Linear Regression Notebook using Traffic Volume

Notebook adapted from linear regression notebook from the Python Data Science Handbook
Modified by: Gábor Major
Last Modified date: 2024-11-12
Description:
This notebook takes in the Dublin traffic data, does some data processing by summing up all cameras at an intersection, and uses an 8-th degree polynomial to predict the volume of traffic at different hours of the day for each junction.

To use preexisting models jump down to the 4-th cell from the bottom.

```
# Imports
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-v0_8-whitegrid')
import numpy as np
import pandas as pd

# Create n-th degree polynomial model
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression
from sklearn.metrics import root_mean_squared_error
import pickle
poly_model = make_pipeline(PolynomialFeatures(8),
                           LinearRegression())
```

Import raw data

To use already cleaned data skip down 6 cells
Data from: data.gov.ie
All of the **Detector** row values for **Sum_Volume** and **Avg_Volume** are summed up to get only 1 per **Site** per **End_Time**.

```
traffic_data_list = pd.read_csv('data/SCATSFebruary2023.csv', sep=',',
header=0, usecols=[0, 1, 2, 4, 5])
print(traffic_data_list)
print(type(traffic_data_list))

# Function for processing raw file
def sum_site_number_vaues(site_number, volume_list):
    specific_site_data = volume_data.loc[volume_data['Site'] ==
site_number]
```

```
    start_row_index = 0
    previous_time = specific_site_data.iloc[0, 0]
    region_code = specific_site_data.iloc[0, 1]
    row_index = 0

    for _, row in specific_site_data.iterrows():
        if row['End_Time'] != previous_time:
            volume_list.append([
                int(previous_time),
                region_code,
                int(site_number),
                int(specific_site_data.iloc[start_row_index:row_index,
3:4].sum().iloc[0]),
                int(specific_site_data.iloc[start_row_index:row_index,
4:5].sum().iloc[0])
            ])

            start_row_index = row_index
            previous_time = row['End_Time']

        row_index += 1

from multiprocessing import Process, Manager
# Sum up data for each camera at each site
summed_traffic_volume_list = Manager().list()

site_numbers = traffic_data_list['Site'].unique()
process_list = []
counter = 0

for number in site_numbers:
    process = Process(target=sum_site_number_vaues,
args=(number,summed_traffic_volume_list,))
    process_list.append(process)
    process.start()
    counter += 1
    if counter % 100 == 0:
        print(counter)

for process in process_list:
    process.join()

print(len(summed_traffic_volume_list))

# Save cleaned file
import csv
columns_names = ['End_Time', 'Region', 'Site', 'Sum_Volume',
'Avg_Volume']
with open('data/summed_data.csv', 'w') as f:
    writer = csv.writer(f)
```

```
    writer.writerow(columns_names)
    writer.writerows(summed_traffic_volume_list)
```

Import cleaned data

The **End_Time** data is converted into **End_Day** and **End_Hour**, and the whole data set is then sorted according to the **End_HOUR**.

```
cleaned_data = pd.read_csv('data/summed_data.csv', sep=',', header=0)
print(cleaned_data)
```

```
              End_Time Region  Site  Sum_Volume  Avg_Volume
0       20230228060000  CCITY   782           0           0
1       20230228050000  CCITY   782          90           7
2       20230228040000  CCITY   782         194          15
3       20230228030000  CCITY   782         121           9
4       20230228060000  CCITY   796         266          18
...                ...    ...   ...         ...         ...
611846  20230228110000    IRE  6381          86           4
611847  20230228100000    IRE  6381         105           5
611848  20230228090000    IRE  6381         133           8
611849  20230228080000    IRE  6381          74           3
611850  20230228070000    IRE  6381          65           3

[611851 rows x 5 columns]
```

```
# Convert End_Time to days and hours
all_times = cleaned_data['End_Time']
days = []
hours = []

for time in all_times:
    time = str(time)
    # year = time[:4]
    # month = time[4:6]
    days.append(time[6:8])
    hours.append(time[8:10])

cleaned_data['End_Day'] = days
cleaned_data['End_Hour'] = hours

print(cleaned_data)
```

```
              End_Time Region  Site  Sum_Volume  Avg_Volume End_Day
End_Hour
0       20230228060000  CCITY   782           0           0      28
06
1       20230228050000  CCITY   782          90           7      28
05
2       20230228040000  CCITY   782         194          15      28
```

```
04
3       20230228030000  CCITY   782          121            9       28
03
4       20230228060000  CCITY   796          266           18       28
06
...                ...    ...   ...          ...          ...      ...
...
611846  20230228110000    IRE  6381           86            4       28
11
611847  20230228100000    IRE  6381          105            5       28
10
611848  20230228090000    IRE  6381          133            8       28
09
611849  20230228080000    IRE  6381           74            3       28
08
611850  20230228070000    IRE  6381           65            3       28
07

[611851 rows x 7 columns]
```

```
# Sort data
cleaned_data_sorted = cleaned_data.sort_values('End_Hour')
```

Get specific site

To process all sites jump down 13 cells.
A specific **Site** data is cleaned removing top and bottom 5% of data for each hour.

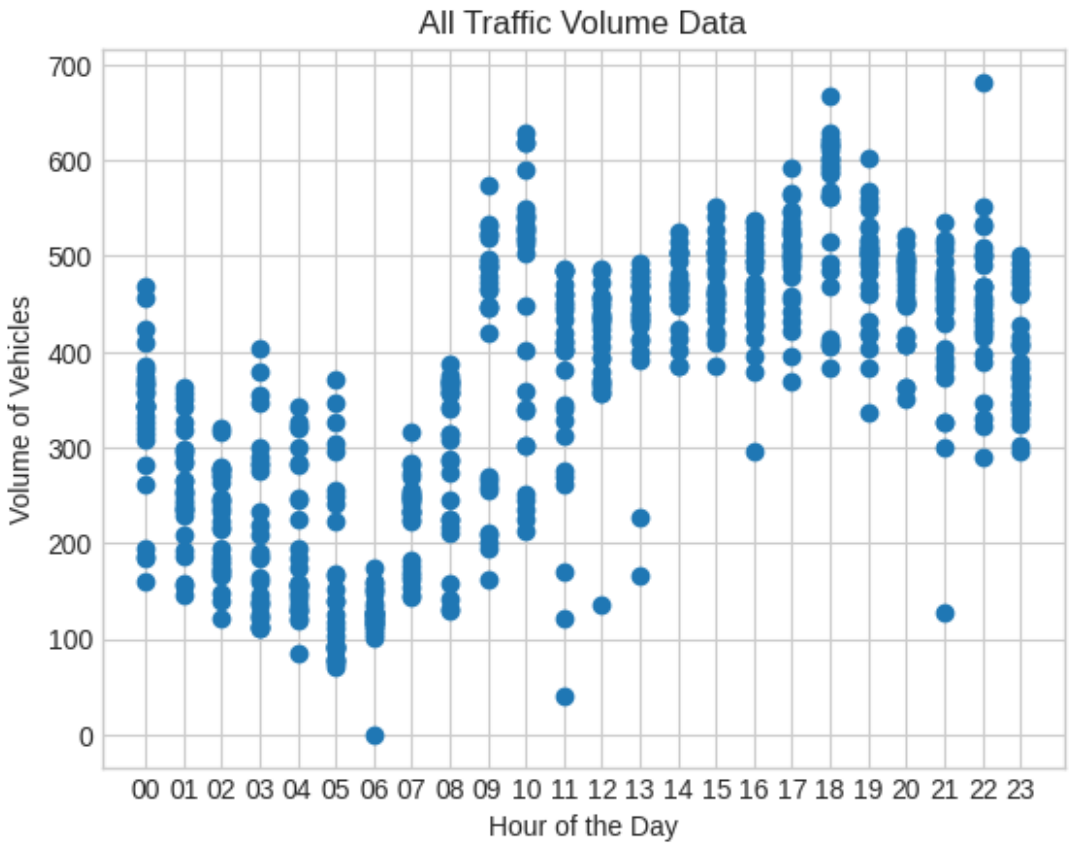
```
use_site = 782
site_data = cleaned_data_sorted.loc[cleaned_data_sorted['Site'] ==
use_site]
print(site_data)
```

```
               End_Time Region  Site  Sum_Volume  Avg_Volume End_Day
End_Hour
90249   20230221000000  CCITY   782         333          25      21
00
139365  20230227000000  CCITY   782         309          23      27
00
50943   20230215000000  CCITY   782         373          29      15
00
147914  20230228000000  CCITY   782         315          24      28
00
83248   20230220000000  CCITY   782         326          25      20
00
...                ...    ...   ...         ...         ...     ...
...
125902  20230224230000  CCITY   782         462          36      24
23
```

```
31594   20230211230000  CCITY   782         407           31      11
23
139318  20230226230000  CCITY   782         331           25      26
23
21692   20230209230000  CCITY   782         415           32      09
23
26618   20230210230000  CCITY   782         472           36      10
23

[672 rows x 7 columns]
```

```
# Use End_Hour and Sum_Volume for calculating
x_data = site_data['End_Hour']
y_data = site_data['Sum_Volume']

# Show original data
x_train = x_data.to_numpy()
y_train = y_data.to_numpy()
array_size = len(y_train)
print(array_size)

# Choose 10% of the data as testing set
random_indexes = np.random.choice(array_size, array_size // 10, False)
x_test = np.take(x_train, random_indexes)
y_test = np.take(y_train, random_indexes)

# Graph training data
plt.title("All Traffic Volume Data")
plt.xlabel("Hour of the Day")
plt.ylabel("Volume of Vehicles")
plt.xticks(range(0, 24))
plt.scatter(x_train, y_train)
```

```
672

<matplotlib.collections.PathCollection at 0x7f0231c3c3b0>
```



Training

An 8-th degree Polynomial model is created and trained on the **Site** data, using the **End_Hour** and the **Sum_Volume**.

```
# Fit data
poly_model.fit(x_train[:, np.newaxis], y_train)
```

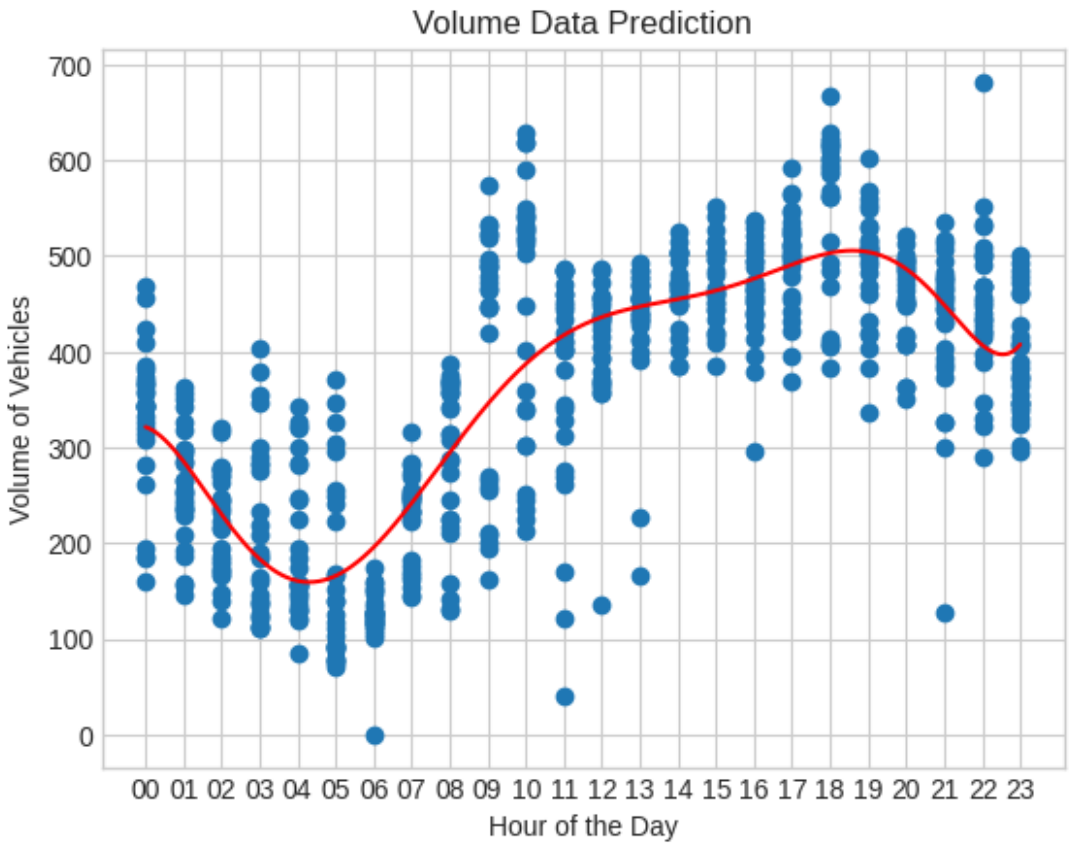
```
Pipeline(steps=[('polynomialfeatures', PolynomialFeatures(degree=8)),
                ('linearregression', LinearRegression())])
```

```
# Use testing set
y_prediction = poly_model.predict(x_test[:, np.newaxis])

# Calculate accuracy using root mean squared error
accuracy = root_mean_squared_error(y_test, y_prediction)
print("Root mean squared error:", accuracy)
```

```
Root mean squared error: 80.54840222656898
```

```
# Create line data
xfit = np.linspace(0, 23, 1000)
yfit = poly_model.predict(xfit[:, np.newaxis])
```

```
# Plot data
plt.title("Volume Data Prediction")
plt.xlabel("Hour of the Day")
plt.ylabel("Volume of Vehicles")
plt.xticks(range(0, 24))
plt.scatter(x_train, y_train)
plt.plot(xfit, yfit, color="red");
```



Clean Up Data

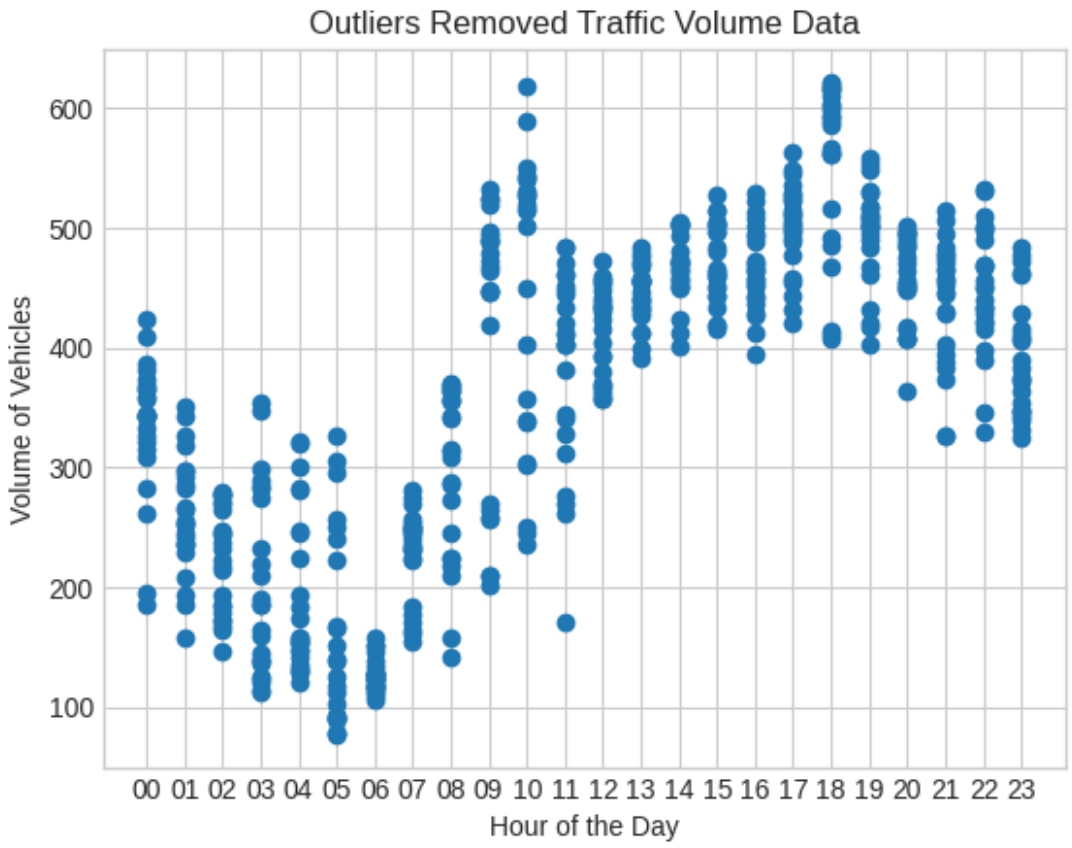
To increase the accuracy of the model and decrease error the outliers of the data are removed

```
# Remove top and bottom 5% of data
def remove_outliers(data_in):
    removal_amount = 5
    upper_threshold = np.percentile(data_in['Sum_Volume'], 100 -
removal_amount)
    lower_threshold = np.percentile(data_in['Sum_Volume'],
removal_amount)
    return data_in.loc[data_in['Sum_Volume'] <=
upper_threshold].loc[data_in['Sum_Volume'] >= lower_threshold]
```

```
# Remove the outliers for each hour
hours_data = []
for hour in range(24):
    if hour < 10:
        hour = '0' + str(hour)
    else:
        hour = str(hour)
    hour_data = site_data.loc[site_data['End_Hour'] == hour]
    if hour_data.empty:
        print('Not enough data!')
        break
    hours_data.append(remove_outliers(hour_data))

if len(hours_data) == 24:
    cleaned_site_data = pd.concat(hours_data)

# Use End_Hour and Sum_Volume for calculating
x_data = cleaned_site_data['End_Hour']
y_data = cleaned_site_data['Sum_Volume']

# Show cleaned data
x_train = x_data.to_numpy()
y_train = y_data.to_numpy()
array_size = len(y_train)
print(array_size)

# Choose 10% of the data as testing set
random_indexes = np.random.choice(array_size, array_size // 10, False)
x_test = np.take(x_train, random_indexes)
y_test = np.take(y_train, random_indexes)

# Graph training data
plt.title("Outliers Removed Traffic Volume Data")
plt.xlabel("Hour of the Day")
plt.ylabel("Volume of Vehicles")
plt.xticks(range(0, 24))
plt.scatter(x_train, y_train)
```

```
577
```

```
<matplotlib.collections.PathCollection at 0x7f01f12b0a10>
```



Training

An 8-th degree Polynomial model is trained on the cleaned **Site** data, using the **End_Hour** and the **Sum_Volume**, which is then saved to disk.

```
# Fit data
poly_model.fit(x_train[:, np.newaxis], y_train)
```

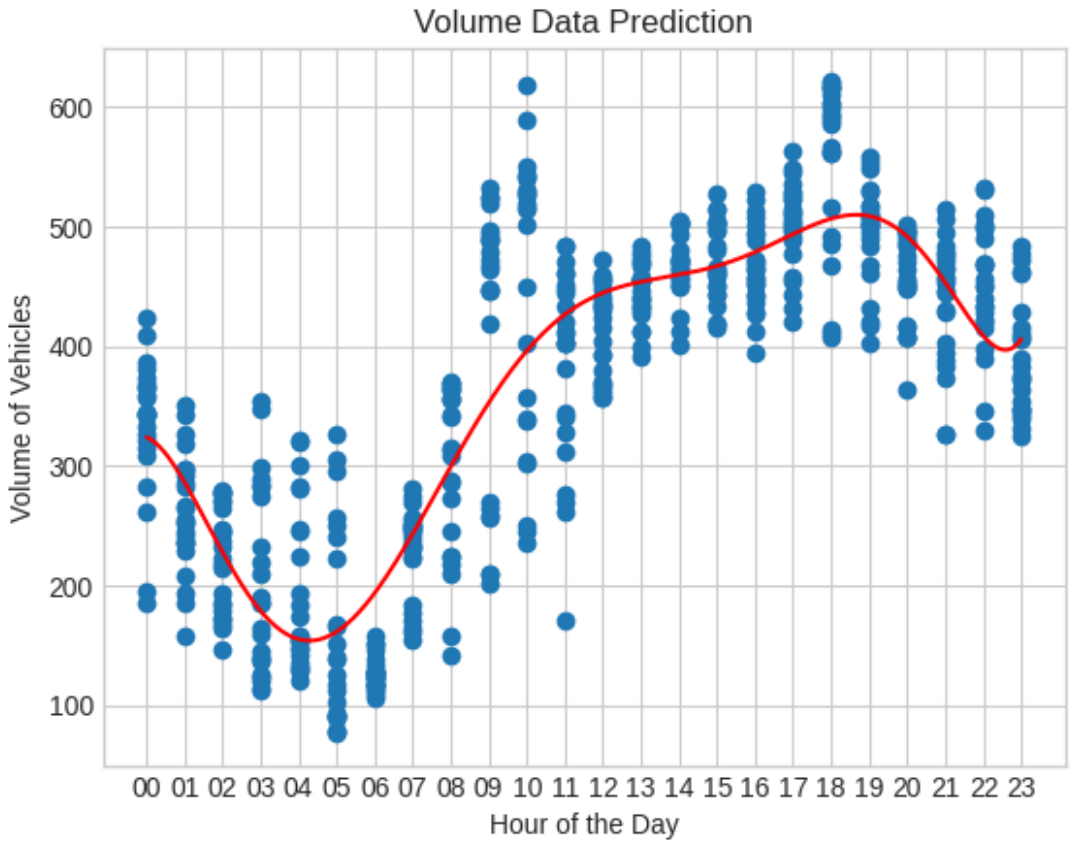
```
Pipeline(steps=[('polynomialfeatures', PolynomialFeatures(degree=8)),
                ('linearregression', LinearRegression())])
```

```
# Use testing set
y_prediction = poly_model.predict(x_test[:, np.newaxis])

# Calculate accuracy using root mean squared error
accuracy = root_mean_squared_error(y_test, y_prediction)
print("Root mean squared error:", accuracy)
```

```
Root mean squared error: 64.97335694061661
```

```
# Save model to disk
with open(f'models/site_{use_site}_model.pkl', 'wb') as f:
    pickle.dump(poly_model,f)
```

```
# Load in model
with open(f'models/site_{use_site}_model.pkl', 'rb') as f:
    loaded_model = pickle.load(f)

# Create line data
xfit = np.linspace(0, 23, 1000)
yfit = loaded_model.predict(xfit[:, np.newaxis])

# Plot data
plt.title("Volume Data Prediction")
plt.xlabel("Hour of the Day")
plt.ylabel("Volume of Vehicles")
plt.xticks(range(0, 24))
plt.scatter(x_train, y_train)
plt.plot(xfit, yfit, color="red");
```



Create and save a model for all sites

The code can then be ran for all **Sites** which have at least one data point for each hour, and the models are then saved.

```
site_numbers = cleaned_data_sorted['Site'].unique()
for site in site_numbers:
```

```
    site_data = cleaned_data_sorted.loc[cleaned_data_sorted['Site'] ==
site]
    # Remove the outliers for each hour
    hours_data = []
    for hour in range(24):
        if hour < 10:
            hour = '0' + str(hour)
        else:
            hour = str(hour)
        hour_data = site_data.loc[site_data['End_Hour'] == hour]
        if hour_data.empty:
            print('Not enough data! - ' + str(site))
            break
        hours_data.append(remove_outliers(hour_data))

    if len(hours_data) != 24:
        continue

    hours_data = pd.concat(hours_data)

    # Use End_Hour and Sum_Volume for calculating
    x_data = hours_data['End_Hour']
    y_data = hours_data['Sum_Volume']
    # Show cleaned data
    x = x_data.to_numpy()
    y = y_data.to_numpy()

    # Fit data
    poly_model.fit(x[:, np.newaxis], y)

    # Save model to disk
    with open(f'models/site_{site}_model.pkl','wb') as f:
        pickle.dump(poly_model,f)
```

Load in model from disk and make predictions

Specified site model is loaded in, and time to predict traffic volume is also taken in. The model than predicts a value and shows the result.

```
# Take site number input
use_site = input("Which site number to predict: ")
```

```
Which site number to predict:  408
```

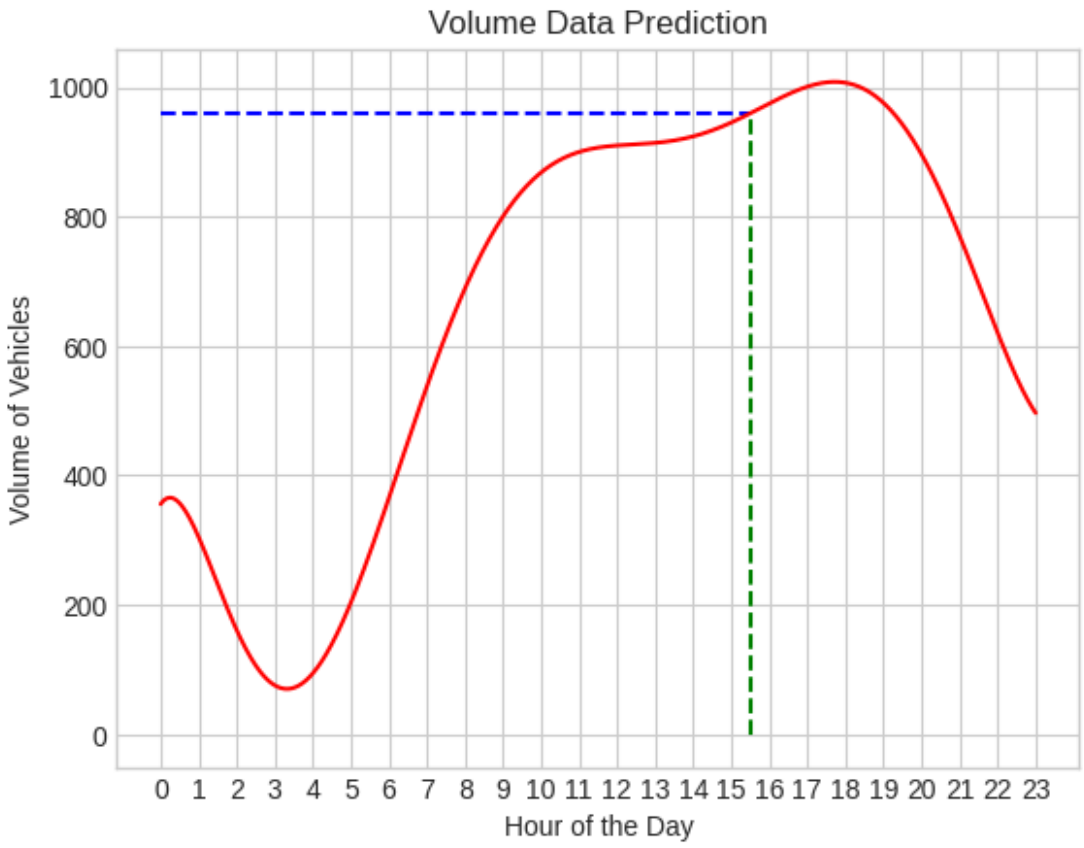
```
# Load in model
with open(f'models/site_{use_site}_model.pkl', 'rb') as f:
    loaded_model = pickle.load(f)

# Take in time to predict, and onvert to decimal
time_to_predict_input = input("Input as 24-hour, example: 13:40.\nWhat
```

```
time to predict traffic volume:")
time_to_predict = time_to_predict_input.split(":")
time_to_predict = int(time_to_predict[0]) + int(time_to_predict[1]) /
60
```

```
Input as 24-hour, example: 13:40.
What time to predict traffic volume: 15:30
```

```
# Create prediction
predicted_traffic_volume =
loaded_model.predict(np.array([time_to_predict]).reshape(1, 1))

print(f"Site {use_site} prediction at time {time_to_predict_input} is:
{round(predicted_traffic_volume[0])}")

plt.vlines(time_to_predict, 0, predicted_traffic_volume, 'green',
'dashed')
plt.hlines(predicted_traffic_volume, 0, time_to_predict, 'blue',
'dashed')

# Create test set
xfit = np.linspace(0, 23, 1000)
yfit = loaded_model.predict(xfit[:, np.newaxis])

# Plot data
plt.title("Volume Data Prediction")
plt.xlabel("Hour of the Day")
plt.ylabel("Volume of Vehicles")
plt.xticks(range(0, 24))
plt.plot(xfit, yfit, color="red");
```

```
Site 408 prediction at time 15:30 is: 960
```


Volume Data Prediction
Volume of Vehicles
1000
800
600
400
200
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Hour of the Day