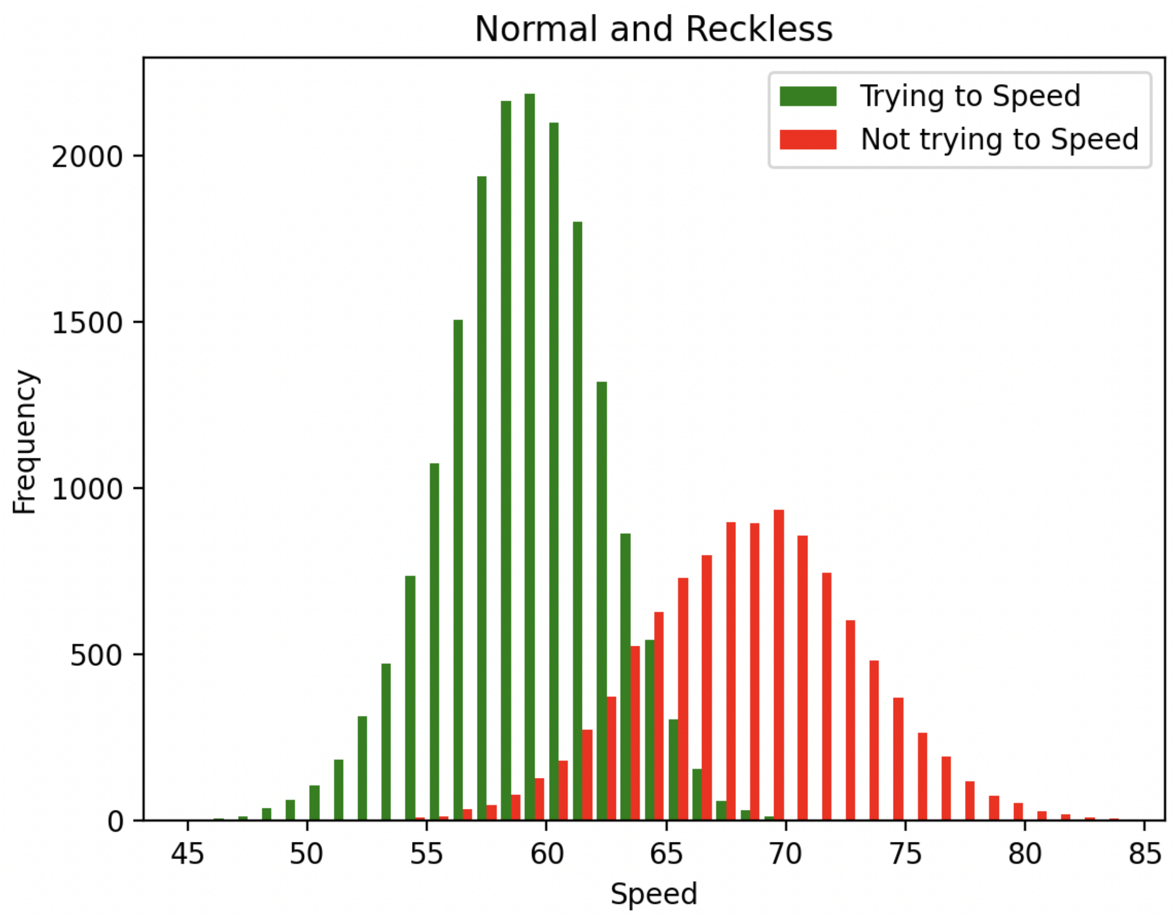
HW_02_Kale_Kushal CSCI720

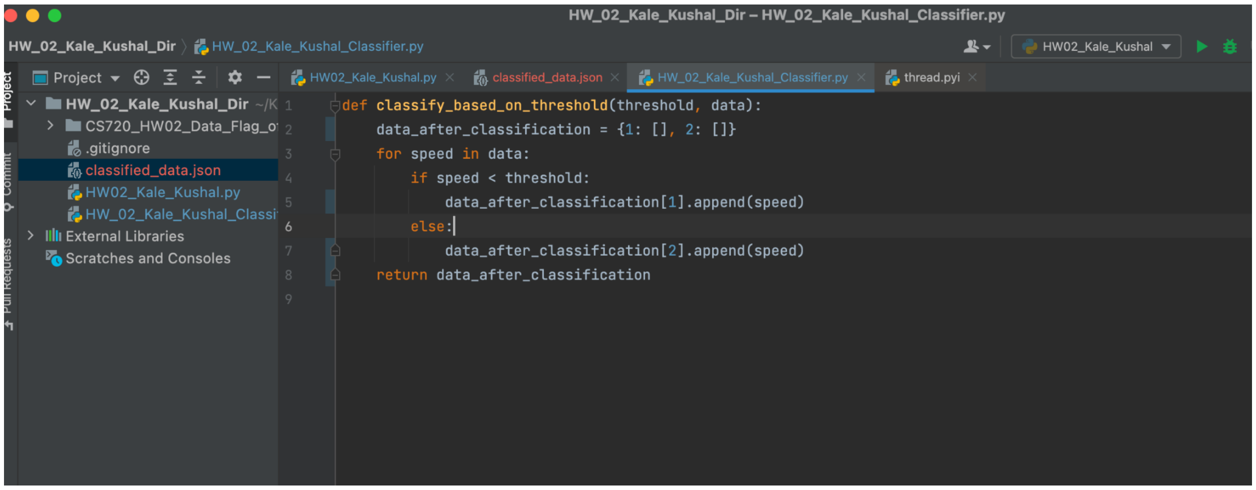
Q3.



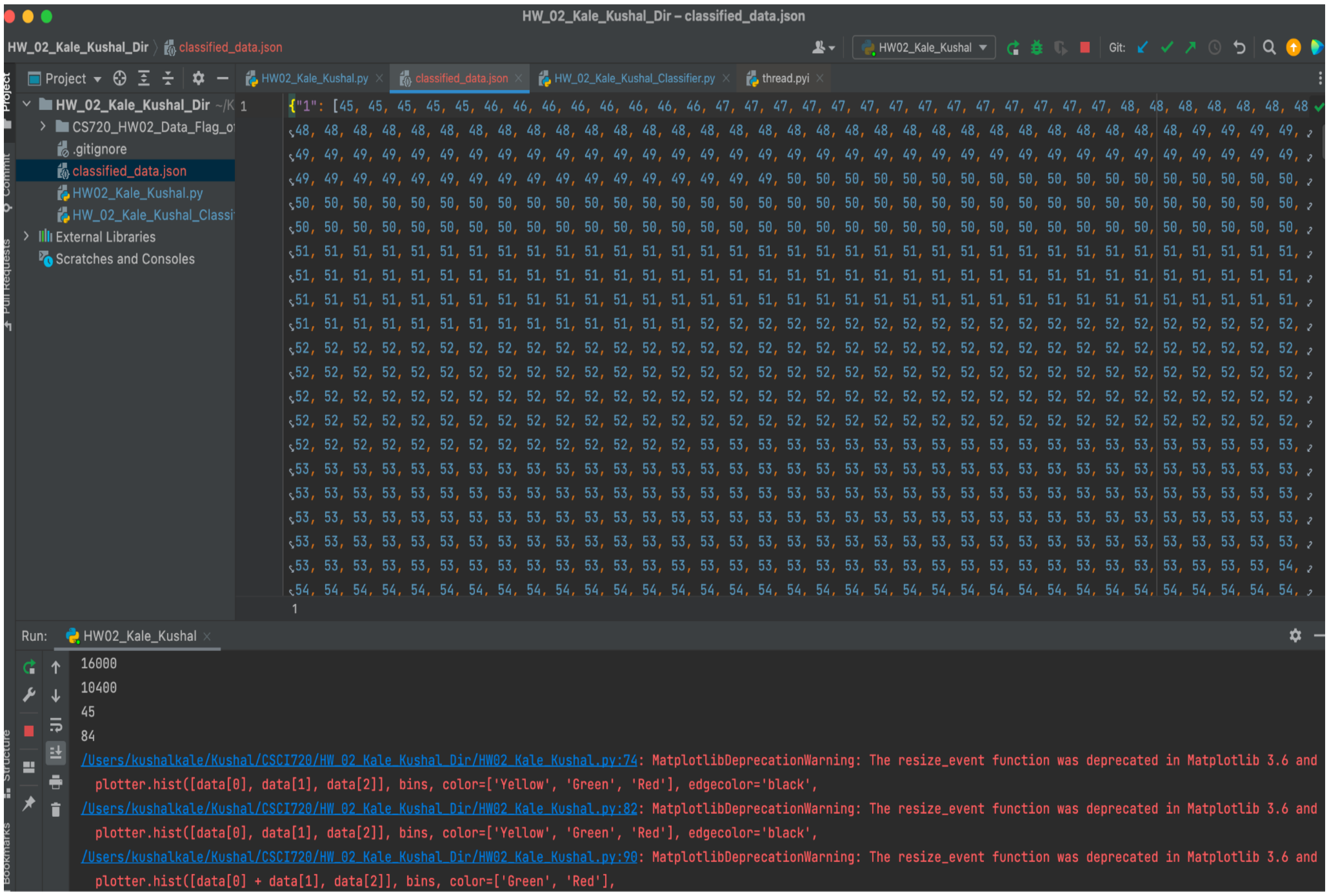
C) Individually, the data read by each thread is a bell curve. The entire data as one looks like a bimodal distribution. The data in the graphs shows that the number of non speeders is way higher than the number of speeders.

D) Threading in python is easy to implement. With the use of ThreadPoolExecutor, Python manages all the necessary threading resources and the developer can focus on the business logic. I tried using a different number of threads and checked the execution time of the file read operation for each number. For me, using 3 threads performed the fastest on average and so I chose to use 3 threads. Using too many threads is never a good idea because it can cause race conditions, thread starvation or deadlocks. Using 32 threads here is not a good idea because the system will spend more time in switching contexts between threads than the overall time saved by using threads.

E) Following is the classifier method and the screenshot of the output JSON file.



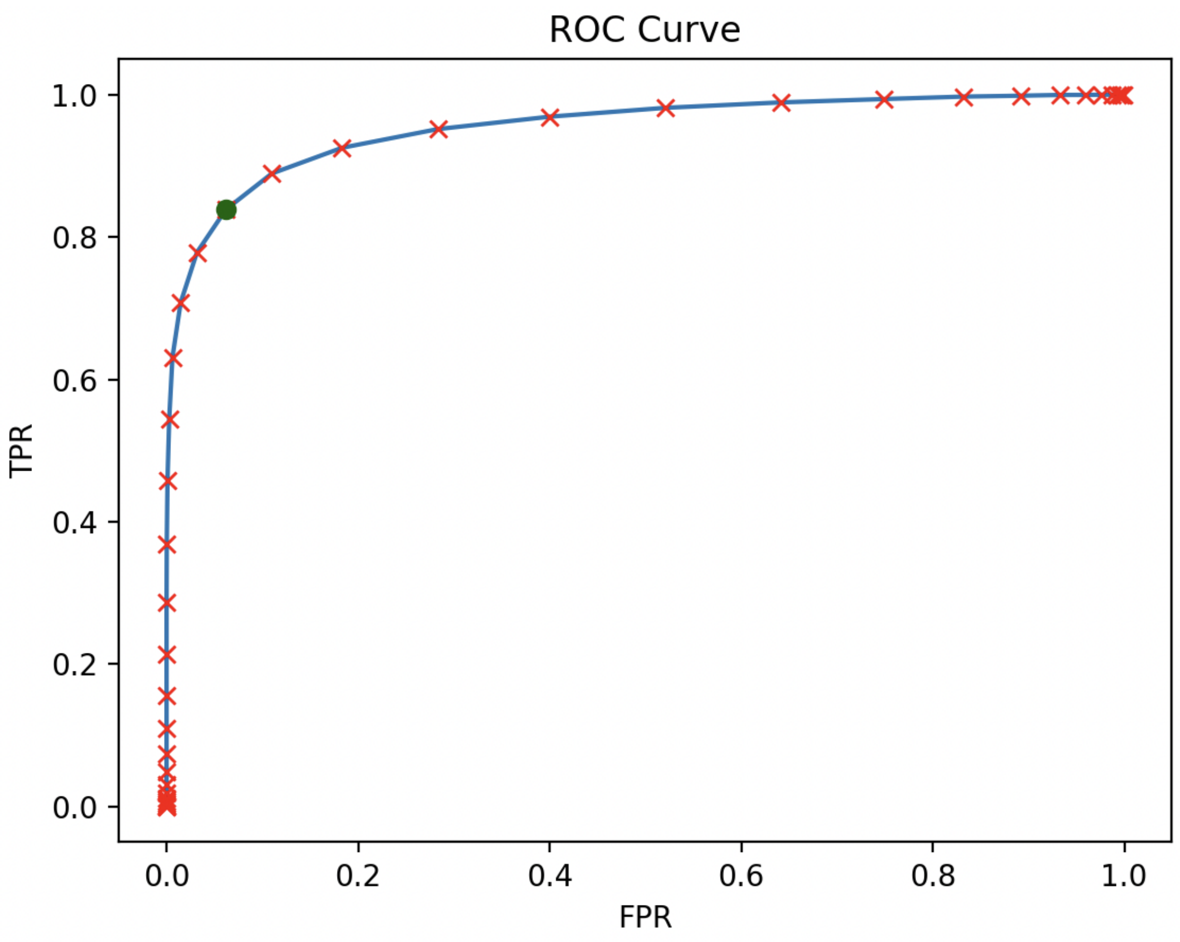
```python
def classify_based_on_threshold(threshold, data):
    data_after_classification = {1: [], 2: []}
    for speed in data:
        if speed < threshold:
            data_after_classification[1].append(speed)
        else:
            data_after_classification[2].append(speed)
    return data_after_classification
```



```
{"1": [45, 45, 45, 45, 45, 46, 46, 46, 46, 46, 46, 46, 46, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 48, 48, 48, 48, 48, 48, 48
48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 49, 49, 49, 49,
49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49,
49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50,
50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50,
50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50,
51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51,
51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51,
51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51,
51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52,
52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52,
52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52,
52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52,
52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52,
52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53,
53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53,
53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53,
53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53,
53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53,
53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 54,
54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54,
```

```
16000
10400
45
84
/Users/kushalkale/Kushal/CSCI720/HW_02_Kale_Kushal_Dir/HW02_Kale_Kushal.py:74: MatplotlibDeprecationWarning: The resize_event function was deprecated in Matplotlib 3.6 and
  plotter.hist([data[0], data[1], data[2]], bins, color=['Yellow', 'Green', 'Red'], edgecolor='black',
/Users/kushalkale/Kushal/CSCI720/HW_02_Kale_Kushal_Dir/HW02_Kale_Kushal.py:82: MatplotlibDeprecationWarning: The resize_event function was deprecated in Matplotlib 3.6 and
  plotter.hist([data[0], data[1], data[2]], bins, color=['Yellow', 'Green', 'Red'], edgecolor='black',
/Users/kushalkale/Kushal/CSCI720/HW_02_Kale_Kushal_Dir/HW02_Kale_Kushal.py:90: MatplotlibDeprecationWarning: The resize_event function was deprecated in Matplotlib 3.6 and
  plotter.hist([data[0] + data[1], data[2]], bins, color=['Green', 'Red'],
```

F)



Conclusion:
I learned about threading. Threading in python is easy to implement. With the use of ThreadPoolExecutor, Python manages all the necessary threading resources and the developer can focus on the business logic. Using too many threads is never a good idea because it can cause race conditions, thread starvation or deadlocks. Using 32 threads here is not a good idea because the system will spend more time in switching contexts between threads than the overall time saved by using threads.

I discovered the new boxplot function in the matplotlib library. We were asked to create a summary of the data read by each thread. I used the boxplot to get a 5 point summary of the data. My program outputs these summaries in a new figure for each thread. The 5 point summary can also give us insights about the data.

The results were all as expected. Each thread reads data which resembles the bell curve.

Python not being my primary language of choice, getting the multithreading to work in Python was particularly challenging for me. After a lot of errors I managed to read the data accurately.