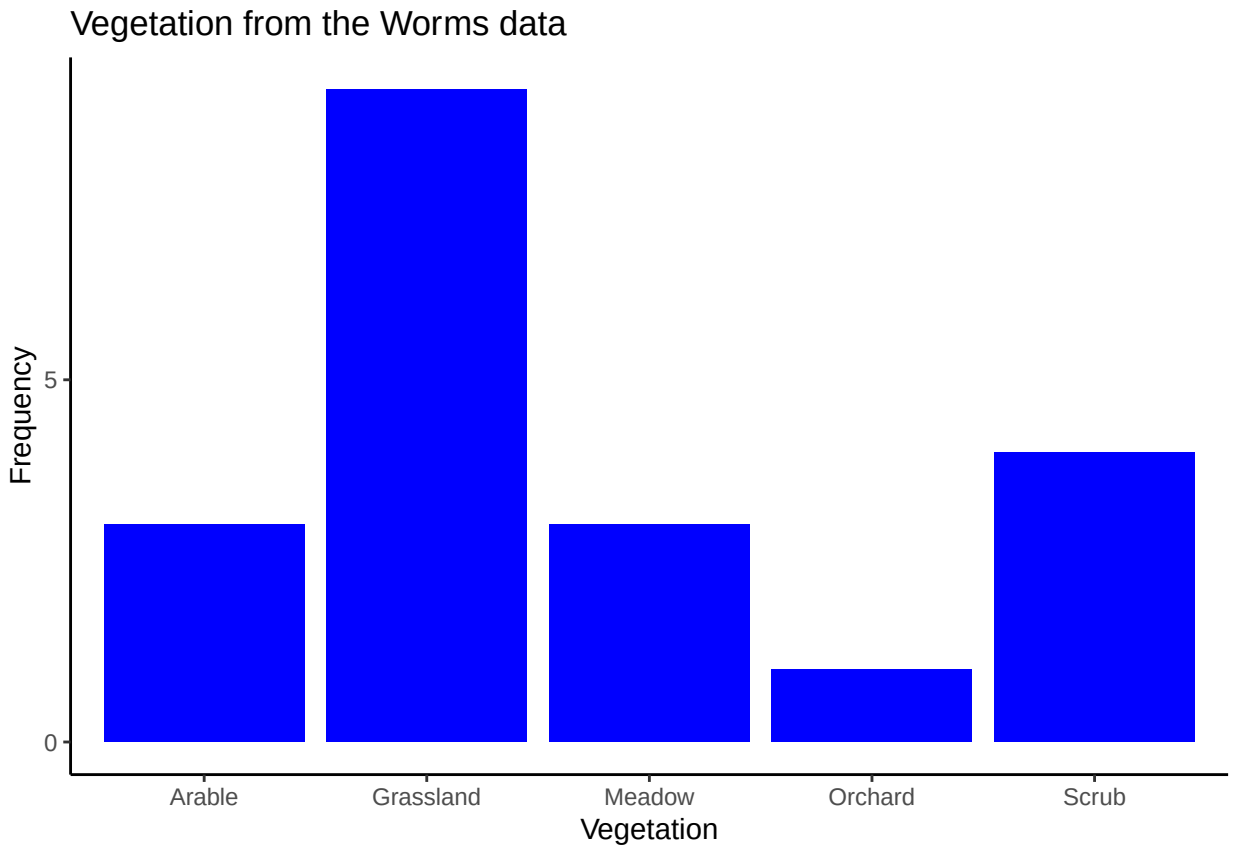


```
ggplot(worms)+ geom_bar(aes(x=Damp), fill="blue")
```



There is a lot you can specify in ggplot - see below for another example which now has well labelled titles and axis and a different theme.

```
ggplot(worms)+ geom_bar(aes(x=Vegetation), fill="blue") +
  labs(title = "Vegetation from the Worms data") +
  scale_y_continuous(name="Frequency", breaks=c(0,5,10)) +
  theme_classic()
```

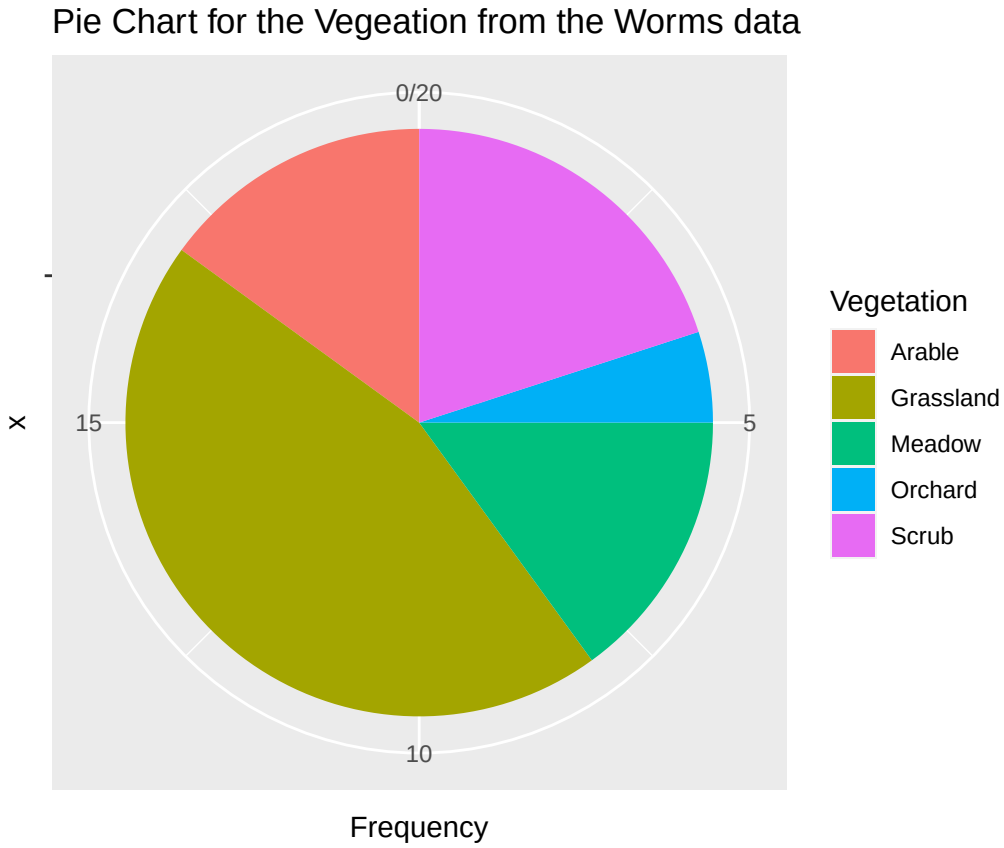


Pie chart

The code chunk below creates a pie chart. In order to do such a plot, the frequency table needs to be computed first, then transformed into a data frame, before finally being plotted using a ggplot option (coord_polar).

Note that giving clear column names makes it easier and quicker to get a better labelled graph.

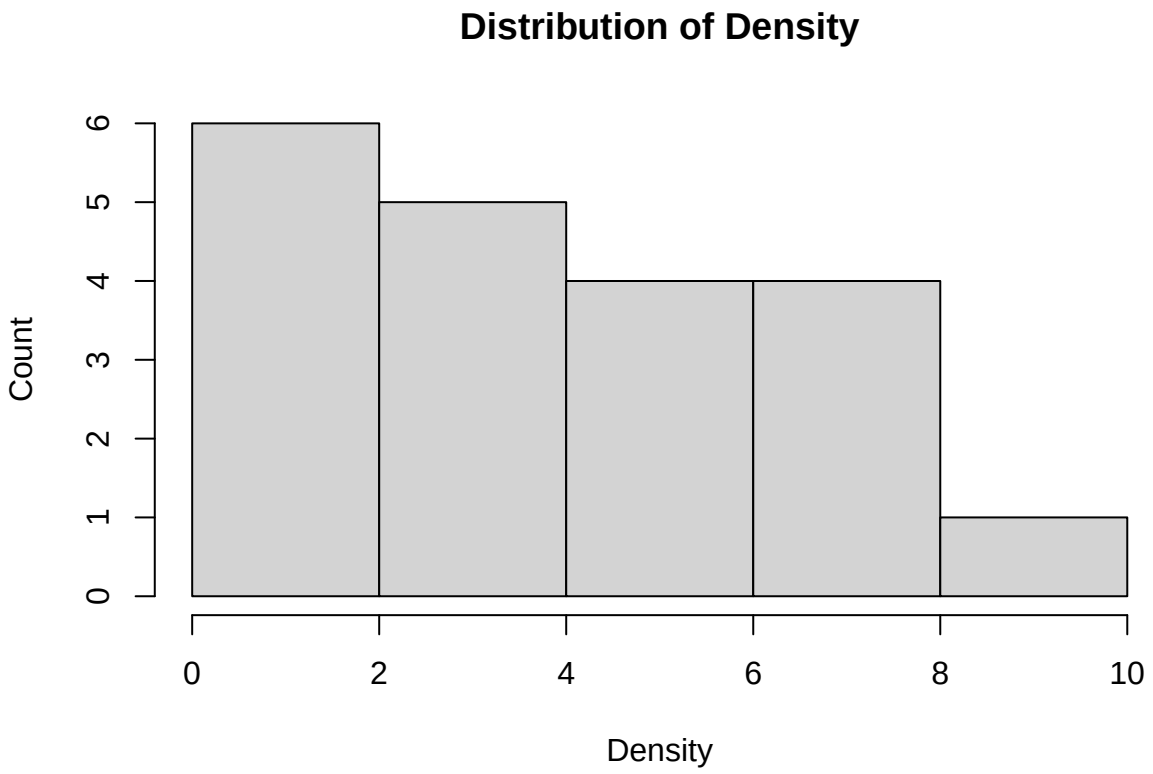
```
w1<-table(worms$Vegetation)
w2<-as.data.frame(w1)
#w2
colnames(w2)<-c("Vegetation","Frequency")

ggplot(w2, aes(x="", y=Frequency, fill=Vegetation))+geom_bar(width=1, stat = "identity") +coord_polar("
```



Graphical summaries for numerical variables

Histogram A histogram to see the distribution of the values for one of the continuous columns can be created in many ways. Using the base R code:

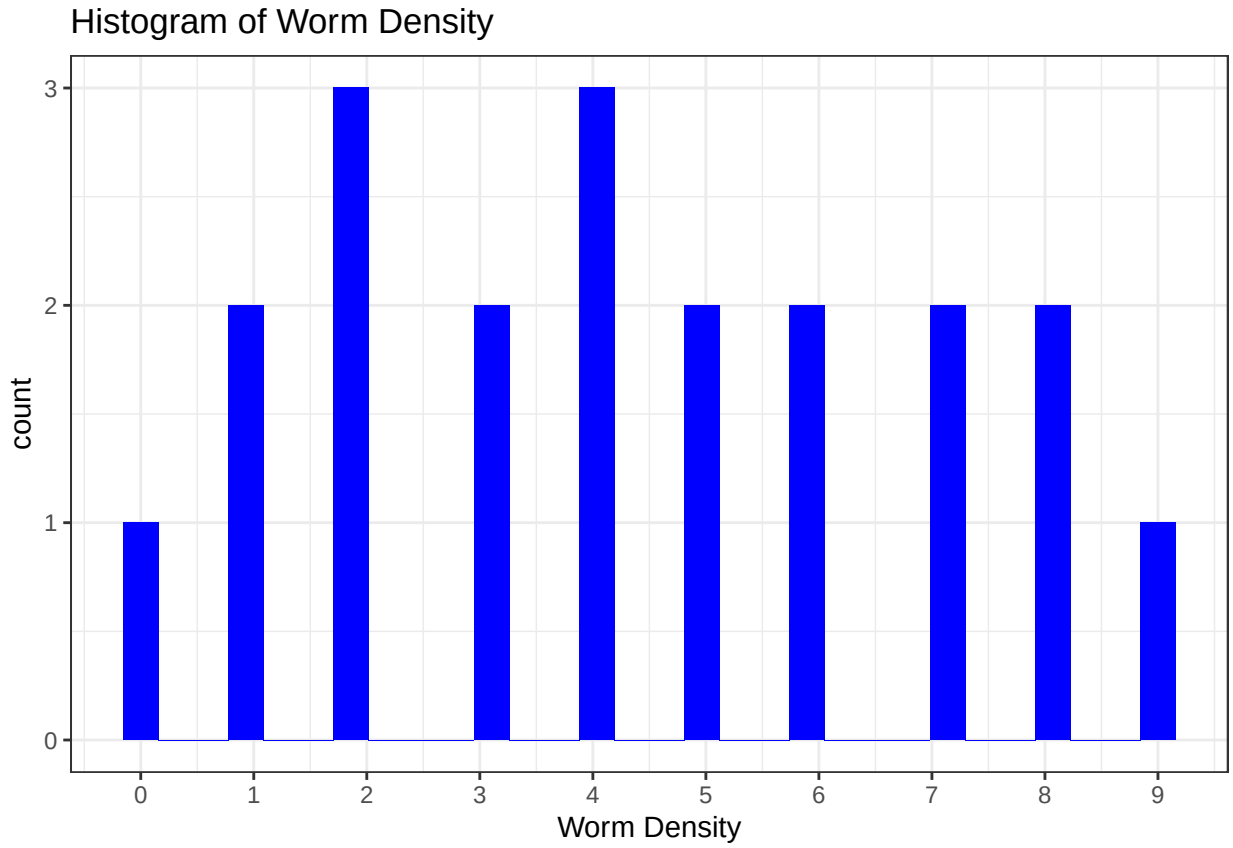
```
hist(worms$Worm.density,xlab="Density",ylab="Count",main="Distribution of Density")
```



OR ggplot can also be used:

```
ggplot(worms, aes(x=Worm.density)) +
  geom_histogram(fill="blue") +
  labs(title="Histogram of Worm Density") +
  scale_x_continuous(name="Worm Density", breaks = c(0,1,2,3,4,5,6,7,8,9)) +
  theme_bw()
```

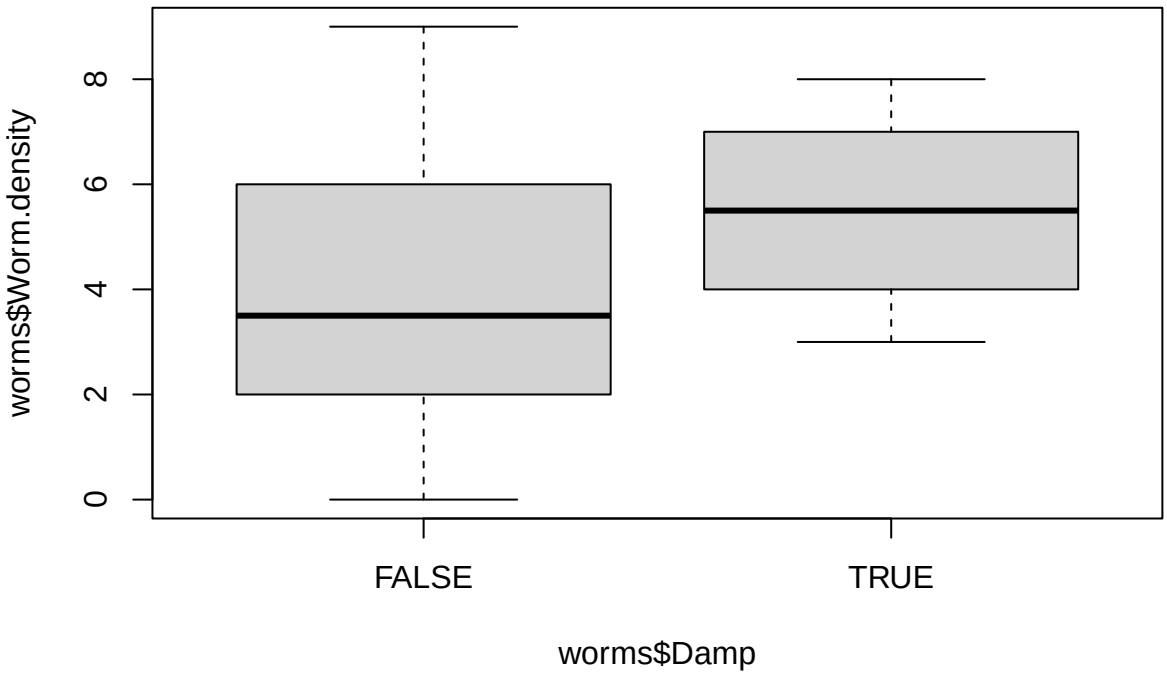
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Box plot Another interesting perspective on a continuous variable, when it is part of a data set with categorical variable is to look at them both at the same time. A **boxplot** is a useful tool to achieve this.

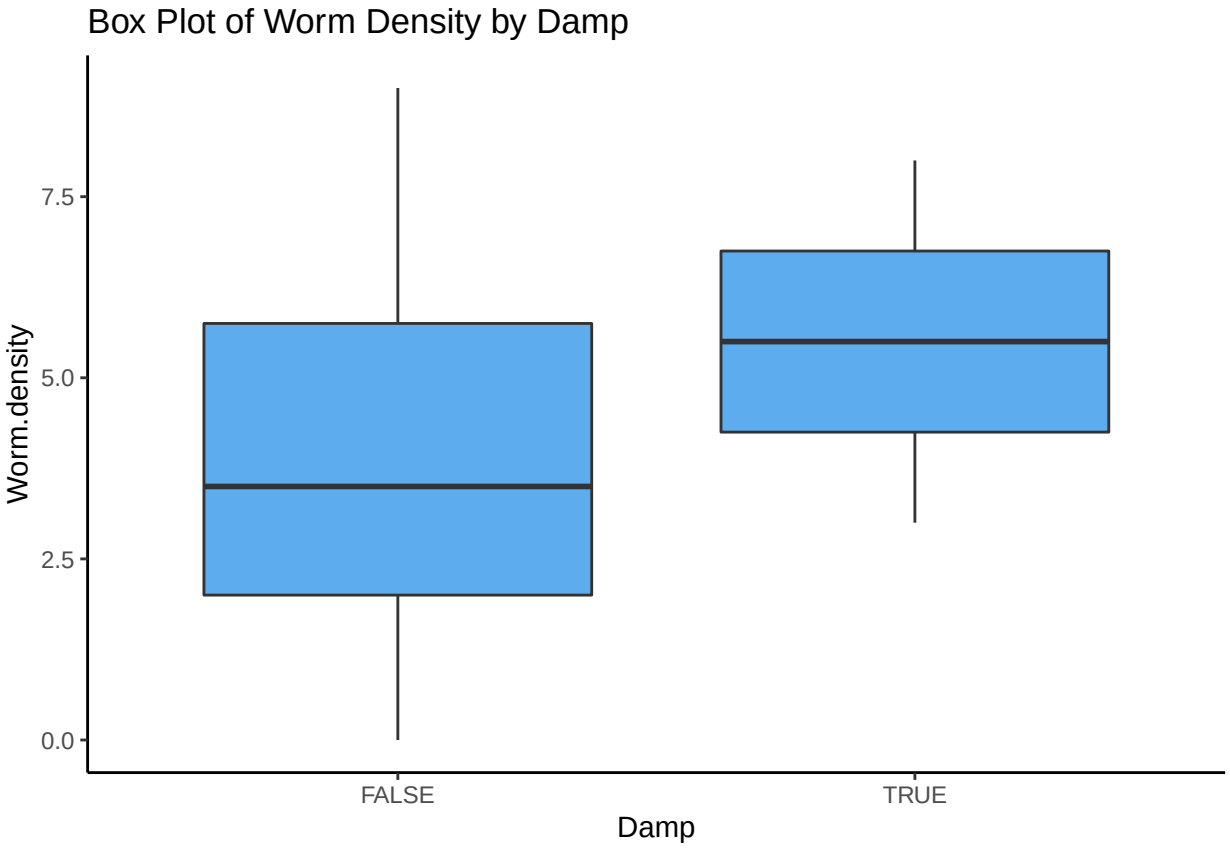
Using R base:

```
boxplot(worms$Worm.density~worms$Damp)
```



or using ggplot

```
#box plot
ggplot(worms, aes(x=Worm.density, y=Damp)) +
  geom_boxplot(fill="steelblue2") +
  theme_classic() +
  labs(title="Box Plot of Worm Density by Damp") +
  coord_flip()
```



6. The diamonds dataset casestudy

Now over to you!

Have a go at performing a descriptive analysis of a data set. In order to do this we are going to make use of the *diamonds* data set that is available in R.

This data set is available in the *{ggplot}* library that was already loaded earlier in this session.

(a) Run the code below to see the description of the diamond data and its contents.

```
help("diamonds")
```

(b) Check that the data set *diamonds* does indeed contain the columns as in the help

(c) The help description of the data set has the ranges for the variables in this data set. Check that this is the case in the data. Are there any variables out of range?

The ranges for the numerical variables can be obtained using the summary function. There are other ways.

(d) How many levels (or different categories) are in the column labelled *cut*?

(e) Generate a numerical summary for each variable in the data set - take care to use the appropriate technique for each different variable type.

(f) For each variable generate a plot, use different options and make sure the plots are labelled, and suitable to the type of variable.

(g) Explore the graphing of one categorical against one of the continuous variables in this data.

Appendix - extra notes on YAML

YAML rhymes with camel! It is an acronym for "**Y**AML **A**in't **M**arkup **L**anguage".

An R Markdown file has three components:

1. Header or front matter written in YAML
2. Markdown: this is text to describe your R analysis
3. Code chunks: Chunks of embedded R code that can be executed and rendered

An RMarkdown file *must* start with a header written using YAML syntax. There are four default elements in the RStudio YAML header:

```
title: The title of your document. NB this is not the same as the file name.
author: Who wrote the document.
date: By default this is the date that the file is created.
output: What format will the output be in.  We will generally use html.  Other options are pdf, various
```

The YAML is delineated by `---` and `---`.

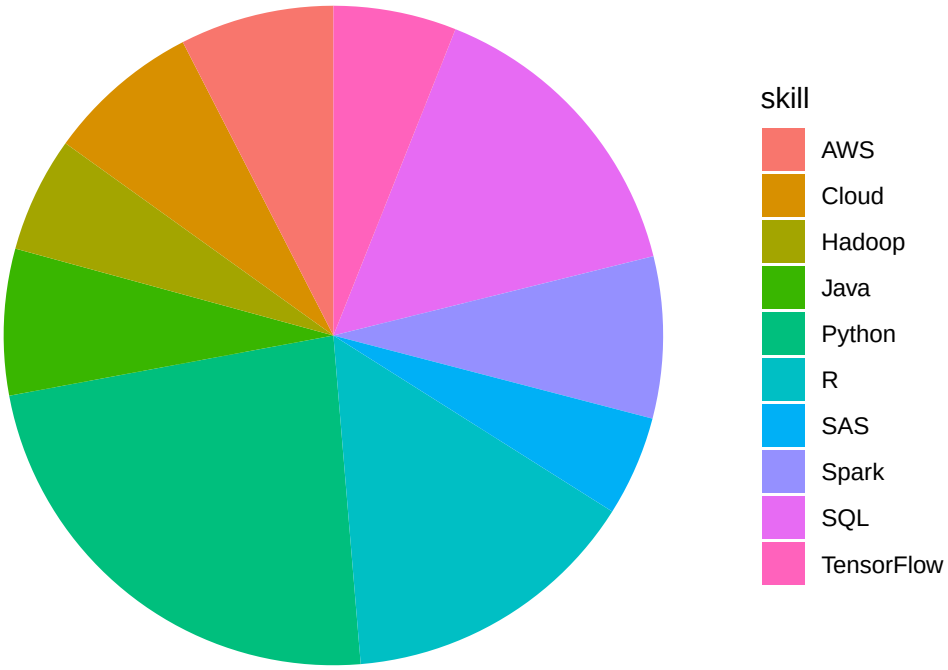
However, if you're unsure, then your best strategy is to copy and adapt, e.g., change the date, title or author.

Appendix 2: Pie charts with ggplot2

Remember that many data scientists have strong reservations about pie charts. In this case a bar plot is probably more effective.

```
# Reproduce the data from the January 2020 required tools for Data Scientists
# using ggplot2

# load the ggplot2 library
library(ggplot2)

percent <- c(62,40,39,21,20,20,19,16,15,13)
skill <- c("Python","SQL","R","Spark","Cloud","AWS","Java","TensorFlow","Hadoop","SAS")
# Merge into a single dataframe which makes ggplot easier
dsTools <- data.frame(skill,percent)

# Generate the pie chart
ggplot(dsTools, aes(x="", y=percent, fill=skill)) +
  geom_bar(stat="identity", width=1) +
  coord_polar("y", start=0) +
  theme_void() +                 # remove background, grid, numeric labels
  labs(title = "In demand data scientist tools (2020)") +
  theme(plot.title = element_text(size = 10))
```

In demand data scientist tools (2020)



Answer to Exercise 4.1

```
# Define a new function called mode
mode <- function(v) {
# calculate mode
  return(names(sort(-table(v)))[1])
}

# Apply the function to our vector
mode(data_vector)
```

```
## [1] "2"
```