Review

My use of my Subroutines:

My code is written within subroutines with no use or creation of global variables. This means it is extremely secure as the memory footprint of the program is miniscule. This is because nothing is needed to be stored in memory for prolonged periods of time whilst it is running apart from the imported libraries. Meaning the system can run for extended periods of time without experiencing any reduction in processing speeds.

Programming like this also means it is more difficult for malicious hackers to try and capture or change one of the variables being used in the code therefore making it more secure from external threats.

When calling the program I made sure to use industry standard practice so that the 'main' subroutine was called using __name__ == "__main__" making the subroutine run without having to call it in terminal

My use of the Subroutines provided:

A good programmer knows how to cut corners and find efficiency within his code. To display this I made sure to call the subroutines Recoats programmers had provided me as well as comment upon their code and find more efficient functions. This can be seen within the get_total_data function of which I reduced the amount of lines in the function and the unnecessary calling of the Day column.

```
89  def get_total_data(total_choice):
90
91      df = read_data_in() # Changed this to fit my subroutine to read in the CSV
92
93      income = df[[total_choice]].sum() #Added .sum() here. The program also called in the day column. It was not needed so I removed it
94
95      # total = income[total_choice].sum() # This line is not needed if you put the .sum() at the end of the initial wrangling
96
97      msg = "The total income from {} was: £{}".format(total_choice, income)
98      return msg
```

In terms of my solution meeting the needs set by Recoats Adventure Park, my program can be applied to all outlined uses. I will display how I met these with evidence below.

The solution must display trends and patterns over time for:

Different Payment types -

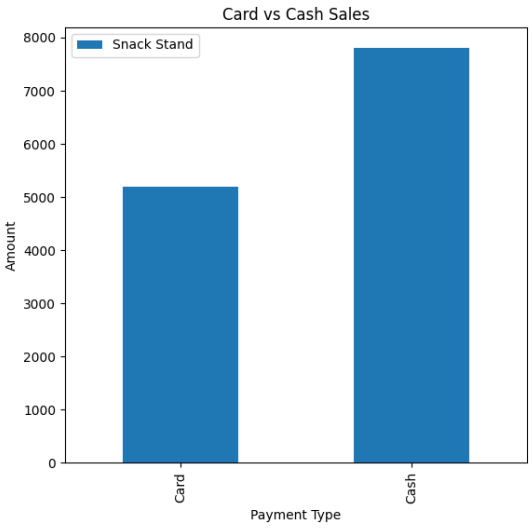
If you select either option 4-5 in my program it will relate to the Card vs Cash sales on a given source. Option 4 will display the total sales on each item in numerical and graphical format. This states a clear difference to the user and shows which one totals more sales. However, it does not display the currency to the user which may cause confusion as to what is the actual meaning behind the number. This should've been something I added before I finalised my work.

I have also shown the average daily sales in both card and cash within option 5 which also falls when it comes to displaying within the correct format of currency. Possible making it unclear to the users what the numbers mean that they are viewing.

Overall I believe I have fully met this criteria, it just needs polishing up in terms of it's outputting.

```
Enter your number selction here: 3
Choice accepted!
          Snack Stand
Pay Type
Card              5197
Cash              7805
```

```
The average sales for Card Transactions was:
185.80769230769232
The average sales for Cash Transactions was:
272.2142857142857
```

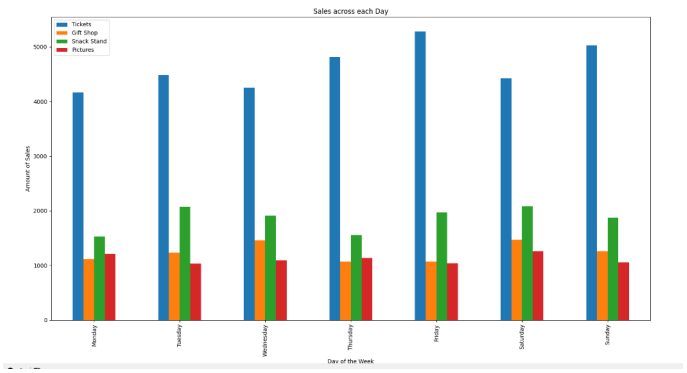


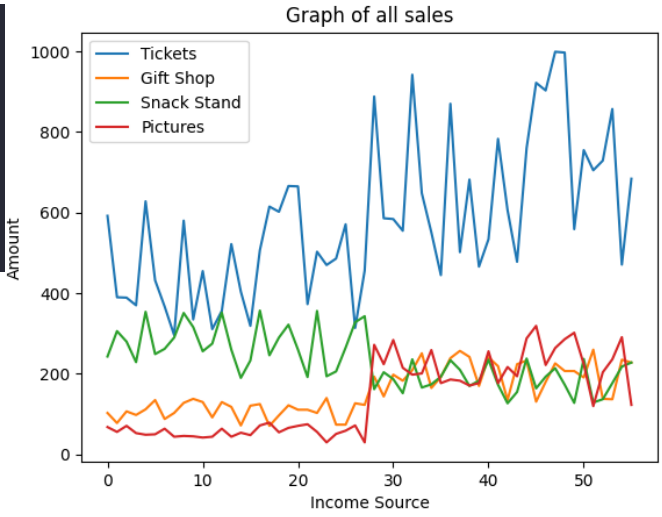
Different income Sources -

My program allows for 7 different ways to display data relating to the income of sources. This is great for the analysis of the data as you can be viewing it in multiple contexts to allow for a full picture. Each option choice goes into it's own subroutine to help keep organisation of the code and the flow of data more controlled. Most options display graphs to help visualise the data as well as print it out to the terminal for more in depth, numerical analysis.

Overall I believe this section is fully met in its criteria as it allows for multiple different outputs of the data from the CSV.

Day	Tickets	Gift Shop	Snack Stand	Pictures
Monday	4163	1116	1525	1209
Tuesday	4481	1228	2076	1034
Wednesday	4252	1458	1911	1093
Thursday	4814	1067	1560	1134
Friday	5280	1071	1973	1040
Saturday	4420	1467	2084	1264
Sunday	5025	1259	1873	1054





Income on different days of the week -

Option 3 on my program groups the sales by the days of the week. Directly meeting this set need and allowing for both graphical and numeric display (as shown in images above)

This criteria is fully met.

Solution must be secure -

The program captures all possible humans errors and deals with them whilst still maintaining a constant flow. This makes it secure. The only way to exit the code is to input '00' when making a menu choice.

The criteria is fully met.

User Requirements:

Solution must be easy to use-

The solution only requires numerical inputs ranging from 0-7 to fully operate the whole program. This makes it extremely easy to navigate as all you need is an enter key and a numpad. It outputs menus for you to select from which clearly outline what functions they will perform and loops continually so if it was not what you needed, you can select another option.

I believe this criteria is fully met.

Display information in a meaningful way -

Most out the outputting performed by the program is simple and therefore easy to digest however, when outputting averages or total values, no currency format is used which can make it quite difficult to understand what you are reading. This is an issue which can be resolved with a rounding function or by capturing integers as floats but I do not know the syntax to perform those operations so was not able to implement them. This is something that can be solves relatively fast if I was to learn those functions.

This condition is partially met as it can be improved upon to display thinks in crystal clear ways.

Make use of ways to output data that would be relevant to the end user -

Most functions that would display a graphical source of data will also print a numerical one to terminal to help with both sides of digestion however, when displaying the sales overtime for all sources it does not print anything to the terminal. This is because this would mean printing the whole CSV file which is not relevant and can be overwhelming so instead only the graph is produced. Multiple graphs are used throughout the program which shows ways of outputting data in bar or line graphs as well as the numerical ways in the terminal.

Overall this condition is fully met.

Further development of my solution:

When creating my program I have made sure to comment on what each section does to allow for other developers to understand my thinking and processing of data. This should make any improvements they would make upon my code easier to implement due to them being able to ready my code as they go through it and the comments alongside. This helps with the scalability of my code as it is easier to continually add to it.

In terms of things that need developing, my numerical display of values needs to be done in the correct currency to allow for users to understand what figures they are looking at. This is something that affects the user end and therefore should be a priority. The impact of this would mean that there is no confusion when outputting the data in this numerical format and therefore users will never be confused at what the big bundle of numbers mean when they look at it.

Overall my program allows itself to work as a stand-alone project, but also be built upon by other developers if they wanted to improve upon it.

Takeaways as a developer

Personally I believe this code is a great display of my experience as a developer and I will be using it to reference my strengths within the pandas library, and my weakness within matplotlib and formatting numerical data as a whole.