# Notebook

March 4, 2020

### 0.0.1 Question 1d

Print a summary of the data selection and cleaning you performed. **Your Python code should not include any number literals, but instead should refer to the shape of `all_taxi`, `clean_taxi`, and `manhattan_taxi`.**

E.g., you should print something like: "Of the original 1000 trips, 21 anomalous trips (2.1%) were removed through data cleaning, and then the 600 trips within Manhattan were selected for further analysis."

(Note that the numbers in the example above are not accurate.)

One way to do this is with Python's f-strings. For instance,

```
name = "Joshua"
print(f"Hi {name}, how are you?")
```

prints out `Hi Joshua, how are you?`.

**Please ensure that your Python code does not contain any very long lines, or we can't grade it.**

*Your response will be scored based on whether you generate an accurate description and do not include any number literals in your Python expression, but instead refer to the dataframes you have created.*

```
In [12]: print(all_taxi.shape)
         print(clean_taxi.shape)
         print(manhattan_taxi.shape)

         clean_taxi.head()

         allTaxiCount = all_taxi.shape
         allTaxiCount = np.array(allTaxiCount).tolist()

         cleanTaxiCount = clean_taxi.shape
         cleanTaxiCount = np.array(cleanTaxiCount).tolist()

         manhattan_taxiCount= manhattan_taxi.shape
         manhattan_taxiCount = np.array(manhattan_taxiCount).tolist()

         print("Of the original ",allTaxiCount[0], " trips, ", allTaxiCount[0] -cleanTaxiCount[0], " an
         ans = (allTaxiCount[0] -cleanTaxiCount[0])/allTaxiCount[0]
         ans = ans * 100

         print("(", round(ans,2),"% ) of were removed through data cleaning, and then ", end="")
         print(manhattan_taxiCount[0], " of the data manhatta's data are selected for further analysis"
```

```
(97692, 9)
(96445, 9)
(82800, 9)
Of the original  97692  trips,  1247  anomalous trips ( 1.28 % ) of were removed through data cleaning,
```

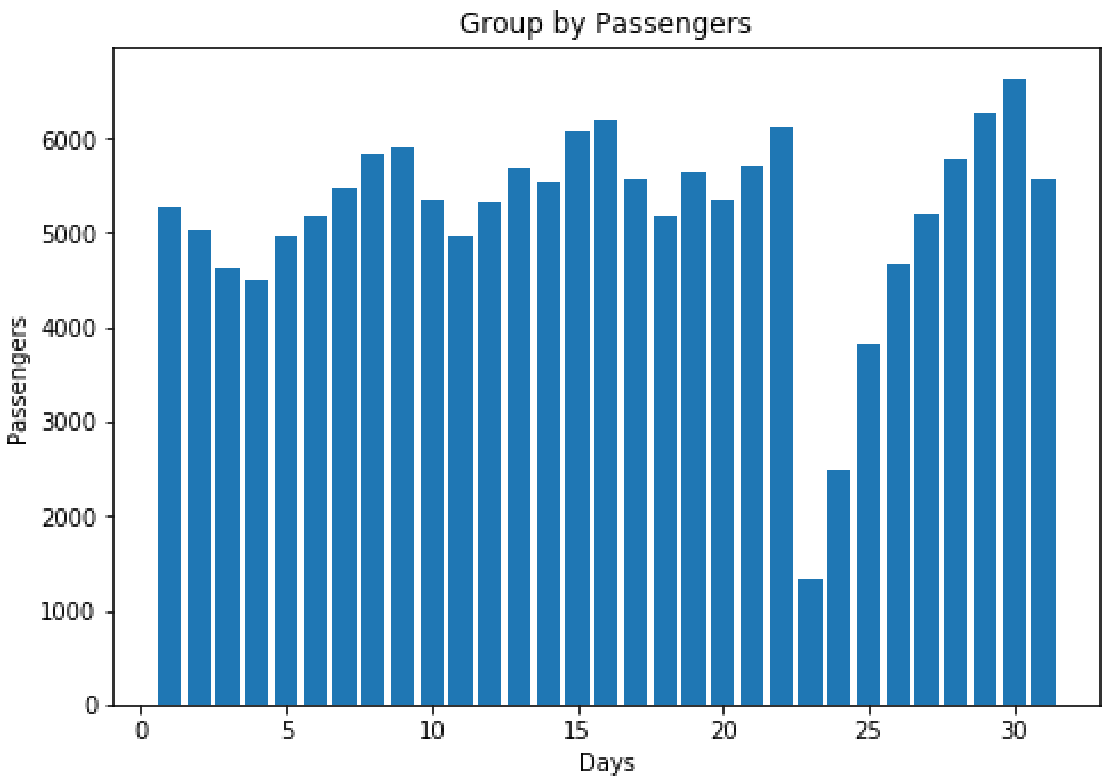
### 0.0.2 Question 2b

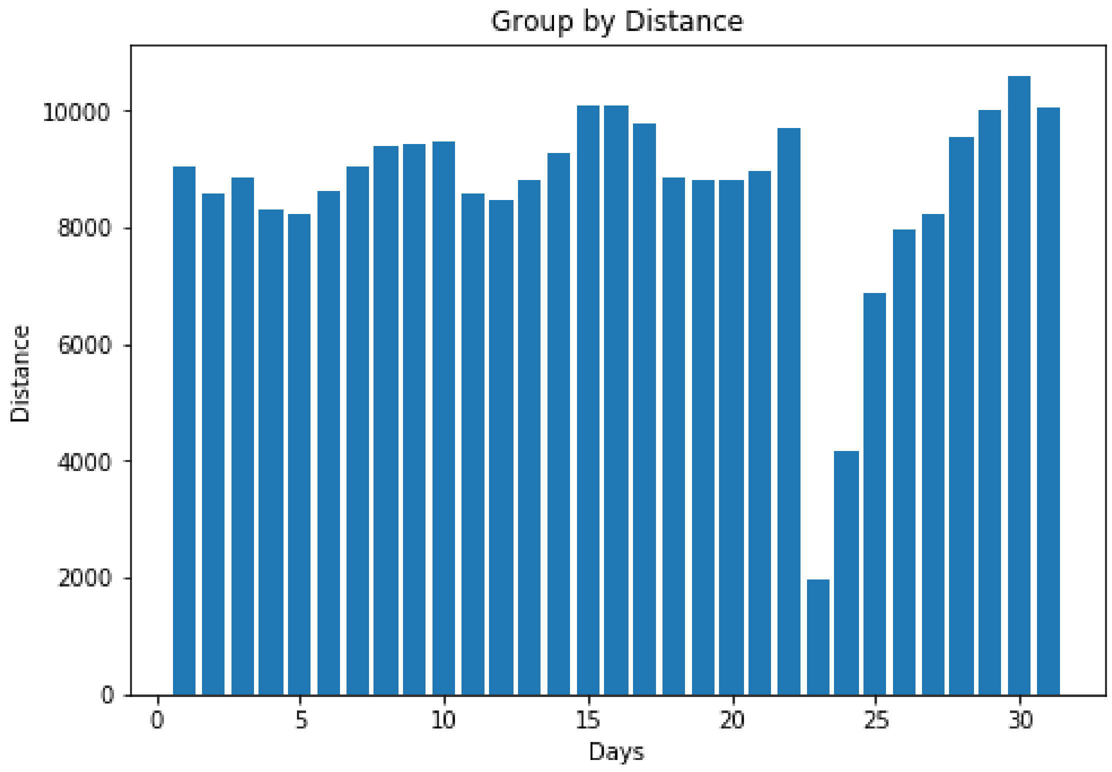
Create a data visualization that allows you to identify which dates were affected by the historic blizzard of January 2016. Make sure that the visualization type is appropriate for the visualized data.

As a hint, consider how taxi usage might change on a day with a blizzard. How could you visualize/plot this?

```
In [35]: import numpy as np
         import matplotlib.pyplot as plt

         visualization = clean_taxi.copy(deep="True")

         visualization["date"]=visualization.pickup_datetime
         visualization["date"]=pd.to_datetime(visualization.date, format='%Y-%m-%d %H:%M:%S')
         visualization= visualization.drop(['pickup_datetime','dropoff_datetime'], axis=1)
         # strftime('%m%d')
         visualization["date"] = visualization["date"].dt.day

         visualization = visualization.groupby(['date']).sum()

         fig = plt.figure()
         ax = fig.add_axes(([0,0,1,1]))
         x =  list(range(32))
         x.pop(0)
         y =  visualization["passengers"]
         y = y.values.tolist()

         # y = pd.DataFrame(y)
         # x = pd.DataFrame(x)
         # print(x.shape)
         # print (y.shape)#
         ax.set_title ('Group by Passengers')
         ax.set_ylabel('Passengers')
         ax.set_xlabel('Days')
         ax.bar(x,y)
         plt.show()



         visualization = clean_taxi.copy(deep="True")

         visualization["date"]=visualization.pickup_datetime
         visualization["date"]=pd.to_datetime(visualization.date, format='%Y-%m-%d %H:%M:%S')
         visualization= visualization.drop(['pickup_datetime','dropoff_datetime'], axis=1)
         # strftime('%m%d')
         visualization["date"] = visualization["date"].dt.day

         visualization = visualization.groupby(['date']).sum()

         fig = plt.figure()
         ax = fig.add_axes(([0,0,1,1]))
         x =  list(range(32))
         x.pop(0)

         y =  visualization["distance"]
```

```
y = y.values.tolist()

ax.set_title ('Group by Distance')
ax.set_ylabel('Distance')
ax.set_xlabel('Days')
ax.bar(x,y)
plt.show()
```

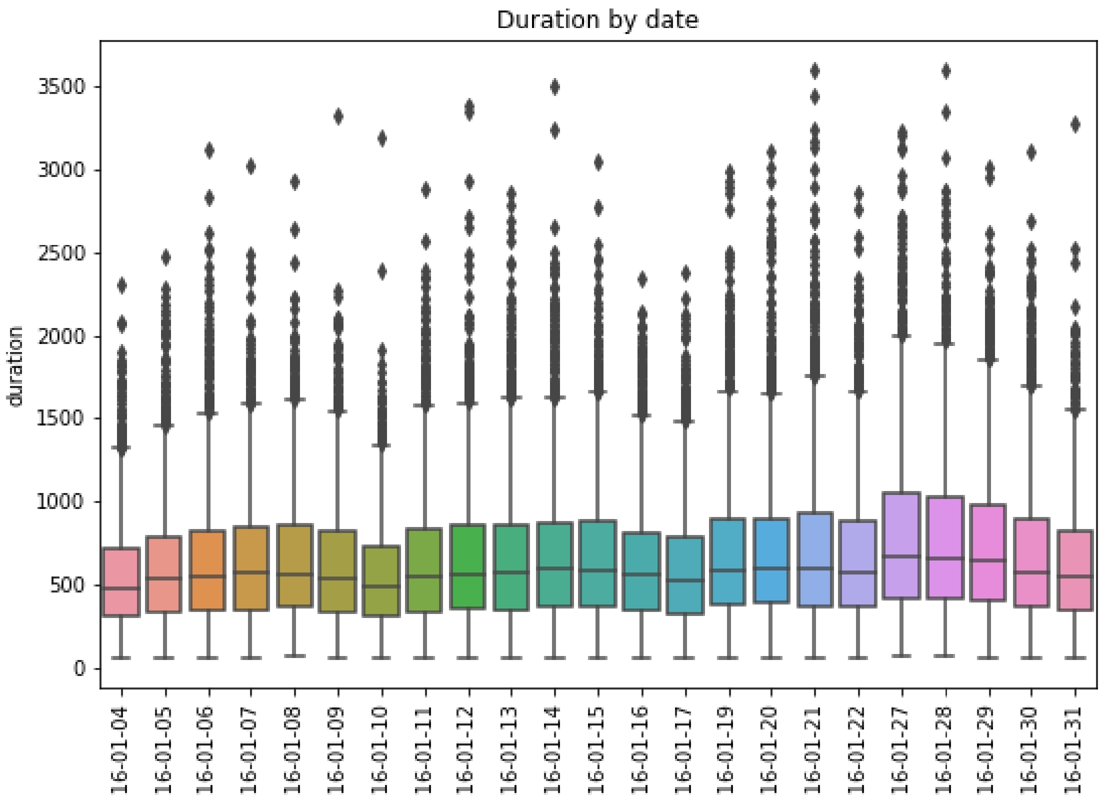



Group by Distance
Distance
Days
0
2000
4000
6000
8000
10000
0
5
10
15
20
25
30

### 0.0.3 Question 3a

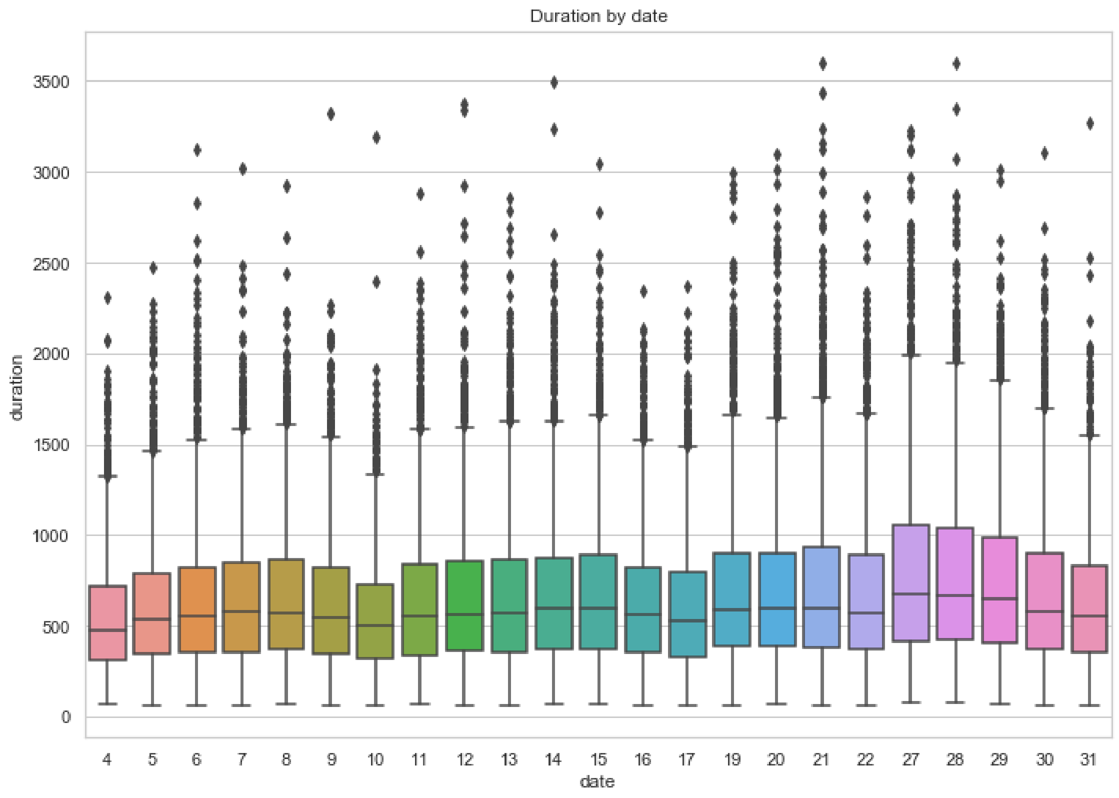
Create a box plot that compares the distributions of taxi trip durations for each day **using `train` only**. Individual dates shoud appear on the horizontal axis, and duration values should appear on the vertical axis. Your plot should look like the one below.

You can generate this type of plot using `sns.boxplot`



```
In [39]: import seaborn as sns

         box_dia = train.copy(deep="True")
         sns.set(style="whitegrid")
         box_dia["date"]=box_dia.pickup_datetime


         box_dia["date"]=pd.to_datetime(box_dia.date, format='%Y-%m-%d %H:%M:%S')
         box_dia= box_dia.drop(['pickup_datetime','dropoff_datetime'], axis=1)

         box_dia["date"] = box_dia["date"].dt.day




         sns.set(style="whitegrid")

         fig, ax = plt.subplots()
         fig.set_size_inches(11.7, 8.27)
         sns.boxplot(x=box_dia["date"], y=box_dia["duration"], ax= ax)
         ax.set_title("Duration by date")
```

```
plt.show()
```


Duration by date
duration
0
500
1000
1500
2000
2500
3000
3500
date
4
5
6
7
8
9
10
11
12
13
14
15
16
17
19
20
21
22
27
28
29
30
31

### 0.0.4 Question 3b

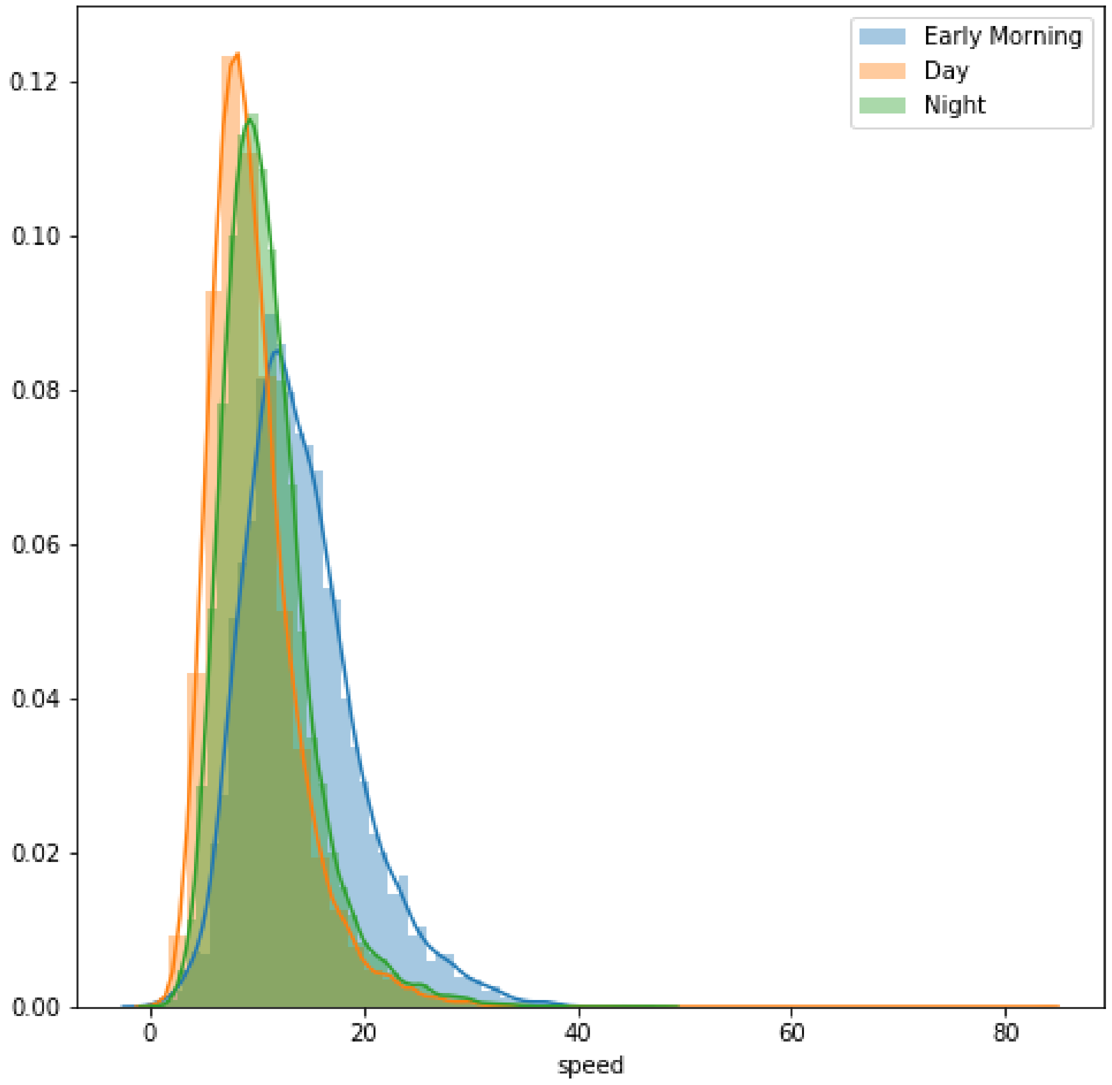
In one or two sentences, describe the assocation between the day of the week and the duration of a taxi trip. Your answer should be supported by your boxplot above.

*Note*: The end of Part 2 showed a calendar for these dates and their corresponding days of the week.

The median of duration of taxi trip is lower in weekends than in weekdays.

### 0.0.5 Question 3c

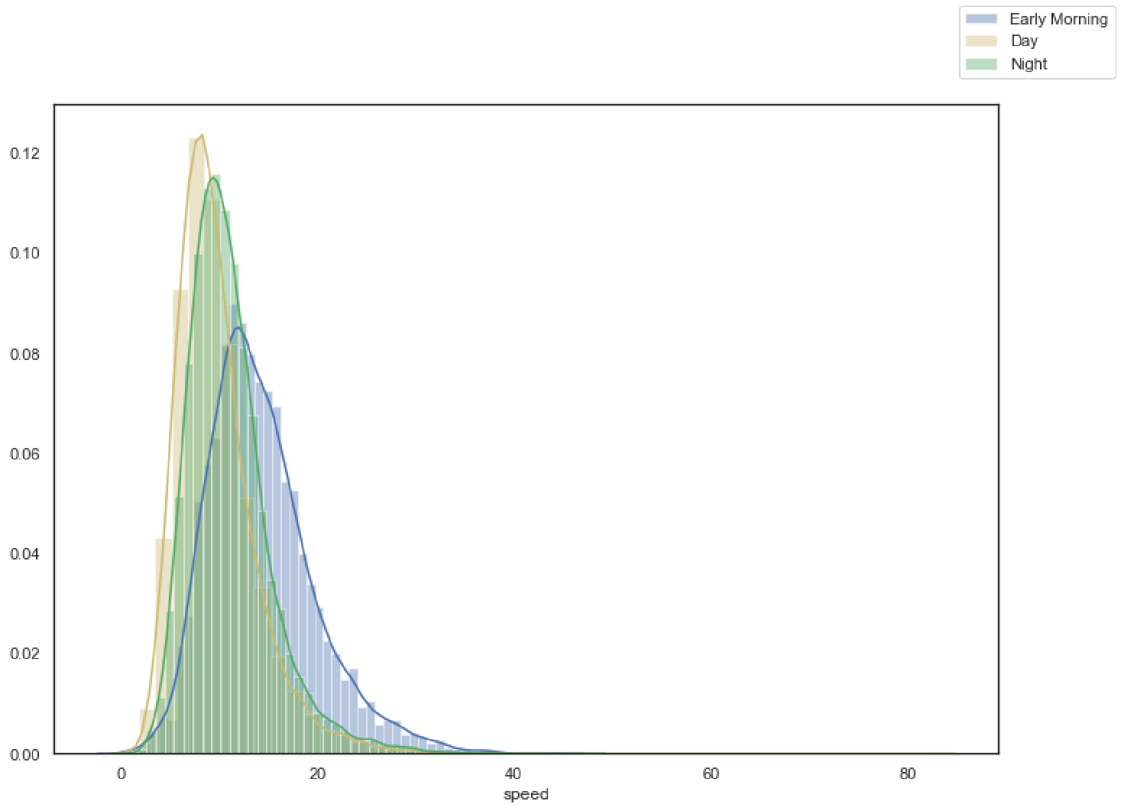
Use `sns.distplot` to create an overlaid histogram comparing the distribution of average speeds for taxi rides that start in the early morning (12am-6am), day (6am-6pm; 12 hours), and night (6pm-12am; 6 hours). Your plot should look like this:



```
In [41]: sns.set(style="white")

         fig, ax = plt.subplots()

         fig.set_size_inches(11.7, 8.27)

         Morning= train.loc[train["period"] == 1]
         Day= train.loc[train["period"] == 2]
         Night= train.loc[train["period"] == 3]

         sns.distplot(Morning["speed"],color = 'b', label = 'Early Morning', ax=ax)
         sns.distplot(Day["speed"],color = 'y', label = 'Day', ax=ax)
         sns.distplot(Night["speed"],color = 'g', label = 'Night', ax=ax)
```

```
fig.legend(labels=['Early Morning', 'Day', 'Night'])
```

```
plt.show()
```



### 0.0.6 Question 4e

In one or two sentences, explain how the `period` regression model above could possibly outperform linear regression when the design matrix for linear regression already includes one feature for each possible hour, which can be combined linearly to determine the `period` value.

The "period" feature are distinguished by labeling 1,2,3, which influences the parameter because 1,2,3, are just labels. But this method treat 1,2,3 fairly

### 0.0.7 Question 5b

Print a summary of your model's performance. You **must** include the RMSE on the train and test sets. Do not hardcode any values or you won't receive credit.

Don't include any long lines or we won't be able to grade your response.

```
In [79]: print("Test RMSE = " ,test_rmse)
         print("Train RMSE = " ,train_rmse)
```

```
Test RMSE =  203.15505756666695
Train RMSE =  189.96678277803375
```

### 0.0.8 Question 5c

Describe why you selected the model you did and what you did to try and improve performance over the models in section 4.

Responses should be at most a few sentences

I use Neural Network. Becasue I think with the big data set and the useful features, we can just let neural network to figure out the duration. More importantly, Neural network has a better mechanism to fit and sift through patterns in the historical data( which is our data set). Neural networks can also deal with nonlinearities and since the data will have nonlinear dependencies, neural networks perform better than linear regerssion.

The first model is train the data and get the degradation in the validation error after about 100 epochs. The second model.fit call to automatically stop training when the validation score doesn't improve. I used an EarlyStopping callback that tests a training condition for every epoch. If a set amount of epochs elapses without showing improvement, then automatically stop the training.