ReadMe Document Python Coursework

Aim of the script

The aim of the script is for the user to be able to plot a graph that displays two datasets on the same plot, adds a line of best fit and displays the equation of the line.
Of course, the graph is also fitted with proper axes attributes such as title, major/minor ticks, data markers and more.

Some tkinters have been added to make the script more interactive, this will allow the user to click some buttons which could help them save the plot as a jpg output file, exit the program or to know more about the developer of the program.

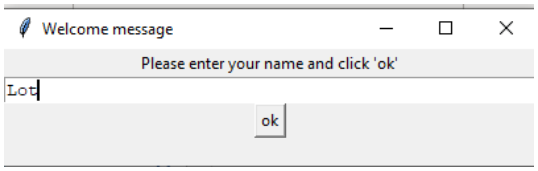
Importing modules

The modules that have been imported are tkinter, matplotlib, csv, numpy and pandas.
This will allow for interactive windows to be displayed, figures to be plotted, datasheets to be read in and lines of best fit to be calculated.

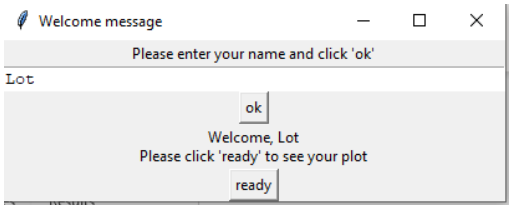
Welcome message display

```
### making a tkinter window with the name 'welcome message'
ws = Tk()
ws.title("Welcome message")

### places a text box in the welcome message window displaying the text below
kj_label1 = Label(ws, text = "Please enter your name and click 'ok'")
kj_label1.pack()

### this function is called when the 'ok' button is clicked, displaying a set of messages using the input given above and displaying a new clickable button
def clicked1():
    msg = "\n" "Please click 'ready' to see your plot"
    inp = inputtxt.get(1.0, "end-1c")
    lbl.config(text = "Welcome, "+inp + msg)
    printButton2 = tk.Button(text = "ready", command = clicked2)
    printButton2.pack()

### this function is called when the 'exit' button is clicked, will close the tkinter window
def clicked2():
    ws.destroy()

### dimensions and display of the input box
inputtxt = tk.Text(width = 50, height=1)
inputtxt.pack()

### displays the clickable 'ok' button which will trigger the clicked1 function if clicked
printButton1 = tk.Button( text = "ok", command = clicked1)
printButton1.pack()
lbl = tk.Label(text = "")
lbl.pack()

ws.mainloop()
```

The code above will display the first interactive window, called “Welcome Message”.
First, it will display the window and ask the user to enter their name and click “ok”.



Next, once the “ok” button is clicked, it will take the input from the user and engrain it into a new set of messages which will appear and the user is asked to click the “ready button”



Once the “ready” button is clicked, the “Welcome Message” window will close and a new window called “pK plots” will appear.
This will display the plotted graph and all its features including a menubar, line of best fit and axes aesthetics.

Reading in data files

```
### data from two different csv files is read
data = pd.read_csv("pK2.csv")
x = data["log(VB/VA)"]
y = data["pH"]

data1 = pd.read_csv("pK3.csv")
x1 = data1["log(VB/VA)"]
y1 = data1["pH"]
```

log(VB/VA)	pH
-0.954242509	5.28
-0.602059991	5.61
-0.367976785	5.85
-0.176091259	6.04
0	6.24
0.176091259	6.4
0.367976785	6.62
0.602059991	6.85
0.753327667	7
0.954242509	7.2

log(VB/VA)	pH
-0.954242509	8.55
-0.602059991	8.88
-0.367976785	9.08
-0.176091259	9.27
0	9.45
0.176091259	9.63
0.367976785	9.84
0.602059991	10.08
0.753327667	10.26
0.954242509	10.46

The code above will tell the program which csv files to use to plot the data. The user needs to make sure that the name before the .csv extension matches the name of the file exactly or an error will be displayed.
The data in the tables above (pK2 and pK3 from left to right) on the right have the headers” log(VB/VA)” for x and “pH” for y.

If the table from which the data is read has any headers, make sure that these are defined in x and y as displayed above in the code. The program will try to run the headers as data points and return an error if these headers are not defined.
For example, to load in datasets from csv-files called “linear1” and “linear2”, the “pK2.csv” line can be replaced with “linear1.csv” and the “pK3.csv” line can be replaced with “linear2.csv”.
Equally, if there is only one dataset to be plotted, then all the repeated code using the second dataset can be removed.
However, this needs to be done all the way throughout as there are multiple stages at which code refers back to data from the second dataset.

Alternatively, a user can input a datafile without any headers and replace the code with “x = data[]” and “y = data[]” instead. However, from an analytical point of view this is less preferred as all data should ideally be accompanied with headers even if it is only used as a small step before actual analysis.

The fact that data is read into the script in this way makes this script very user friendly as now any csv file can be read into the script so long as the notes above are followed.

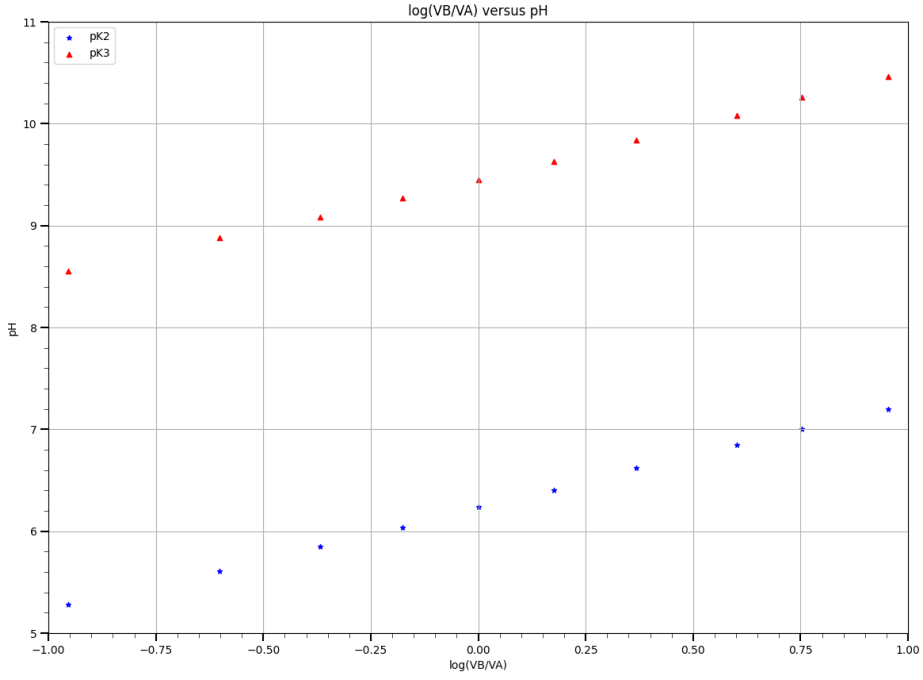
Output of the plot

```
### both datasets are plotted as scatterplots on the same graph
plt1.scatter(x,y,s = 20, c = 'b', marker='*', label = 'pK2')
plt1.scatter(x1,y1,s = 20, c = 'r', marker='^', label = 'pK3')

### any characteristics of the graph like axes titles, graph title and location of the legend
plt.xlabel("log(VB/VA)")
plt.ylabel("pH")
plt.title("log(VB/VA) versus pH")
plt.legend(loc='upper left')

### any axes characteristics such as x and y axes range, major and minor axes and grid-view
plt.xlim([-1,1])
plt.ylim([5,11])
plt.minorticks_on()
plt.tick_params(which="major", width=1.5)
plt.tick_params(which="major", length=7)
plt.tick_params(which="minor", width=0.5, length=4, color="black")
plt.grid(True)
```

The code above takes the data from datasets one and two and will plot the datapoints using markers and assign a label to them.
It will then take this plot and add axes titles (“plt.xlabel” and “plt.ylabel”), a graph title (“plt.title”) and a legend using the markers indicated in (“plt1.scatter”)
Next, some limits for the axes and major and minor ticks are added.
For readability of this graph, it was chosen to manually define the x- and y-axis limit as I think this looks better.
However, for functionality reasons, the code “plt.xlim([-1,1])” and “plt.ylim([5,11])” can be replaced with “plt.xlim()” and “plt.ylim()”. This will let the program automatically define its own x- and y-axis limits depending on the data.

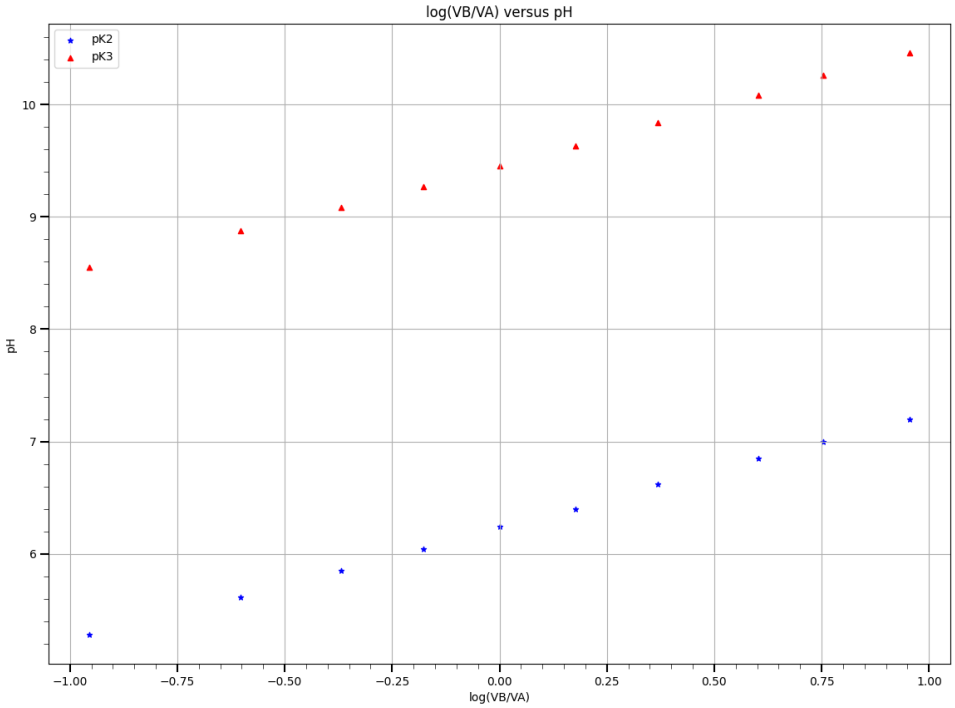
Output of the plot reading in pK2 and pK3 files with axes limit definition

With axes limit definition (i.e. using the code as shown in the script snippet above), the graph looks as follows:
As you can see, the axes are very neat and the data points are all there.



Output of the plot reading in pK2 and pK3 files without axes limit definition

Without axes limit definition (i.e. using “plt.xlim()” and “plt.ylim()”), the graph looks as follows:
As you can see, the graph is less neat and less aesthetically pleasing.

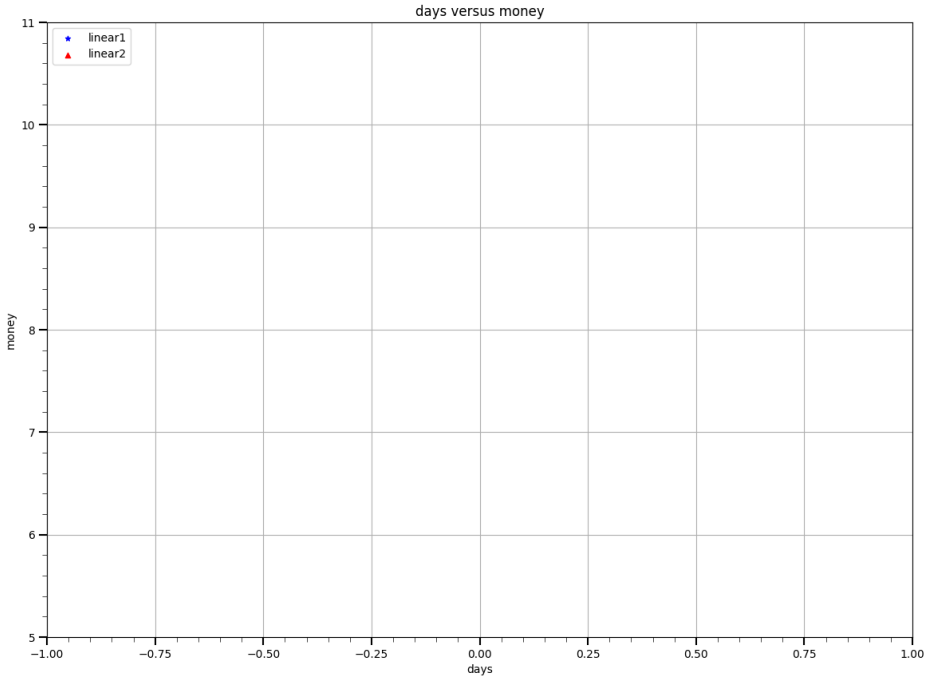


Why would I not define the axes limits

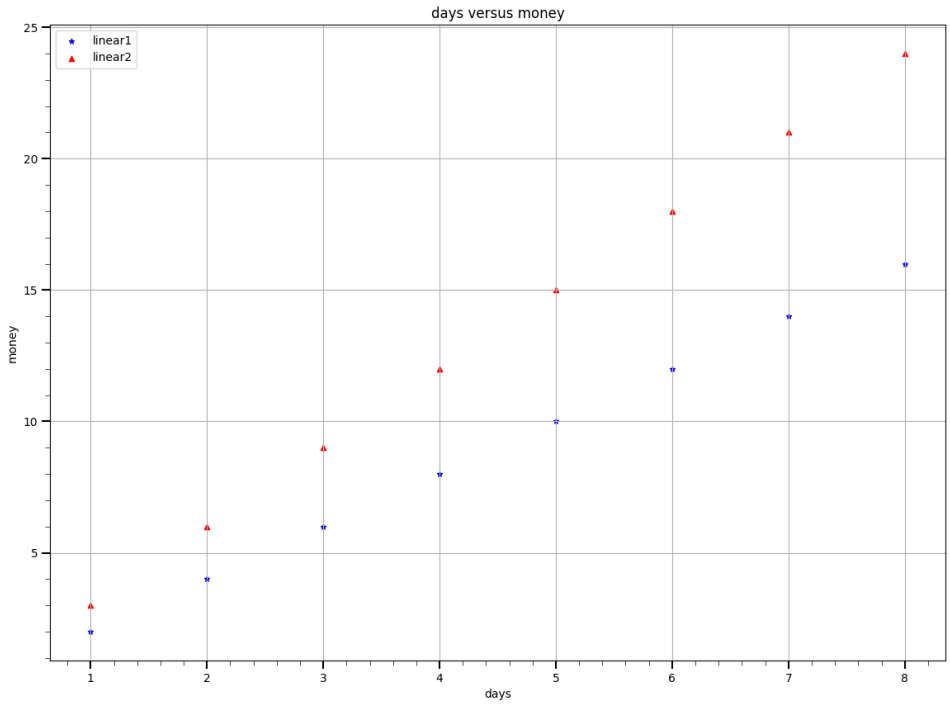
If the user would want to input other data, it is useful to not define the axes limits.
If the script snippet above is changed to include different data, then this might not be plotted if x doesn’t fall in the range -1,1 and y doesn’t fall in the range 5,11.
For example: the code below has still got the axes definitions as put in by me

```
### plots a graph with dimensions 14x10 with resolution 100
fig = plt.figure(figsize=(14,10), dpi=100)
plt1 = fig.add_subplot(111)

### data from two different csv files is read
data = pd.read_csv("linear1.csv")
x = data["days"]
y = data["money"]

data1 = pd.read_csv("linear2.csv")
x1 = data1["days"]
y1 = data1["money"]


### both datasets are plotted as scatterplots on the same graph
plt1.scatter(x,y,s = 20, c = 'b', marker='*', label = 'linear1')
plt1.scatter(x1,y1,s = 20, c = 'r', marker='^', label = 'linear2')

### any characteristics of the graph like axes titles, graph title and location of the legend
plt.xlabel("days")
plt.ylabel("money")
plt.title("days versus money")
plt.legend(loc='upper left')

### any axes characteristics such as x and y axes range, major and minor axes and grid-view
plt.xlim([-1,1])
plt.ylim([5,11])
plt.minorticks_on()
plt.tick_params(which="major", width=1.5)
plt.tick_params(which="major", length=7)
plt.tick_params(which="minor", width=0.5, length=4, color="black")
plt.grid(True)
```

This will produce the following output:



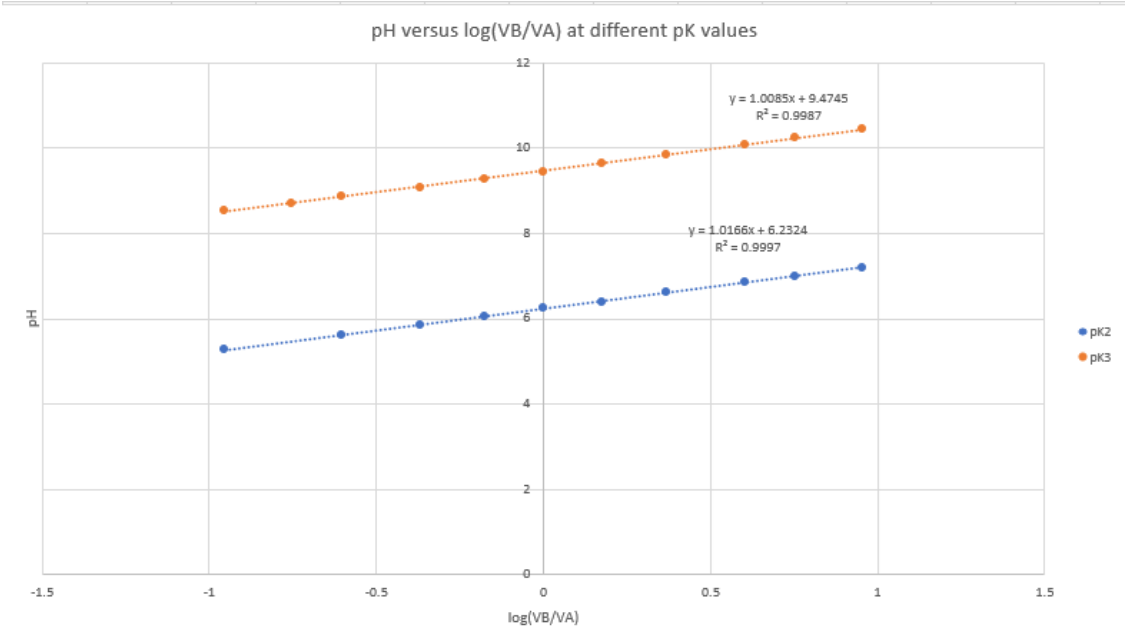
Now, if we change the code from “plt.xlim([-1,1])” and “plt.ylim([5,11])” to “plt.xlim()” and “plt.ylim()”, the output will look like this:



As you can see, this plotted data looks a lot better and is therefore better when it comes to the functionality and reusability of the code.
However, as explained before, for the sake of the data that is used to plot and the readability of the plot, I chose to include axes limits.

Determination of the line of best fit, R^2 and the plotting of these variables

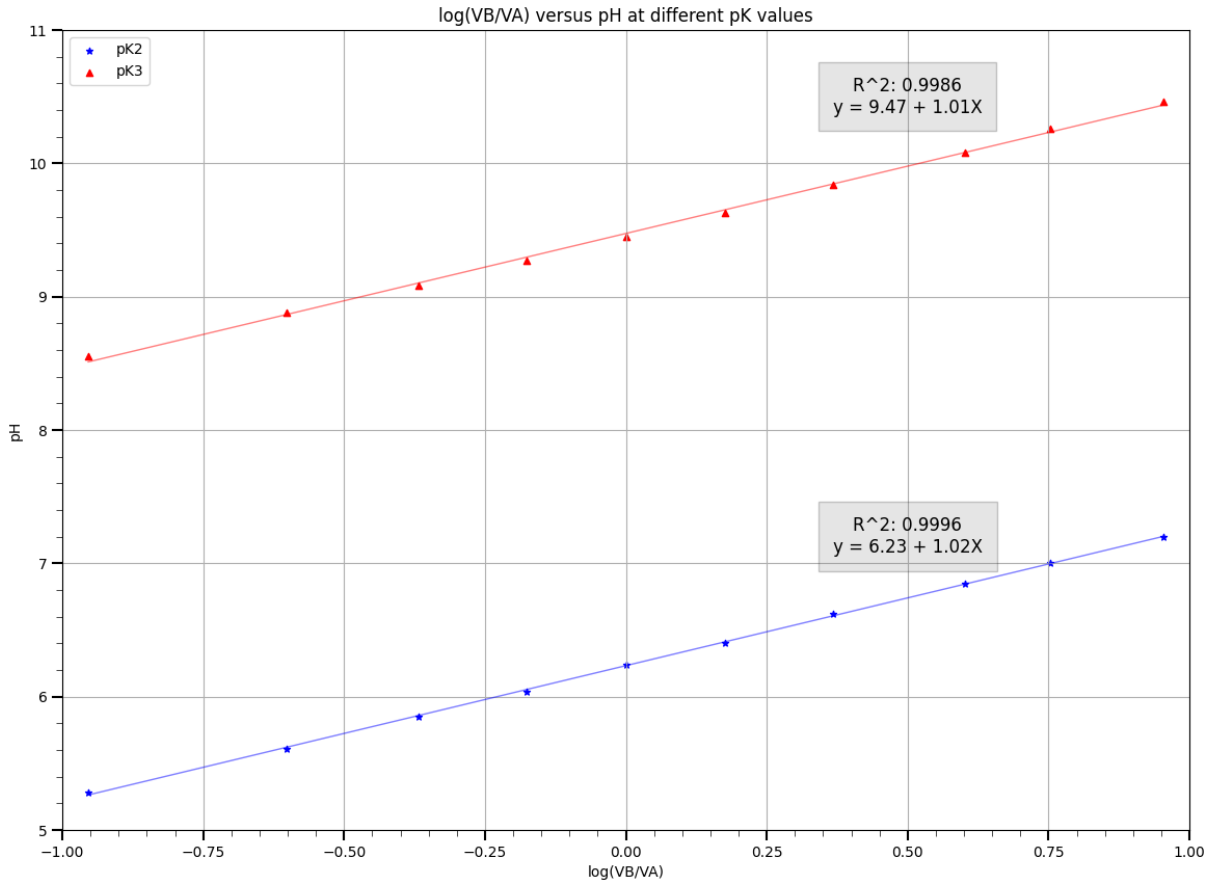
If the data in the tables above for pK2 and pK3 is plotted, then using Excel, the following graph is expected (line of best fit and R^2 are shown on the graph)



To determine the line of best fit and R^2 and plotting these on the graph, the code in the snippet below is added to the snippet before:

```
### define function for linear regression for dataset 1 and dataset 2
def linear_regression(x,y):
    N = len(x)
    x_mean = x.mean()
    y_mean = y.mean()

    B1_num = ((x - x_mean)*(y - y_mean)).sum()
    B1_den = ((x - x_mean)**2).sum()
    B1 = B1_num/B1_den
    B0 = y_mean - (B1*x_mean)

    reg_line = "y = {} + {}b".format(round(B0,3), round(B1,3))
    return (B0, B1, reg_line)

def linear_regression(x1,y1):
    N = len(x1)
    x1_mean = x1.mean()
    y1_mean = y1.mean()

    B2_num = ((x1 - x1_mean)*(y1 - y1_mean)).sum()
    B2_den = ((x1 - x1_mean)**2).sum()
    B2 = B2_num/B2_den
    B3 = y1_mean - (B2*x1_mean)

    reg_line = "y = {} + {}b".format(round(B3,3), round(B2,3))
    return (B3, B2, reg_line)
### define function for correlation coefficient for dataset 1 and dataset 2
def corr_coef(x,y):
    N = len(x)                                                          ### calculates the number of datasets in this dataset
    num = (N*(x*y).sum()) - (x.sum() * y.sum())                         ### calculates the numerator of this equation for R^2
    den = np.sqrt((N*(x**2).sum() - x.sum()**2) * (N*(y**2).sum() - y.sum()**2))        ### calculates the denominator of this equation for R^2
    R = num/den
    return round(R,4)

def corr_coef(x1,y1):
    N = len(x1)                                                         ### calculates the number of datasets in this dataset
    num = (N*(x1*y1).sum()) - (x1.sum() * y1.sum())                     ### calculates the numerator of this equation for R^2
    den = np.sqrt((N*(x1**2).sum() - x1.sum()**2) * (N*(y1**2).sum() - y1.sum()**2))    ### calculates the denominator of this equation for R^2
    R1 = num/den
    return round(R1,4)

### call functions for the two datasets
B0,B1, reg_line = linear_regression(x,y)
B3,B2, reg_line = linear_regression(x1,y1)
R = corr_coef(x,y)
R1 = corr_coef(x1,y1)

text = '''R^2: {}
y = {} + {}X'''.format(round(R1**2, 4), round(B3,2), round(B2,2))
text_1 = '''R^2: {}
y = {} + {}X'''.format(round(R**2, 4), round(B0,2), round(B1,2))

### plots text boxes with R^2, the equation for line of best fit and the line of best fit itself for the two datasets
plt.text(x = 0.50, y = 7.2, s= text_1, fontsize = 12, ha = "center", va = "center", bbox={"facecolor": "grey", "alpha": 0.2, "pad":10})
plt.text(x = 0.50, y = 10.5, s= text, fontsize = 12, ha = "center", va = "center", bbox={"facecolor": "grey", "alpha": 0.2, "pad":10})
plt.plot(x, B0+B1*x, c="b", linewidth=1, alpha=.5,solid_capstyle="round")
plt.plot(x1, B3+B2*x1, c="r", linewidth=1, alpha=.5,solid_capstyle="round")
```

The code added above as well as all the previous code then returns the output below:



When comparing this graph to the graph that was expected using Excel, you can see that they match up almost perfectly, indicating that the code works as expected.

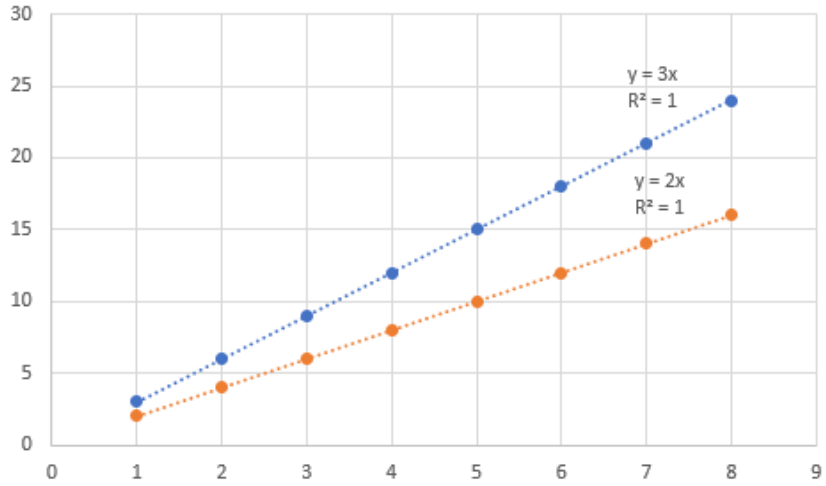
Testing code for maintainability/reusability through different data sets

All the code in the snippets before is now changed to reflect the two random made-up data tables (linear1 and linear2) below:

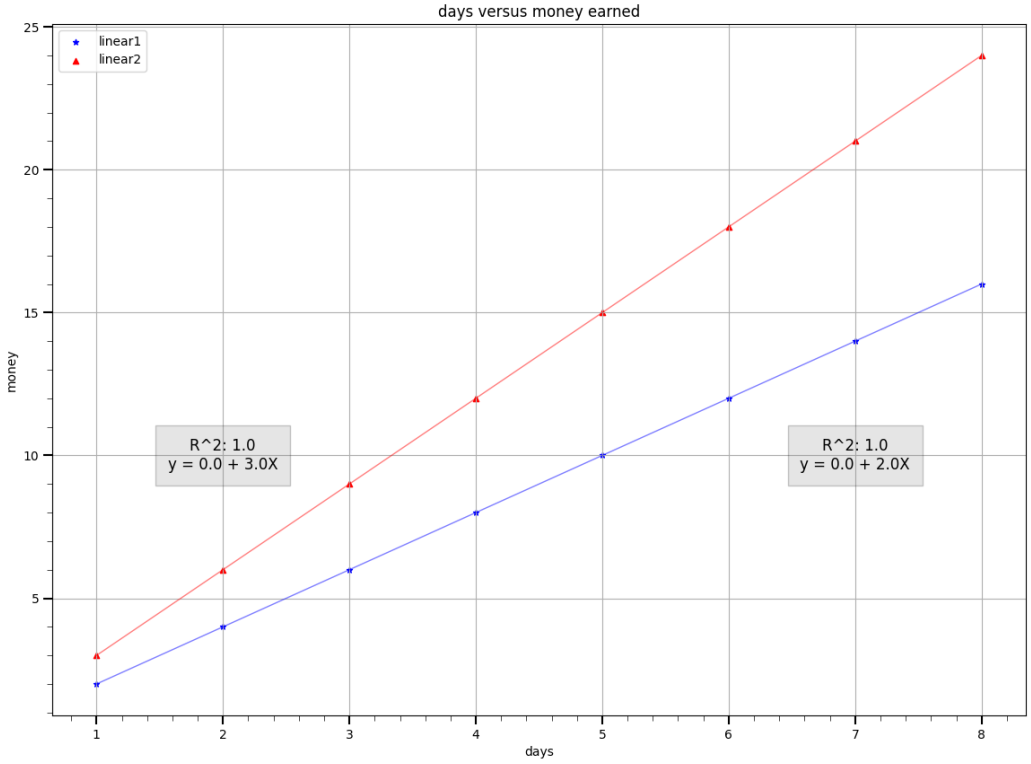
days	money
1	2
2	4
3	6
4	8
5	10
6	12
7	14
8	16

days	money
1	3
2	6
3	9
4	12
5	15
6	18
7	21
8	24

In Excel, the data above will get the following graph:



Using the code written in the script, the output will be as follows:

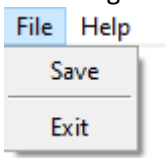


This shows that the code works if the appropriate pieces are changed as explained above.

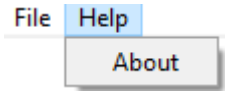
The code below is the code to output a menu bar with drop down modes

```
### opens a new tkinter window called pK plots
ws = Tk()
ws.title("pK plots")

### this function is called when 'save' button is clicked
def save_button():
     plt.savefig("output.jpg")

### this function is called when the 'about' button is clicked, opening a new tkinter window, displaying a message
def clicked3():
    msg2 = "Hi!" "\n" "This application was made by Lot Burgstra, B914713. \nPlease, if you have any questions make sure to email me on: l.n.burgstra-19@student.lboro.ac.uk"
    ws = Tk()
    ws.title("About")
    kj_label3 = Label(ws, text = msg2)
    kj_label3.pack()

### creates a menubar a dropdown file menu called "file", once cursor is on the file bar it will show a drop down menubar with clickable buttons "save" and "exit"
menubar = Menu(ws)

filemenu = Menu(menubar, tearoff = 0)
filemenu.add_command(label = "Save", command = save_button)     ### once clicked, the graph plot will be saved as a jpg output in the directory the user is working in
filemenu.add_separator()                                        ### separates the "Save" button from the "Exit" button with a little line
filemenu.add_command(label= "Exit", command = clicked2)         ### once clicked, exits the program
menubar.add_cascade(label="File", menu=filemenu)

helpmenu = Menu(menubar, tearoff = 0)
helpmenu.add_command(label = "About", command = clicked3)       ### once clicked, a new window will pop up, disaplaying a message about the developer of this script
menubar.add_cascade(label = "Help", menu=helpmenu)

ws.config(menu = menubar)
```

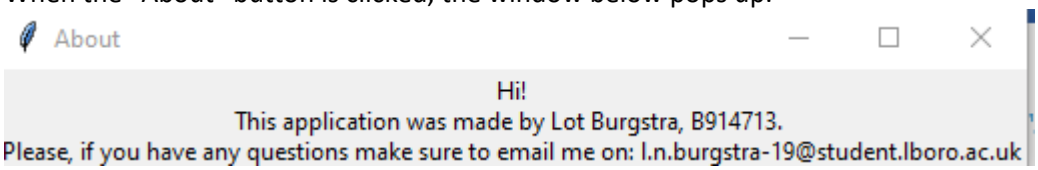
This will give the following options:



If “Save” is clicked above, it will give the user an output file in the form of a jpg with the full plot saved on it.
The “Exit” button will close the plot window.



When the “About” button is clicked, the window below pops up:



This was everything needed to explain in the ReadMe Document.
Personally, I think the code is user friendly and lends itself well to any other data if provided with this ReadMe Document.