Predicting Metro Interstate Traffic Volume: A Multivariate Time Series Forecasting Approach with Vector Auto Regression

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 Date of Change: 4/15/2023

Change## : 2

Change(s) Made: visualized dataset and inspected for missing values Date of Change: 4/26/2023

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()
```

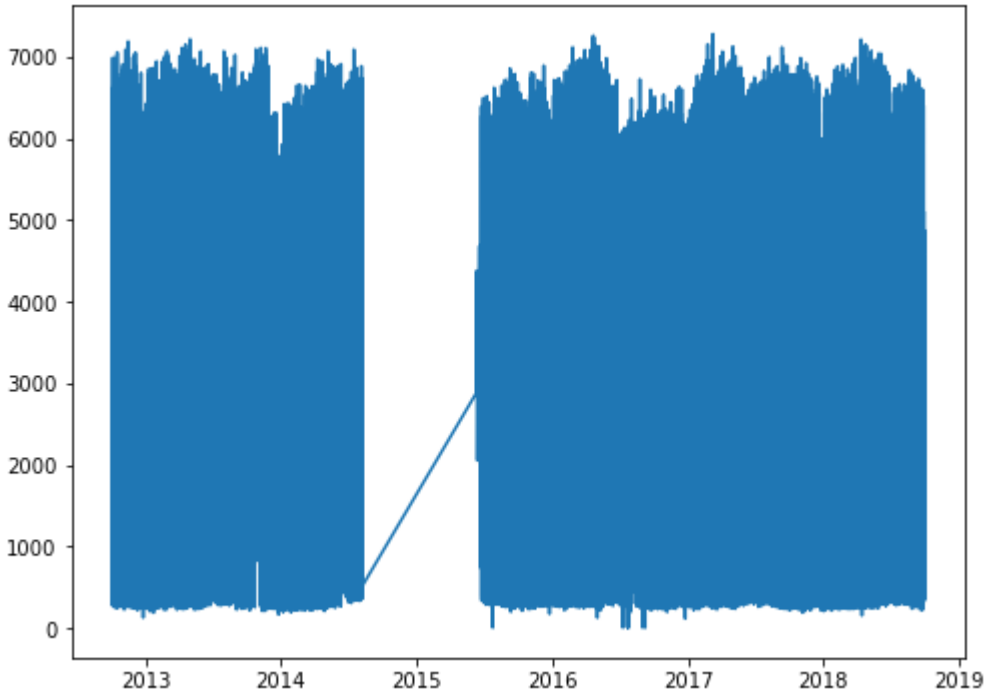
Out[2]:

	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	

	holiday	temp	rain_1h	snow_1h	clouds_all	weather_main	weather_description	date_time	traffic_v
4	None	291.14	0.0	0.0	75	Clouds	broken clouds	2012-10-02 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"])
```

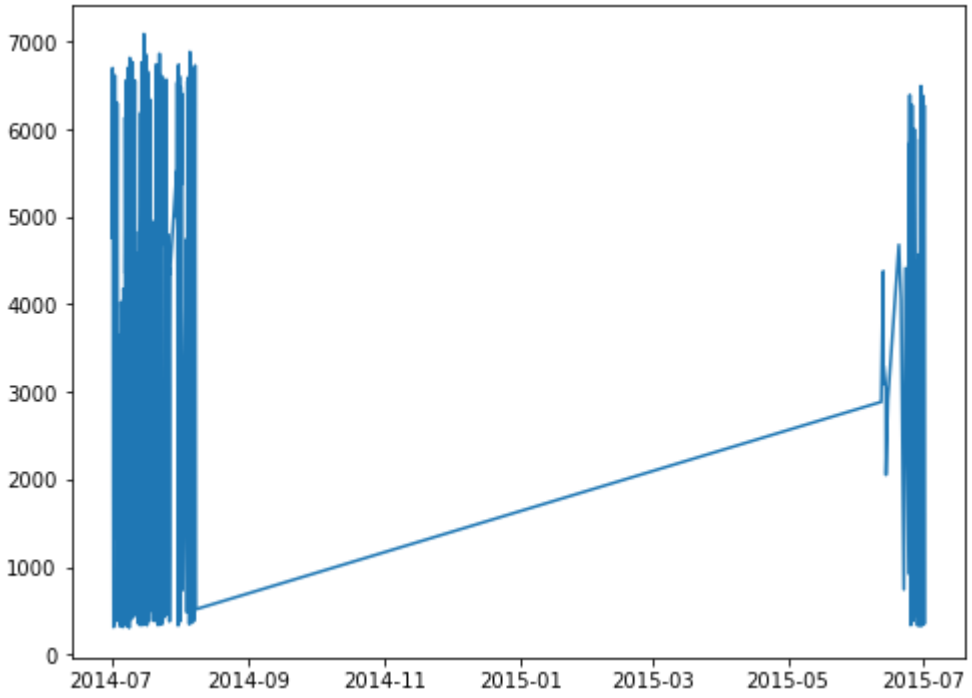
In [4]:
```
fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(df["date_time"], df["traffic_volume"]);
```



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]:
```
# 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 [8]:
```
# 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 [9]:
```
from statsmodels.tsa.stattools import adfuller
```

In [10]:
```
target_df = df_complete[["date_time", "traffic_volume"]]
target_df.head()
```

Out[10]:

	date_time	traffic_volume
16166	2015-07-01 10:00:00	4273
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 [11]:
```
series = target_df["traffic_volume"].values
```

In [12]:
```
# ADF Test
result = adfuller(series, autolag='AIC')
```

In [13]:
```
# 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 [14]:
```
from statsmodels.tsa.stattools import grangercausalitytests
```

In [15]:
```
column_names = df_encoded.columns.values.tolist()
column_names.remove('date_time')
column_names.remove('traffic_volume')
```

In [16]:
```
column_names
```

Out[16]:
```
['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 [ ]:
```
for name in column_names:
    print(name)
    grangercausalitytests(df_encoded[[name, 'traffic_volume']], maxlag=4)
    print('------------------------------------')
```

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

In [18]:
```
df_dropped = df_encoded[['date_time', 'temp', 'clouds_all', 'holiday_None', 'weather_ma
```

In [19]:
```
df_dropped.head()
```

Out[19]:

	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 [20]:
```
df_features = df_dropped.drop('date_time', axis = 1)
```

In [21]:
```
df_features.astype('int32').dtypes
```

Out[21]:
```
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
dtype: object
```

In [22]:
```
df_datetime = df_dropped['date_time']
```

In [23]:
```
df_concat = pd.concat([df_datetime, df_features], axis=1)
```

In [24]:
```
df_final = df_concat.set_index('date_time')
```

In [25]:
```
df_final.index = pd.DatetimeIndex(df_final.index).to_period('H')
```

In [26]:
```
df_final.dtypes
```

Out[26]:
```
temp                                           float64
clouds_all                                       int64
holiday_None                                     uint8
weather_main_Clear                               uint8
weather_main_Clouds                              uint8
weather_main_Fog                                 uint8
weather_main_Mist                                uint8
weather_description_broken clouds                uint8
weather_description_few clouds                   uint8
weather_description_fog                          uint8
weather_description_light intensity drizzle      uint8
weather_description_mist                         uint8
weather_description_overcast clouds              uint8
weather_description_scattered clouds             uint8
weather_description_sky is clear                 uint8
weather_description_proximity shower rain        uint8
traffic_volume                                   int64
dtype: object
```

Lets choose 6 months of data for validation, and train the model on the other 2.5 years of data.

In [27]:
```
train = df_final[:int(0.8*(len(df_final)))]
valid = df_final[int(0.8*(len(df_final))):]
test_plot = df_datetime[int(0.8*(len(df_final))):]
```

Train the Model

https://www.machinelearningplus.com/time-series/vector-autoregression-examples-python/

In [28]:
```
# Import Statsmodels
from statsmodels.tsa.vector_ar.var_model import VAR
```

In [29]:
```
model = VAR(endog=train)
```

In [30]:
```
model_fit = model.fit()
```

In [31]:
```
prediction = model_fit.forecast(model_fit.endog, steps=len(valid))
```

In [32]:
```
cols = train.columns
```

In [33]:
```
from sklearn.metrics import mean_squared_error
```

In [34]:
```
#converting predictions to dataframe
pred = pd.DataFrame(index=range(0,len(prediction)),columns=[cols])
for j in range(0,17):
    for i in range(0, len(prediction)):
       pred.iloc[i][j] = prediction[i][j]
```

In [35]:
```
pred.index = valid.index
```

In [36]:
```
pred
```

Out[36]:

date_time	temp	clouds_all	holiday_None	weather_main_Clear	weather_main_Clouds	weather_mai
2018-02-24 08:00	256.086808	85.798617	0.997794	0.055161	0.01652	0.0
2018-02-24 08:00	256.040588	82.28748	0.997334	0.085165	0.036888	0.0
2018-02-24 08:00	256.026357	79.28077	0.99736	0.106321	0.057967	0.0
2018-02-24 08:00	256.037967	76.676979	0.997409	0.12495	0.077918	0.0
2018-02-24 09:00	256.071479	74.396979	0.997476	0.14221	0.095727	0.0
...	...	...	...	...	...	
2018-09-30 19:00	281.605124	47.932118	0.998516	0.304643	0.270037	0.0
2018-09-30 20:00	281.605124	47.932118	0.998516	0.304643	0.270037	0.0
2018-09-30 21:00	281.605124	47.932118	0.998516	0.304643	0.270037	0.0

	temp	clouds_all	holiday_None	weather_main_Clear	weather_main_Clouds	weather_mai
date_time						
2018-09-30 22:00	281.605124	47.932118	0.998516	0.304643	0.270037	0.0
2018-09-30 23:00	281.605124	47.932118	0.998516	0.304643	0.270037	0.0

6408 rows × 17 columns

In [37]:
```
valid
```

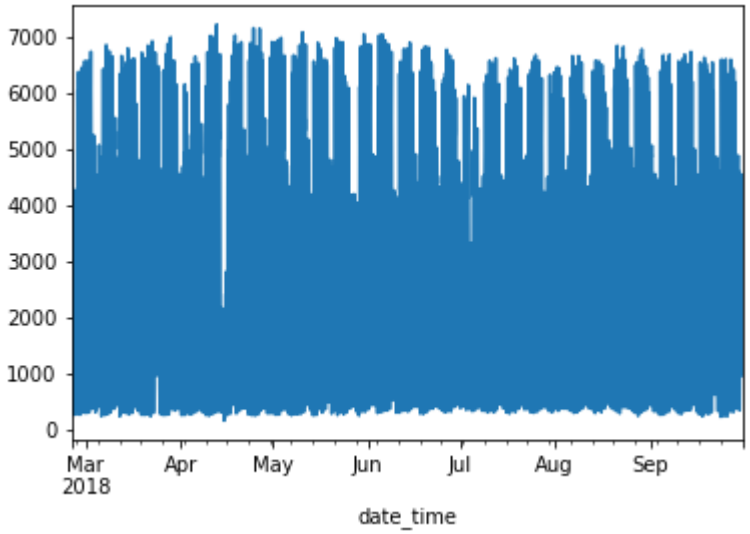
Out[37]:

	temp	clouds_all	holiday_None	weather_main_Clear	weather_main_Clouds	weather_main_Fo
date_time						
2018-02-24 08:00	257.44	1	1	0	0	
2018-02-24 08:00	257.44	1	1	0	0	
2018-02-24 08:00	257.44	1	1	0	0	
2018-02-24 08:00	257.44	1	1	0	0	
2018-02-24 09:00	259.90	75	1	0	0	
...	...	...	...	...	...	
2018-09-30 19:00	283.45	75	1	0	1	
2018-09-30 20:00	282.76	90	1	0	1	
2018-09-30 21:00	282.73	90	1	0	0	
2018-09-30 22:00	282.09	90	1	0	1	
2018-09-30 23:00	282.12	90	1	0	1	

6408 rows × 17 columns

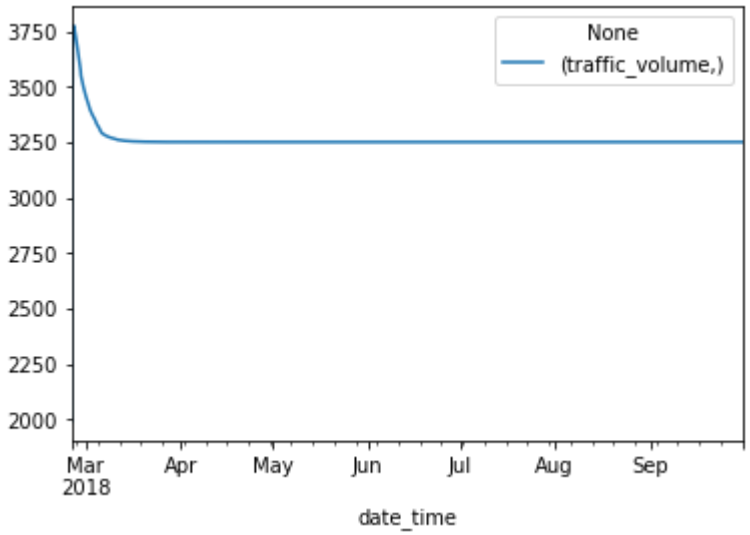
In [38]:
```
valid['traffic_volume'].plot()
```

Out[38]:
```
<AxesSubplot:xlabel='date_time'>
```

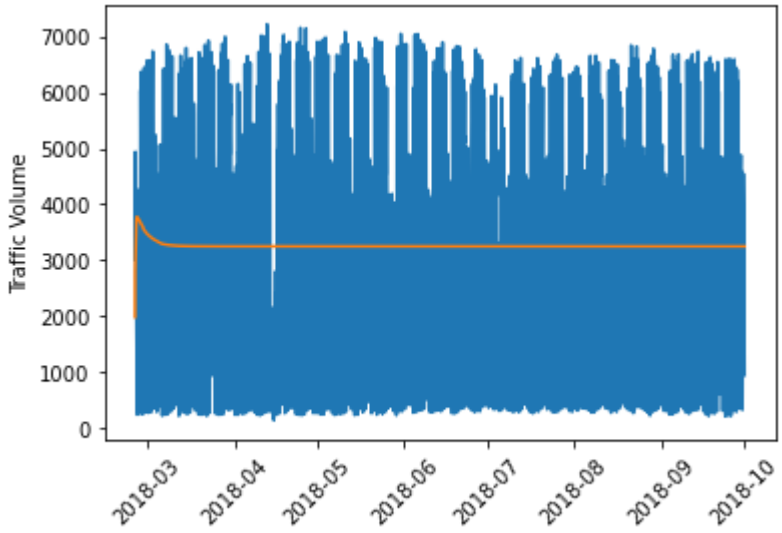


In [39]:
```
pred['traffic_volume'].plot()
```

Out[39]:
```
<AxesSubplot:xlabel='date_time'>
```



In [45]:
```
plt.plot(test_plot, valid['traffic_volume'])
plt.plot(test_plot, pred['traffic_volume'])
plt.xticks(rotation =45)
plt.ylabel('Traffic Volume')
plt.show()
```



In [41]:

```
#check rmse
for i in cols:
    print('rmse value for', i, 'is : ', sqrt(mean_squared_error(pred[i], valid[i])))
```

```
rmse value for temp is :  11.937561741919643
rmse value for clouds_all is :  39.32969146106891
rmse value for holiday_None is :  0.027931605625447536
rmse value for weather_main_Clear is :  0.44864794636650385
rmse value for weather_main_Clouds is :  0.43570137034247647
rmse value for weather_main_Fog is :  0.1526829275013179
rmse value for weather_main_Mist is :  0.3403281863381954
rmse value for weather_description_broken clouds is :  0.27779170903965306
rmse value for weather_description_few clouds is :  0.1704581822089871
rmse value for weather_description_fog is :  0.1526829275013179
rmse value for weather_description_light intensity drizzle is :  0.1656733528846707
rmse value for weather_description_mist is :  0.3403281863381954
rmse value for weather_description_overcast clouds is :  0.25323246886774164
rmse value for weather_description_scattered clouds is :  0.2591050202517264
rmse value for weather_description_sky is clear is :  0.4469846017480361
rmse value for weather_description_proximity shower rain is :  0.0807484153864075
rmse value for traffic_volume is :  1992.4853550673063
```