COMP 120: Lab 4 Worksheet

In this lab, we will explore some of the basic routines provided by R to visualise data. Following that, we will explore the ggplot2 library, which can greatly simplify many plotting and visualisation tasks.

Remember that anything in a code block should be copied and pasted into the RStudio console so that you can see the desired result for yourself.

# Remember to set your working directory and library path (if working on the Student Desktop) at the start of your session. See the Lab 01 for advice on how to do this.

# Plotting in R

The basics of plotting were introduced in the lectures this week. Plotting in R is fairly straightforward once you get a handle on some basic commands:

* plot() creates a new plot
* points() adds additional x-y points to an existing plot (which already may have x-y points)
* lines() adds x-y lines to an existing plot
* text() adds character strings at provided x-y coordinates to an existing plot
* legend() adds a legend to an existing plot

Note that only plot() and legend() functions were covered in the lectures this week and you’ll use the lines() function in this lab. The other functions are for noting only.

In addition to 2D X-Y style plots (scatter plots) which can be generated using the plot() function, R has a number of functions that enable production of other visualisations common in statistics (histograms, bar plots and box plots). These were covered in the lectures.

All the plotting functions in R have many useful arguments to control how they generate the resulting visualisation. The online help (accessed through ?<function name> at the console) provides excellent detail on what these arguments do.

For the examples in this lab, we will use the data that you worked on in lab 03 regarding student performance in 2017’s INFO 204 cohort. We’ve added a few more columns and the dataset can be downloaded from Blackboard. Once downloaded into your workspace, it can be loaded as follows:

library(tidyverse)

info.204 <- read\_tsv("INFO204-2017-results-mod.tsv", na="?")

info.204$Gender <- factor(info.204$Gender,

levels=c("F", "M", "U"),

labels=c("Female", "Male", "Unspecified"))

info.204$Grade <- ordered(info.204$Grade,

levels=c("E", "D",

"C-", "C", "C+",

"B-", "B", "B+",

"A-", "A", "A+"))

Line 2 above loads the data into info.204. Line 3 converts Gender into a factor, which helps with easier visualisation and analysis. Also, the grade is also a factor that is ordered into levels using the ordered function in line 4. Doing so enables comparison of grades (i.e., grade D is higher than E).

Answer the following questions:

1. How many rows and columns does the tibble info.204 contain? Hint: you can also use nrow() and ncol() functions. These functions expect the data frame name as an argument. What do functions rownames() and colnames() produce? These functions also expect the same arguments as the nrow() and ncol() functions.
2. By default, the first 10 rows of a tibble is printed. To print, say the first 20 rows you can use print(info.204, n = 20). Here the number of rows that should be printed is specified using the *n* argument.
3. Are all columns visible when you print info.204? If not, use print(info.204, width = Inf) command to see all the columns.
4. How many rows and columns will be printed by the following line of code? print(info.204, n = 20, width = Inf)
5. How many variables contain the factor data type? Is there a logical variable in the data?
6. Of the variables that have factor data type, is there an unordered factor?

## Scatter Plots

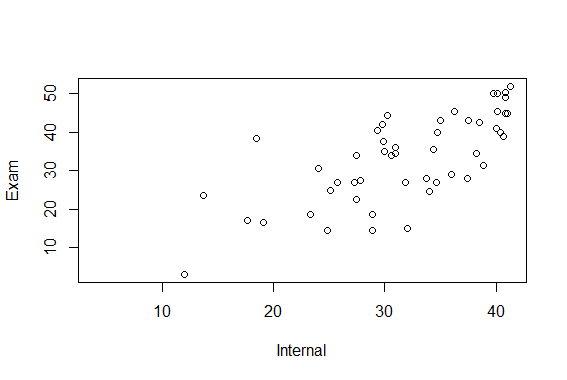
Scatter plots are generated through the plot() function. In this case, we supply two primary arguments, a vector for the x coordinates, and a corresponding vector of y coordinates. These are then combined in a pairwise fashion to produce a set of points that are subsequently plotted. Let’s start by plotting the INFO 204 results from before, comparing Internal with Exam performance. There are two ways of achieving this as given in the two lines below.

plot(info.204$Internal, info.204$Exam)

with(info.204, plot(Internal, Exam))

*Note: in the second line above, we used the* with() *function to make it easier to access the columns of the data frame. By using* with()*, we get direct access to the columns without having to prefix them with the data frame name and dollar operator.*

The resulting plot for line 2 should resemble the following. The plot for line 1 above should look similar but there is a difference. Can you spot the difference?



The default scatter plot provided by plot() is, at best, “okay”. However, we can do a lot to it to make it better. Here are a few ways:

* we could give the x and y axes proper and accurate ranges (0-45 for the x axis, 0-55 for the y axis)
* we could give the plot a title
* we could change the symbols and/or their colour

Let’s incorporate these changes:

with(info.204, plot(Internal, Exam,

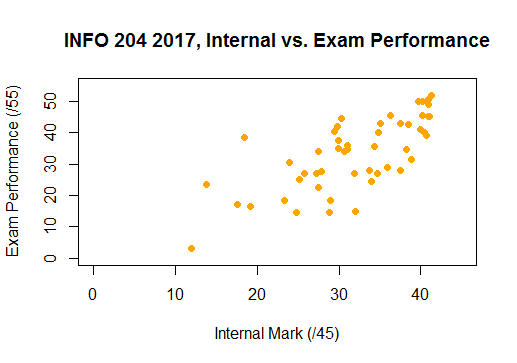
main="INFO 204 2017, Internal vs. Exam Performance",

xlim=c(0, 45), xlab="Internal Mark (/45)",

ylim=c(0, 55), ylab="Exam Performance (/55)",

pch=19, col="orange"))

These changes should result in the following plot. The best way to understand what the code above does is to include one argument at a time inside the plot function. Initially, make sure there are two arguments for the plot function (Internal and Exam). Then add the third argument. Execute the code to see how the plot looks. Then, add the fourth argument and investigate the output. Continue till you reach the last argument.



One last thing that we might want to do is colour the points in the scatter plot based upon some condition (e.g., did the student fail the tutorial part of INFO 204?). By using the ifelse() function in conjunction with the PassedTutorials column in the tibble, we can colour the points on a per-student basis:

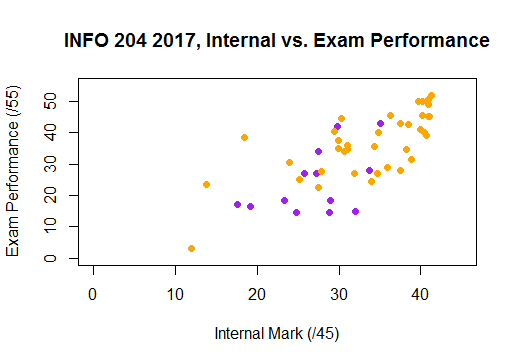
with(info.204, plot(Internal, Exam,

main="INFO 204 2017, Internal vs. Exam Performance",

xlim=c(0, 45), xlab="Internal Mark (/45)",

ylim=c(0, 55), ylab="Exam Performance (/55)",

pch=19, col=ifelse(PassedTutorials, "orange", "purple")))



Finally, we can make this plot easier to understand by placing a legend on the plot. Read the comments placed next to each argument to understand the purpose of the argument.

legend("topleft", ## the legend position - the top-left corner,

c("Passed Tutorials", "Failed Tutorials"), ## the legend text

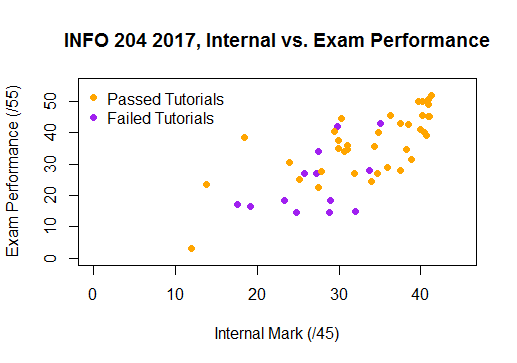
pch=19, ## point style to use for the legend (solid dots)

col=c("orange", "purple"), ## the colours to use for each dot

## in the legend

bty="n" ## should a box be placed around the legend (NO!)

)



## Layered Plots

Sometimes, you need to create plots with multiple features (e.g., a scatter plot, with a trend line), and this will require building the plot incrementally. To show an example of this, let’s generate some trivial data (*and you don’t need to understand the exact workings of the data generation part*):

plot.data <- data.frame(x=seq(-2\*pi, 2\*pi, length.out=201) +

rnorm(201, sd=0.05))

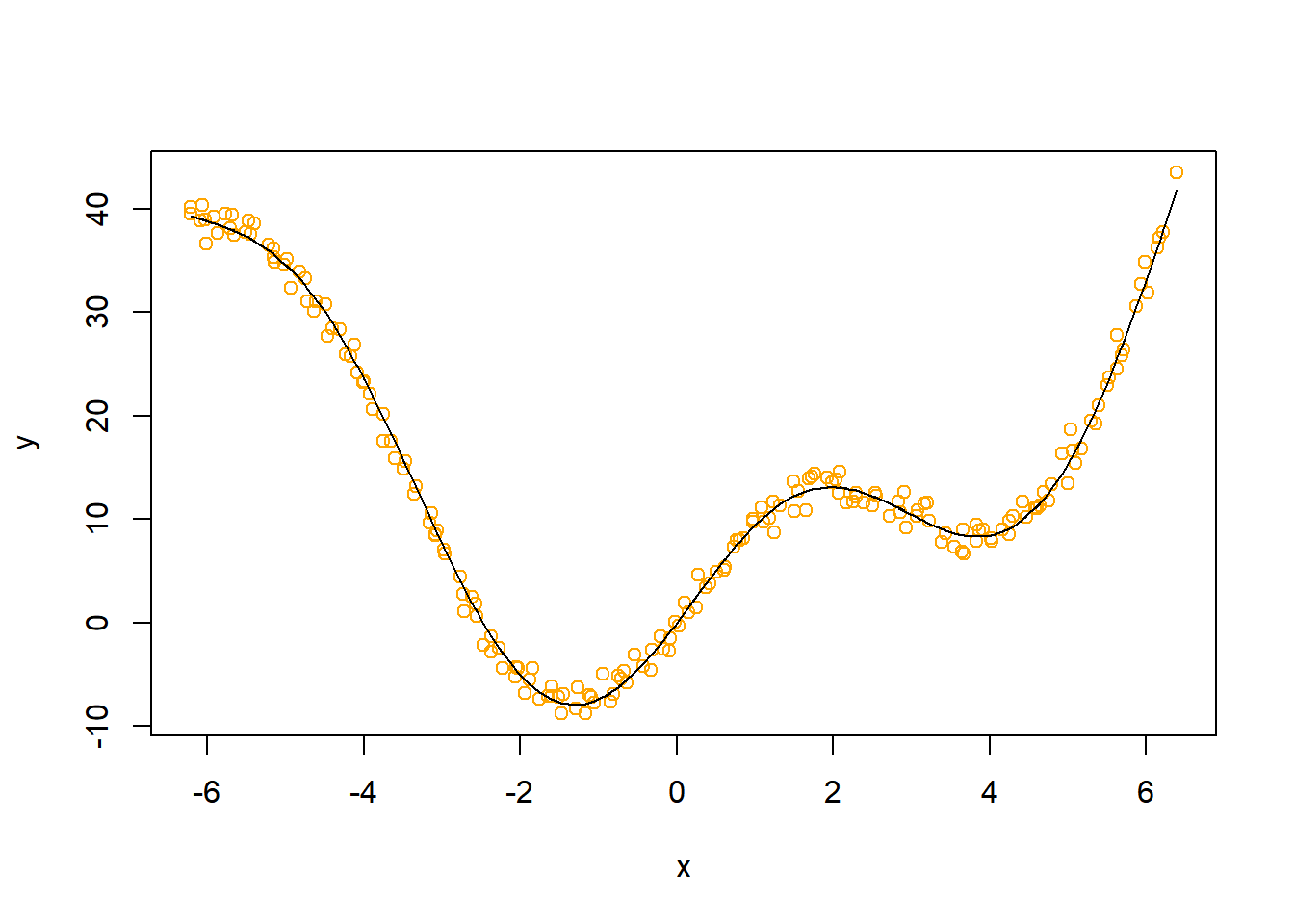
plot.data$f <- with(plot.data, x^2 + 10\*sin(x))

plot.data$y <- plot.data$f + rnorm(201)

Now, let’s build a simple scatter plot, and then add a second layer to this plot using the lines() function:

with(plot.data, plot(x, y, col="orange"))

with(plot.data, lines(x, f, col="black"))



Again, we could place a legend on this to make it easier to interpret:

legend("bottomright", ## legend position - the bottomright corner,

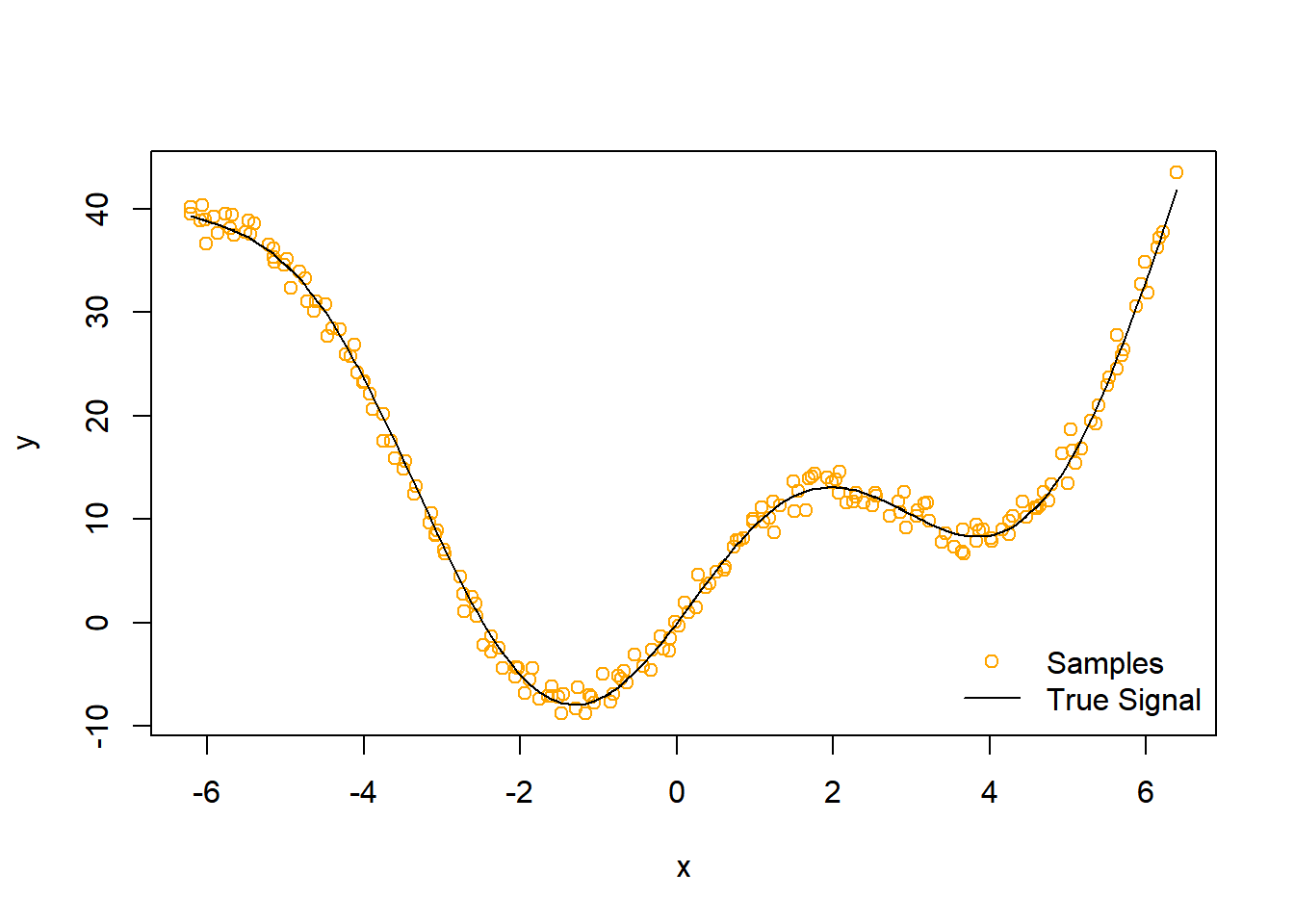
c("Samples", "True Signal"),## legend text

pch=c(1, NA),## symbols used: 1 (unfilled circle) and NA

lty=c(NA, 1),## line type used: NA and 1 (solid line)

col=c("orange", "black"), ## Colours used

bty="n") ## No box around the legend



## Histograms and Bar Plots

Histograms are useful for capturing the distribution of observations from a continuous variable. For example, we could generate a histogram of the final marks of students in INFO 204 using the following:

otago.blue <- "#00508F" ## the official colour code for the blue in

## the University of Otago crest!

## Hexadecimal RGB colour value(0, 80, 143)

breaks <- seq(0, 100, 5) ## the histogram broken down into bins

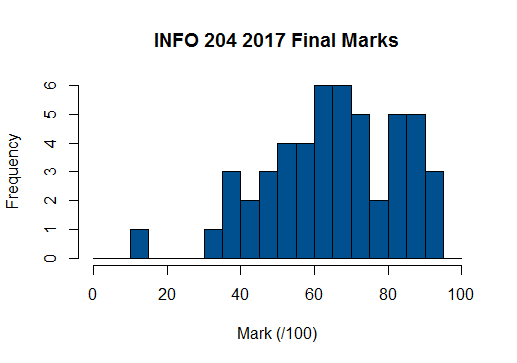
## of five marks from 0 to 100

with(info.204, hist(Internal+Exam, breaks=breaks,

col=otago.blue,

xlab="Mark (/100)",

main="INFO 204 2017 Final Marks"))



Likewise, bar plots are useful for depicting summaries for categorical data. For example, say we wanted to tally the number of students for each grade level in the INFO 204 results. First, we’d need the tally (we can use the *table()* function to do this for us):

grade.tally <- table(info.204$Grade)

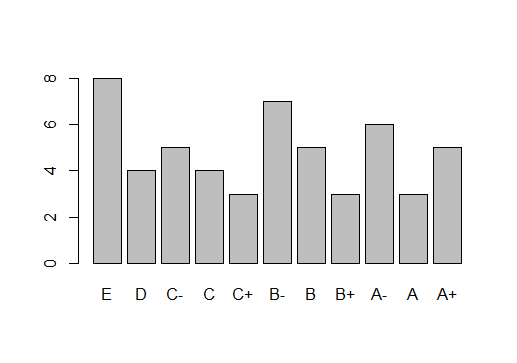
grade.tally

## E D C- C C+ B- B B+ A- A A+

## 8 4 5 4 3 7 5 3 6 3 5

And now we can generate the bar plot using the code below:

barplot(grade.tally)



Modify the code that produces the graph above to include appropriate x- and y-axis labels and a title.

# Plotting with ggplot2

The basic plotting routines in R are very powerful and can create very professional looking visualisations of data. However, for anything beyond relatively basic plotting, they can require large amounts of code to be written to produce the required effect. Plotting can be made easier if you use the ggplot2 library that is part of the tidyverse.

Plotting in ggplot2 was introduced in the last lecture (lecture 8). Overall, plotting in ggplot2 is governed by the ggplot() function and follows a general structure (don’t run the code below – this just a template!):

ggplot(data = **<DATA>**) +

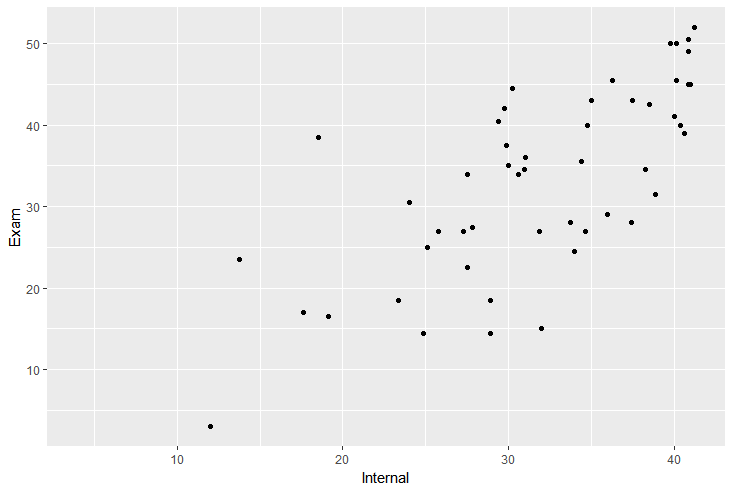
**<GEOM\_FUNCTION>**(mapping = aes(**<MAPPINGS>**))

Essentially, to plot using ggplot(), you identify the data frame that you want to plot, and then identify the layer(s) that you desire (e.g., points, lines, smoothed trend line, histograms or bar plots). Each layer will draw upon a set of specified aesthetic mappings (i.e., which columns to use for various aspects like the x and y axes, colours etc.) to produce the plot. For example, consider the very first plot that we produced in this lab (Internal vs. Exam INFO 204 performance). The equivalent ggplot() call would look like:

ggplot(info.204) +

geom\_point(aes(x=Internal, y=Exam))

which should produce the plot shown on the next page.



Aesthetics are mapped through the aes() function, where you supply the required column name for the corresponding mapping. For example, to change the colour of the points according to whether or not students passed tutorials, you would use the following:

ggplot(info.204, aes(x=Internal, y=Exam)) +

geom\_point(aes(colour=PassedTutorials))

Notice that you don’t need to use the ifelse() function like you did before – ggplot will handle this for you.

To change various aspects of the plot (e.g., axis title, plot title), we “add” the required information. For example, to recreate the “final” version of the Internal vs. Exam plot using ggplot, we could use the following

ggplot(info.204, aes(x=Internal, y=Exam)) +

geom\_point(aes(colour=PassedTutorials)) +

xlab("Internal Mark (/45)") + ylab("Exam Mark (/55)") +

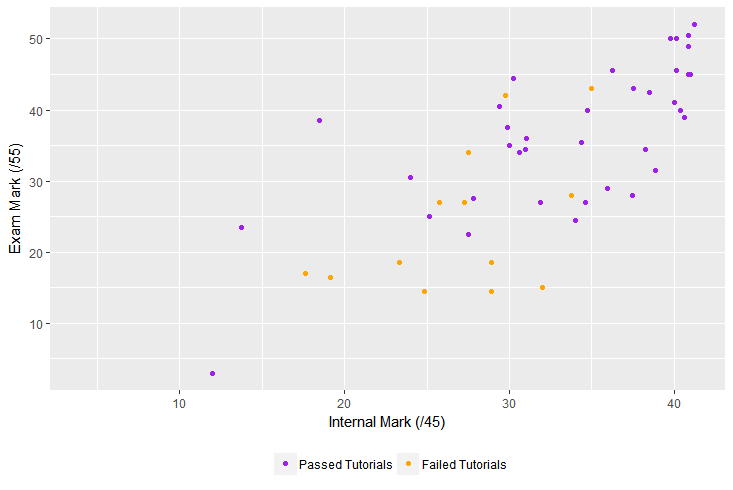
scale\_colour\_manual( ## sets colours manually

values=c("orange", "purple"),

labels=c("Passed Tutorials", "Failed Tutorials")) +

theme(legend.position="bottom") ## legend is set at the bottom

which produces the plot shown on the next page.



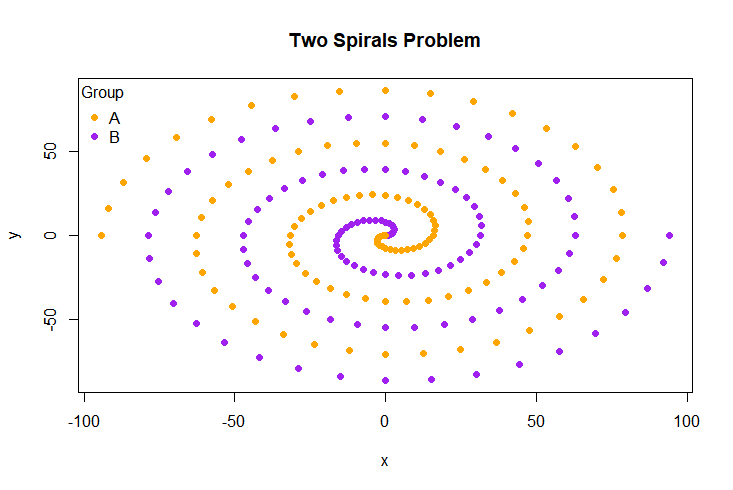
You now should have the basic skills to complete the mastery tasks for this week. It would be a good idea to try all the examples presented in the lectures this week. This would be a good preparation for the upcoming practical test. The source code for the examples covered in the lectures are available on Blackboard (look for the demo R files for each lecture). There is also a challenge task that was posed in Lecture 8. If you haven’t tried this already, give it a go – visualisations are a lot of fun!

# Mastery Tasks

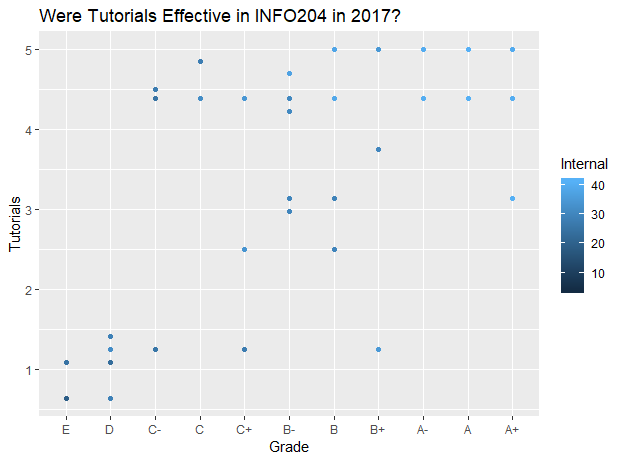
This lab has presented the basics of R plotting. We’ve really only scratched the surface, and learning the complete suite of plotting routines can take many months (and sometimes years!). However, the basics presented here should be sufficient for creating moderately complex but elegant plots. To test your mastery of the basics, complete the *four* plotting tasks given below. Try to match the plots as best possible. Put all the required code to generate the plots in the appropriate placeholders in the script provided to you (mastery-04.R). **Place a short comment before each block of code that you use to complete a given task to explain what the piece of code does.** Documenting your code in plain English is an important skill that you need to cultivate. Your comment will be beneficial to others to understand your code and sometimes it might help yourself if you revisit the code after some time. So, document your code well so that you can understand it when you come back to it. **Not writing good comments for each code block will attract a 5% penalty (0.15 out of 6). A couple of sample comments of code are given at the end of this document.**

When you have completed the tasks, **submit your work on Blackboard before 4pm on Thursday the 6th of August**.

1. Download the file spiral.csv from Blackboard into your workspace and then read it into a data frame in R. Then, use this data frame to create the following plot (using the built-in plotting routines – i.e., *base R plotting functions* covered in lecture 7). Note the legend has a title called Group. You can create a title using the *title* argument within the legend function. For an example see this link - <http://www.sthda.com/english/wiki/add-legends-to-plots-in-r-software-the-easiest-way>.

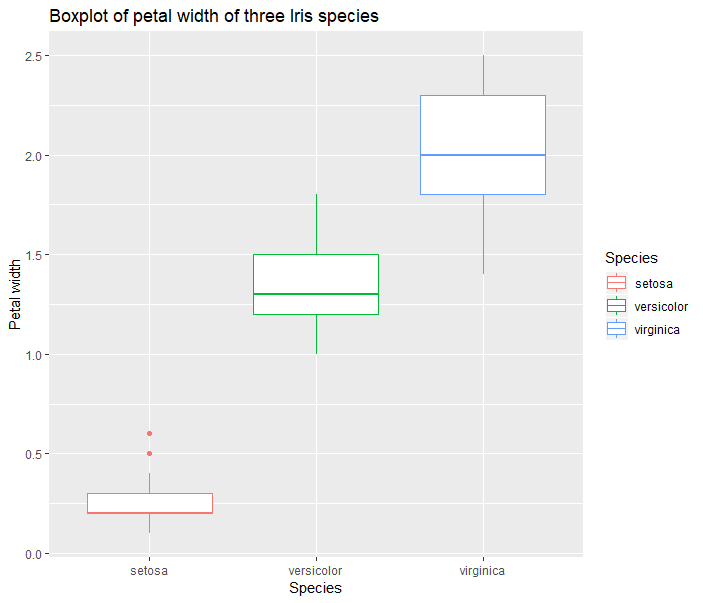


1. Using the info.204 data frame you used in this lab (which was created using the code given on page 1), create the following plot using *ggplot()* covered in lecture 8:



Note that the points in the graph are coloured according to the corresponding Internal marks’ value, and that a title can be “added” to a ggplot using the ggtitle() function. As part of your answer, provide a comment to address the question: “were tutorials effective in INFO 204 in 2017?” (Note: there is a placeholder for this comment in the mastery-04.R file). In other words, does the visualisation suggest that there was a strong correlation between final grade and tutorials mark? Note that we do not expect you to perform a correlation test (a statistical test) as we haven’t covered this yet. Your answer should be just based on your interpretation of this plot. If you haven’t heard of the correlation before, here is a gentle introduction - <https://www.youtube.com/watch?v=qC9_mohleao>. Your answer should state whether by looking at the graph you see a strong correlation, weak correlation or no correlation. If you think a correlation exists, you should also indicate whether the correlation might be positive or negative.

1. Using ggplot() write code to create a the box plot shown below. This plot should use the data from the iris dataset that is preloaded in R environment. This iris data set gives the measurements in centimeters of the variables sepal length, sepal width, petal length and petal width, respectively, for 50 flowers from each of 3 species of the Iris plant. These species are Iris setosa, versicolor, and virginica. Type iris to see the data in this dataset. Before writing the code, you will need to work out which variables (columns in the dataset) will be used for the plot.



1. Using ggplot() write code to create the plot shown below. This plot uses the same iris dataset that you have used for question 3. Hint: The plot uses multiple geoms.



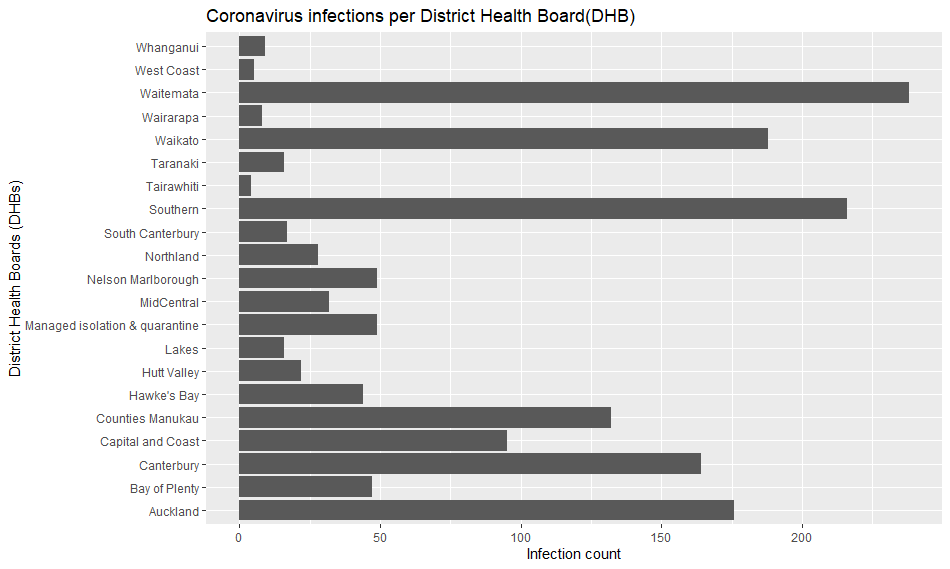
1. For the three plots you would create for this question (i.e., plots 5a-c) using ggplot(), you will use the coronavirus dataset provided in the file titled “coronavirus\_dataset.csv”. This data was sourced and processed from the ministry of health’s website on 24th July 2020 (see link here[[1]](#footnote-1)). The dataset contains details about all coronavirus cases (i.e., infections) in New Zealand. The dataset has 10 columns (given below at the end of this paragraph in italics). The columns of interest for us are *DHB*, *Age\_group, Overseas\_travel* and *Status*. *DHB* stands for District Health Board (i.e., the district a case belongs to). *Age\_group* stands for the age group of a case (e.g., 30 to 39). *Overseas\_travel* indicates whether the case was linked to overseas travel (which takes one of three values - Yes, No or missing value), and the *Status* column takes one of two values – whether a case is *Confirmed* or *Probable*. Note that the dataset isn’t perfect (i.e., it considers “Managed isolation & quarantine” as a DHB). But, that’s OK since this dataset is used for developing your R skills (i.e., we are not making predictions based on this data!). Also, we can always omit problematic rows, and we will get to this in the future labs.

Columns included in the coronavirus dataset:

1. *Date notified of potential case*
2. *Sex*
3. *Age\_group*
4. *DHB*
5. *Overseas\_travel*
6. *Last country before return*
7. *Flight number*
8. *Flight departure date*
9. *Arrival date*
10. *Status*

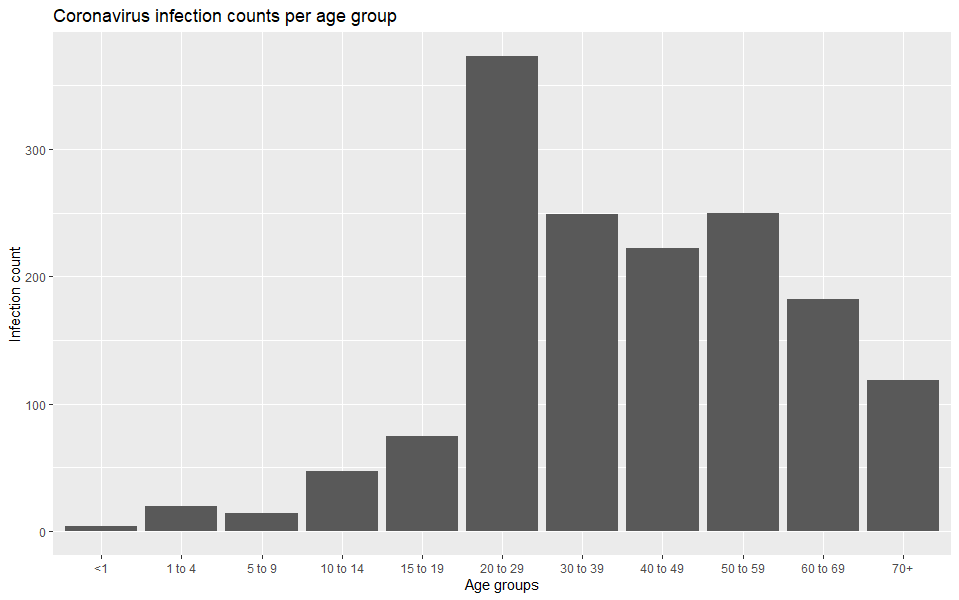
First, import this data into a tibble called coronavirus. Using this dataset and appropriate ggplot functions to produce the following plots (a-c):

1. A plot showing the total number of cases in each DHB.



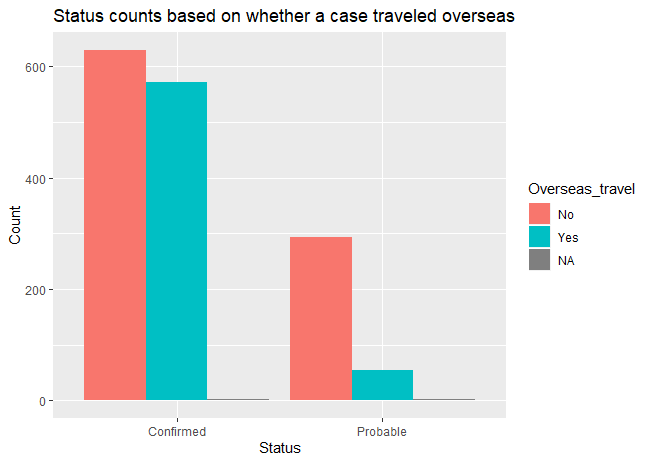
Hint: To swap X and Y axes of a graph, you can use a ggplot function called coord\_flip(). See an example here - <http://www.cookbook-r.com/Graphs/Axes_(ggplot2)/>. Try the first two snippets of code given on that page to see how X and Y axes are flipped!

1. A plot showing coronavirus infection counts per age group across New Zealand.



Hint: To obtain the age groups in the right order (<1, 1 to 4, 5 to 9 etc.), you will need to convert the age\_group variable to a factor before you write code to plot this graph. It is a good idea to store the factor as a different variable (e.g. age\_group\_factor) and use that variable for plotting. The code that may come handy to create the factor variable is: factor(coronavirus$Age\_group, levels = c("<1", "1 to 4", "5 to 9", "10 to 14", "15 to 19", "20 to 29" ,"30 to 39", "40 to 49", "50 to 59", "60 to 69", "70+")).

1. A plot showing the counts of confirmed and probable cases across New Zealand split for each sex (male and female).



Hint: You may realise the chart above is a grouped bar graph. You can use position = "dodge" *or* position = position\_dodge()inside the geom\_bar function to convert a stacked bar group into a grouped bar graph. An example can be found here - <https://www.learnbyexample.org/r-bar-plot-ggplot2/>. Search for *position = “dodge”* on that page.

## Examples of commenting code

Example 1:

An example of commenting a function is given below in blue.

# The function named count\_elements takes two arguments V and T. For each element in V the code checks whether the value is less than T. If the value is less than T, then the value of the counter variable is incremented by 1. At the end of the loop, the value in the counter variable is returned.

count\_elements <- function(V,T){

  counter <- 0

  for (i in V){

    if(i < T){

      counter <- counter + 1

      }

  }

  return(counter)

}

Example 2:

# In the code shown below, the data used for plotting the graph is the mpg dataset which is passed as the argument to the ggplot() function. Since the graph is a scatter plot, geom\_point() function has been used. The x and y variables inside the aesthetic function are mapped to drv and hwy variables respectively. The titles, x-axis and y-axis labels have been set using the appropriate functions.

ggplot(data = mpg) +

  geom\_point(mapping = aes(x = drv, y = hwy)) +

  ggtitle("Box plot of drive type vs. highway miles") +

  xlab("Drive") +

  ylab("Highway miles")

1. <https://www.health.govt.nz/our-work/diseases-and-conditions/covid-19-novel-coronavirus/covid-19-current-situation/covid-19-current-cases/covid-19-current-cases-details> [↑](#footnote-ref-1)