Predicting Metro Interstate Traffic Volume: Random Forest Regressor Method

KJ MoChroi

Department of Data Science, Bellevue University

DSC680: Applied Data Science

Dr. Brett Werner

Spring 2023

Change Control Log:

Change## : 1

Change(s) Made: Found and imported dataset, visualizations

Date of Change: 5/1/2023

Change## : 2

Change(s) Made: ttrained model

Date of Change: 5/2/2023

https://archive.ics.uci.edu/ml/datasets/Metro+Interstate+Traffic+Volume

In [1]:
```
# Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from numpy import sqrt
```

In [2]:
```
df = pd.read_csv("Potential_Datasets/Metro_Interstate_Traffic_Volume.csv.gz", compressi
df.head()
```

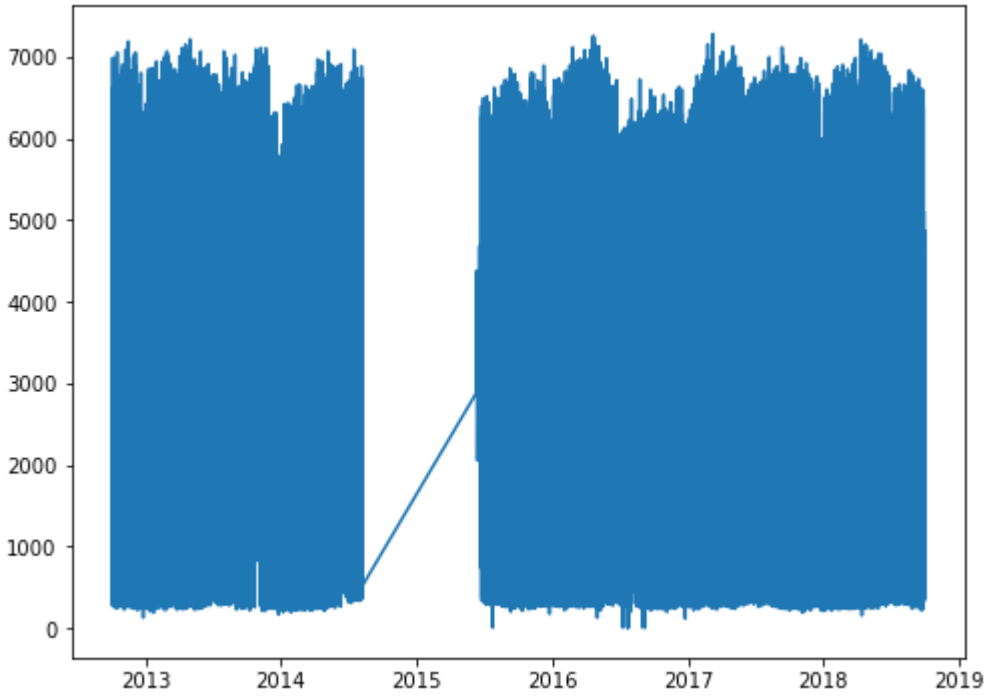
Out[2]:

	holiday	temp	rain_1h	snow_1h	clouds_all	weather_main	weather_description	date_time	traffic_v
0	None	288.28	0.0	0.0	40	Clouds	scattered clouds	2012-10-02 09:00:00	
1	None	289.36	0.0	0.0	75	Clouds	broken clouds	2012-10-02 10:00:00	
2	None	289.58	0.0	0.0	90	Clouds	overcast clouds	2012-10-02 11:00:00	
3	None	290.13	0.0	0.0	90	Clouds	overcast clouds	2012-10-02 12:00:00	
4	None	291.14	0.0	0.0	75	Clouds	broken clouds	2012-10-02	

holiday	temp	rain_1h	snow_1h	clouds_all	weather_main	weather_description	date_time	traffic_v
							13:00:00	

In [3]:
```
# Let's make sure 'date' is actually a date in pandas
df["date_time"] = pd.to_datetime(df["date_time"])
```

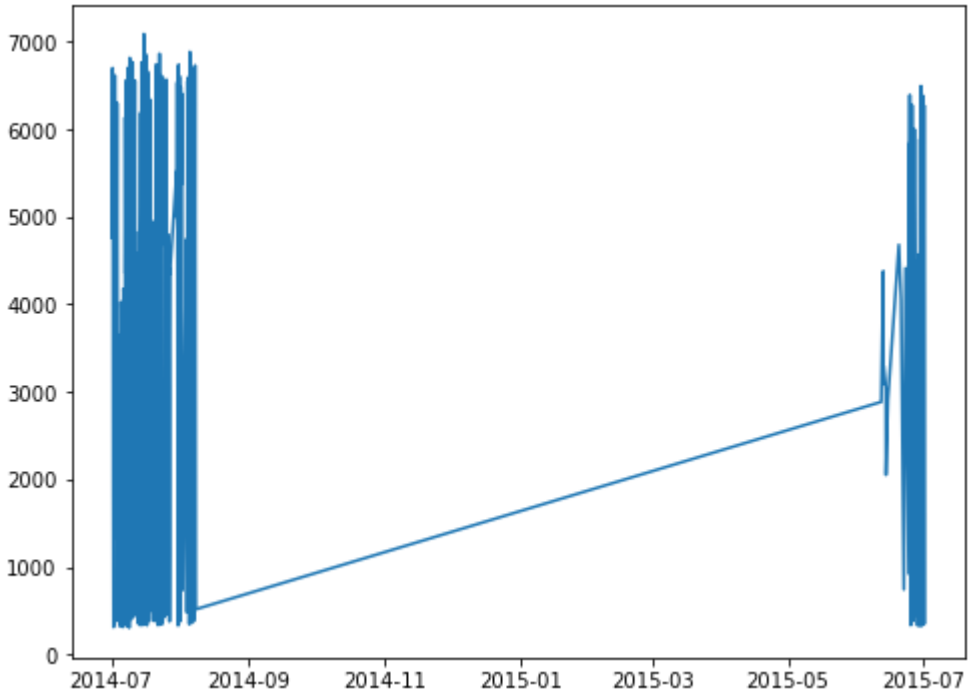
In [4]:
```
fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(df["date_time"], df["traffic_volume"]);
```



Clean the dataset

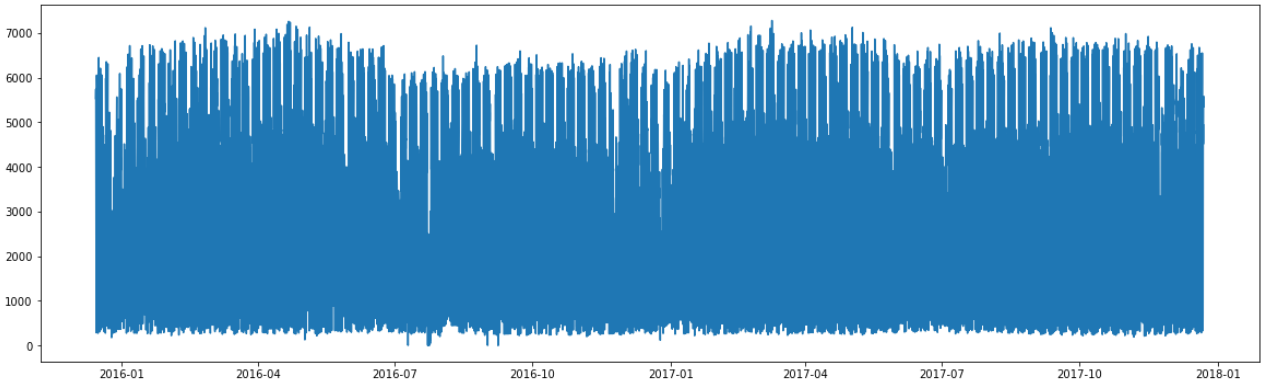
In [5]:
```
# subset of dataframe
dates_2014_to_2015 = df[(df["date_time"] > '2014-07-01 09:00:00') & (df["date_time"] <
```

In [6]:
```
fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(dates_2014_to_2015["date_time"], dates_2014_to_2015["traffic_volume"]);
```



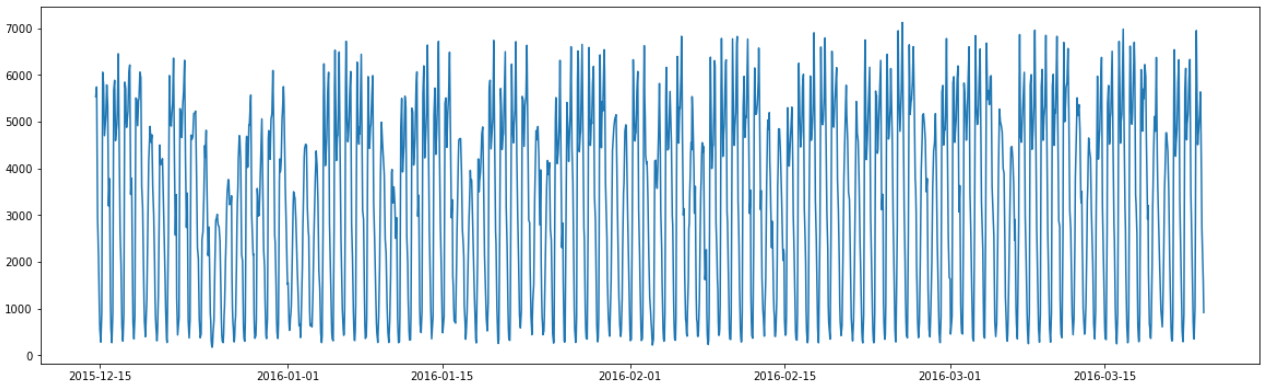
In [7]:

```
# plot the dataset
fig, ax = plt.subplots(figsize=(20, 6))
ax.plot(df["date_time"][20000:40000], df["traffic_volume"][20000:40000]);
```

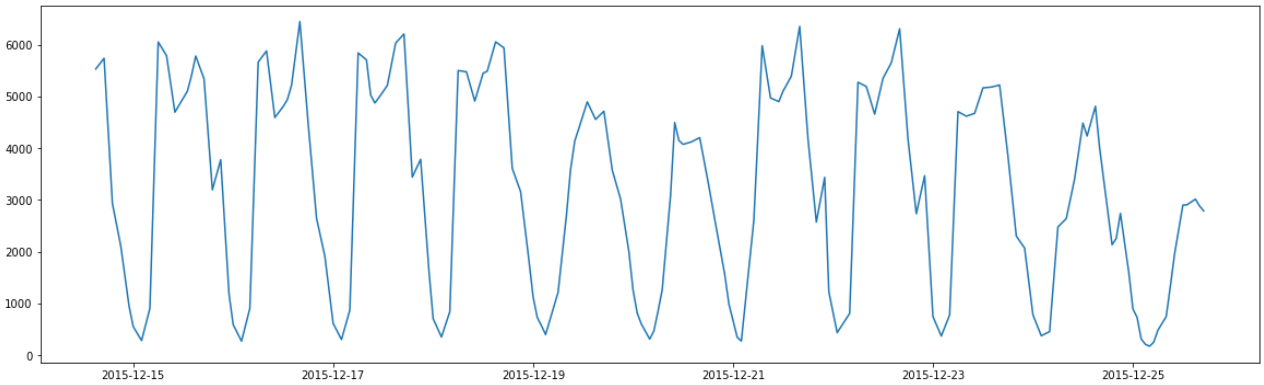


In [8]:

```
# plot the dataset
fig, ax = plt.subplots(figsize=(20, 6))
ax.plot(df["date_time"][20000:22000], df["traffic_volume"][20000:22000]);
```



In [9]:
```
# plot the dataset
fig, ax = plt.subplots(figsize=(20, 6))
ax.plot(df["date_time"][20000:20200], df["traffic_volume"][20000:20200]);
```



In [10]:
```
# drop data prior to 2015 - 07
df_complete = df[df["date_time"] > '2015-07-01 09:00:00']
```

- checking stationarity
- making non-stationary data stationary
- checking for feature correlation with target
- choosing appripriate train-test-split
- Creating the model
- Choosing validation metrics

https://www.analyticsvidhya.com/blog/2018/09/multivariate-time-series-guide-forecasting-modeling-python-codes/

In [11]:
```
# one hot encoding categorical variables
df_encoded = pd.get_dummies(df_complete)
```

Checking Stationarity with Augmented Dicky-Fuller Test

https://analyticsindiamag.com/complete-guide-to-dickey-fuller-test-in-time-series-analysis/

A key point to remember here is: Since the null hypothesis assumes the presence of a unit root, the p-value obtained by the test should be less than the significance level (say 0.05) to reject the null hypothesis. Thereby, inferring that the series is stationary.

In [12]:
```
from statsmodels.tsa.stattools import adfuller
```

In [13]:
```
target_df = df_complete[["date_time", "traffic_volume"]]
target_df.head()
```

Out[13]:

	date_time	traffic_volume
16166	2015-07-01 10:00:00	4273

	date_time	traffic_volume
16167	2015-07-01 11:00:00	4469
16168	2015-07-01 12:00:00	4625
16169	2015-07-01 13:00:00	4462
16170	2015-07-01 14:00:00	4996

In [14]:
```
series = target_df["traffic_volume"].values
```

In [15]:
```
# ADF Test
result = adfuller(series, autolag='AIC')
```

In [16]:
```
# cite source here: https://analyticsindiamag.com/complete-guide-to-dickey-fuller-test-

print('ADF Statistic: %f' % result[0])

print('p-value: %f' % result[1])

print('Critical Values:')

for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))
if result[0] < result[4]["5%"]:
    print ("Reject Ho - Time Series is Stationary")
else:
    print ("Failed to Reject Ho - Time Series is Non-Stationary")
```

```
ADF Statistic: -21.926192
p-value: 0.000000
Critical Values:
        1%: -3.431
        5%: -2.862
        10%: -2.567
Reject Ho - Time Series is Stationary
```

Granger's Causality Test

In [17]:
```
from statsmodels.tsa.stattools import grangercausalitytests
```

In [18]:
```
column_names = df_encoded.columns.values.tolist()
column_names.remove('date_time')
column_names.remove('traffic_volume')
```

In [19]:
```
column_names
```

Out[19]:
```
['temp',
 'rain_1h',
 'snow_1h',
 'clouds_all',
 'holiday_Christmas Day',
```

```
'holiday_Columbus Day',
'holiday_Independence Day',
'holiday_Labor Day',
'holiday_Martin Luther King Jr Day',
'holiday_Memorial Day',
'holiday_New Years Day',
'holiday_None',
'holiday_State Fair',
'holiday_Thanksgiving Day',
'holiday_Veterans Day',
'holiday_Washingtons Birthday',
'weather_main_Clear',
'weather_main_Clouds',
'weather_main_Drizzle',
'weather_main_Fog',
'weather_main_Haze',
'weather_main_Mist',
'weather_main_Rain',
'weather_main_Smoke',
'weather_main_Snow',
'weather_main_Squall',
'weather_main_Thunderstorm',
'weather_description_SQUALLS',
'weather_description_Sky is Clear',
'weather_description_broken clouds',
'weather_description_drizzle',
'weather_description_few clouds',
'weather_description_fog',
'weather_description_freezing rain',
'weather_description_haze',
'weather_description_heavy intensity drizzle',
'weather_description_heavy intensity rain',
'weather_description_heavy snow',
'weather_description_light intensity drizzle',
'weather_description_light intensity shower rain',
'weather_description_light rain',
'weather_description_light rain and snow',
'weather_description_light shower snow',
'weather_description_light snow',
'weather_description_mist',
'weather_description_moderate rain',
'weather_description_overcast clouds',
'weather_description_proximity shower rain',
'weather_description_proximity thunderstorm',
'weather_description_proximity thunderstorm with drizzle',
'weather_description_proximity thunderstorm with rain',
'weather_description_scattered clouds',
'weather_description_shower drizzle',
'weather_description_sky is clear',
'weather_description_sleet',
'weather_description_smoke',
'weather_description_snow',
'weather_description_thunderstorm',
'weather_description_thunderstorm with drizzle',
'weather_description_thunderstorm with heavy rain',
'weather_description_thunderstorm with light drizzle',
'weather_description_thunderstorm with light rain',
'weather_description_thunderstorm with rain',
'weather_description_very heavy rain']
```

In [20]:
```
"""for name in column_names:
    print(name)
    grangercausalitytests(df_encoded[[name, 'traffic_volume']], maxlag=4)
    print('------------------------------------')"""
```

Out[20]:
```
"for name in column_names:\n    print(name)\n    grangercausalitytests(df_encoded[[name,
'traffic_volume']], maxlag=4)\n    print('------------------------------------')"
```

According to the results above, the columns with p values less than 0.05 should be included in the training model

In [21]:
```
df_dropped = df_encoded[['date_time', 'temp', 'clouds_all', 'holiday_None', 'weather_ma
```

In [22]:
```
df_dropped.head()
```

Out[22]:

	date_time	temp	clouds_all	holiday_None	weather_main_Clear	weather_main_Clouds	weather_
16166	2015-07-01 10:00:00	289.24	40	1	0	1	
16167	2015-07-01 11:00:00	289.44	75	1	0	1	
16168	2015-07-01 12:00:00	290.53	1	1	1	0	
16169	2015-07-01 13:00:00	292.17	1	1	1	0	
16170	2015-07-01 14:00:00	293.61	1	1	1	0	

Training and Validation Data Split

In [23]:
```
df_dropped['hour'] = df_dropped['date_time'].dt.hour
df_dropped['weekday'] = df_dropped['date_time'].dt.dayofweek
```

```
C:\Users\karli\AppData\Local\Temp/ipykernel_26216/430161332.py:1: SettingWithCopyWarnin
g:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df_dropped['hour'] = df_dropped['date_time'].dt.hour
C:\Users\karli\AppData\Local\Temp/ipykernel_26216/430161332.py:2: SettingWithCopyWarnin
g:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df_dropped['weekday'] = df_dropped['date_time'].dt.dayofweek
```

In [24]:
```
df_features = df_dropped.drop('date_time', axis = 1)
```

In [25]:
```
df_features.astype('int32').dtypes
```

Out[25]:
```
temp                                          int32
clouds_all                                    int32
holiday_None                                  int32
weather_main_Clear                            int32
weather_main_Clouds                           int32
weather_main_Fog                              int32
weather_main_Mist                             int32
weather_description_broken clouds             int32
weather_description_few clouds                int32
weather_description_fog                       int32
weather_description_light intensity drizzle   int32
weather_description_mist                      int32
weather_description_overcast clouds           int32
weather_description_scattered clouds          int32
weather_description_sky is clear              int32
weather_description_proximity shower rain     int32
traffic_volume                                int32
hour                                          int32
weekday                                       int32
dtype: object
```

In [26]:
```
# df_datetime = df_dropped['date_time']
```

In [27]:
```
# df_concat = pd.concat([df_datetime, df_features], axis=1)
```

In [28]:
```
# df_final = df_concat.set_index('date_time')
```

In [29]:
```
# df_final.dtypes
```

In [30]:
```
# df_final.index = pd.DatetimeIndex(df_final.index).to_period('H')
```

In [31]:
```
train = df_features[:int(0.8*(len(df_features)))]
valid = df_features[int(0.8*(len(df_features))):]
train_x = train.drop('traffic_volume', axis=1)
train_y = train[['traffic_volume']]
valid_x = valid.drop('traffic_volume', axis=1)
valid_y = valid[['traffic_volume']]
x_axis = df_complete['date_time']
x_axis_train = x_axis[:int(0.8*(len(df_features)))]
x_axis_valid = x_axis[int(0.8*(len(df_features))):]
```

Train the Model

In [32]:
```
# Fitting Random Forest Regression to the dataset
# import the regressor
from sklearn.ensemble import RandomForestRegressor
```

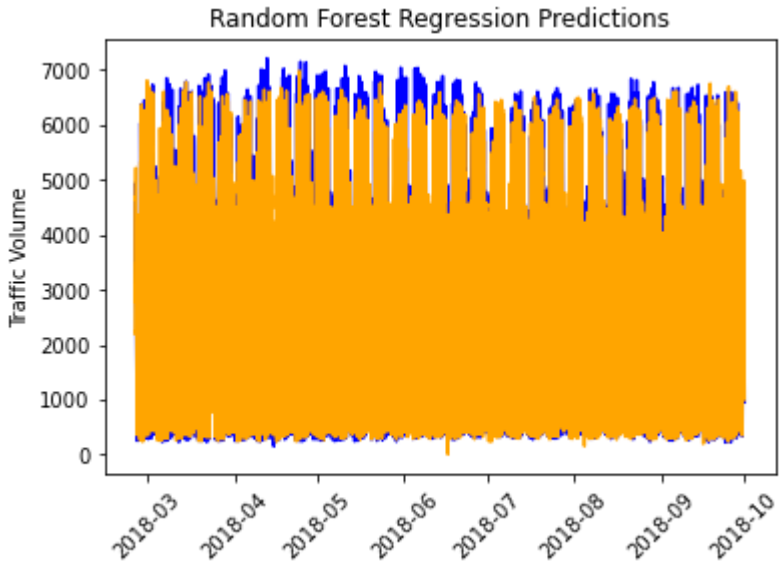
In [33]:
```
regressor = RandomForestRegressor(n_estimators=100, random_state=0, oob_score = True)
```

In [34]:
```
regressor.fit(train_x, train_y.values.ravel())
```

Out[34]:
```
RandomForestRegressor
RandomForestRegressor(oob_score=True, random_state=0)
```

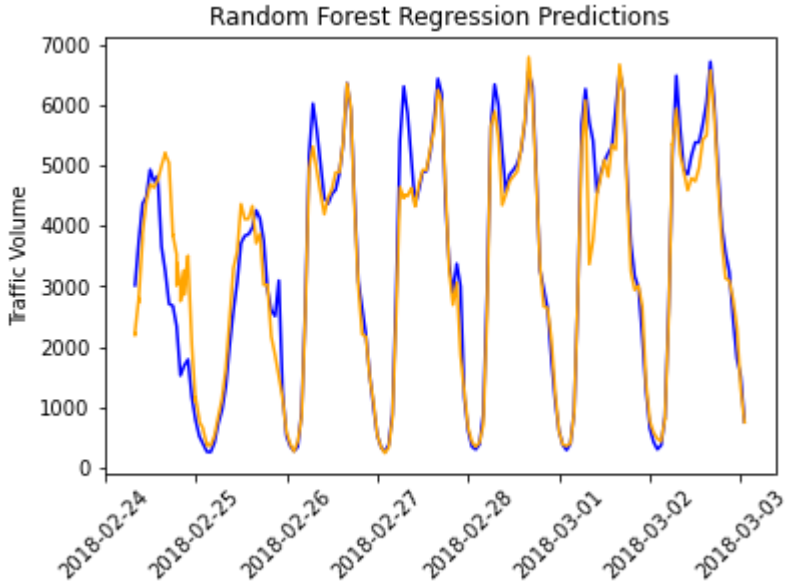
In [35]:
```
y_pred = regressor.predict(valid_x)  # test the output by changing values
```

In [39]:
```
plt.plot(x_axis_valid, valid_y, color = 'blue')
plt.plot(x_axis_valid, y_pred, color = 'orange')
plt.title('Random Forest Regression Predictions')
plt.xticks(rotation=45)
plt.ylabel('Traffic Volume')
plt.show()
```



In [40]:

```
plt.plot(x_axis_valid[0:200], valid_y[0:200], color = 'blue')
plt.plot(x_axis_valid[0:200], y_pred[0:200], color = 'orange')
plt.title('Random Forest Regression Predictions')
plt.xticks(rotation=45)
plt.ylabel('Traffic Volume')
plt.show()
```



In [38]:

```
print(regressor.oob_score_)
```

```
0.9469373959325946
```

In [ ]:

In [ ]:

In [ ]: