STANDARD DEVIATION CALC

TABLE OF CONTENTS

- Why create this app?
- Development/build process
- Code walkthrough
- Terminal walkthrough
- Conculsion

WHY CREATE THIS TOOL?

First it would make sense to briefly explain what we are trying to accomplish with the app.

The intention is that we are comparing two data sets from different cameras in terms of performance, with this tool we can measure data in mass, display the mean, standard deviation and have the data displayed visually to allow us to see trends where possible.

DEVELOPMENT/BUILD PROCESS

For the development process I kept track of progress via Trello.

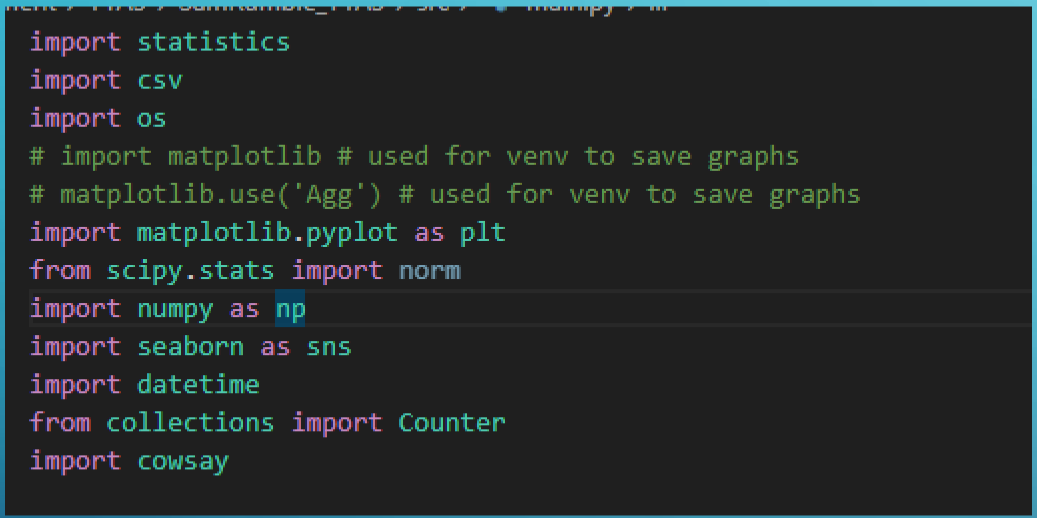
I originally created cards at the start before I started coding to give myself an idea on how the app would be structured. After I hit a roadblock or unexpected step that would be added as an additional card that would need to be completed in order to progress.

In a later section we will show how the build progressed while looking at the functions.

CODE WALKTHROUGH

- Import statements
- Functions
- Variables
- Try/except

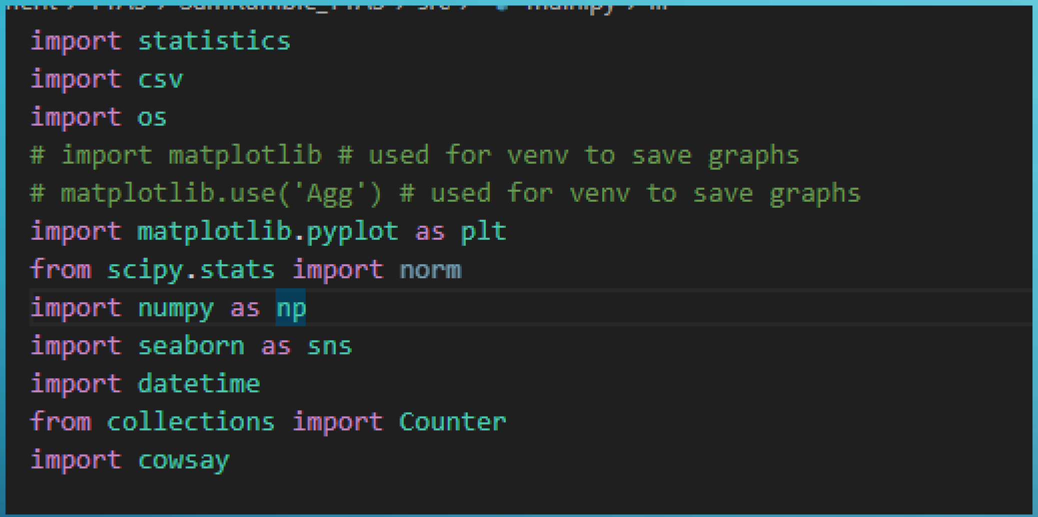
IMPORTS



```
import statistics
import csv
import os
# import matplotlib # used for venv to save graphs
# matplotlib.use('Agg') # used for venv to save graphs
import matplotlib.pyplot as plt
from scipy.stats import norm
import numpy as np
import seaborn as sns
import datetime
from collections import Counter
import cowsay
```

- Statistics
  - Used for calculating statistical data
- Csv
  - Allows us access to read/write csv files
- Os
  - Gives us a way to access os dependant functions
- Matplotlib
  - Is used to create the graphs we use
- Scipy
  - Is redundant for now but will be used in future for normal distribution plots

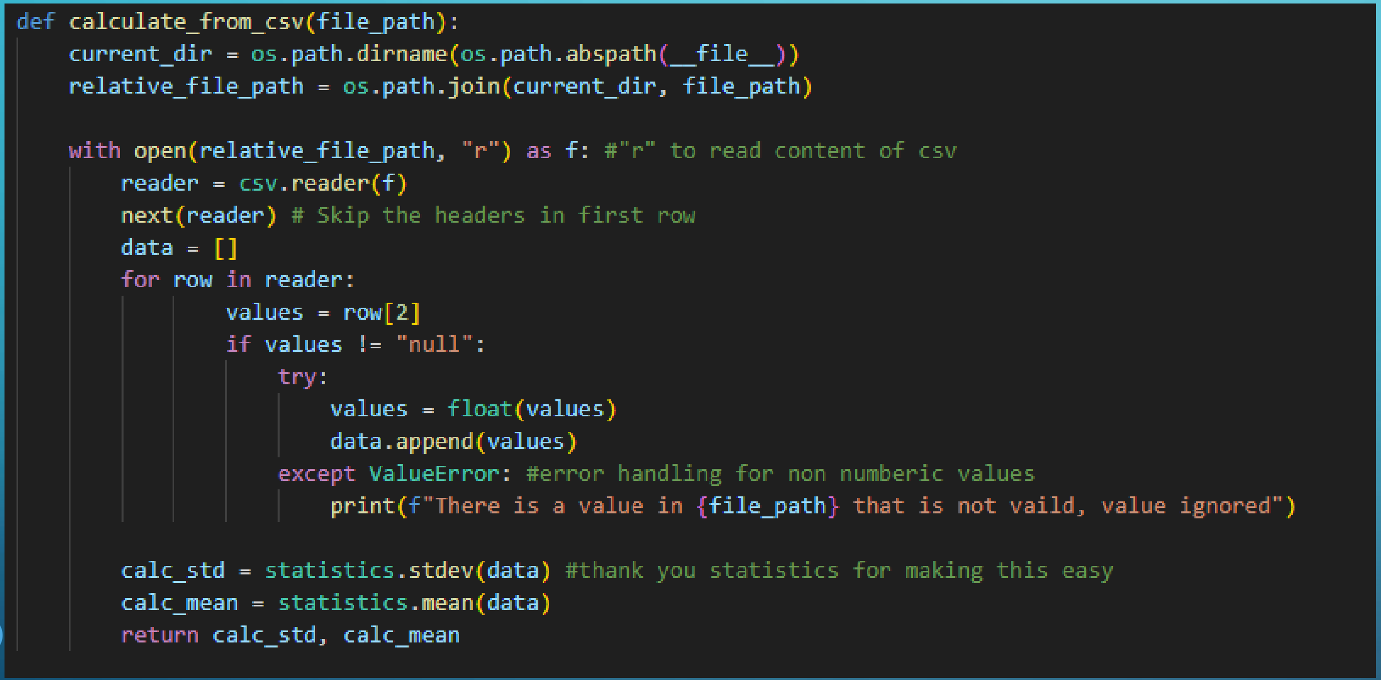
IMPORTS CONT.



```python
import statistics
import csv
import os
# import matplotlib # used for venv to save graphs
# matplotlib.use('Agg') # used for venv to save graphs
import matplotlib.pyplot as plt
from scipy.stats import norm
import numpy as np
import seaborn as sns
import datetime
from collections import Counter
import cowsay
```

- Numpy
  - Used to manipulate our arrays for the graphs
- Seaborn
  - Purely used cosmetically for our matplotlib graphs
- Datetime
  - The output we store to the csv will display the date the program was run
- Counter
  - A nice tool that allows us to count occurrences of an element
- Cowsay
  - A goofy tool that allows us to use ascii art

FUNCTIONS

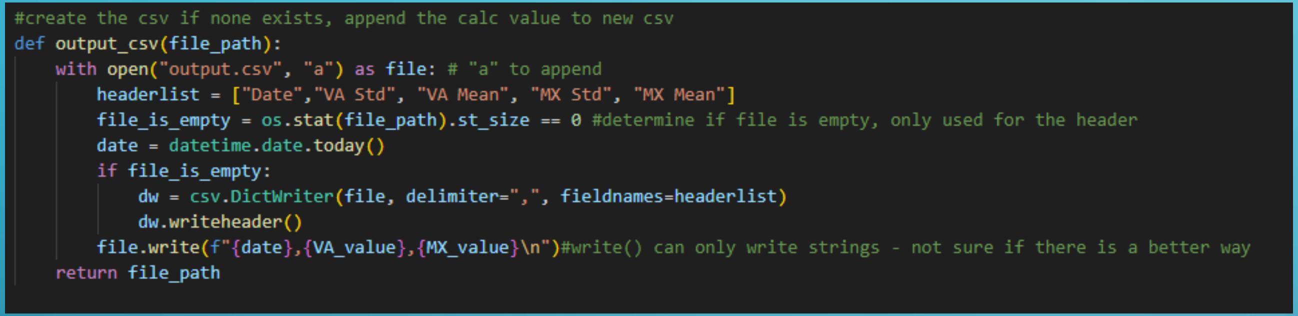


```
def calculate_from_csv(file_path):
    current_dir = os.path.dirname(os.path.abspath(__file__))
    relative_file_path = os.path.join(current_dir, file_path)

    with open(relative_file_path, "r") as f: #"r" to read content of csv
        reader = csv.reader(f)
        next(reader) # Skip the headers in first row
        data = []
        for row in reader:
                values = row[2]
                if values != "null":
                    try:
                        values = float(values)
                        data.append(values)
                    except ValueError: #error handling for non numberic values
                        print(f"There is a value in {file_path} that is not vaild, value ignored")

        calc_std = statistics.stdev(data) #thank you statistics for making this easy
        calc_mean = statistics.mean(data)
        return calc_std, calc_mean
```

calculate_from_csv
Our first core function that makes use of os, csv and statistics imports

We iterate through row two of our csv's, ignoring any null values and printing an error when a non-numeric value is displayed

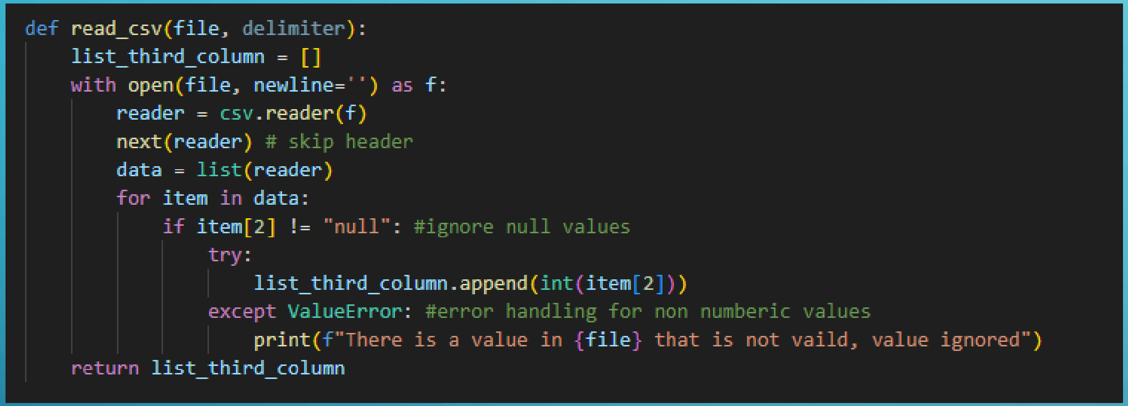
FUNCTIONS



```python
#create the csv if none exists, append the calc value to new csv
def output_csv(file_path):
    with open("output.csv", "a") as file: # "a" to append
        headerlist = ["Date","VA Std", "VA Mean", "MX Std", "MX Mean"]
        file_is_empty = os.stat(file_path).st_size == 0 #determine if file is empty, only used for the header
        date = datetime.date.today()
        if file_is_empty:
            dw = csv.DictWriter(file, delimiter=",", fieldnames=headerlist)
            dw.writeheader()
        file.write(f"{date},{VA_value},{MX_value}\n")#write() can only write strings - not sure if there is a better way
    return file_path
```

The output from the previous function is used here which will append to “output.csv” if such a csv doesn’t exist it will create it and append to it.

FUNCTIONS

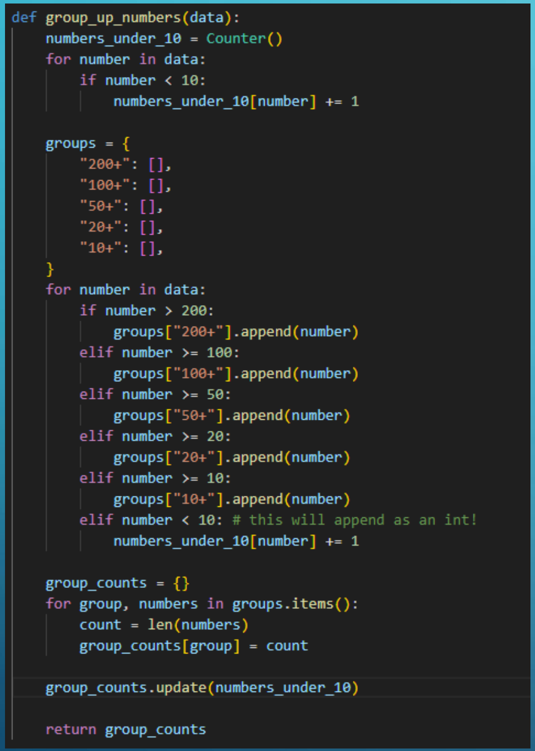


```python
def read_csv(file, delimiter):
    list_third_column = []
    with open(file, newline='') as f:
        reader = csv.reader(f)
        next(reader) # skip header
        data = list(reader)
        for item in data:
            if item[2] != "null": #ignore null values
                try:
                    list_third_column.append(int(item[2]))
                except ValueError: #error handling for non numberic values
                    print(f"There is a value in {file} that is not vaild, value ignored")
    return list_third_column
```

Here we read the data from our csv and store its contents into an empty string for later

The function also ignores null values and raises an error if a non-numeric value is detected. The data is not affected by any values that aren’t integers

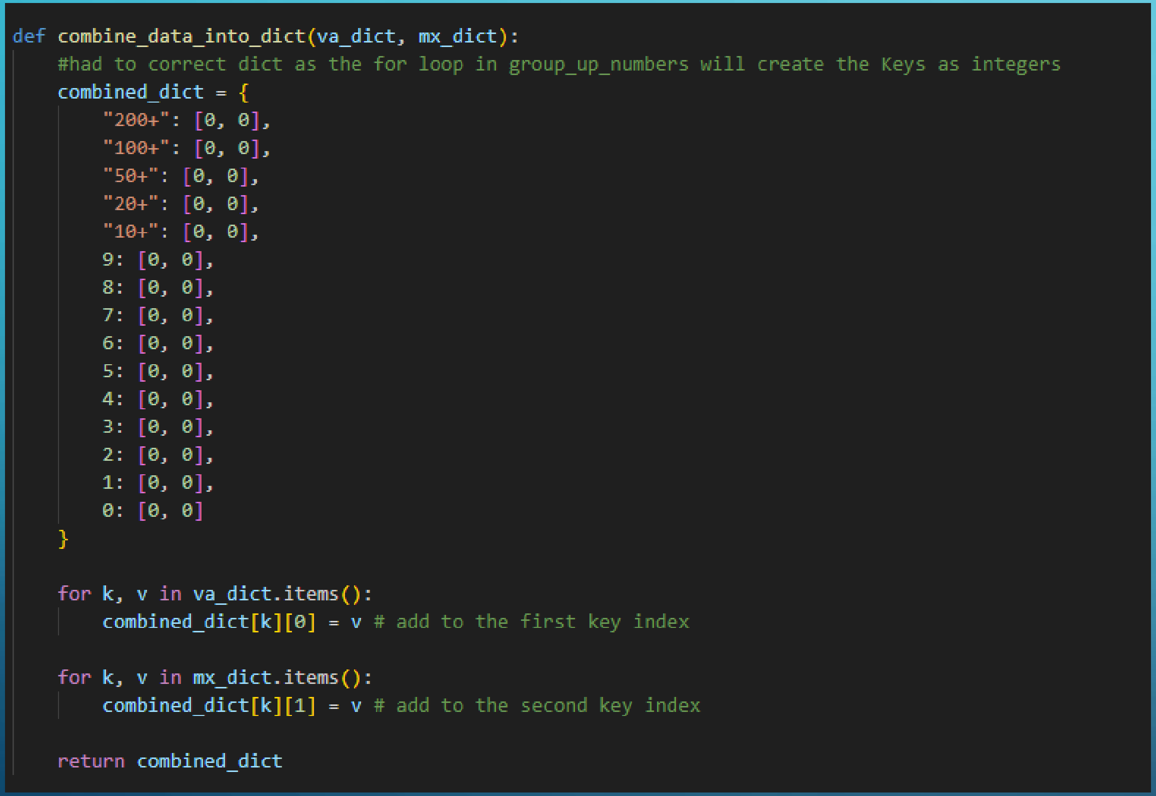
FUNCTIONS



```python
def group_up_numbers(data):
    numbers_under_10 = Counter()
    for number in data:
        if number < 10:
            numbers_under_10[number] += 1

    groups = {
        "200+": [],
        "100+": [],
        "50+": [],
        "20+": [],
        "10+": [],
    }
    for number in data:
        if number > 200:
            groups["200+"].append(number)
        elif number >= 100:
            groups["100+"].append(number)
        elif number >= 50:
            groups["50+"].append(number)
        elif number >= 20:
            groups["20+"].append(number)
        elif number >= 10:
            groups["10+"].append(number)
        elif number < 10: # this will append as an int!
            numbers_under_10[number] += 1

    group_counts = {}
    for group, numbers in groups.items():
        count = len(numbers)
        group_counts[group] = count

    group_counts.update(numbers_under_10)

    return group_counts
```

Next the data is to be sorted and grouped into a dictionary, i.e. all counts of “0” will be added to the dictionary under the 0 Key. And the Value will increment by one for each count of 0 occurs.

FUNCTIONS

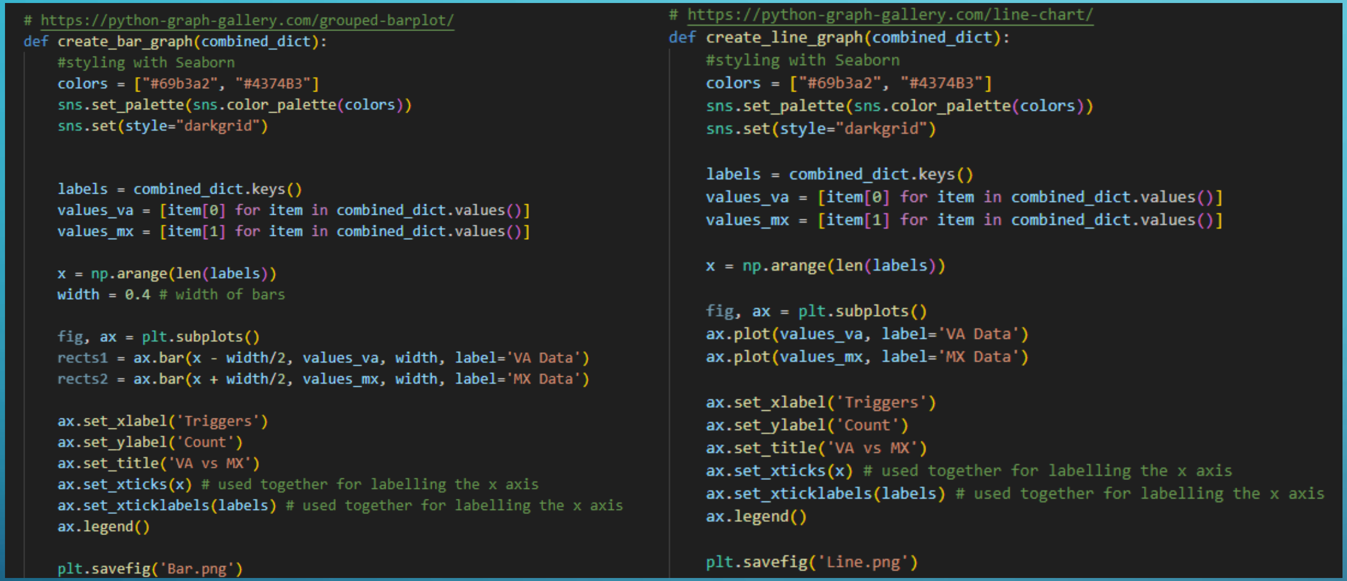


```
def combine_data_into_dict(va_dict, mx_dict):
    #had to correct dict as the for loop in group_up_numbers will create the Keys as integers
    combined_dict = {
        "200+": [0, 0],
        "100+": [0, 0],
        "50+": [0, 0],
        "20+": [0, 0],
        "10+": [0, 0],
        9: [0, 0],
        8: [0, 0],
        7: [0, 0],
        6: [0, 0],
        5: [0, 0],
        4: [0, 0],
        3: [0, 0],
        2: [0, 0],
        1: [0, 0],
        0: [0, 0]
    }

    for k, v in va_dict.items():
        combined_dict[k][0] = v # add to the first key index

    for k, v in mx_dict.items():
        combined_dict[k][1] = v # add to the second key index

    return combined_dict
```

Because we have two different data sets that need to be analysed, we will need to store them both in a combined dictionary to be able to plot them later for comparison.

The data is sorted and stored in the values depending on its index

FUNCTIONS



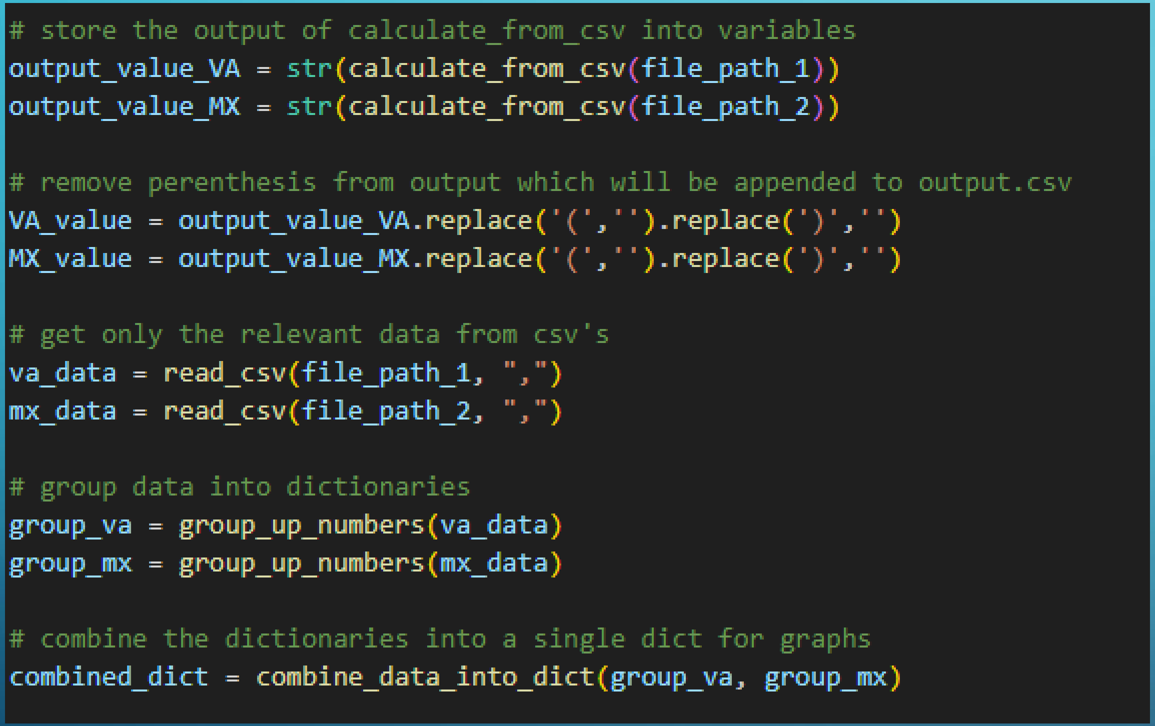
```python
# https://python-graph-gallery.com/grouped-barplot/
def create_bar_graph(combined_dict):
    #styling with Seaborn
    colors = ["#69b3a2", "#4374B3"]
    sns.set_palette(sns.color_palette(colors))
    sns.set(style="darkgrid")


    labels = combined_dict.keys()
    values_va = [item[0] for item in combined_dict.values()]
    values_mx = [item[1] for item in combined_dict.values()]

    x = np.arange(len(labels))
    width = 0.4 # width of bars

    fig, ax = plt.subplots()
    rects1 = ax.bar(x - width/2, values_va, width, label='VA Data')
    rects2 = ax.bar(x + width/2, values_mx, width, label='MX Data')

    ax.set_xlabel('Triggers')
    ax.set_ylabel('Count')
    ax.set_title('VA vs MX')
    ax.set_xticks(x) # used together for labelling the x axis
    ax.set_xticklabels(labels) # used together for labelling the x axis
    ax.legend()

    plt.savefig('Bar.png')
```

```python
# https://python-graph-gallery.com/line-chart/
def create_line_graph(combined_dict):
    #styling with Seaborn
    colors = ["#69b3a2", "#4374B3"]
    sns.set_palette(sns.color_palette(colors))
    sns.set(style="darkgrid")

    labels = combined_dict.keys()
    values_va = [item[0] for item in combined_dict.values()]
    values_mx = [item[1] for item in combined_dict.values()]

    x = np.arange(len(labels))

    fig, ax = plt.subplots()
    ax.plot(values_va, label='VA Data')
    ax.plot(values_mx, label='MX Data')

    ax.set_xlabel('Triggers')
    ax.set_ylabel('Count')
    ax.set_title('VA vs MX')
    ax.set_xticks(x) # used together for labelling the x axis
    ax.set_xticklabels(labels) # used together for labelling the x axis
    ax.legend()

    plt.savefig('Line.png')
```

Here are the functions for the graphs which take their input from the previous function the combined dictionary.

The data is plotted from the Values based on their index.

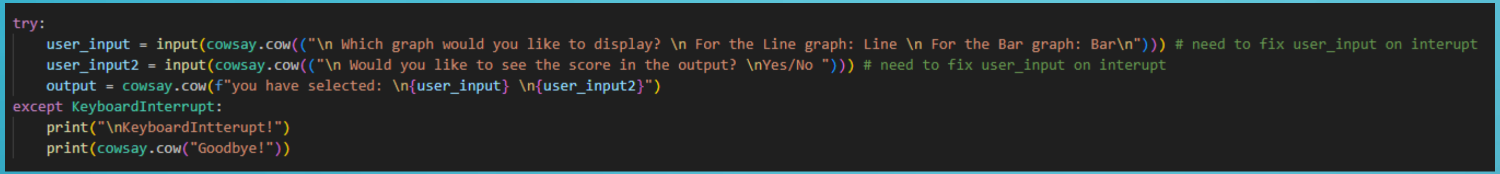
GLOBAL VARIABLES



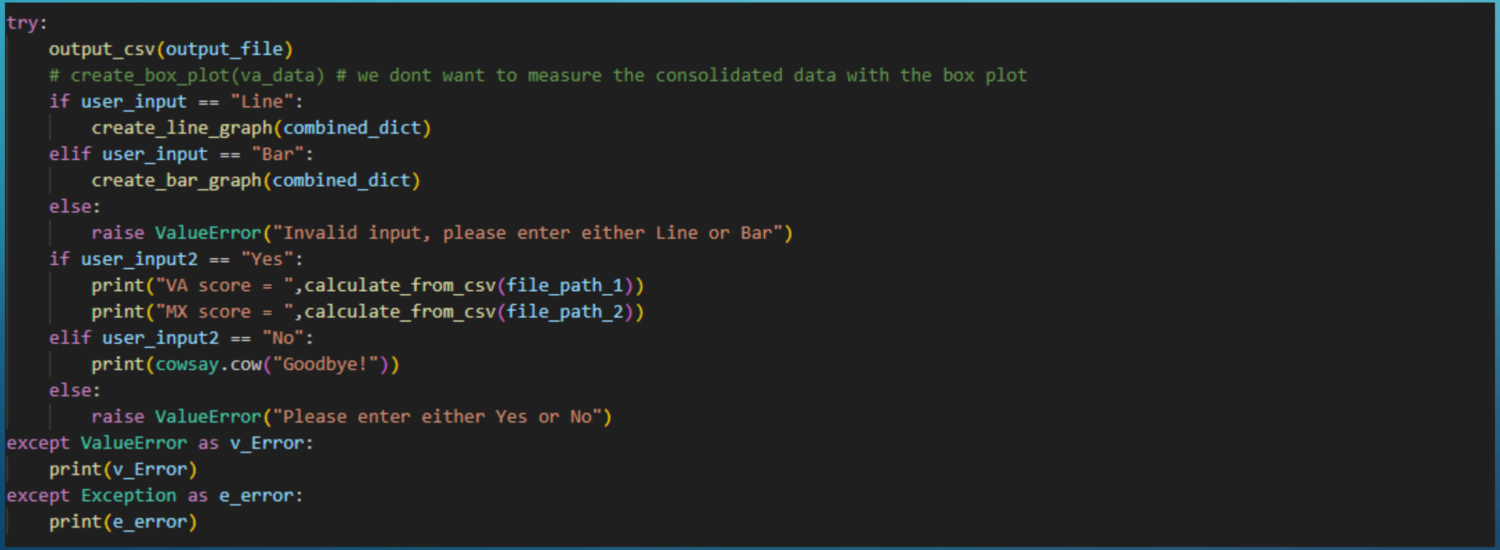
```
# store the output of calculate_from_csv into variables
output_value_VA = str(calculate_from_csv(file_path_1))
output_value_MX = str(calculate_from_csv(file_path_2))

# remove perenthesis from output which will be appended to output.csv
VA_value = output_value_VA.replace('(','').replace(')','')
MX_value = output_value_MX.replace('(','').replace(')','')

# get only the relevant data from csv's
va_data = read_csv(file_path_1, ",")
mx_data = read_csv(file_path_2, ",")

# group data into dictionaries
group_va = group_up_numbers(va_data)
group_mx = group_up_numbers(mx_data)

# combine the dictionaries into a single dict for graphs
combined_dict = combine_data_into_dict(group_va, group_mx)
```

Here are the variables in the global space which utilize the functions, the comments I feel summarize their purpose well enough to not need to repeat it.

TRY/EXCEPT



```
try:
    user_input = input(cowsay.cow(("\n Which graph would you like to display? \n For the Line graph: Line \n For the Bar graph: Bar\n"))) # need to fix user_input on interupt
    user_input2 = input(cowsay.cow(("\n Would you like to see the score in the output? \nYes/No "))) # need to fix user_input on interupt
    output = cowsay.cow(f"you have selected: \n{user_input} \n{user_input2}")
except KeyboardInterrupt:
    print("\nKeyboardIntterupt!")
    print(cowsay.cow("Goodbye!"))
```

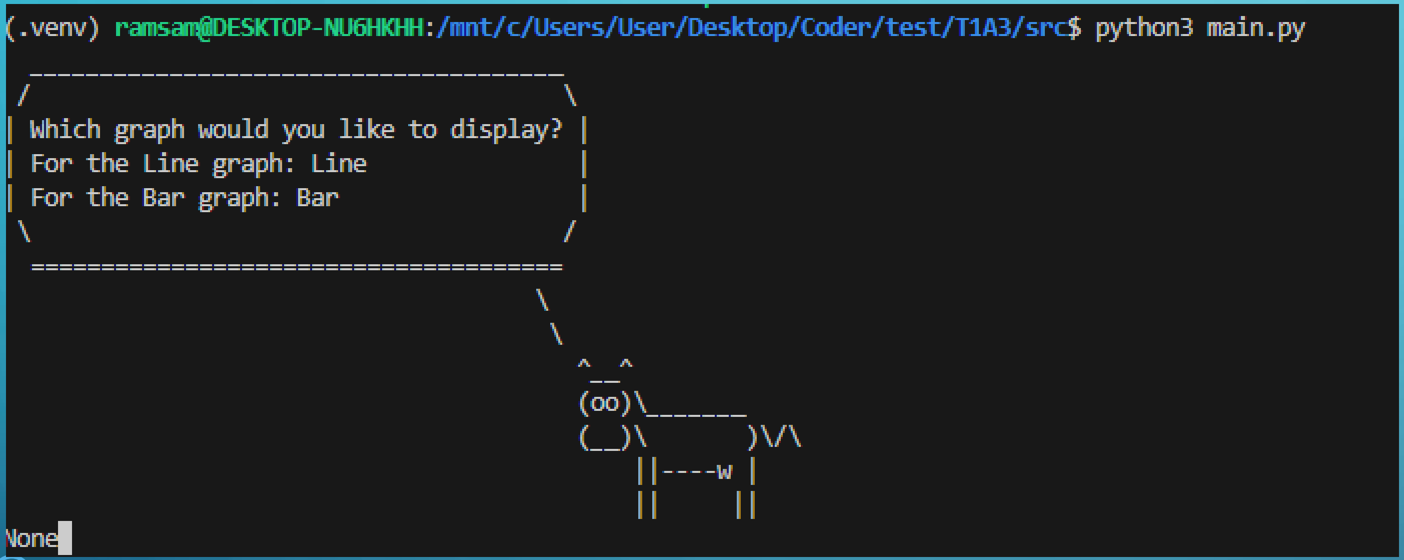


```
try:
    output_csv(output_file)
    # create_box_plot(va_data) # we dont want to measure the consolidated data with the box plot
    if user_input == "Line":
        create_line_graph(combined_dict)
    elif user_input == "Bar":
        create_bar_graph(combined_dict)
    else:
        raise ValueError("Invalid input, please enter either Line or Bar")
    if user_input2 == "Yes":
        print("VA score = ",calculate_from_csv(file_path_1))
        print("MX score = ",calculate_from_csv(file_path_2))
    elif user_input2 == "No":
        print(cowsay.cow("Goodbye!"))
    else:
        raise ValueError("Please enter either Yes or No")
except ValueError as v_Error:
    print(v_Error)
except Exception as e_error:
    print(e_error)
```

Here is how we end our program.

- Grab the user's input
- Output the contents to csv
- Determine which graph to display
- Determine if we need to print the score

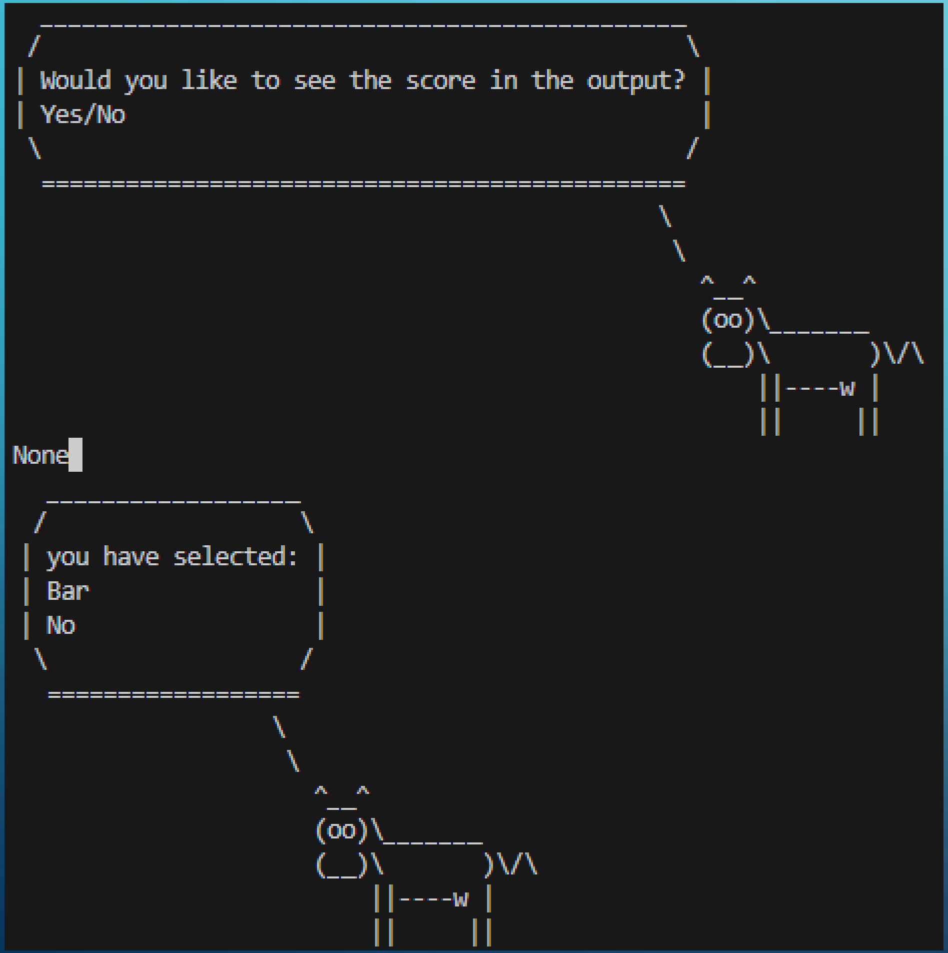
WALKTHROUGH APP



```
(.venv) ramsam@DESKTOP-NU6HKHH:/mnt/c/Users/User/Desktop/Coder/test/T1A3/src$ python3 main.py
  ______________________________________
 /                                      \
| Which graph would you like to display? |
| For the Line graph: Line               |
| For the Bar graph: Bar                 |
 \                                      /
  ======================================
                                      \
                                       \
                                         ^__^
                                         (oo)\_______
                                         (__)\       )\/\
                                             ||----w |
                                             ||     ||
None
```

Upon running the code you will be prompted with following choices to save either a Line or Bar graph

WALKTHROUGH APP

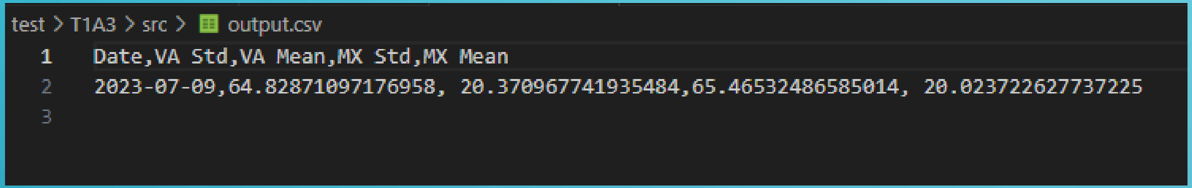


```
  ______________________________________________
 /                                              \
| Would you like to see the score in the output? |
| Yes/No                                         |
 \                                              /
  ==============================================
                                               \
                                                \
                                                  ^__^
                                                  (oo)\_______
                                                  (__)\       )\/\
                                                      ||----w |
                                                      ||     ||
None
  __________________
 /                  \
| you have selected: |
| Bar                |
| No                 |
 \                  /
  ==================
                   \
                    \
                      ^__^
                      (oo)\_______
                      (__)\       )\/\
                          ||----w |
                          ||     ||
```

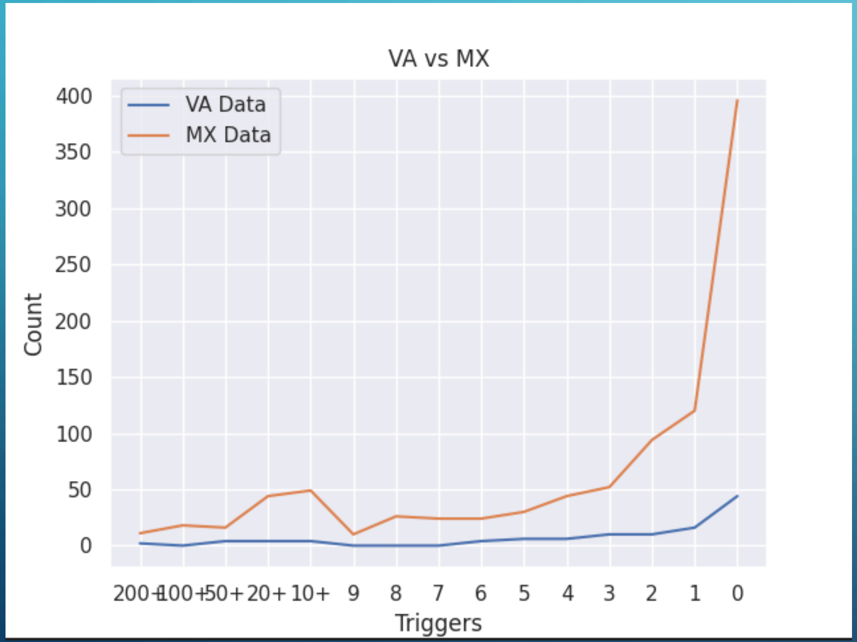
Next you will need to either specify if you want the output score to be displayed with a Yes or No answer

Our friend will then re-iterate our input.

WALKTHROUGH APP



```
Date,VA Std,VA Mean,MX Std,MX Mean
2023-07-09,64.82871097176958, 20.370967741935484,65.46532486585014, 20.023722627737225
```



We then get out output in the following:

- A graph with the data

- The score is appended to our csv

CONCLUSION

The program will likely evolve further as it will be the main tool I use over the next few months to measure performance.