THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

DATS 6450: TIME SERIES ANALYSIS AND MODELING

PREDICTING TRAFFIC VOLUME

OSEMEKHIAN SOLOMON EHILEN

OSE

5/2/2022

CONTENT

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This time series project uses weather and holiday features to predict hourly traffic volume between Minneapolis and St Paul, MN with data from October 2, 2012 till September 30, 2018. Several time series techniques were used to find an appropriate prediction model. The project progresses from data preprocessing, time series decomposition, multiple model development and evaluation using standard metrics. At the end of this research, the Average method performed best in predicting traffic volume.

INTRODUCTION

Time series analysis helps us model time dependent occurrences which further enables trustworthy prediction. With the dataset in concern, we will discover factors influencing traffic volumes hourly and get best feature(s) that predict hourly traffic volume between Minneapolis and St Paul, MN.

Before we explore various models such as base models (average, naïve, drift, simple exponential smoothing) then Holt-Winters method, multiple linear regression with feature selection using backward stepwise regression and ARMA/ARIMA/SARIMA models.

At the end a more accurate model will be selected based on various metrics.

DATA DESCRIPTION

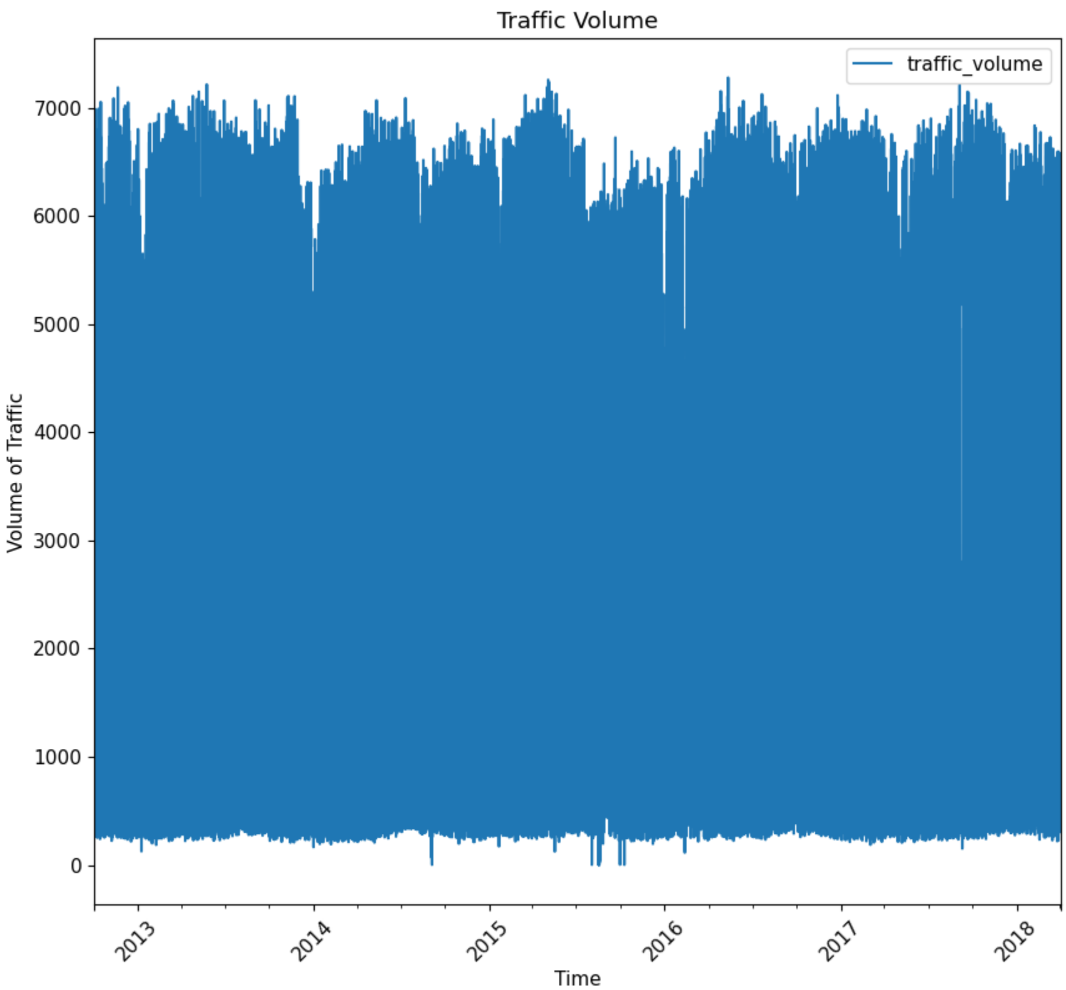
The dataset is gotten from UCI's Machine Learning repository. Hourly Interstate 94 Westbound traffic volume for MN DoT ATR station 301, roughly midway between Minneapolis and St Paul, MN. Hourly weather features and holidays included for impacts on traffic volume.

Attribute Information

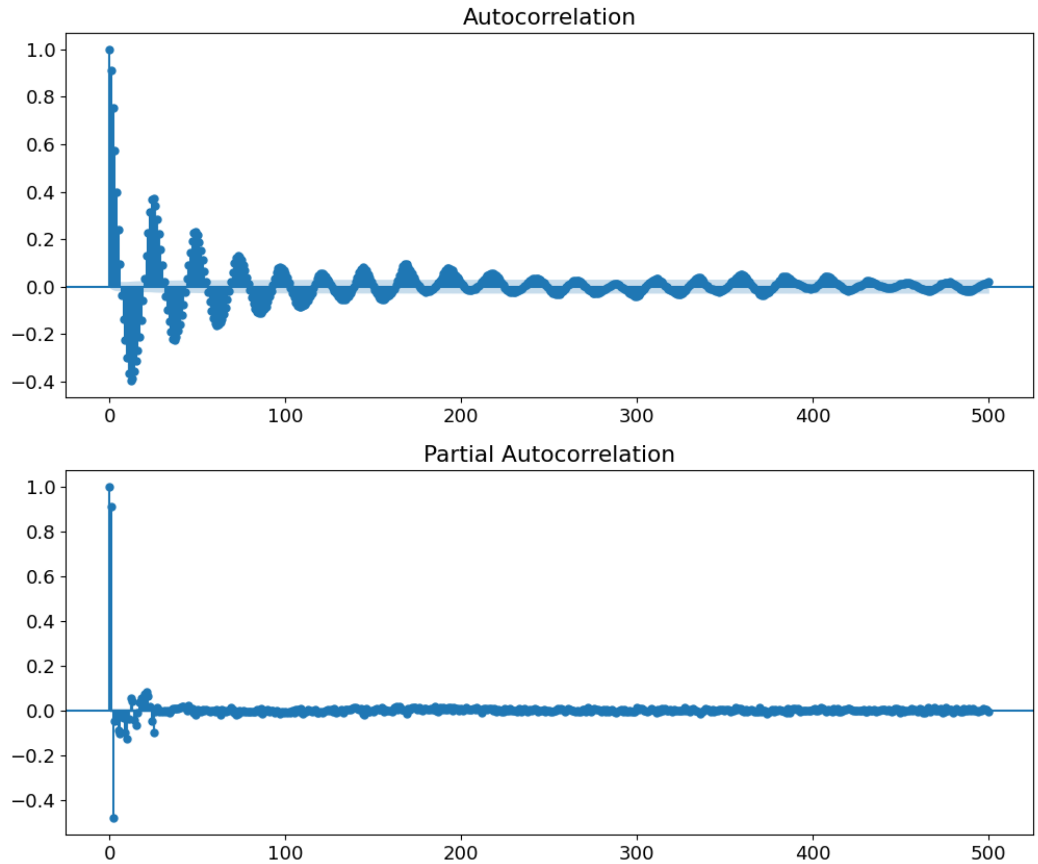
Variable	Description
Holiday	Categorical US National holidays plus regional holiday, Minnesota State Fair
Temp	Numeric Average temp in kelvin
Rain_1h	Numeric Amount in mm of rain that occurred in the hour
Snow_1h	Numeric Amount in mm of snow that occurred in the hour
Clouds_all	Numeric Percentage of cloud cover
Weather_main	Categorical Short textual description of the current weather
Weather_description	Categorical Longer textual description of the current weather
Date_time	DateTime Hour of the data collected in local CST time
Traffic_volume	Numeric Hourly I-94 ATR 301 reported westbound traffic volume

Metro Interstate Traffic Volume Data Set

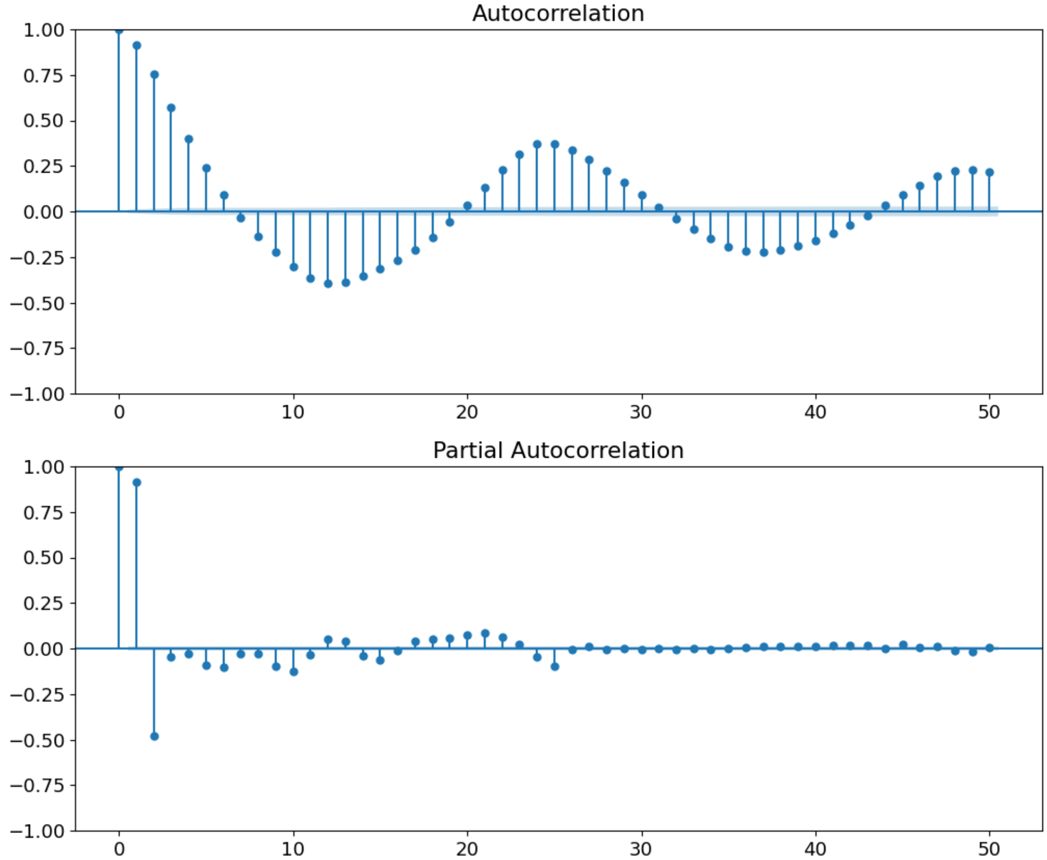
Time Plot for Dependent Variable



Autocorrelation and Partial Autocorrelation Function (ACF/PACF) Plots



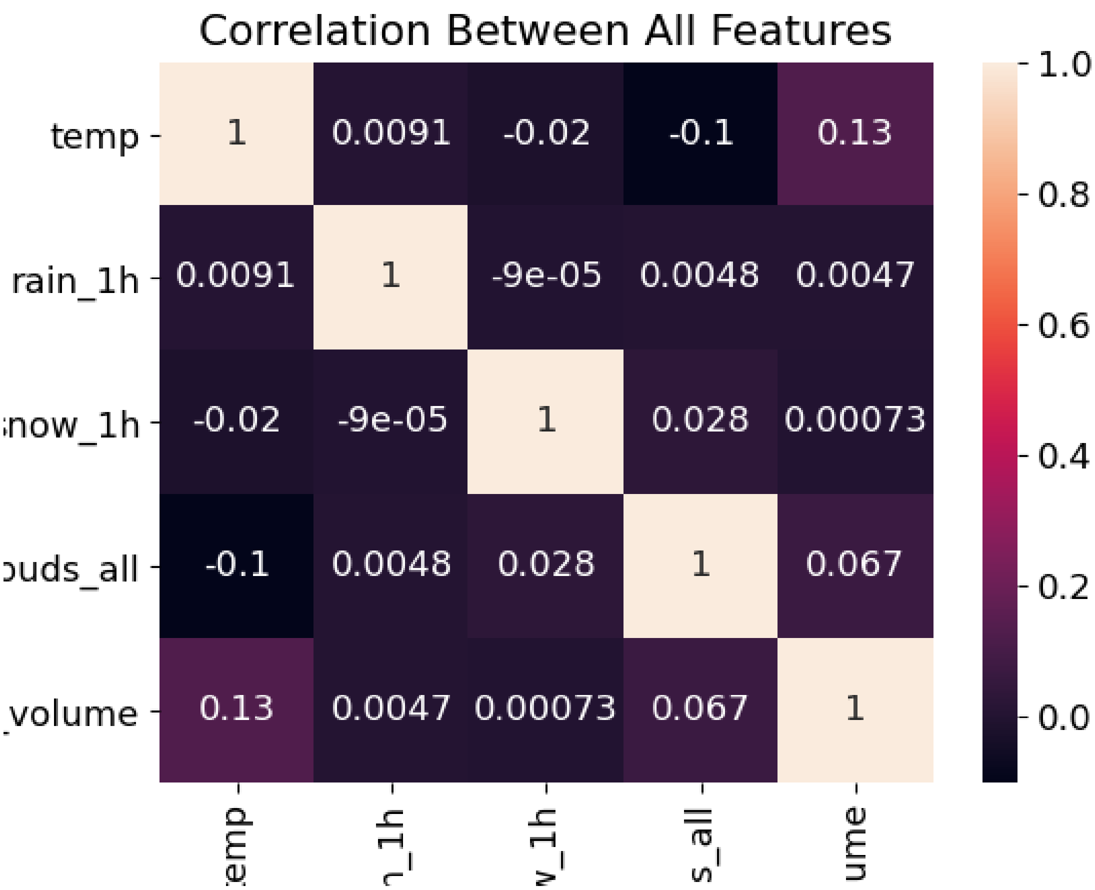
ACF/PACF for 500 Sample



ACF/PACF for 50 Sample

The ACF clearly shows oscillations which indicates the presence of seasonality. The ACF shows a tail-off between lags and a cut-off in PACF at lag 2. This can indicate this particular dataset to be an Autoregressive process of order 2.

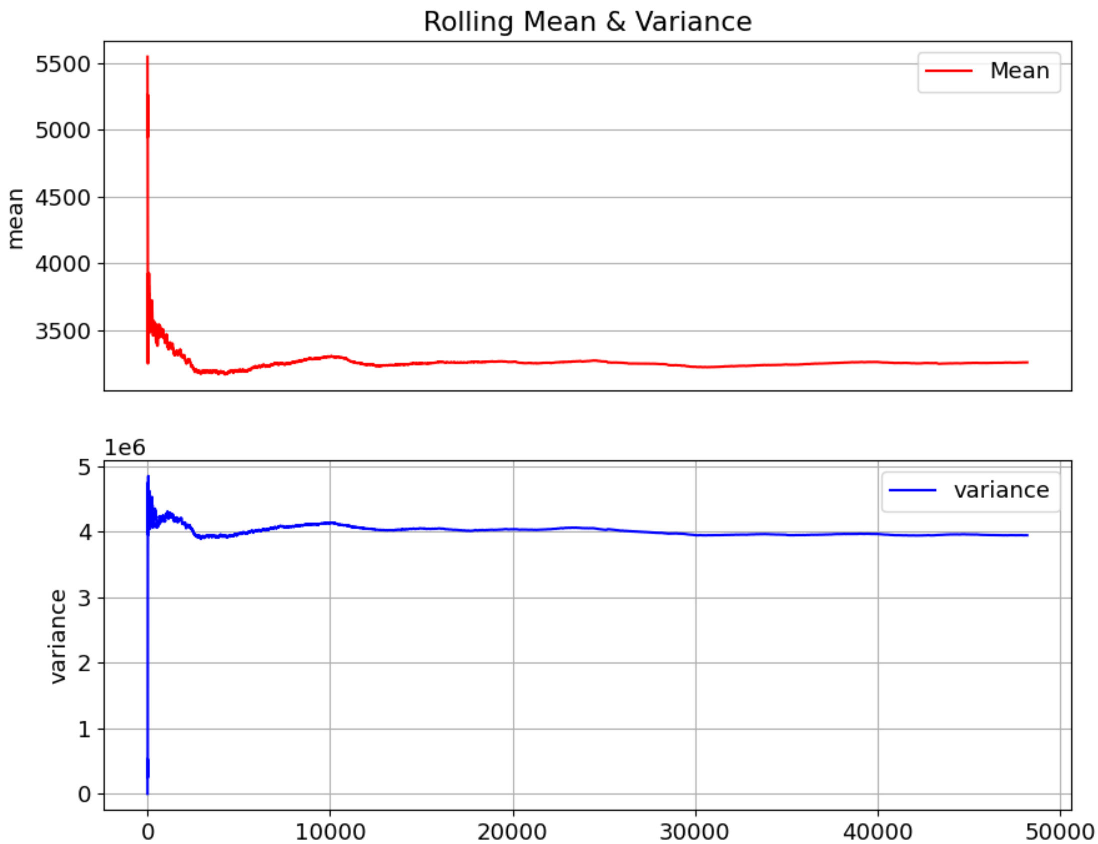
Correlation Plot Between Numeric features



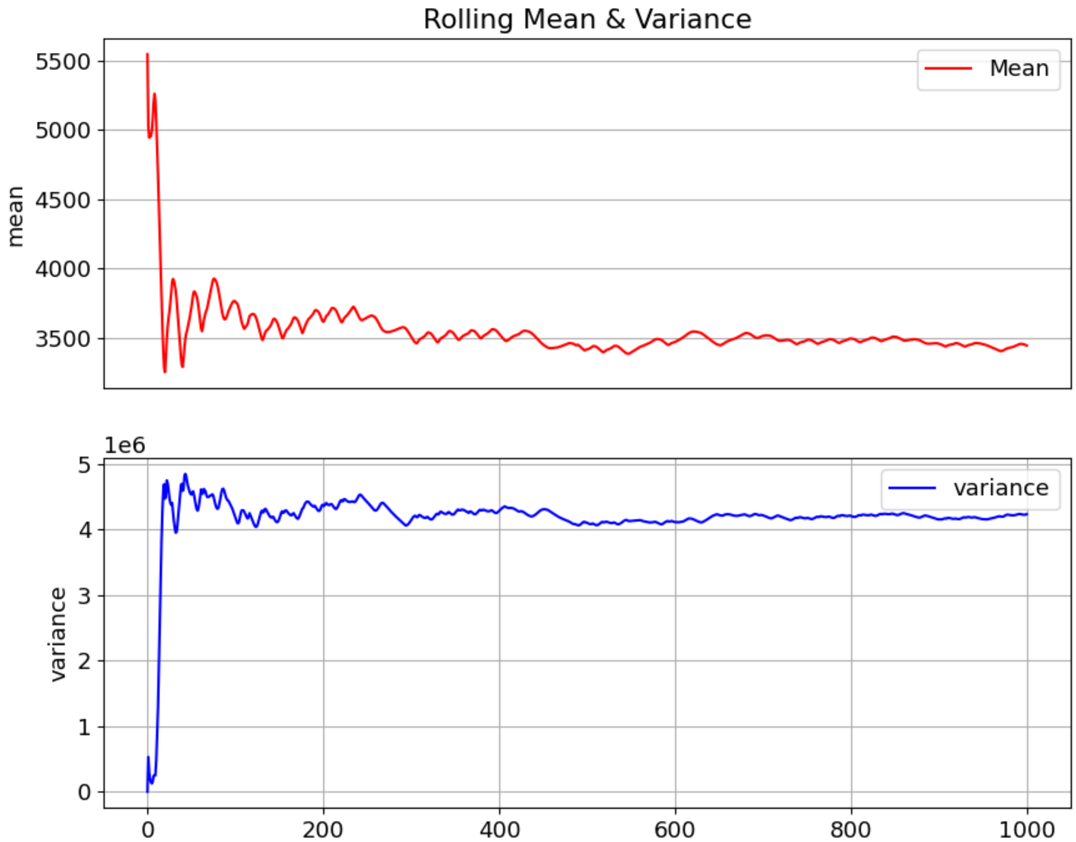
The correlation plot above shows that temp is positively correlated with traffic volume in the sense that as temp increases traffic volume increases by 13% vice versa. Also, clouds_all has a 6.7% positive correlation with traffic volume, rain_1h has a weak positive correlation of 0.47% with traffic volume, and snow_1h has a weak positive correlation of 0.073% with traffic volume.

With no missing values, we can proceed in splitting the dataset into training and testing set with ratio 80:20 respectively.

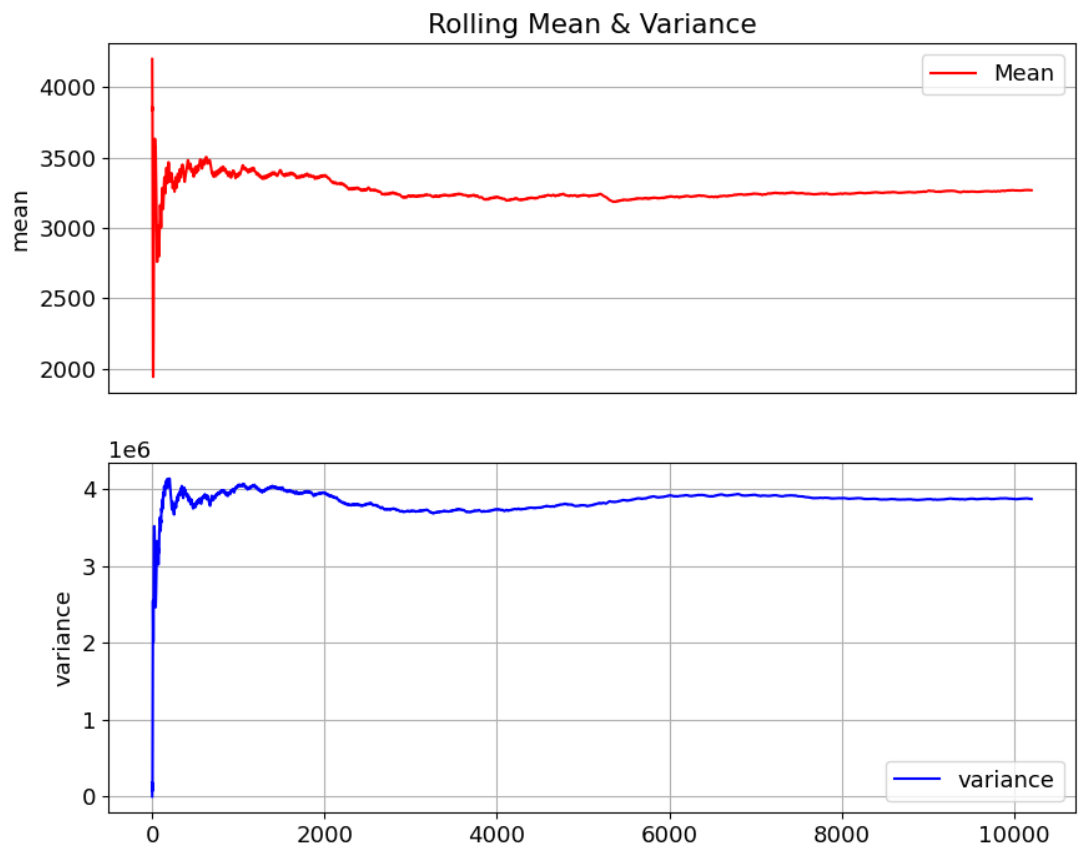
Check for Stationarity on Target Variable



Rolling mean & variance of the whole dataset



Rolling mean & variance of the first 1000 samples



Rolling mean & variance of the last 1000 samples

The rolling mean and rolling variance plot above does not diverge. This indicates stationarity of our dataset, but an ADF and KPSS tests will be performed to confirm the claim.

Hypothesis for Augmented Dickey-Fuller (ADF) Test:

- Null Hypothesis - Traffic volume is non-stationary
- Alternate Hypothesis - Traffic volume is stationary

ADF Statistic: -28.016624

p-value: 0.000000

Critical Values:

1%: -3.430

5%: -2.862

10%: -2.567

With p-value less than 0.05 we reject the null hypothesis and conclude that the traffic volume is stationary.

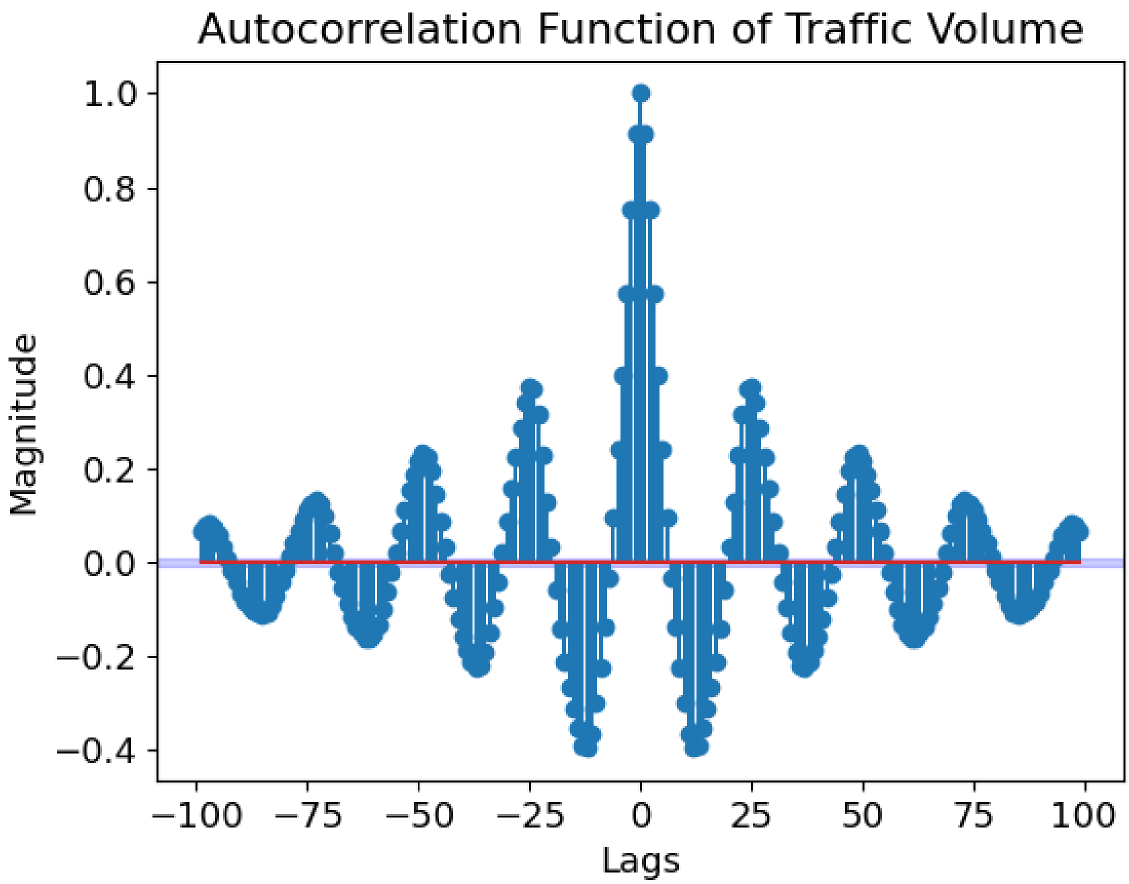
Hypothesis for Kwiatkowski-Phillips-Schmidt-Shin (KPSS) Test:

- Null Hypothesis - Traffic volume is stationary
- Alternate Hypothesis - Traffic volume is non-stationary

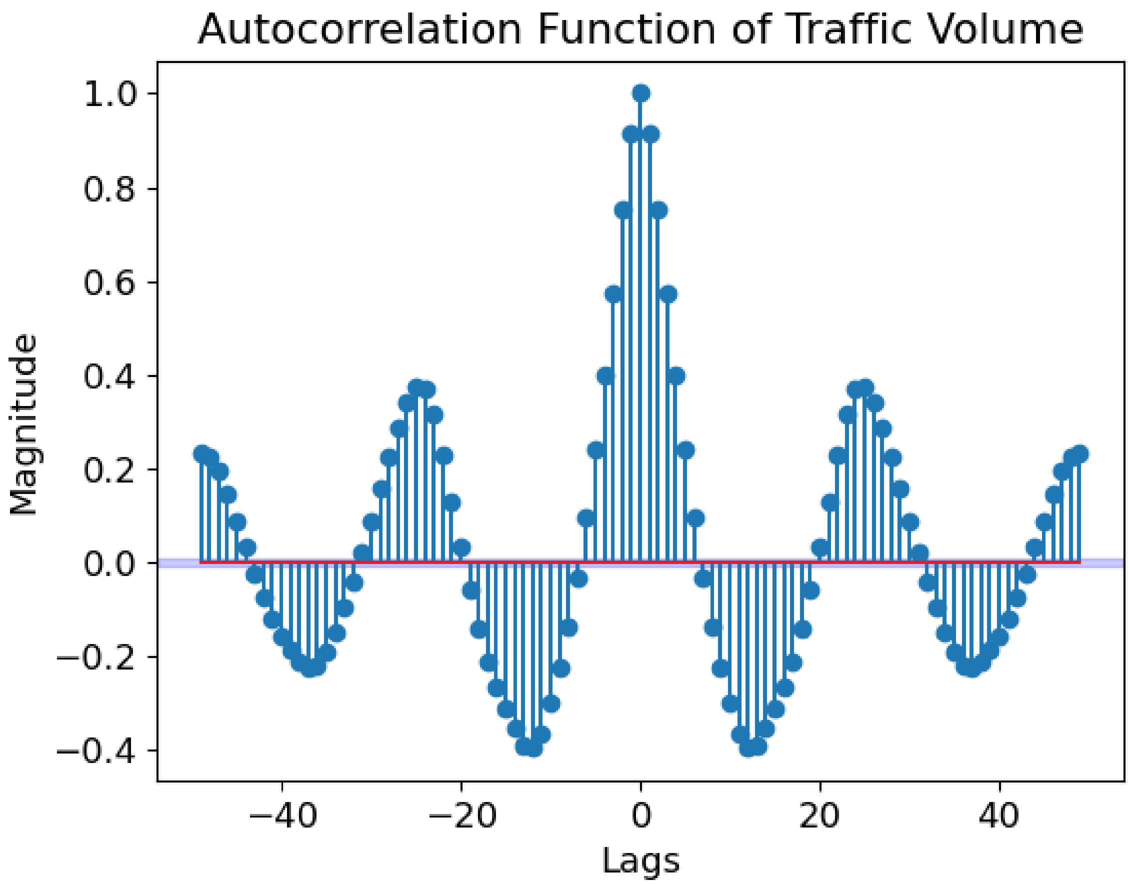
Results of KPSS Test:

Test Statistic	0.212032
p-value	0.100000
Lags Used	23.000000
Critical Value (10%)	0.347000
Critical Value (5%)	0.463000

With p-value is greater than 0.05 we do not reject the null hypothesis and conclude that traffic volume is stationary.

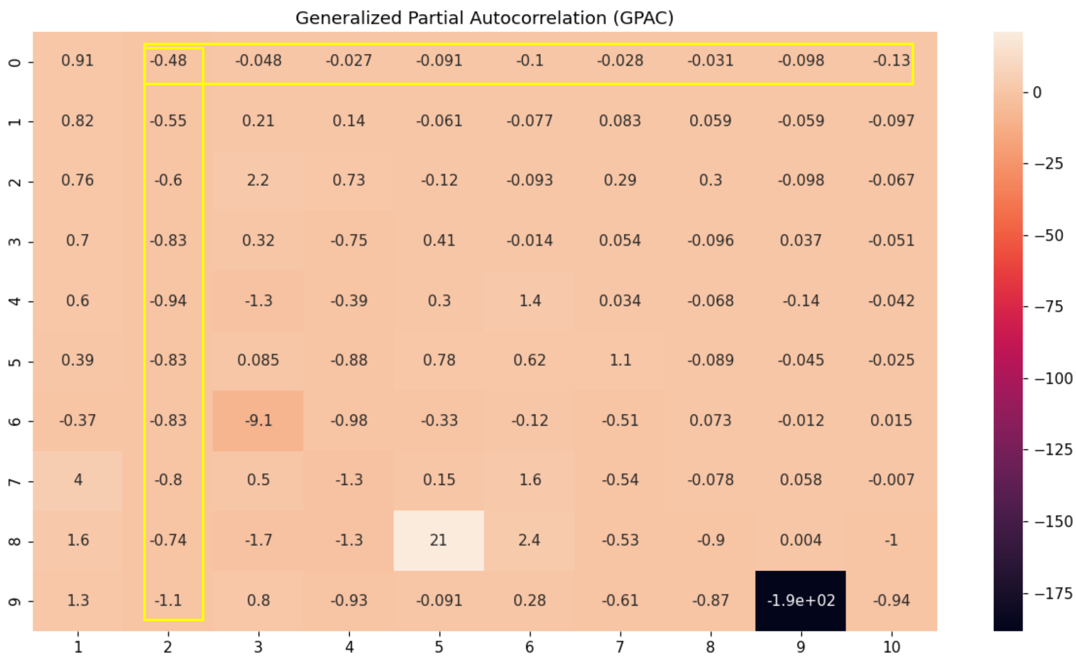


ACF for 100 Lags



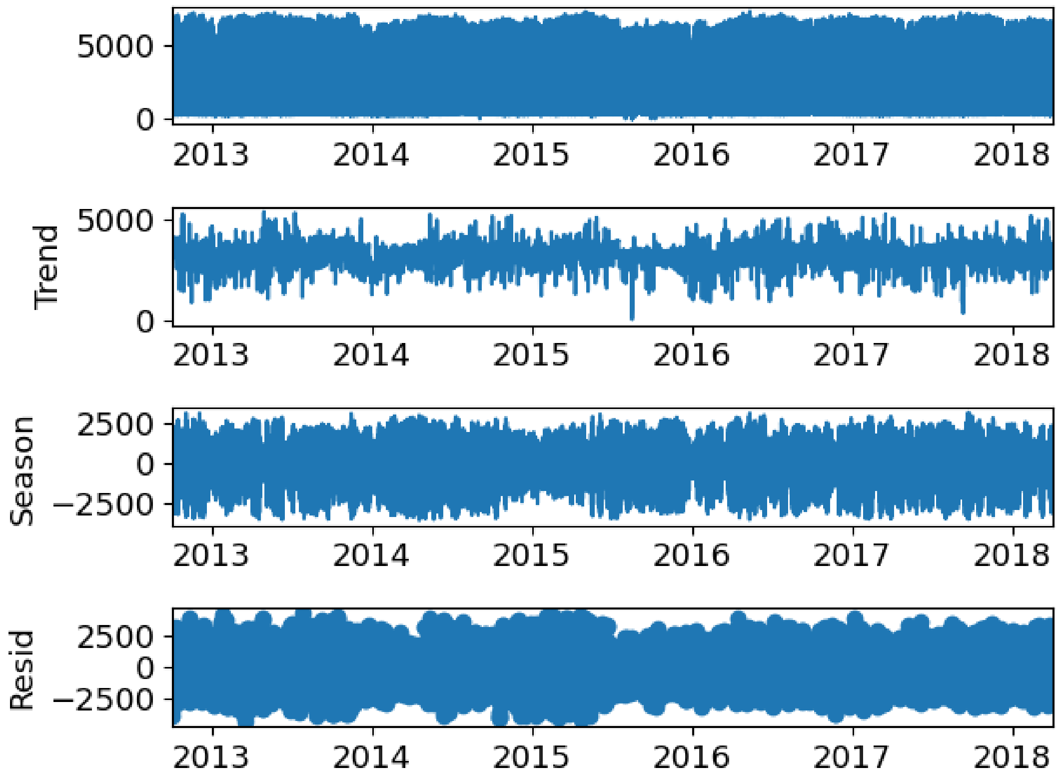
ACF for 50 Lags

The ACF above infer seasonality with period 12. That is, at every lag of 12, there is a rise and fall in traffic volume which will be 12 hours' period.

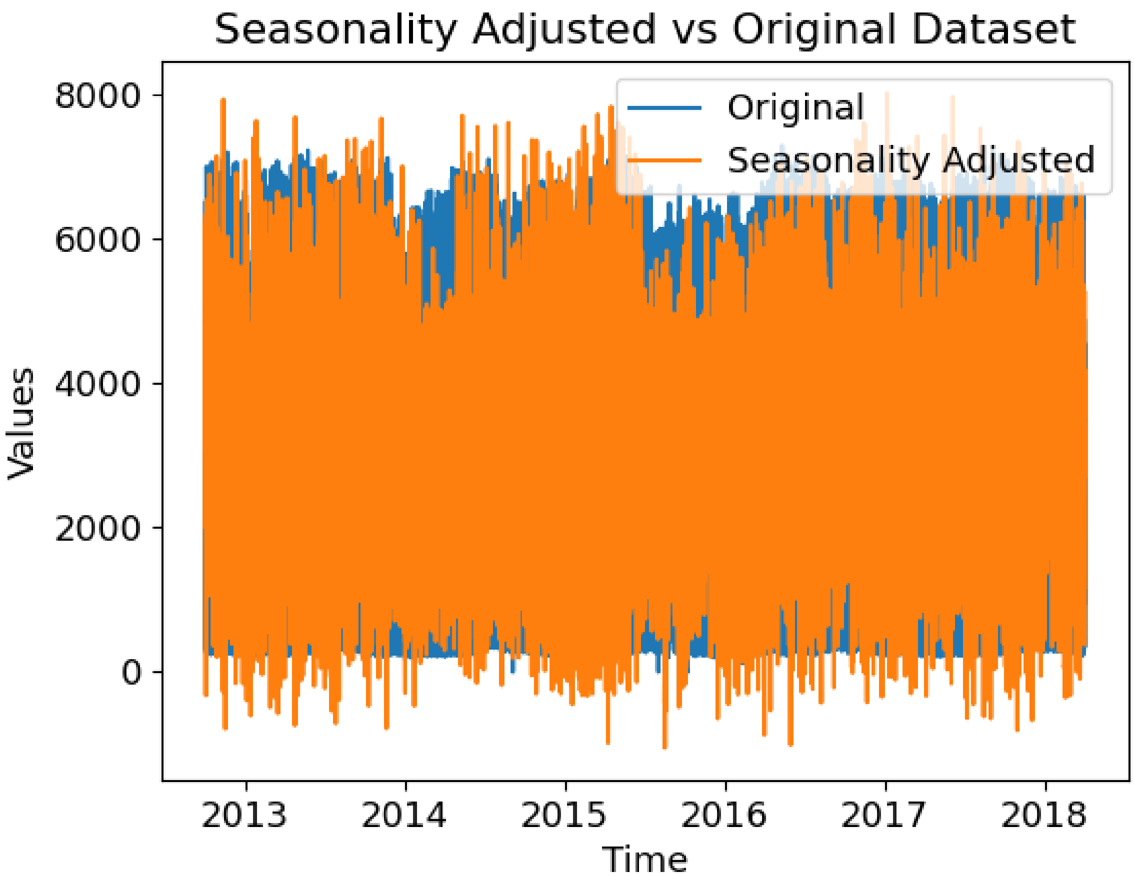


The Generalized Partial Autocorrelation Function (GPAC) indicates an ARMA (2,0)

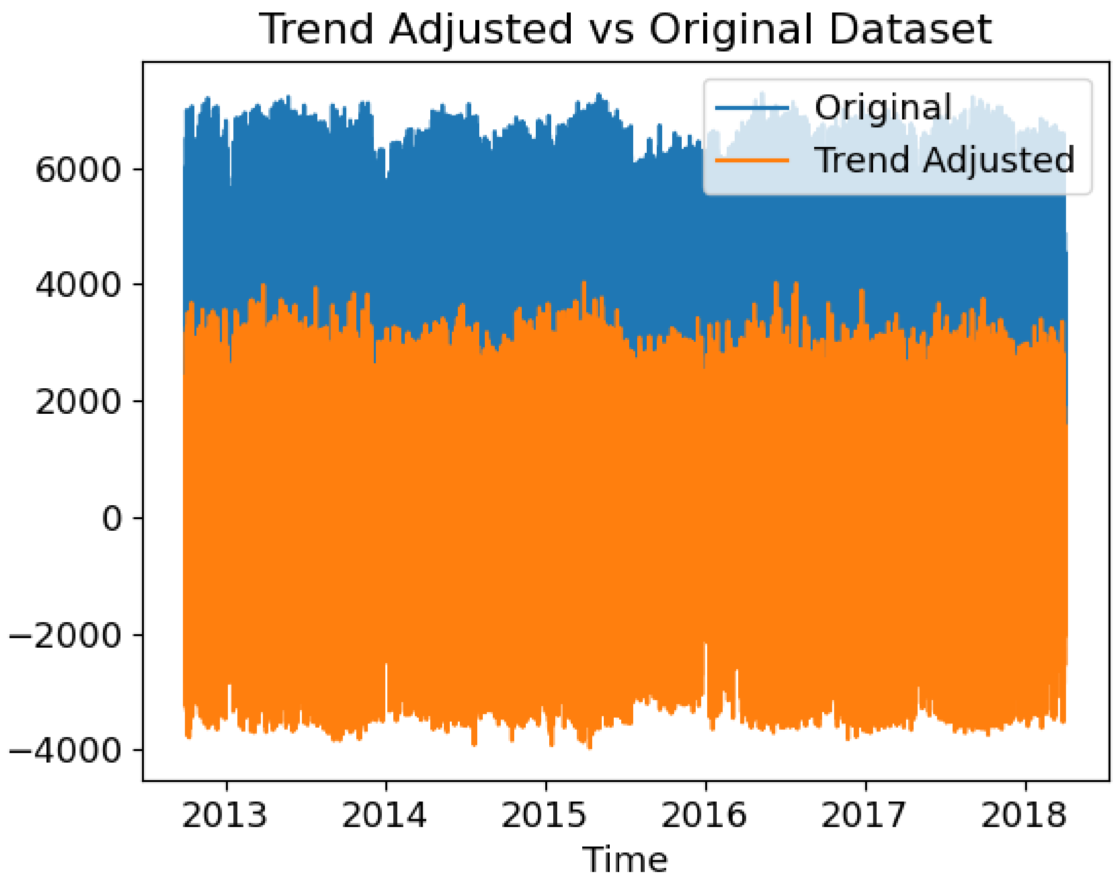
Time Series Decomposition with STL (Seasonal and Trend Decomposition using Loess)



The additive decomposition will be used to decompose the traffic volume because the seasonal variation is relatively constant over time.



The strength of seasonality for this data set is: 0.6636 which indicates about 66.4% seasonality in traffic volume. This seasonal effect is significant as it is beyond 50% of strength.



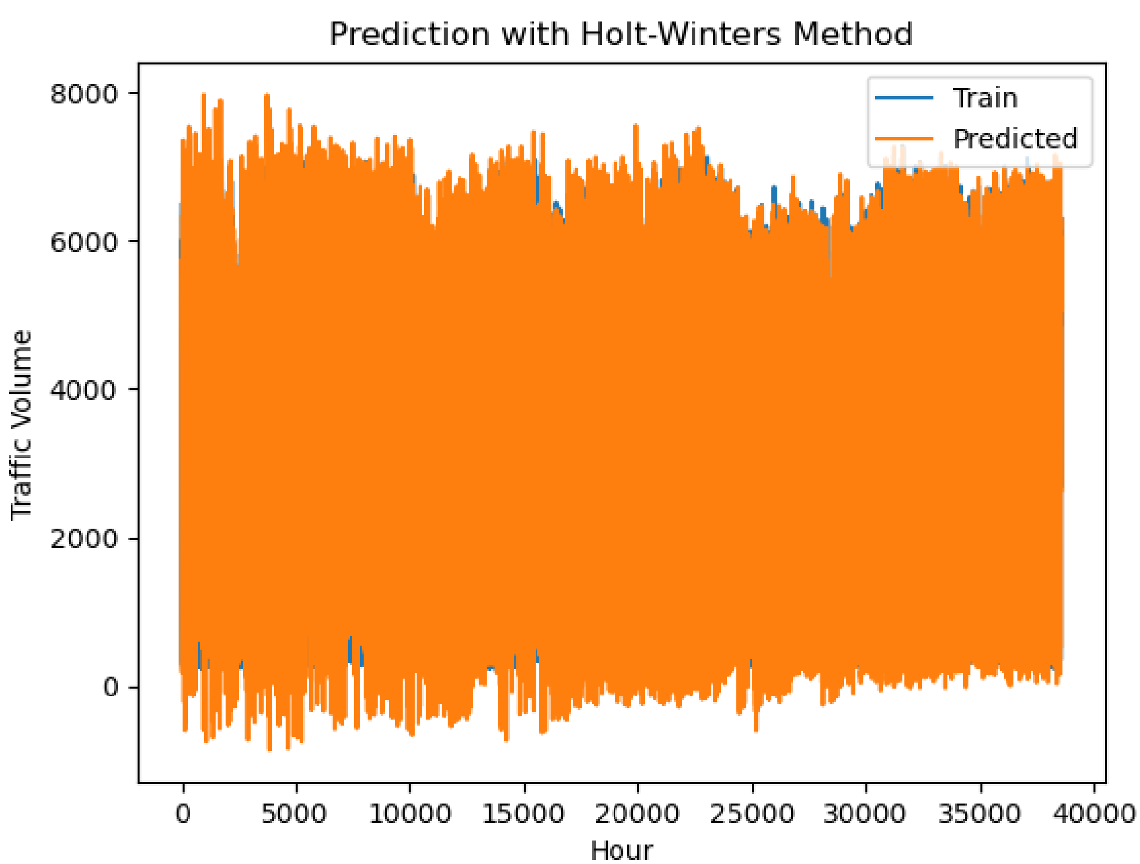
The strength of trend for this data set is: 0.4063 which indicates about 41% trend in traffic volume. This is less than 50% and it shows a weak trend.

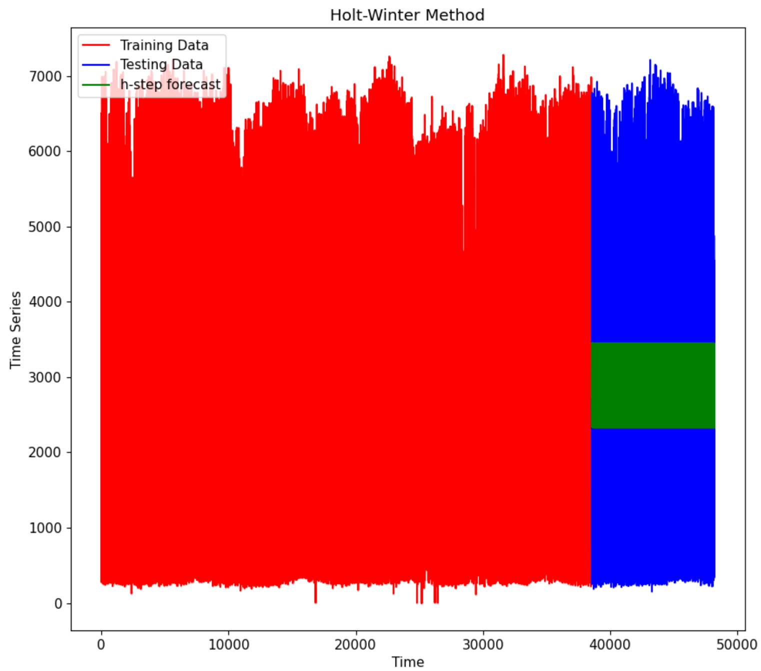
Holt-Winters

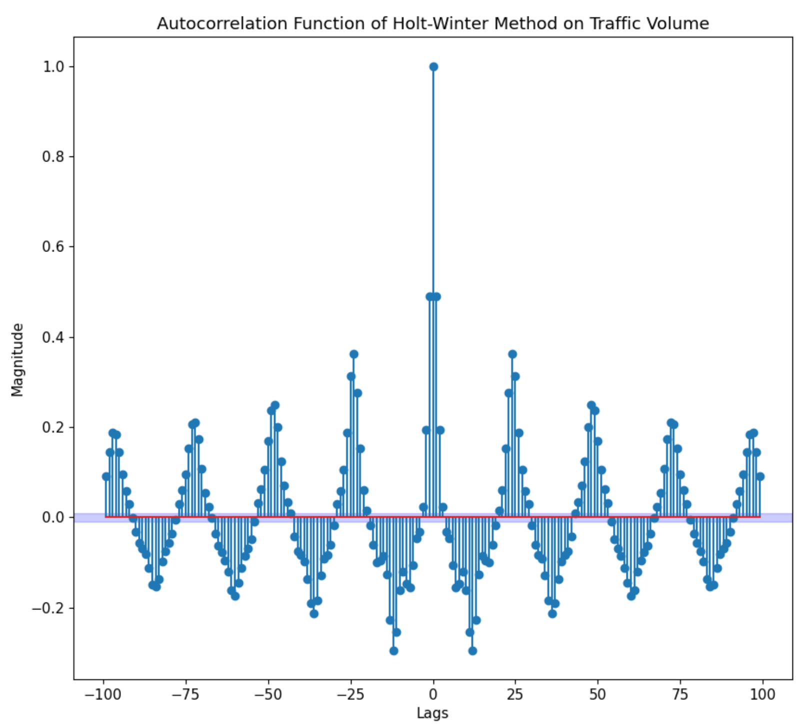
Now we will take a look at Holt-Winters method of forecasting as this can handle the trend and seasonality.

```
Train MSE: 867741.7464269776
Test MSE: 4171421.057763724
Variance: Prediction variance: 867741.7438812318 Forecast variance: 3971003.0393295153
Ratio between variance of prediction & forecast: 0.21851953657223736
Ljung-Box test:
         lb_stat  lb_pvalue
100  76652.905299        0.0
```

The mean squared error for the forecast is way higher than the prediction mean squared error. The Q-value's p-value is indicating a not-white residual.







Also the ACF of residual errors is not white.

The Holt-Winters does not capture the variation of this particular dataset as we see the ratio between prediction and forecast variance to be very low 0.2185 (22% variation explained).

Feature Selection with Backward Selection

Check for collinearity:

```
            Singular Values
temp           3.156165e+09
rain_1h        9.669609e+07
snow_1h        5.747731e+07
clouds_all     3.211887e+00
Condition number for X is 31347.271140857665
```

The least singular value 3.2 is closer to zero compared to the other values. This indicate presence of co-linear feature(s). The condition number indicates severe degree of co-linearity with condition number greater than 1000.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.022
Model:                            OLS   Adj. R-squared:                  0.022
Method:                 Least Squares   F-statistic:                     219.8
Date:                Wed, 13 Apr 2022   Prob (F-statistic):          7.54e-187
Time:                        23:28:21   Log-Likelihood:            -3.4724e+05
No. Observations:               38563   AIC:                         6.945e+05
Df Residuals:                   38558   BIC:                         6.945e+05
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -2880.8829    217.539    -13.243      0.000   -3307.265   -2454.501
temp          21.1835      0.765     27.685      0.000      19.684      22.683
rain_1h        0.1381      0.200      0.689      0.491      -0.254       0.531
snow_1h      389.8256   1099.078      0.355      0.723   -1764.396    2544.047
clouds_all     3.6388      0.260     13.976      0.000       3.129       4.149
==============================================================================
```

With a constant vector of 1 added, the model's Adjusted R-squared value is very low.

```
                            OLS Regression Results
=======================================================================================
Dep. Variable:                      y   R-squared (uncentered):                   0.733
Model:                            OLS   Adj. R-squared (uncentered):              0.733
Method:                 Least Squares   F-statistic:                          2.649e+04
Date:                Wed, 13 Apr 2022   Prob (F-statistic):                        0.00
Time:                        23:30:04   Log-Likelihood:                     -3.4732e+05
No. Observations:               38563   AIC:                                  6.947e+05
Df Residuals:                   38559   BIC:                                  6.947e+05
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
temp          11.0790      0.058    192.455      0.000      10.966      11.192
rain_1h        0.1656      0.201      0.825      0.409      -0.228       0.559
snow_1h      122.2823   1101.375      0.111      0.912   -2036.440    2281.005
clouds_all     3.0019      0.256     11.705      0.000       2.499       3.505
==============================================================================
```

Removing the constant made the model jump to 73% adjusted r-squared which is very good.

Next we remove feature snow_all with p>|t| 0.912:

```
                            OLS Regression Results
=======================================================================================
Dep. Variable:                      y   R-squared (uncentered):                   0.733
Model:                            OLS   Adj. R-squared (uncentered):              0.733
Method:                 Least Squares   F-statistic:                          3.532e+04
Date:                Wed, 13 Apr 2022   Prob (F-statistic):                        0.00
Time:                        23:32:24   Log-Likelihood:                     -3.4732e+05
No. Observations:               38563   AIC:                                  6.947e+05
Df Residuals:                   38560   BIC:                                  6.947e+05
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
temp          11.0790      0.058    192.462      0.000      10.966      11.192
rain_1h        0.1656      0.201      0.825      0.409      -0.228       0.559
clouds_all     3.0028      0.256     11.715      0.000       2.500       3.505
==============================================================================
```

The F-statistic improves. Now we remove feature rain_1h with p>|t| 0.409

```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:                      y   R-squared (uncentered):                   0.733
Model:                            OLS   Adj. R-squared (uncentered):              0.733
Method:                 Least Squares   F-statistic:                          5.298e+04
Date:                Wed, 13 Apr 2022   Prob (F-statistic):                        0.00
Time:                        23:34:08   Log-Likelihood:                     -3.4732e+05
No. Observations:               38563   AIC:                                  6.947e+05
Df Residuals:                   38561   BIC:                                  6.947e+05
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
temp          11.0791      0.058    192.464      0.000      10.966      11.192
clouds_all     3.0037      0.256     11.719      0.000       2.501       3.506
==============================================================================
```

The final model has the best F-statistics with p-value and a significant t-value for the two features left (temp & clouds_all).

From the above feature selection with backward selection, features temperature and clouds_all are best for predicting traffic volume.

```
Condition number after Feature Selection for X is 7.410057871927594
```

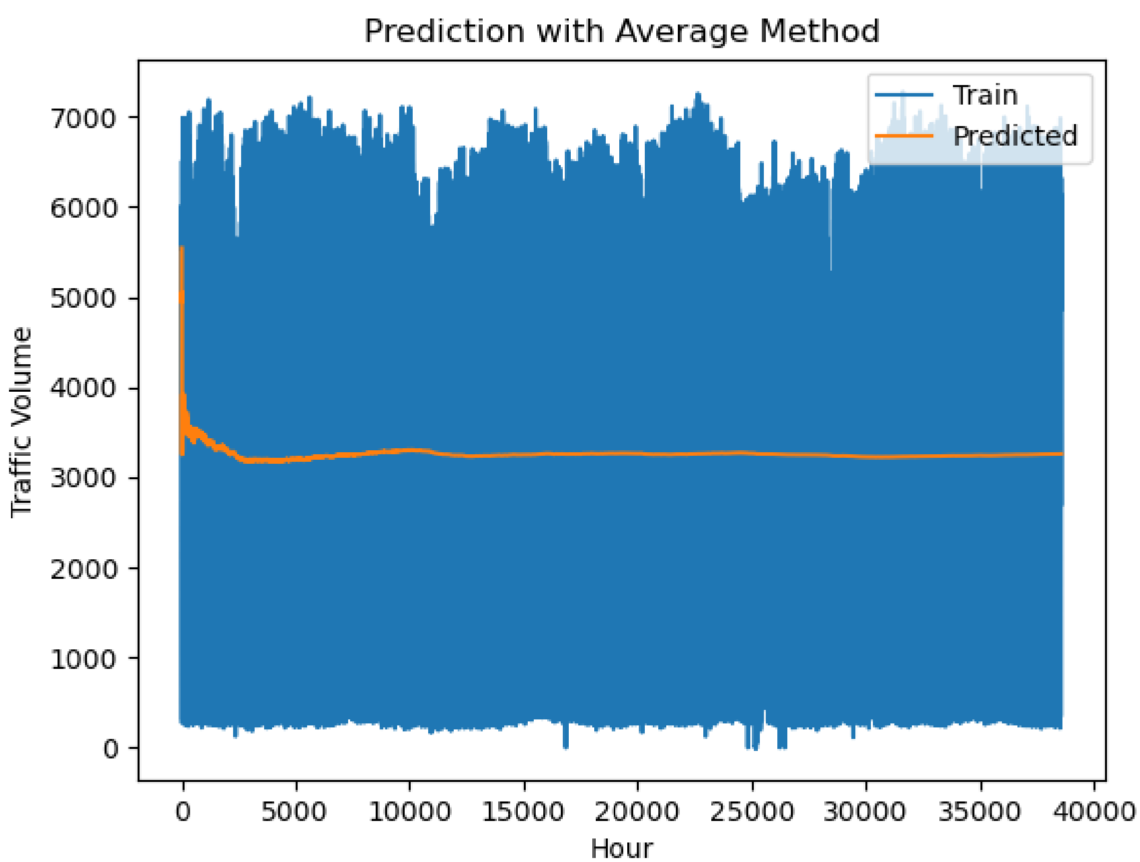
The condition number indicate weak degree of collinearity; with this the features selected using backward selection are valid.

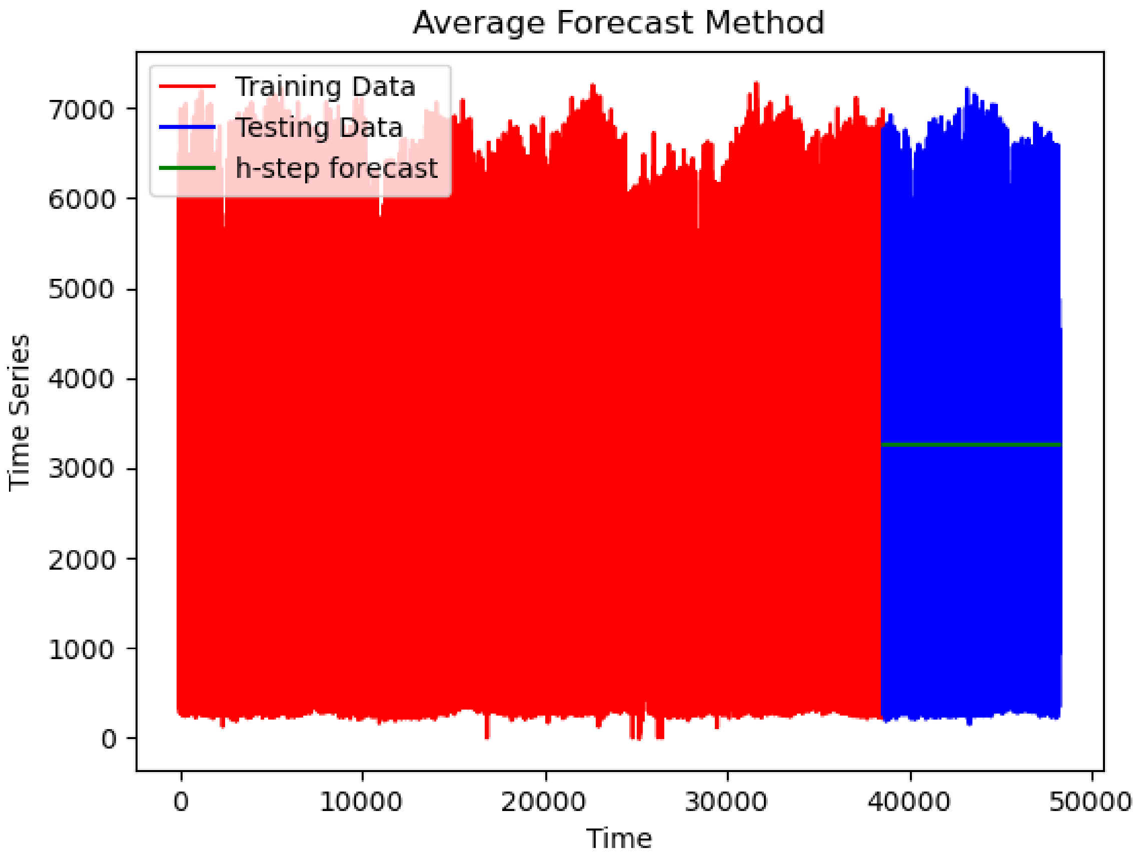
BASE MODELS

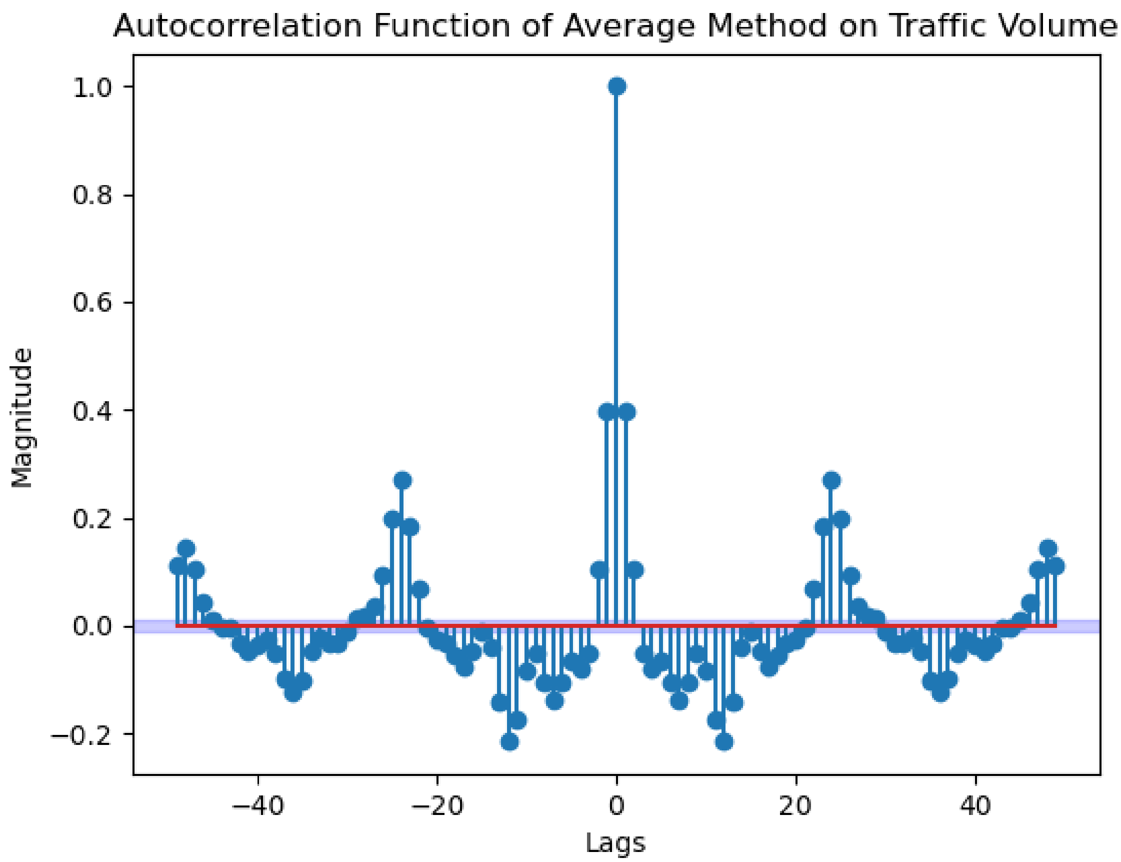
Average Forecast Method

The results from the average method on this dataset is presented below using python:

```
Train MSE: 3967564.65
Test MSE: 3871753.24
Ljung-Box test:
           lb_stat  lb_pvalue
100  172321.014358        0.0
```




Average Forecast Method
Training Data
Testing Data
h-step forecast
Time Series
Time
0
1000
2000
3000
4000
5000
6000
7000
0
10000
20000
30000
40000
50000

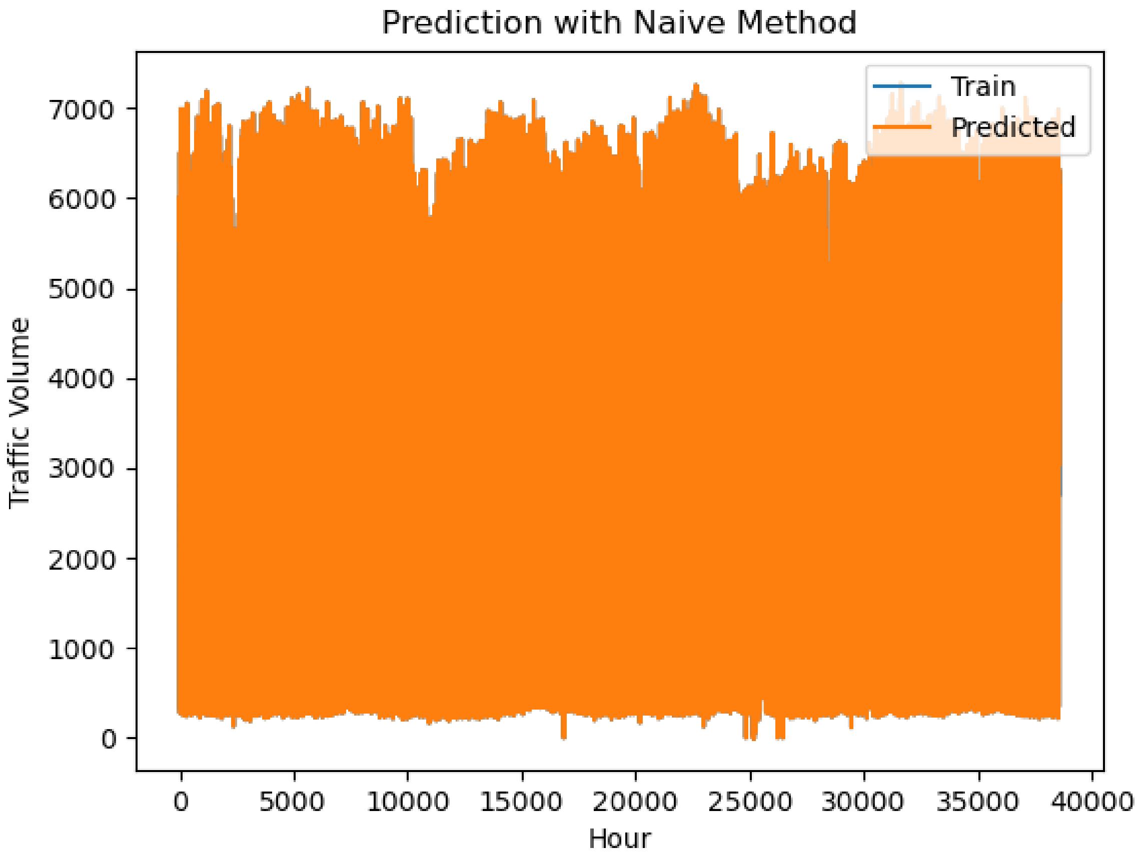


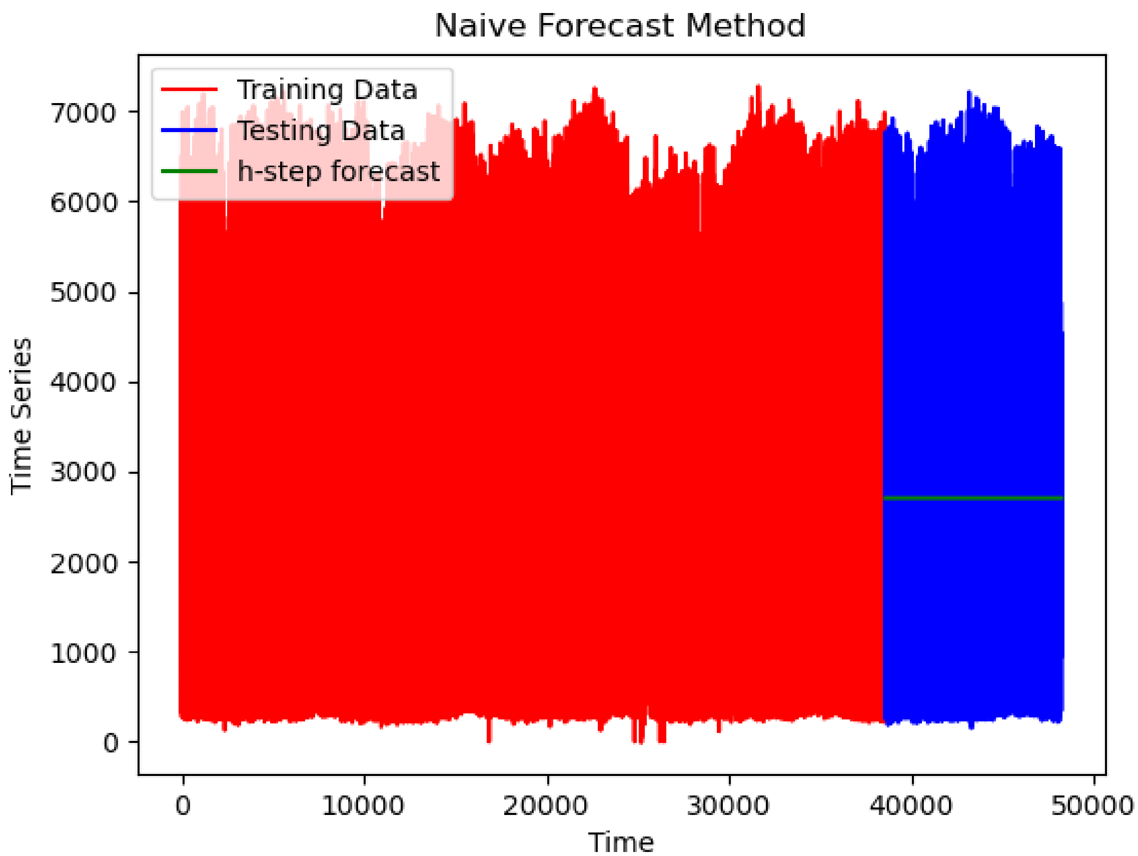
The residual is not white with p-value of Ljung Box test less than 0.05 and many correlated lags in the ACF of residual above. It can be noticed that both the train and test set have close MSE value.

Naïve Forecast Method

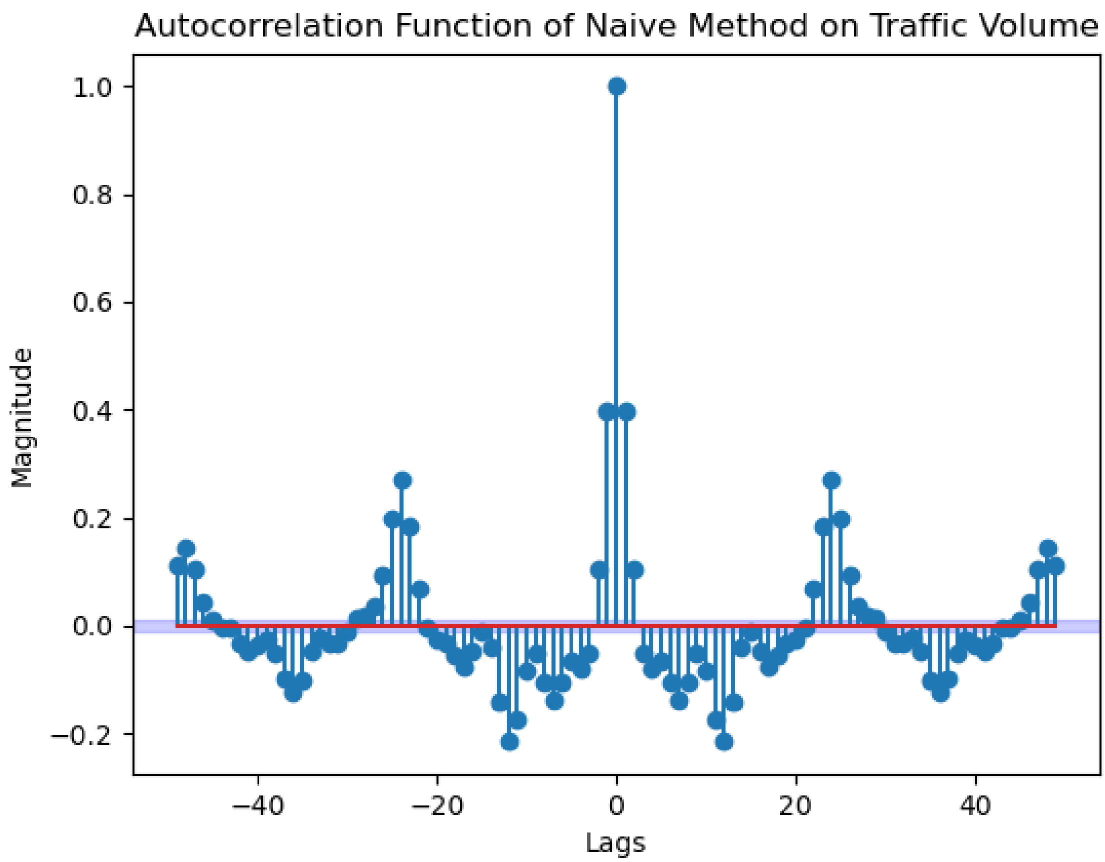
The results from the Naïve method on this dataset is presented below using python:

```
Train MSE: 725728.7022198018
Test MSE: 4175686.816823981
Ljung-Box test:
          lb_stat  lb_pvalue
100  26059.182387        0.0
```


Prediction with Naive Method
Train
Predicted
7000
6000
5000
4000
3000
2000
1000
0
Traffic Volume
0
5000
10000
15000
20000
25000
30000
35000
40000
Hour



Naive Forecast Method
Training Data
Testing Data
h-step forecast
7000
6000
5000
4000
3000
2000
1000
0
Time Series
0
10000
20000
30000
40000
50000
Time

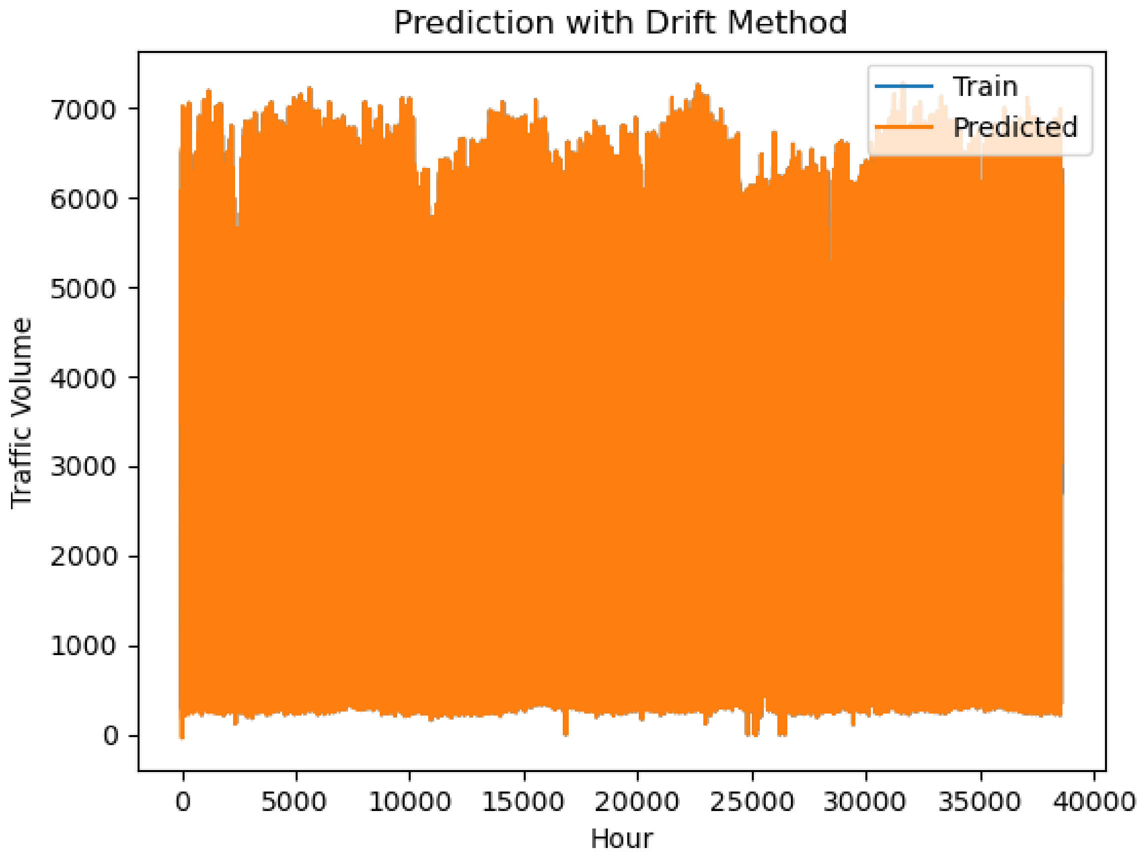


The residual is not white with p-value of Ljung Box test less than 0.05 and many correlated lags in the ACF of residual above.

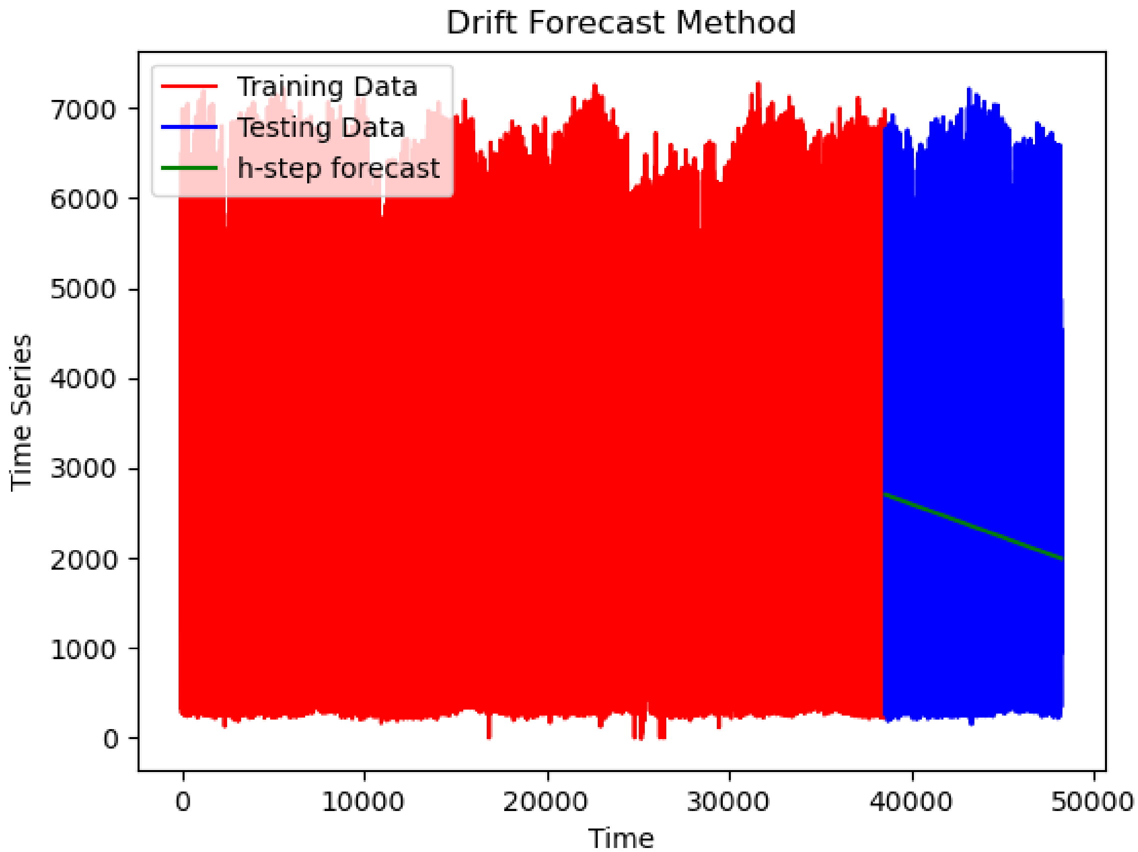
Drift Forecast Method

The results from the Drift method on this dataset is presented below using python:

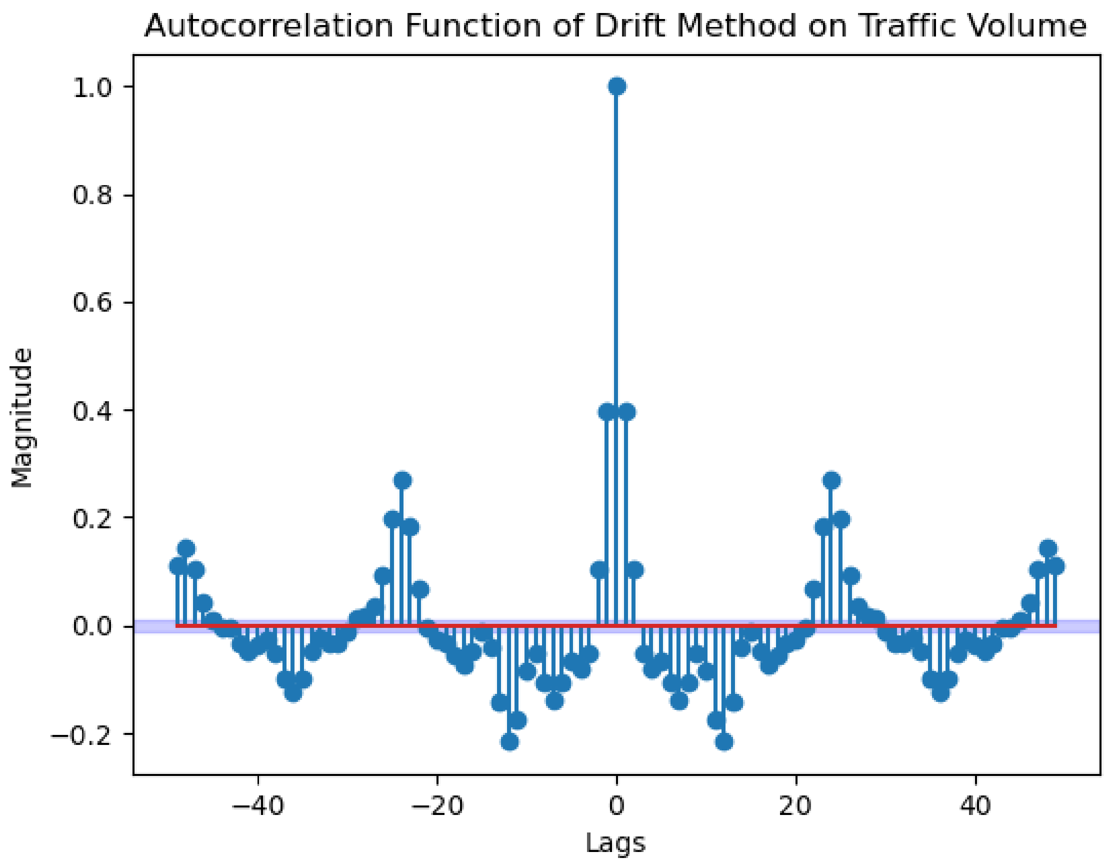
```
Train MSE: 725946.6029443607
Test MSE: 4761470.679660877
Ljung-Box test:
          lb_stat  lb_pvalue
100  26064.655464        0.0
```



Prediction with Drift Method
Train
Predicted
Traffic Volume
Hour
0
1000
2000
3000
4000
5000
6000
7000
0
5000
10000
15000
20000
25000
30000
35000
40000



Drift Forecast Method
Training Data
Testing Data
h-step forecast
7000
6000
5000
4000
3000
2000
1000
0
Time Series
0
10000
20000
30000
40000
50000
Time

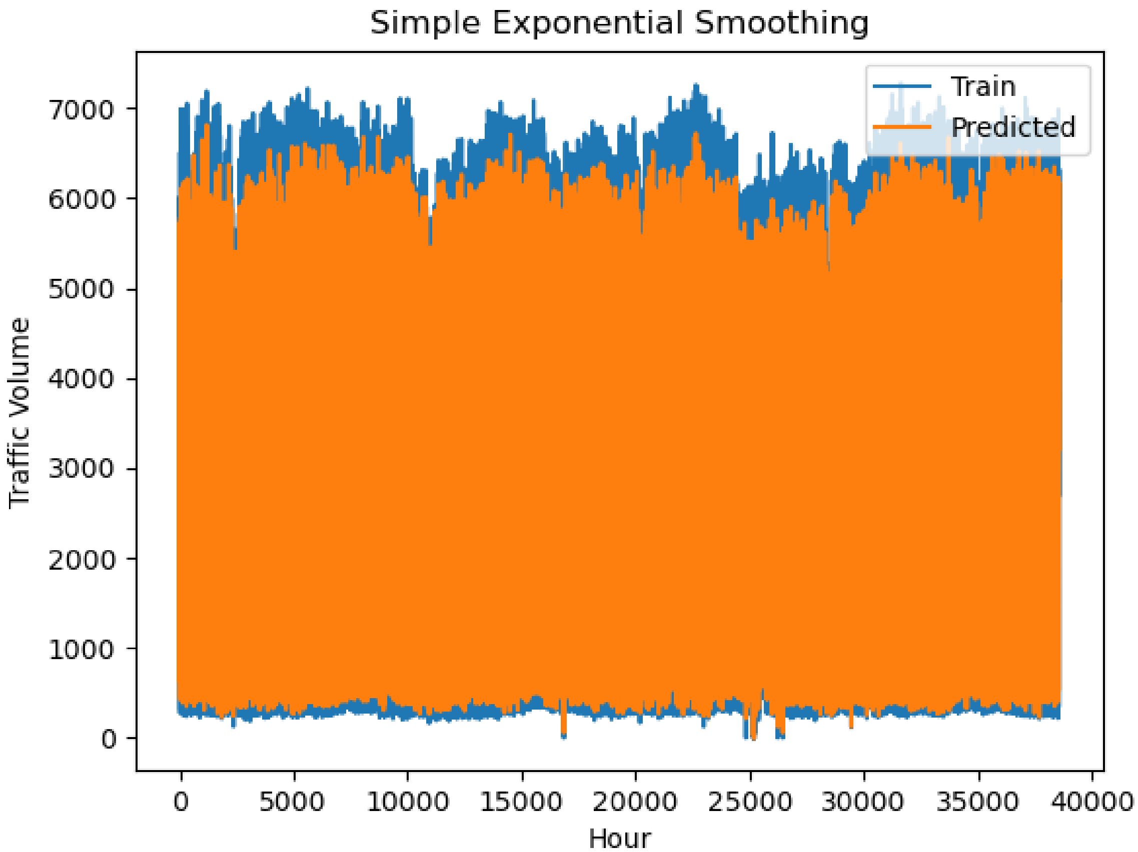


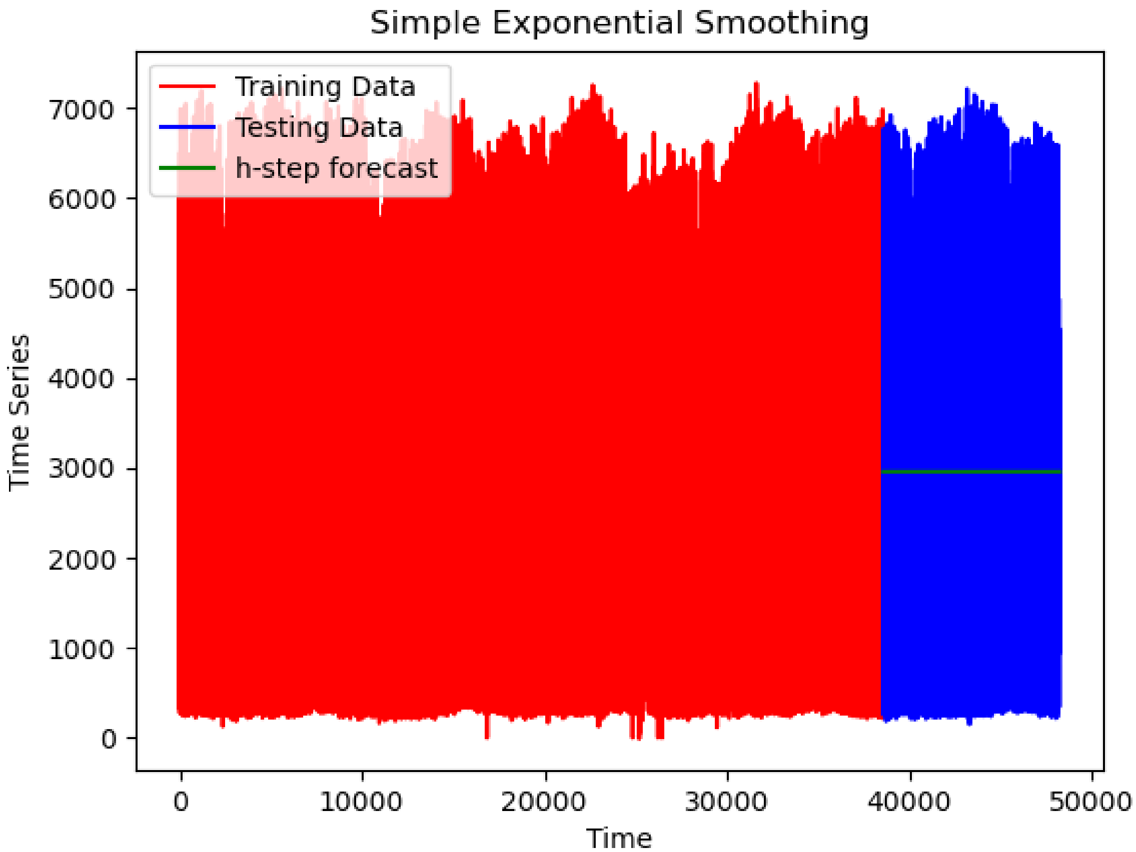
The residual is not white with p-value of Ljung Box test less than 0.05 and many correlated lags in the ACF of residual above.

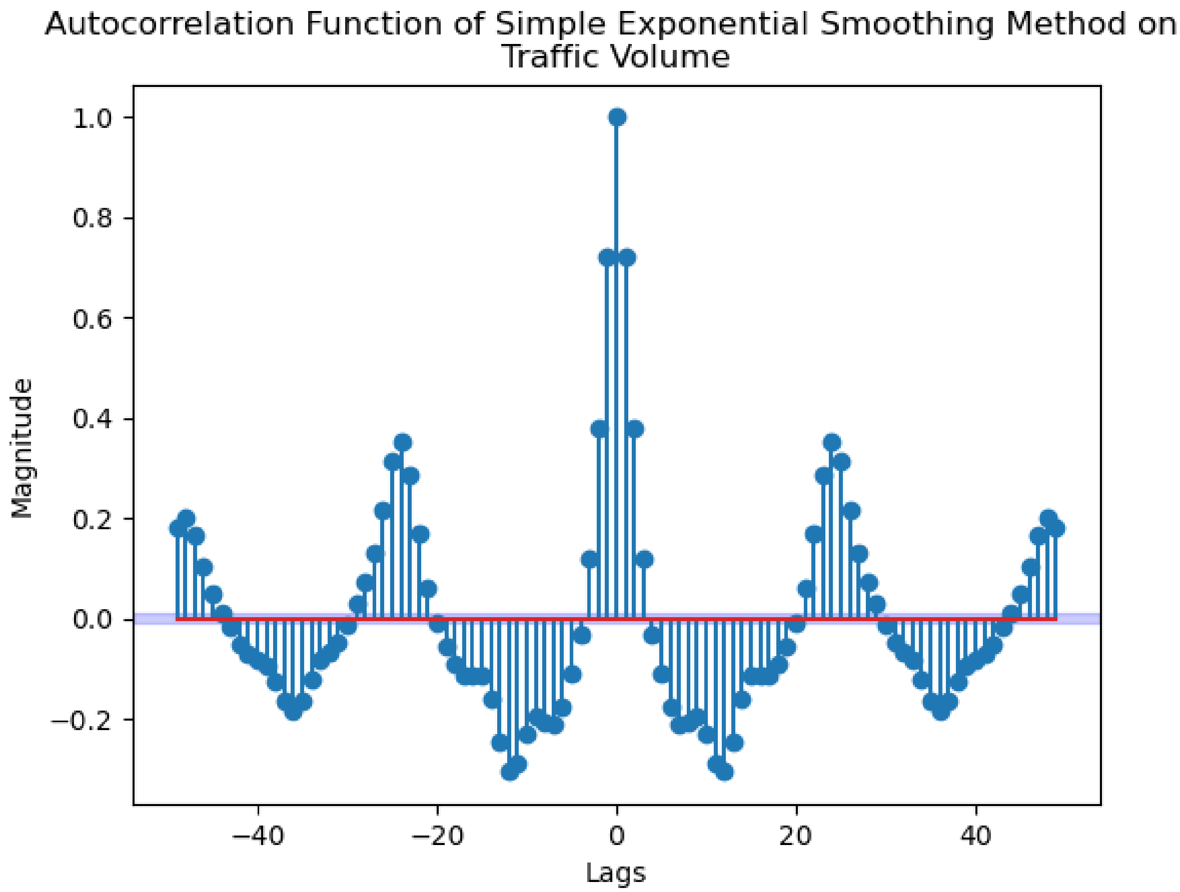
Simple Exponential Smoothing

The results from the Simple Exponential Smoothing method on this dataset is presented below using python:

```
Train MSE: 1370194.370994163
Test MSE: 3959576.6660246924
Ljung-Box test:
          lb_stat  lb_pvalue
100  81939.395933        0.0
```


Simple Exponential Smoothing
Train
Predicted
Traffic Volume
7000
6000
5000
4000
3000
2000
1000
0
0
5000
10000
15000
20000
25000
30000
35000
40000
Hour


Simple Exponential Smoothing
Training Data
Testing Data
h-step forecast
Time Series
Time
0
1000
2000
3000
4000
5000
6000
7000
0
10000
20000
30000
40000
50000



The residual is not white with p-value of Ljung Box test less than 0.05 and many correlated lags in the ACF of residual above.

Model in-view

	Q	MSE_pred	MSE_f	var_pred_er	var_forecast_er
Holt-Winters mtd	76652.91	867741.7	4171421	867741.7	3971003
Average mtd	172321	3967565	3871753	3967558	3871722
Naive mtd	26059.18	725728.7	4175687	725728.7	3871722
Drift mtd	26064.66	725946.6	4761471	725946.4	3939709
SES mtd	81939.4	1370194	3959577	1370194	3871722

Comparing base models above, Average method performed better amongst all.

Multiple Linear Regression

We will be using the two features from our backward selection which are temp and clouds_all to predict traffic volume between Minneapolis and St Paul, MN.

First, we will check for collinearity among this two features using singular value decomposition.

	Singular Values
temp	3.938931e+09
clouds_all	7.229093e+07

There is no presence of collinearity among the two features because their singular values are far from zero.

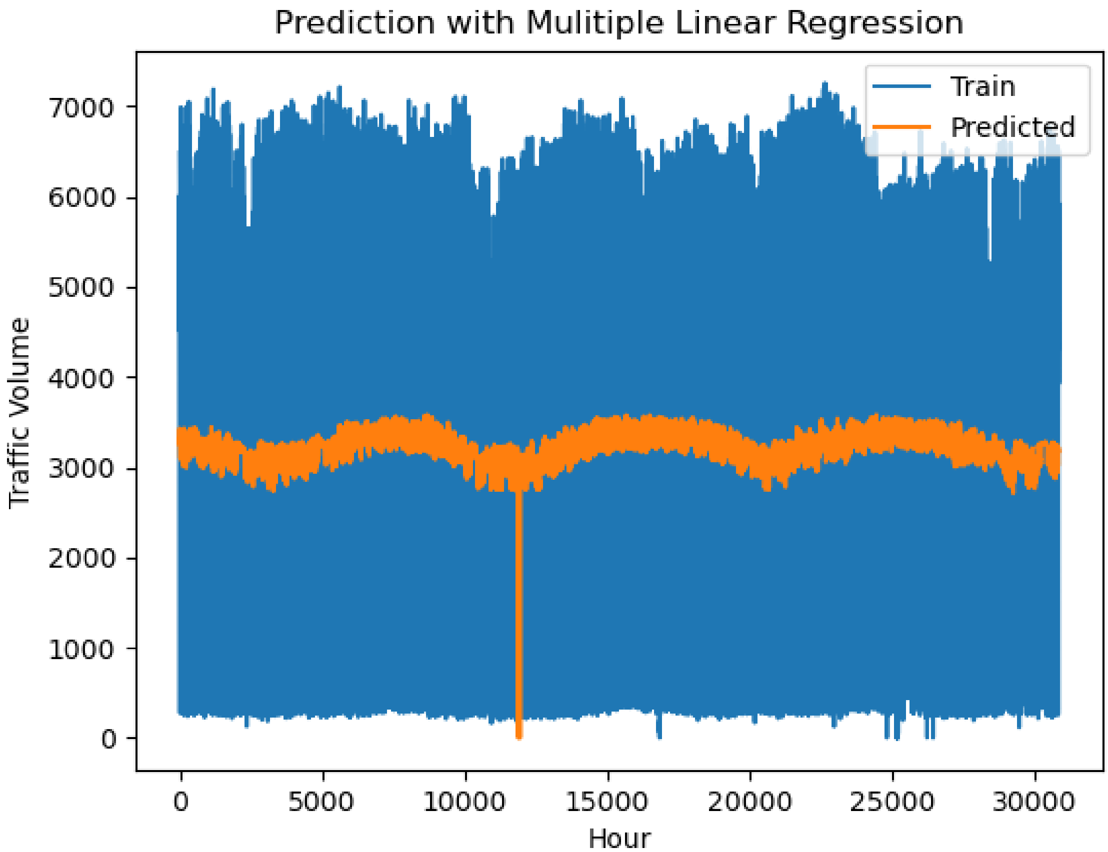
```
Condition number for X is 7.381545601028764
```

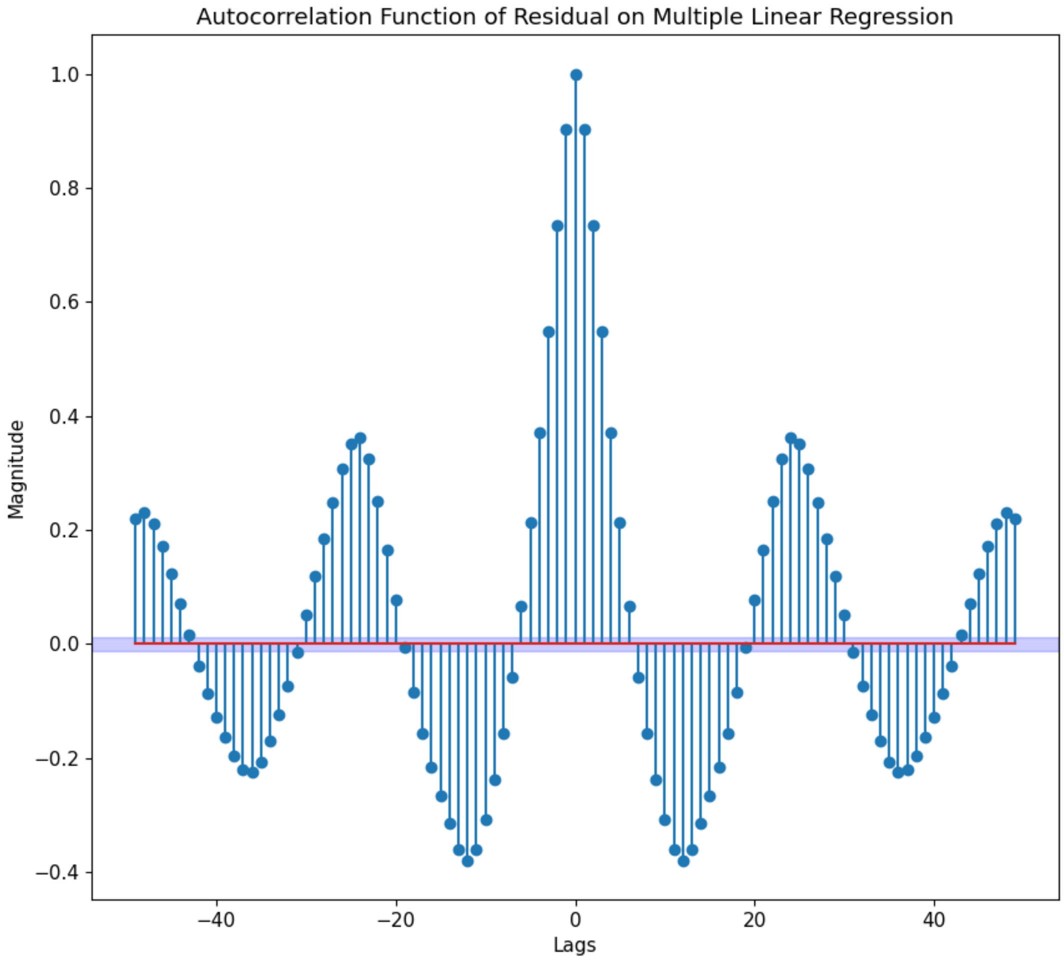
The condition number which is less than 100 indicate a very weak degree of collinearity. With this we can fit the regression model.

```
                            OLS Regression Results
=======================================================================================
Dep. Variable:                      y   R-squared (uncentered):                   0.733
Model:                            OLS   Adj. R-squared (uncentered):              0.733
Method:                 Least Squares   F-statistic:                          5.298e+04
Date:                Mon, 02 May 2022   Prob (F-statistic):                        0.00
Time:                        23:31:00   Log-Likelihood:                     -3.4732e+05
No. Observations:               38563   AIC:                                  6.947e+05
Df Residuals:                   38561   BIC:                                  6.947e+05
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
temp          11.0791      0.058    192.464      0.000      10.966      11.192
clouds_all     3.0037      0.256     11.719      0.000       2.501       3.506
==============================================================================
Omnibus:                   104860.012   Durbin-Watson:                   0.186
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2696.478
Skew:                          -0.080   Prob(JB):                         0.00
Kurtosis:                       1.714   Cond. No.                         7.41
==============================================================================
```

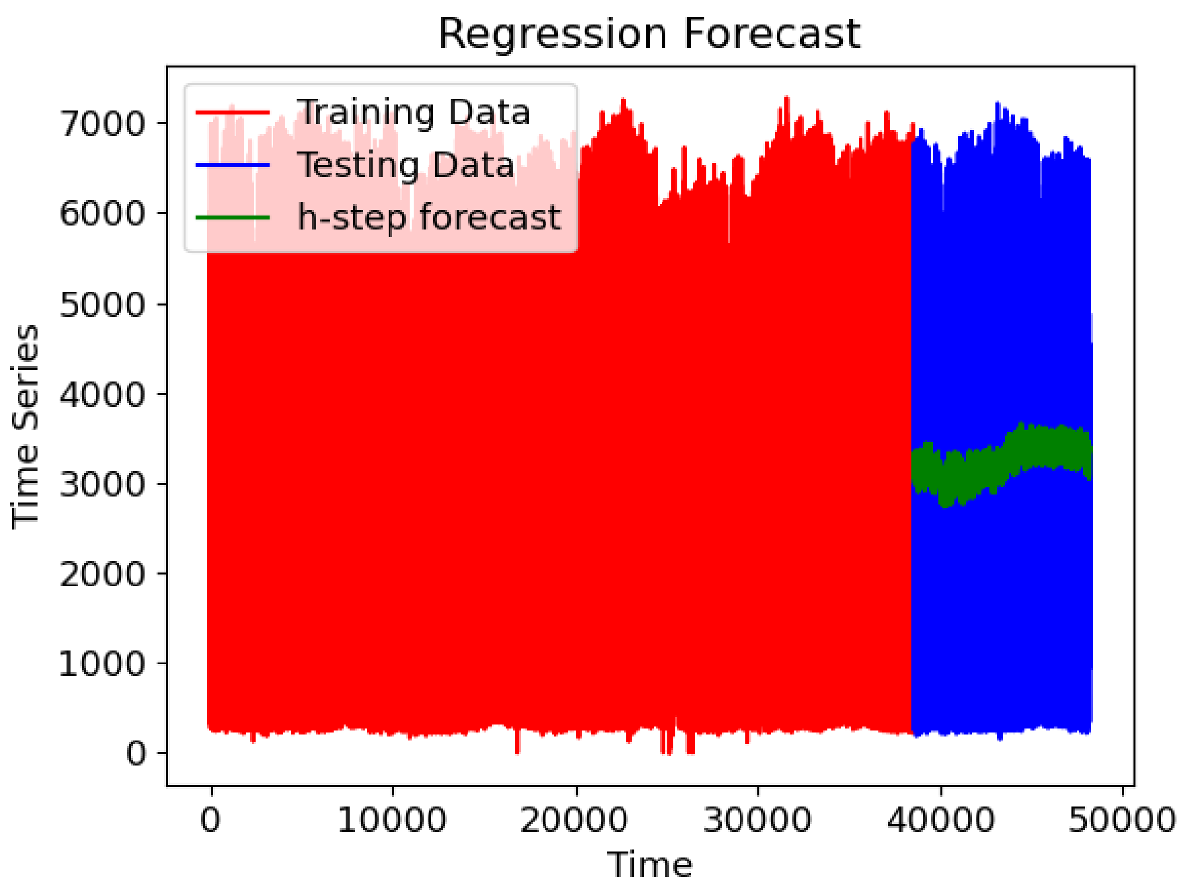
The probability of t-values is significant with values less than 0.05, this indicates that the features are important in this model.

The probability of F-statistic is significant which tells us that this model is better than a model with constant-only feature.


Prediction with Mulitiple Linear Regression
Train
Predicted
Traffic Volume
Hour
0
1000
2000
3000
4000
5000
6000
7000
0
5000
10000
15000
20000
25000
30000



The ACF of residuals for the multiple linear regression shows lots of correlation in lags; hence the residuals are not white.



Residual Analysis for Multiple Linear Regression

```
Ljung-Box test:
           lb_stat  lb_pvalue
100  168259.218223        0.0
```

The probability of the Q value is less than 0.05; this indicate that the residual is not white.

```
Ljung-Box test:
           lb_stat  lb_pvalue
100  168259.218223        0.0
MSE of prediction: 3895768.9828714193
 R Squared 0.7331703988892266
Adjusted R Squared 0.7331565595385298
```

The Adjusted R-squared indicates that the model explains about 73% information of the dataset.

Table: Model in-view

	Q	MSE_PRED	MSE_F	VAR_PRED_ER	VAR_FORECAST_ER	VARIANCE RATIO
HOLT-WINTERS MTD	76652.91	867741.7	4171421	867741.7	3971003	0.21852
AVERAGE MTD	172321	3967565	3871753	3967558	3871722	1.024753
NAIVE MTD	26059.18	725728.7	4175687	725728.7	3871722	0.187443
DRIFT MTD	26064.66	725946.6	4761471	725946.4	3939709	0.184264
SES MTD	81939.4	1370194	3959577	1370194	3871722	0.353898
MULTIPLE LR	134071.1	3883146	3953303	3898774	3988637	0.97747

From the table above Average method performed better than the others with the least forecast MSE which improved from the prediction MSE. Also, the variance ratio for average method is closer to one.

The multiple linear regression is noticed to be the second best performing model with variance ratio close to one and least second least MSE amongst others.

ARMA Process

Recalling from the GPAC our tentative model will be ARMA(2,0). We will now feed this to Levenberg-Marquardt Algorithm for parameter estimation.

Result from simulation:

```
Parameters from LM Algorithm:
 [[-1.39826651]
 [ 0.43216121]]
```

The ARMA model can now be expressed as:

$$y_t - 1.39826651y_{t-1} + 0.43216121y_{t-2} = e_t$$

```
Covariance Matrix
 [[ 1.68715766e-05 -1.64723061e-05]
 [-1.64723061e-05  1.68716271e-05]]
```

```
Confidence interval:
 [-1.40648152],[-1.39005151]
Confidence interval:
 [0.42394619],[0.44037623]
```

The estimated parameters are found in their respective range of values of confidence interval.

```
Zero: []
Pole: [0.93709565 0.46117086]
```

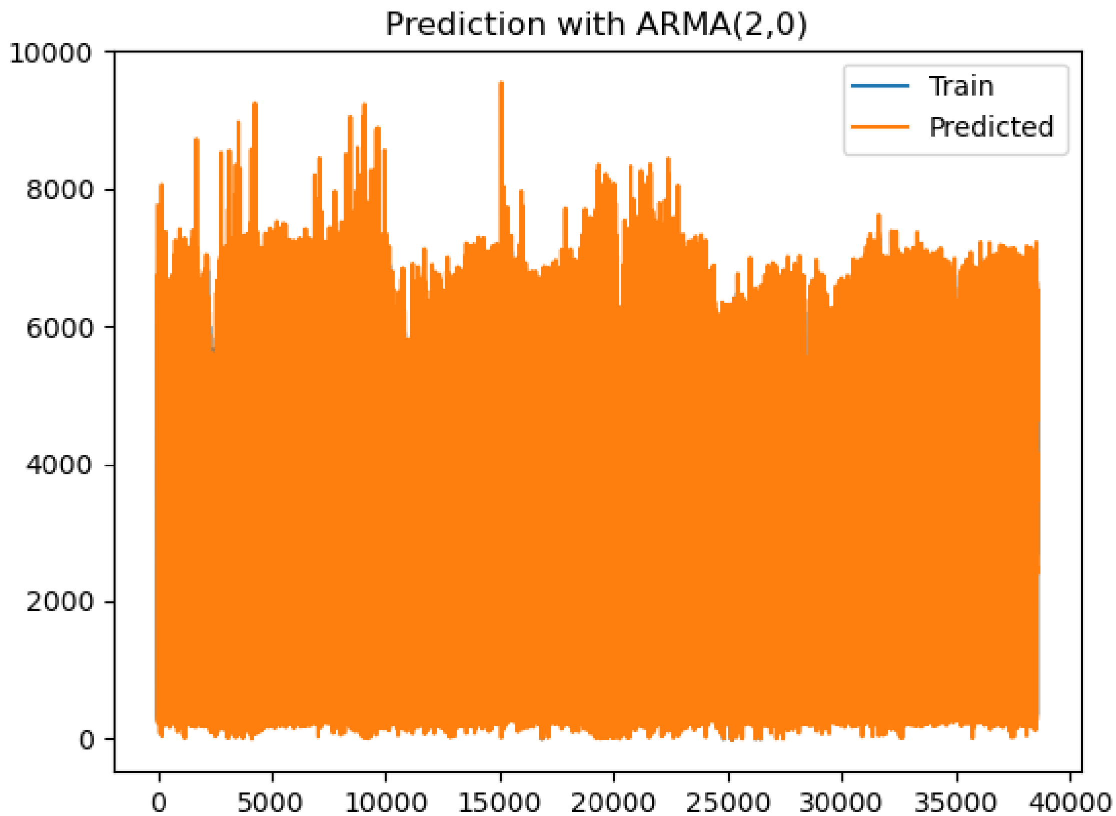
No evidence of zero-pole cancellation from the roots of coefficients above.

Using Statsmodels package in python we get similar results:

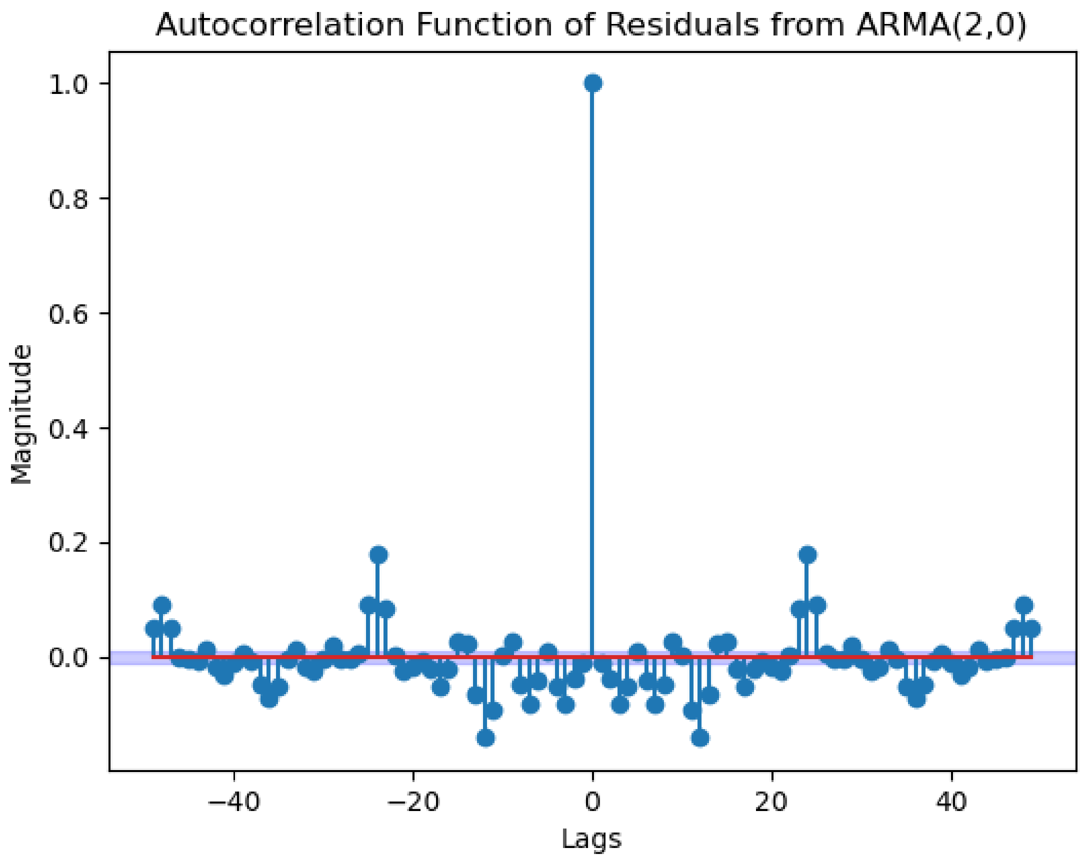
```
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                48204
Model:                     ARMA(2, 0)   Log Likelihood             -387127.576
Method:                       css-mle   S.D. of innovations            744.008
Date:                Tue, 19 Apr 2022   AIC                         774261.151
Time:                        21:53:17   BIC                         774287.501
Sample:                             0   HQIC                        774269.419

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1.y        1.3988      0.004    340.665      0.000       1.391       1.407
ar.L2.y       -0.4327      0.004   -105.378      0.000      -0.441      -0.425
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0672           +0.0000j            1.0672            0.0000
AR.2            2.1656           +0.0000j            2.1656            0.0000
-----------------------------------------------------------------------------
```

```
The AR Coefficient a1: -1.3988334825725308
The AR Coefficient a2: 0.4327069153246223
```

Results from ARMA(2,0)

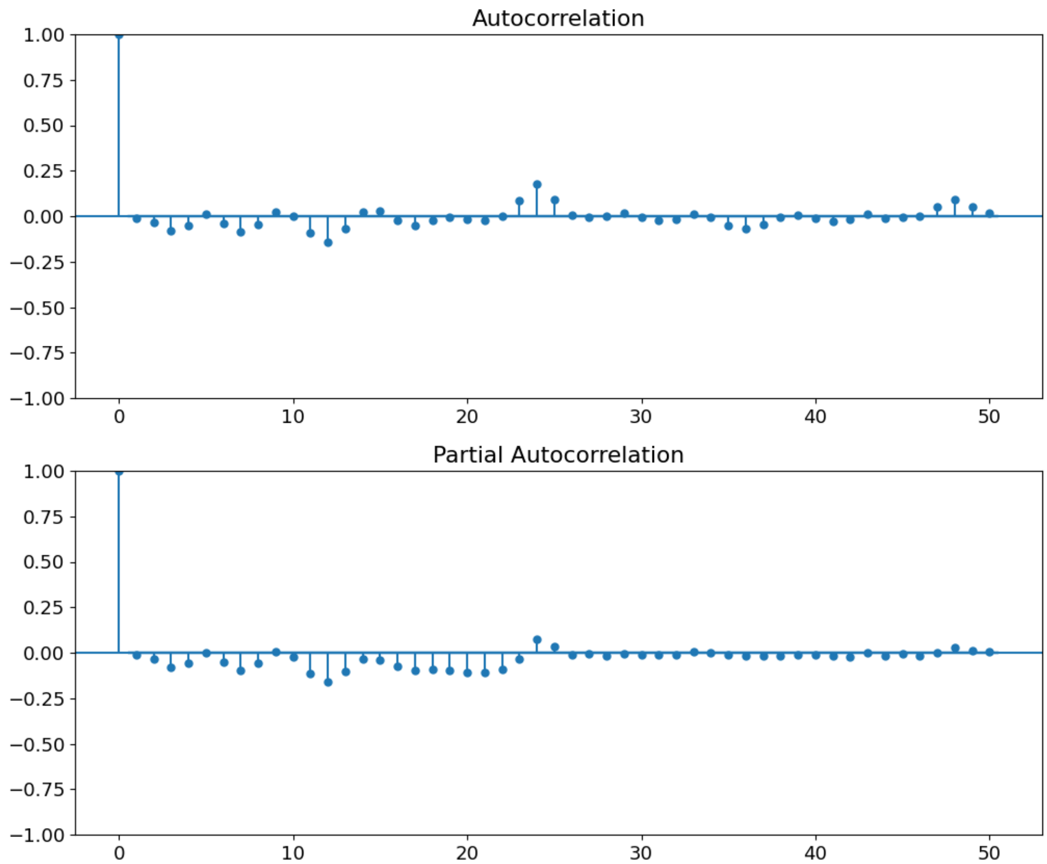


The plot above shows the model captured the training data with some errors.

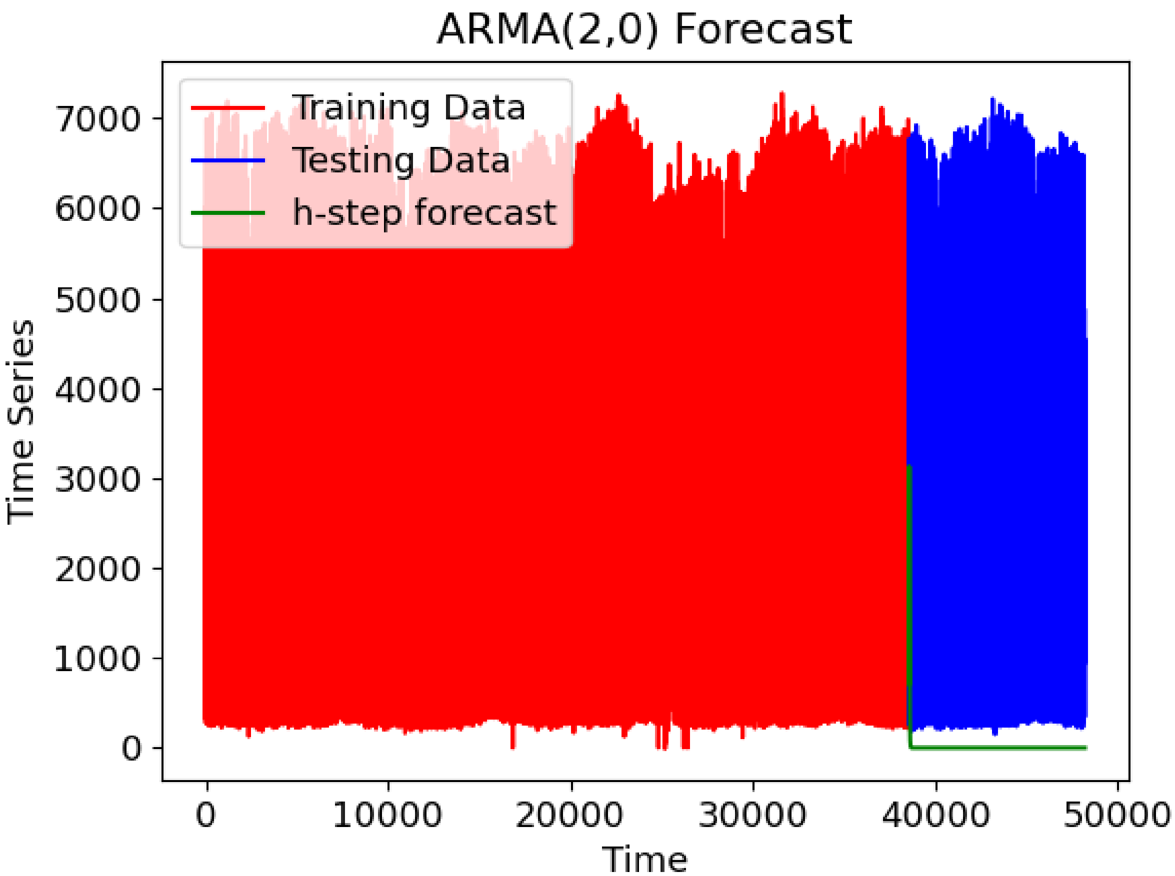



```
Ljung-Box test:
        lb_stat  lb_pvalue
50  5192.967398        0.0
```

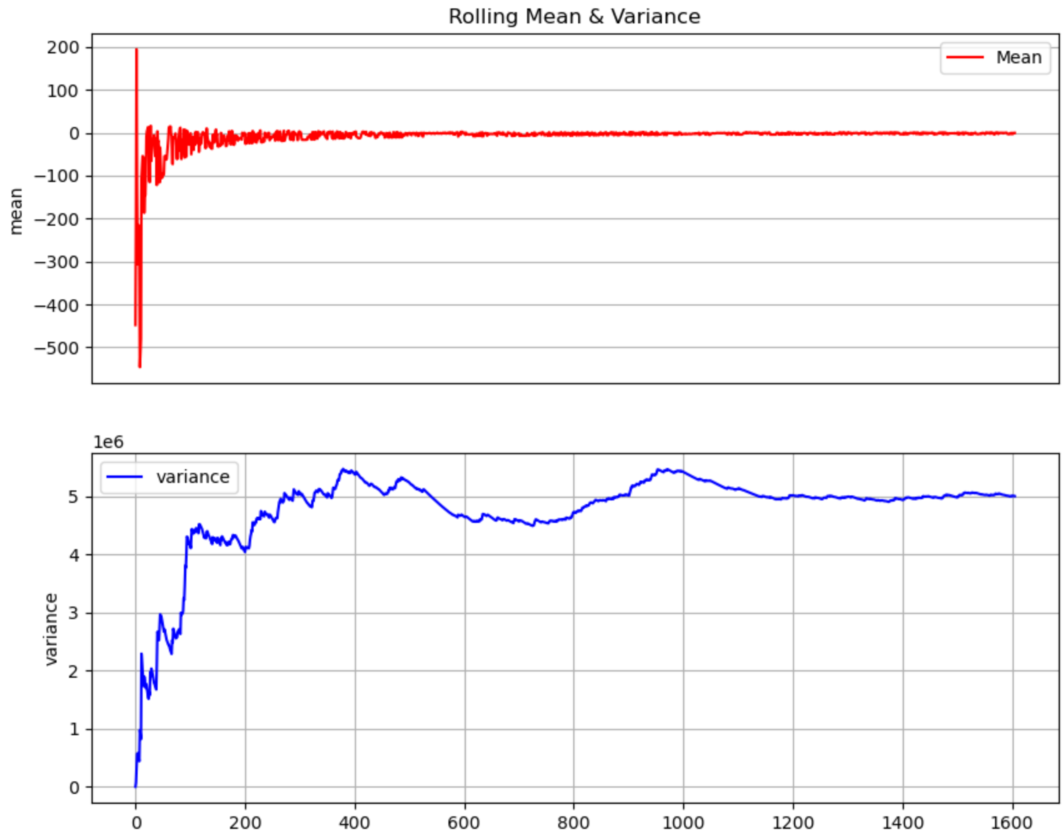
The Q value above with p-value less than 0.05 indicates that the residual is not white noise.



ACF/PACF from residual of ARMA(2,0)


ARMA(2,0) Forecast
Training Data
Testing Data
h-step forecast
7000
6000
5000
4000
3000
2000
1000
0
Time Series
0
10000
20000
30000
40000
50000
Time

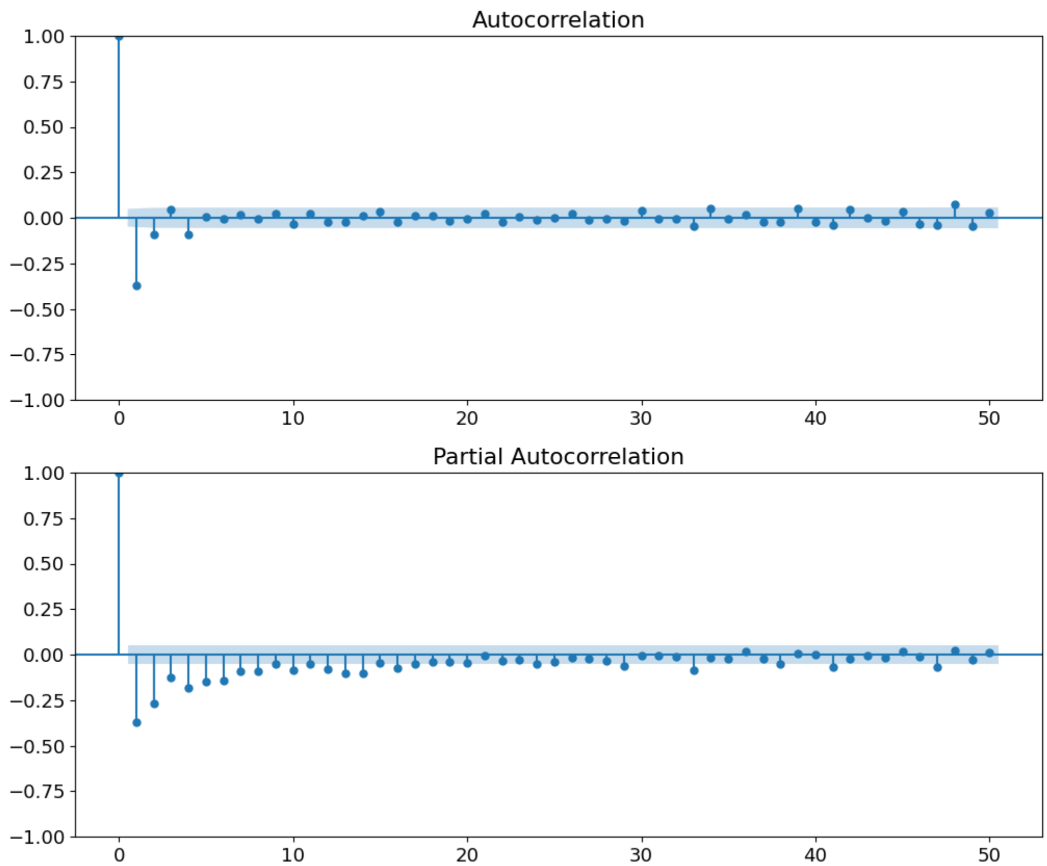
Experimenting with SARIMA $(0,0,1)_{24}$



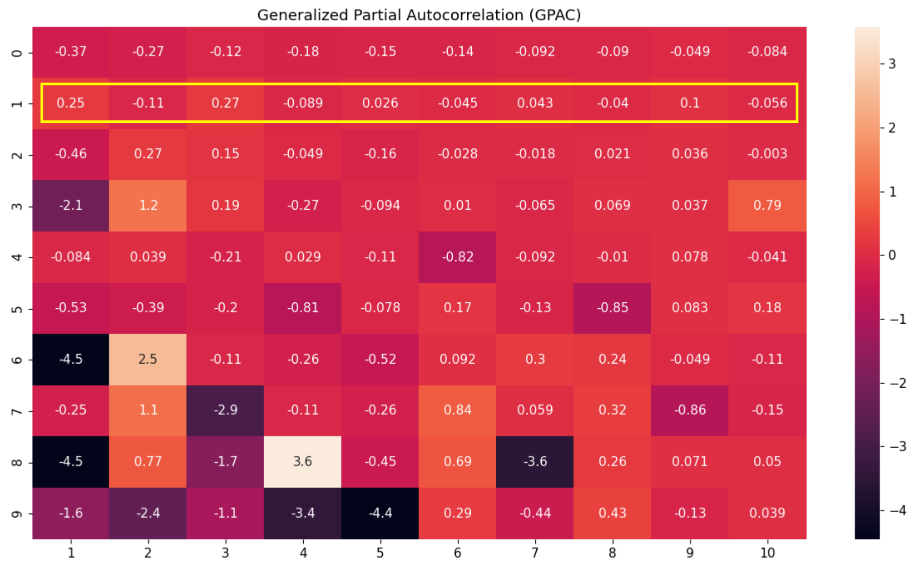

```
p-value: 0.000000
Critical Values:
    1%: -3.434
    5%: -2.863
    10%: -2.568
```

```
Results of KPSS Test:
Test Statistic              0.232222
p-value                     0.100000
LagsUsed                  518.000000
Critical Value (10%)        0.347000
dtype: float64
Test Statistic              0.232222
p-value                     0.100000
LagsUsed                  518.000000
Critical Value (10%)        0.347000
Critical Value (5%)         0.463000
```

After the seasonal differencing of 24 periods, the rolling and variance as well as the ADF and KPSS test indicates stationarity.



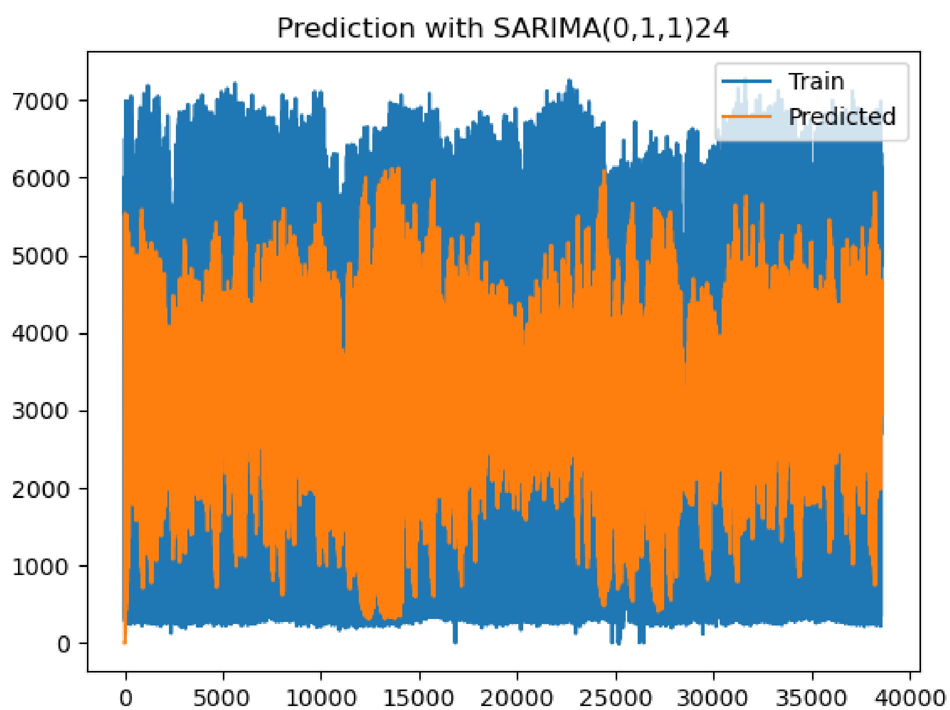
The ACF is cut off at lag 1 while PACF is tailing off. This can indicate SARIMA $(0,0,1)_{24}$



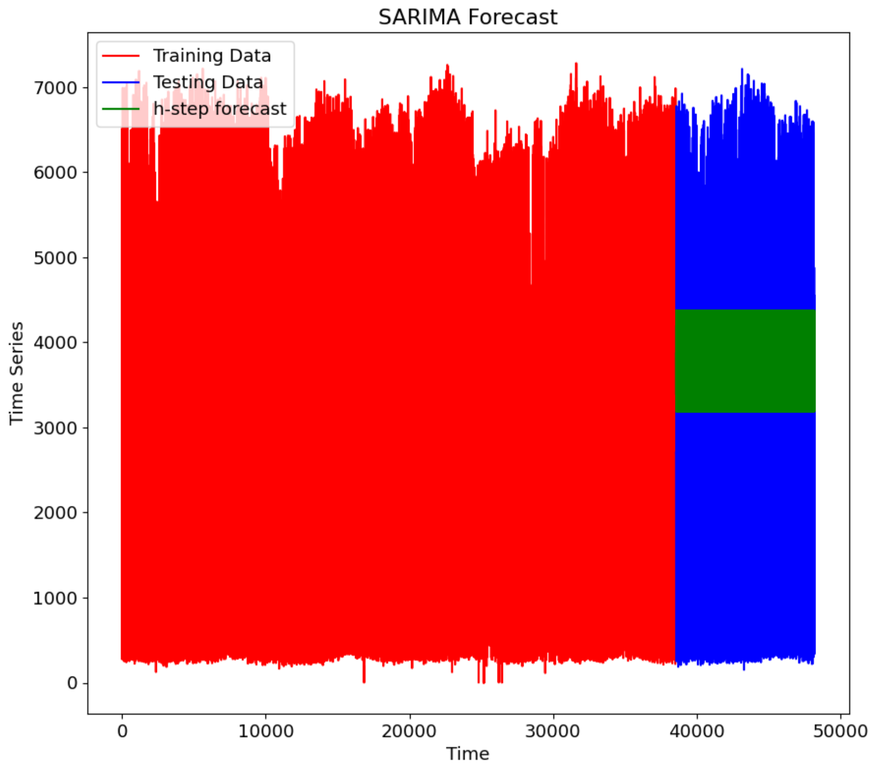
The ACF is cut off at lag 1 while PACF is tailing off. This can indicate SARIMA $(0,1,1)_{24}$. The GPAC confirms the order also.

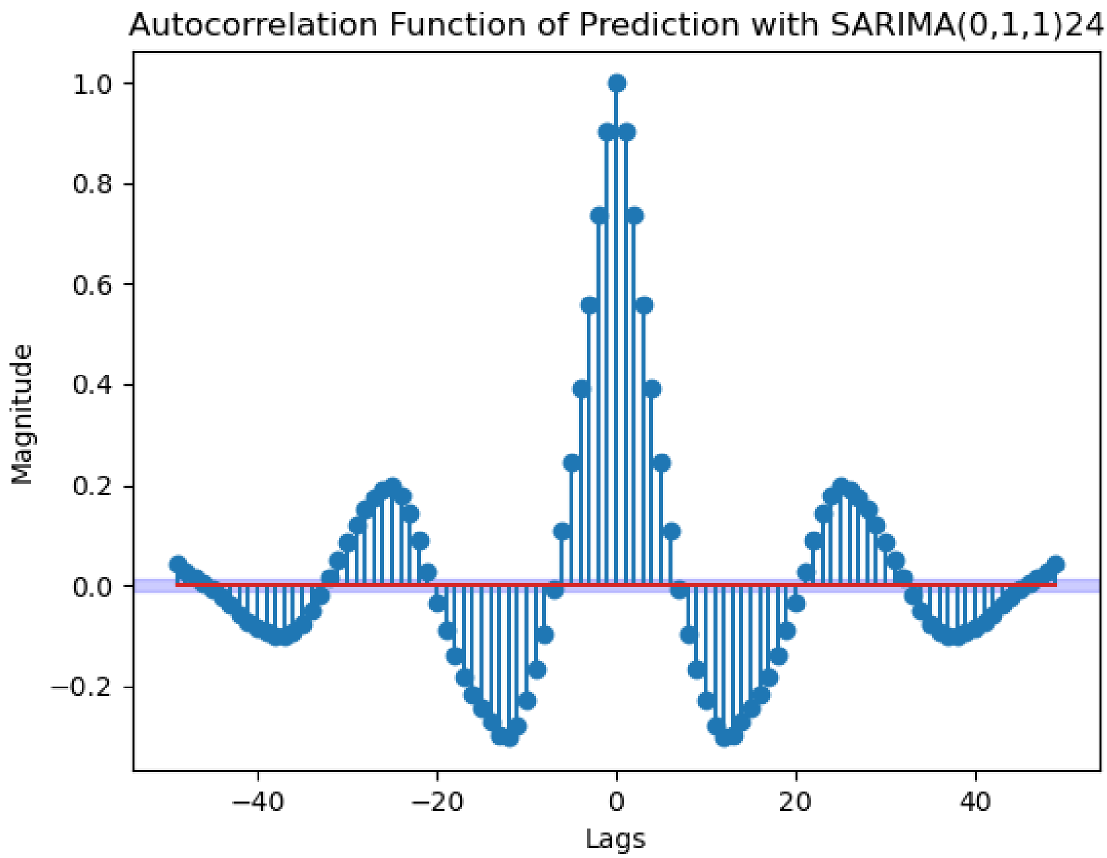
SARIMA $(0,1,1)_{24}$

```
                                     SARIMAX Results
==================================================================================
Dep. Variable:                             y   No. Observations:             38563
Model:             SARIMAX(0, 1, [1], 24)   Log Likelihood          -345833.289
Date:                    Fri, 29 Apr 2022   AIC                      691670.579
Time:                            15:53:21   BIC                      691687.696
Sample:                                 0   HQIC                     691676.007
                                  - 38563
Covariance Type:                      opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.S.L24      -0.8136      0.003   -260.932      0.000      -0.820      -0.808
sigma2      3.687e+06   3.44e+04    107.258      0.000    3.62e+06    3.75e+06
===================================================================================
Ljung-Box (L1) (Q):             31440.29   Jarque-Bera (JB):               800.78
Prob(Q):                            0.00   Prob(JB):                         0.00
Heteroskedasticity (H):             0.99   Skew:                            -0.12
Prob(H) (two-sided):                0.76   Kurtosis:                         2.33
===================================================================================
```



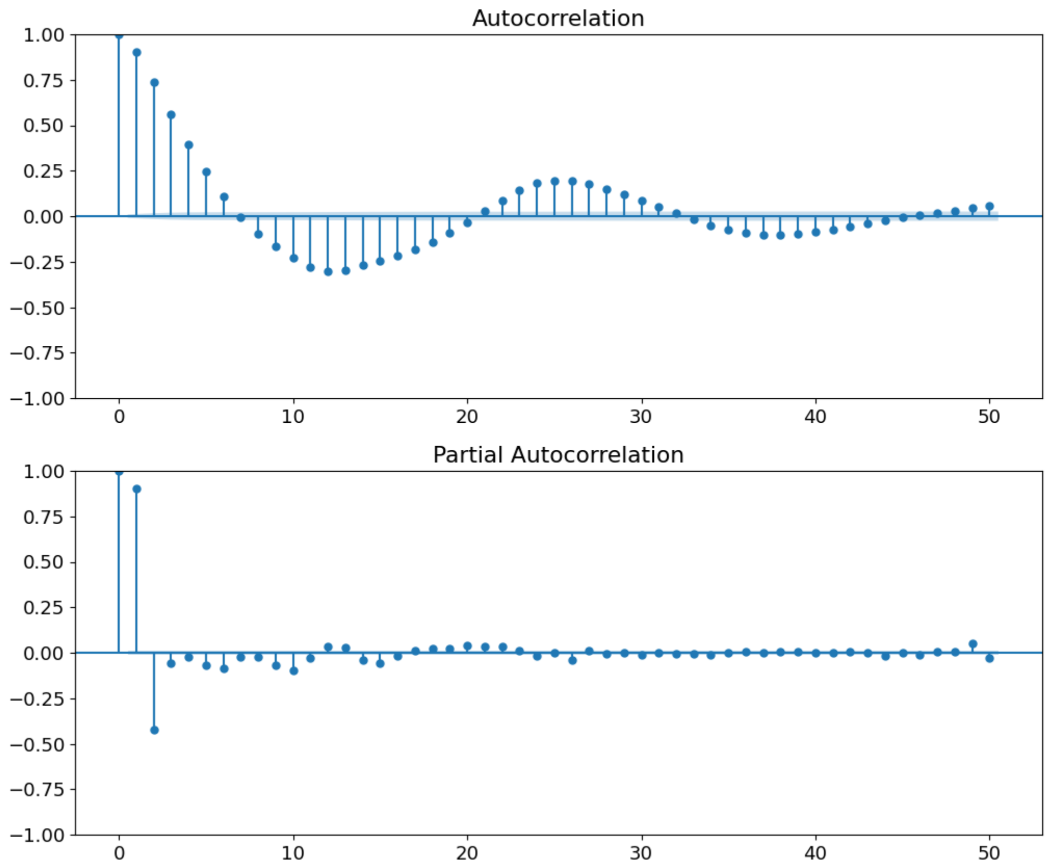
The SARIMA $(0,1,1)_{24}$ seem to capture the flow of the traffic volume from the plot above.






```
Ljung-Box test:
          lb_stat  lb_pvalue
50  107174.612713        0.0
```

The residual is not white with p-value of Ljung Box test less than 0.05 and many correlated lags in the ACF of residual above.



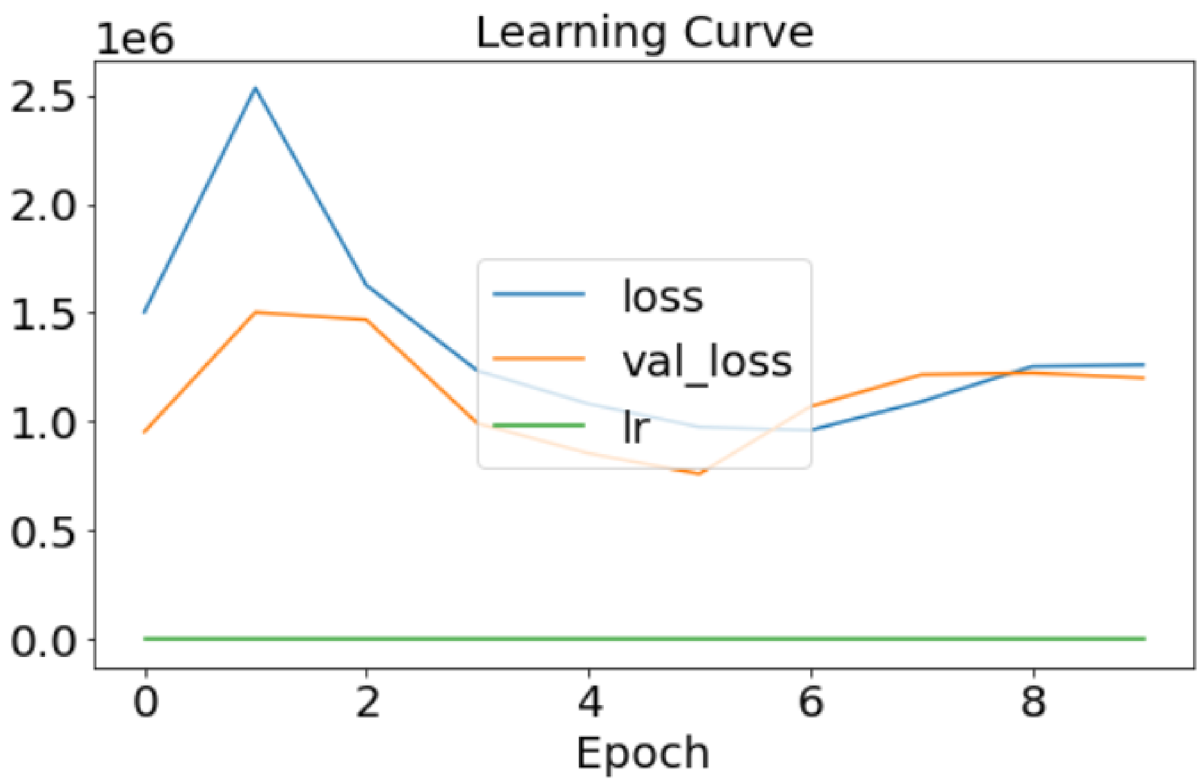

```
MSE Prediction SARIMA: 3699886.6533114836
```

Residual from the above SARIMA $(0,1,1)_{24}$ shows that the PACF is cut off at lag 2 and ACF is tailing off.

Bidirectional LSTM Model

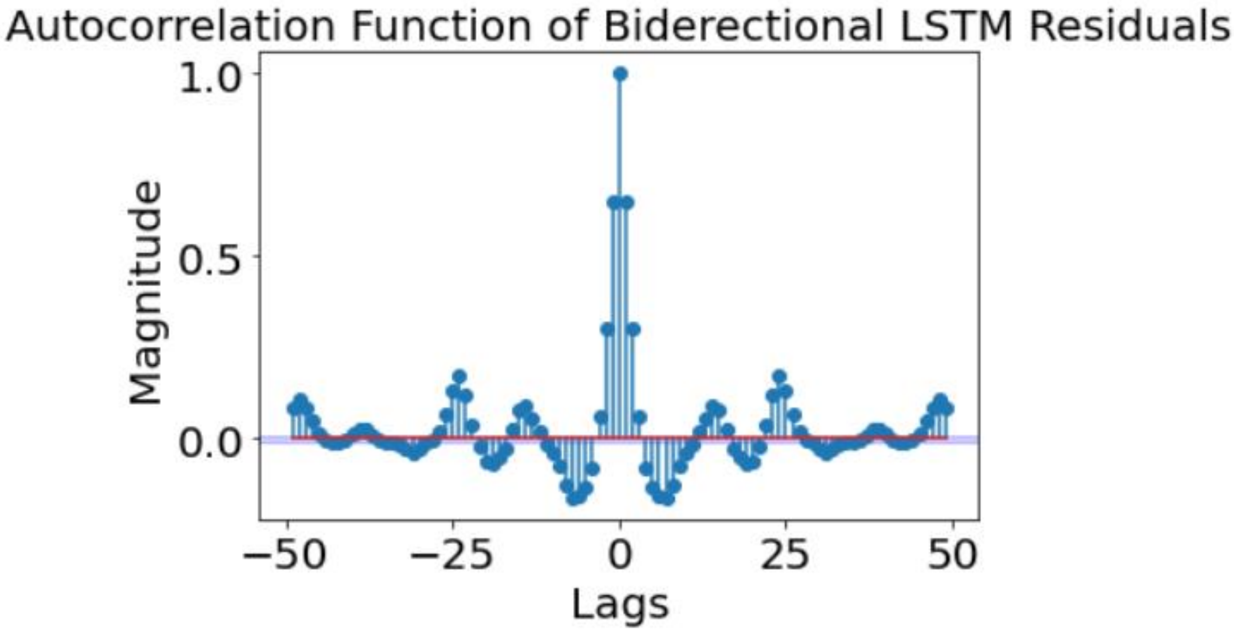
Results from training Bidirectional LSTM are shown below:

```
Epoch 1/10
964/964 [==============================] - 75s 78ms/step - loss: 1503981.3750 - val_loss: 954234.8125 - lr: 0.0010
Epoch 2/10
964/964 [==============================] - 74s 76ms/step - loss: 2532186.0000 - val_loss: 1502302.3750 - lr: 0.0010
Epoch 3/10
964/964 [==============================] - 74s 77ms/step - loss: 1627304.5000 - val_loss: 1468555.2500 - lr: 0.0010
Epoch 4/10
964/964 [==============================] - 74s 77ms/step - loss: 1234758.1250 - val_loss: 993209.0625 - lr: 1.0000e-04
Epoch 5/10
964/964 [==============================] - 74s 77ms/step - loss: 1082706.5000 - val_loss: 855656.2500 - lr: 1.0000e-04
Epoch 6/10
964/964 [==============================] - 74s 76ms/step - loss: 976253.9375 - val_loss: 761405.1250 - lr: 1.0000e-04
Epoch 7/10
964/964 [==============================] - 74s 77ms/step - loss: 961559.5000 - val_loss: 1069841.8750 - lr: 1.0000e-04
Epoch 8/10
964/964 [==============================] - 73s 75ms/step - loss: 1092752.1250 - val_loss: 1216224.3750 - lr: 1.0000e-04
Epoch 9/10
964/964 [==============================] - 74s 76ms/step - loss: 1254351.6250 - val_loss: 1224172.0000 - lr: 1.0000e-05
Epoch 10/10
964/964 [==============================] - 74s 77ms/step - loss: 1260723.8750 - val_loss: 1201084.5000 - lr: 1.0000e-05
```




```
MSE prediction: 1251498.2047935843
```

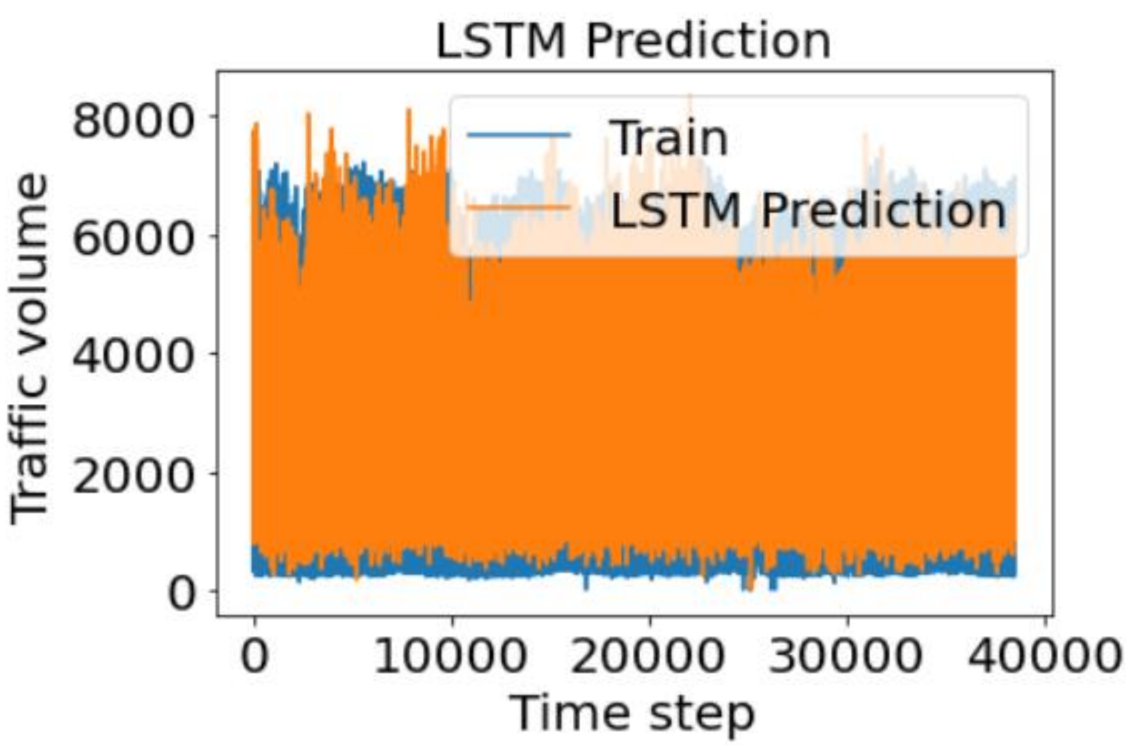
The epoch with the best loss (MSE) will be chosen. (see ipython notebook)

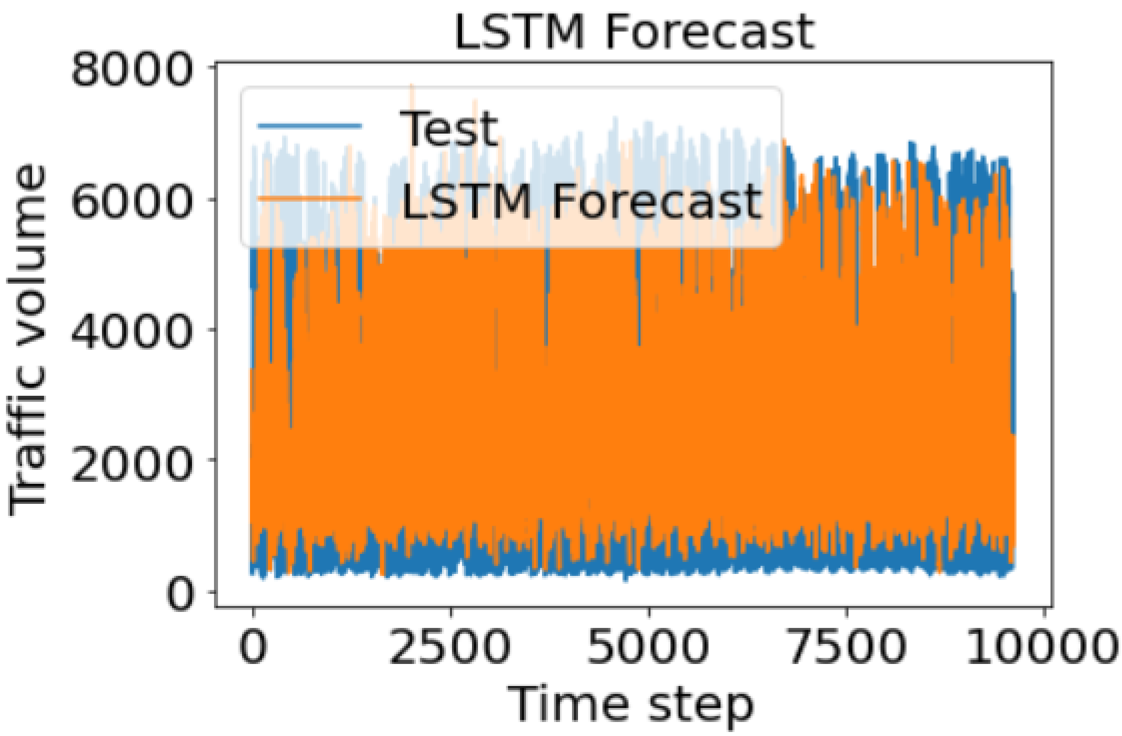


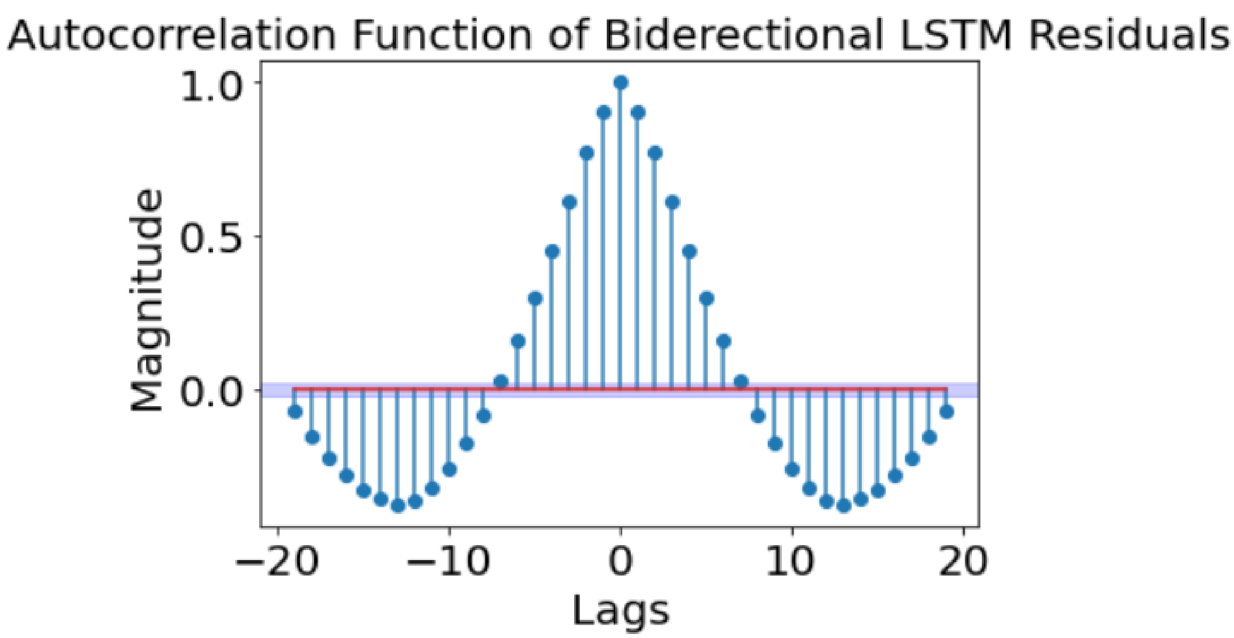

```
Qvalue: (array([28721.48425235]), array([0.]))
```

```
Variance of Residual: 2001939
```

The residual is not white with p-value of Ljung Box test less than 0.05 and many correlated lags in the ACF of residual above.








```
Qvalue: (array([28693.32268179]), array([0.]))
```

```
MSE Forecast: 6719125.784854772

Variance of Forecast Error: 6627908
```

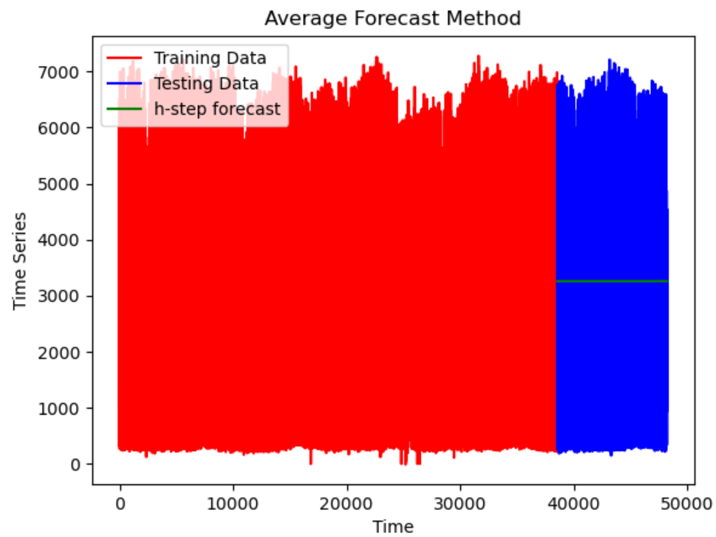
Model Selection

Below, we will review the metrics of the various models explored and select the best model.

Table: Model Generalization

	Q	MSE_pred	MSE_f	var_pred_er	var_forecast_er	variance ratio	RMSE_Forecast
Average mtd	172321	3967565	3871753	3967558	3871722	1.024753	1967.677
Multiple LR	134071.1	3883146	3953303	3898774	3988637	0.97747	1988.291
SES mtd	81939.4	1370194	3959577	1370194	3871722	0.353898	1989.869
Holt-Winters mtd	76652.91	867741.7	4171421	867741.7	3971003	0.21852	2042.406
Naive mtd	26059.18	725728.7	4175687	725728.7	3871722	0.187443	2043.45
SARIMA(0,1,1,24)	107174.6	3699887	4234742	3699841	4017493	0.920933	2057.849
Drift mtd	26064.66	725946.6	4761471	725946.4	3939709	0.184264	2182.079
LSTM	28721.48	1251498	6719126	2001939	6627908	0.302047	2592.128
ARMA(200)	5432.939	588943.9	14449181	576310.4	3890256	0.148142	3801.208

Average method has the least forecast MSE and RMSE with similar variances (residual and forecast) and closest ratio between variance of residual and forecast errors to one, which makes my best model for predicting traffic volume though the residual analysis indicated not white.



Root Mean Square: 1967.677

The h-step prediction obviously gives the average forecast.

Summary and Conclusion

Looking at predicting traffic volumes between Minneapolis and St Paul, MN with data from October 2, 2012 till September 30, 2018; we saw various models such as multiple linear regression, average method, naïve method, drift method, simple exponential smoothing method, holt-winters method, LSTM, ARMA process and SARIMA method.

From residual analysis the residual errors for all models were not white as their p-values for Ljung Box test were less than 0.05. However, the least forecast MSE and RMSE was observed with **Average method** and also the variance ratio between the residual and forecast errors is closest to one.

Taking a look at LSTM, it has the potential of following the future pattern of the test set. Due, to its high computational power I could not further try more hidden layers. Also, the SARIMAX python package requires high computational power so I could not dive deeper by adding more orders to the function based on the ACF/PACF of the residual error.

I would also suggest for future analysis on this kind of dataset, that a complex model be used to forecast since the models used here have very high MSE.

References

How to Develop LSTM Models for Time Series Forecasting. (2018, November 14). https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/

Time Series Analysis and Modeling. https://github.com/rjafari979/Time-Series-Analysis-and-Moldeing

Appendix

```python
#-----------Importing Libraries--------------
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.tsa.holtwinters as ets
import seaborn as sns
import datetime
from statsmodels.tsa.stattools import kpss
from statsmodels.tsa.stattools import adfuller
from pandas.plotting import register_matplotlib_converters
from sklearn.model_selection import train_test_split
from statsmodels.tsa.seasonal import STL
from scipy import signal
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.metrics import mean_squared_error
from scipy.signal import dlsim
import statsmodels.api as sm
from scipy.linalg import norm
pd.set_option('display.max_columns', None)
import sys
sys.path.append(r'C:\Users\oseme\PycharmProjects\pythonProject2\mytools')
from mytools import *
import warnings
warnings.filterwarnings('ignore')
random_seed=42
target='traffic_volume'
#--------------------------------------------------------------
#===============================================
# Developing Levenberg-Marquardt (LM) Algorithm
def gen_e(y,num, den):
    '''
    :param y: time series values
    :param num: list of numerator values (MA params)
    :param den: list of denominator values (AR params)
    :return: errors
    '''
    np.random.seed(42)
    y=np.array(y)
    sys= (den, num,1)
    _,e= dlsim(sys,y)
    e= e.ravel()
    # e[(e == -(np.inf)) | (e == np.inf) | (e == np.nan)] = 0
    e[np.isnan(e)] = np.zeros(1)[0]
    e[np.isinf(e)] = np.zeros(1)[0]
    return np.array(e)

def LMA_params(teta, e, delta, na, nb):
    '''
    :param teta: list of coefficients
    :param e: list of error
    :param delta: update parameter
    :param na: int of AR order
```

```
    :param nb: int of MA order
    :return: X, A, g
    '''
    e=e.reshape(-1,1)
    n= na + nb
    N= len(e)
    x=[]
    # tetas = []
    teta= teta.ravel()
    p1=teta[:na] #AR params
    p1=np.append(p1,[0]*(np.maximum(na,nb)-na))
    p2=teta[na:] #MA params
    p2=np.append(p2,[0]*(np.maximum(na,nb)-nb))
    e_orig = gen_e(y, den=np.r_[1,p1], num=np.r_[1,p2])  # regular teta
errors
    e_orig= e_orig.reshape(-1,1)
    for i in range(n):
        a= teta.copy()
        a[i]= a[i]+ delta
        # tetas.append(a)
        p11 = a[:na]  # AR params
        p11 = np.r_[p11, [0] * (np.maximum(na, nb) - na)]
        p22 = a[na:]  # MA params
        p22 = np.r_[p22, [0] * (np.maximum(na, nb) - nb)]
        e_new = gen_e(y, den=np.r_[1, p11], num=np.r_[1, p22])  # updated
tetas errors
        e_new = e_new.reshape(-1, 1)
        sol = (e_orig - e_new) / delta
        sol= sol.ravel()
        sol = sol.tolist()
        x.append(sol)
    X=np.array(x)
    X= (X.T).reshape(-1,n)
    A= X.T @ X
    g= X.T @ e
    g[np.isnan(g)] = np.zeros(1)[0]
    g[np.isinf(g)] = np.zeros(1)[0]
    return X, A, g


def SSE(e):
    '''
    :param e: numpy array of error
    :return: Sum of Squared Error
    '''
    # e[(e == -(np.inf)) | (e == np.inf)| (e == np.nan)] = 0
    e[np.isnan(e)] = np.zeros(1)[0]
    e[np.isinf(e)] = np.zeros(1)[0]
    sse= e.T @ e
    return sse

def teta_change(A, mu, g,na, nb):
    n= na + nb
    I= np.eye(n,n)
```

```
    change_in_teta= np.linalg.inv(A + mu*I)
    change_in_teta= change_in_teta @ g
    return change_in_teta

def teta_new(teta_old, change_in_teta):
    new= teta_old + change_in_teta
    return new


def initial(y,na,nb,mu=0.01):
    global teta,X,A,g,sse_old,sse_new,ch_teta,ch_teta_norm,new_teta,e
    # ==========================
    maxx= np.maximum(na,nb)
    n = na + nb
    AR, MA = np.r_[1, [0] * maxx], np.r_[1, [0] * maxx]  # make params
zeros
    N = len(y)
    e = gen_e(y,num=MA,den=AR)
    e = e.reshape(-1, 1)
    delta = 0.000001
    teta = [0] * (na + nb)
    teta = np.array(teta,dtype='float').reshape(-1, 1)
    sse_old = SSE(e)
    X, A, g = LMA_params(teta, e, delta, na, nb)
    #sse_old = SSE(e)
    ch_teta = teta_change(A, mu, g, na, nb)
    new_teta = teta_new(teta, ch_teta)
    pp1=new_teta[:na].ravel() #AR params
    pp1= np.r_[pp1,[0]*(np.maximum(na,nb)-na)]
    pp2= new_teta[na:].ravel() #MA params
    pp2= np.r_[pp2,[0]*(np.maximum(na,nb)-nb)]
    AR = np.r_[1,pp1]
    MA = np.r_[1,pp2]
    e = gen_e(y, num=MA, den=AR)
    e= e.reshape(-1,1)
    sse_new = SSE(e)
    ch_teta_norm = norm(ch_teta)
    # ===========================

# teta= new_teta
def step1(teta,na,nb,N,e):
    global X,A,g,sse_old
    e=e.reshape(-1,1)
    # ===========================
    ppp1 = teta[:na].ravel()   # AR params
    ppp1 = np.r_[ppp1, [0] * (np.maximum(na, nb) - na)]
    ppp2 = teta[na:].ravel()   # MA params
    ppp2 = np.r_[ppp2, [0] * (np.maximum(na, nb) - nb)]
    AR= np.r_[1,ppp1]
    MA= np.r_[1,ppp2]
    e = gen_e(y, num=MA, den=AR)
    e= e.reshape(-1,1)
    n = na + nb
    delta = 0.0000001
```

```
    X, A, g = LMA_params(teta, e, delta, na, nb)
    sse_old = SSE(e)

def step2(A,g,teta,na,nb,mu=0.01):
    global ch_teta,new_teta,e,sse_new,ch_teta_norm,var_error,cov_teta
    n=na+nb
    ch_teta = teta_change(A, mu, g, na, nb)
    new_teta = teta_new(teta, ch_teta)

    pppp1 = new_teta[:na].ravel()  # AR params
    pppp1 = np.r_[pppp1, [0] * (np.maximum(na, nb) - na)]
    pppp2 = new_teta[na:].ravel()  # MA params
    pppp2 = np.r_[pppp2, [0] * (np.maximum(na, nb) - nb)]

    AR = np.r_[1,pppp1]
    MA = np.r_[1,pppp2]
    e = gen_e(y, num=MA, den=AR)
    e= e.reshape(-1,1)
    sse_new = SSE(e)
    #===========
    if np.isnan(sse_new):
        sse_new = 10 ** 10



def
levenberg_marquardt(y,na,nb,e,mu=0.01,max_iter=100,n_iter=0,act=False):
    global teta, teta_estimate, cov_teta
    n=na+nb
    sse_tracker.append(sse_new)
    if act==False:
        if n_iter < max_iter:
            if sse_new< sse_old:
                if ch_teta_norm< 0.001:
                    teta_estimate= new_teta
                    var_error = sse_new / (N - n)
                    cov_teta = var_error * np.linalg.pinv(A)
                    print('===Results Available===')
                    print(teta_estimate)
                    act = True
                else:
                    teta= new_teta
                    mu= mu/10
                    step1(teta, na, nb, N, e)
                    step2(A, g, teta, na, nb, mu)
            while sse_new >= sse_old:
                mu=mu*10
                #step2(A, g, teta, na, nb, mu=mu)
                step2(A, g, teta, na, nb, mu=mu)
                if mu>1e20: #1e20
                    # print(f' "max_mu"')
                    print(new_teta)
                    act=True
                    n_iter=max_iter
```

```
                    break
                # step2(A,g,teta,na,nb,mu=mu)
            var_error = sse_new / (N - n)
            cov_teta = var_error * np.linalg.pinv(A)
            n_iter += 1
            if n_iter> max_iter:
                # print(f'Error: "max_iteration" ')
                # print(new_teta)
                act = True

            teta= new_teta
            step1(teta,na,nb,N,e)
            step2(A,g,teta,na,nb,mu)
            levenberg_marquardt(y, na, nb, e=e, mu=mu, max_iter=50,
n_iter=n_iter)
        else:
            pass
    else:
        pass


def LM_algorithm(y,na,nb):
    '''
    :param y: numpy array of values
    :param na: AR order
    :param nb: MA order
    :return: estimated parameters and covariance(as global variable)
    '''
    global sse_tracker
    sse_tracker=[]
    with np.errstate(divide='ignore'):
        np.float64(1.0) / 0.0
    initial(y,na,nb)
    sse_tracker.append(sse_old)
    levenberg_marquardt(y,na,nb,e)
    return new_teta
#============================================================
def zero_pole(na,nb,teta):
    ar= np.r_[1,teta[:na]]
    ma = np.r_[1, teta[na:]]
    print(f'Zero: {np.roots(ma)}')
    print(f'Pole: {np.roots(ar)}')

def print_coef(na,nb,model_obj):
  for i in range(na):
    print(f'The AR Coefficient a{i+1}:
{np.negative(model_obj.params[i])}')
  for i in range(nb):
    print(f'The MA Coefficient b{i+1}: {model_obj.params[na+i]}')
#============================================================
#---------Importing Dataset------------------------------------
df= pd.read_csv('Metro_Interstate_Traffic_Volume.csv')
df_copy= df.copy(deep=True)
datetime= pd.date_range(start='2012-10-02 09:00:00',
```

```
                            periods=len(df_copy), freq='1H') #set accurate
date_range
df_copy['datetime']=datetime
#df_copy['date_time']=pd.DatetimeIndex(df_copy['date_time']) # change to
type- datetime
# pd.DatetimeIndex(df_copy['date_time']).year

#---------------Missing Values & transformation---------------------------
------
print(f'Missing Values: {df_copy.isna().sum().any()}')
df_copy2=datetime_transformer(df_copy,['date_time'])
df_copy2.drop(columns=['date_time_minute','date_time_second'],inplace=True
)
df_copy_dum=
pd.get_dummies(df_copy2,columns=['holiday','weather_main','weather_descrip
tion'])

#---------------Dependent Variable time Plot-------------------
df_copy.plot(x='datetime',y='traffic_volume')
plt.title('Traffic Volume')
plt.xlabel('Time')
plt.ylabel('Volume of Traffic')
plt.xticks(rotation=45)
plt.show()

#---------------ACF/PACF Plot-------------------
ACF_PACF(df_copy[target],50)

#---------------Heatmap-------------------
cor= df_copy.corr(method='pearson')
sns.heatmap(cor, annot=True)
plt.title('Correlation Between All Features')
plt.show()

#----------------Splitting Dataset----------------------------
df_train, df_test= train_test_split(df_copy2,train_size=0.8,shuffle=False)

#----------------Stationarity Check----------------------------
# Initial check for stationarity on training and testing data set

# call rolling mean/variance
cal_rolling_mean_var(df_copy[target])
# ADF Test
ADF_Cal(df_copy[target])
# KPSS Test
kpss_test(df_copy[target])
ACF(df_copy[target],50,'Traffic Volume')


# Time Series Decomposition with
# STL (Seasonal and Trend Decomposition using Loess)
datetime= pd.date_range(start='2012-10-02 09:00:00',
                            periods=len(df_copy), freq='1H') #set accurate
date_range
```

```
traffic_volume= pd.Series(np.array(df_copy[target]),
                          index=datetime)

STL= STL(traffic_volume)
result= STL.fit()
plt.figure(figsize=(25,20))
fig= result.plot()
# fig.suptitle('STL Decomposition for Traffic Volume')
plt.show()

# Seasonality, Trend & Residual
T= result.trend
S= result.seasonal
R= result.resid
# strength of seasonality and trend
F= np.maximum(0,1- np.var(np.array(R))/np.var(np.array(S)+np.array(R)))

print(f'The strength of seasonality for this data set is: {F:.4f}')
F2= np.maximum(0,1- np.var(np.array(R))/np.var(np.array(T)+np.array(R)))
print(f'The strength of trend for this data set is: {F2:.4f}')

# Adjusted seasonality
adj_seasonal= df_copy[target].values - S.values # additive decomposition

plt.plot(datetime,df_copy[target],label='Original')
plt.plot(datetime,adj_seasonal,label='Seasonality Adjusted')
plt.xlabel('Time')
plt.ylabel('Values')
plt.title('Seasonality Adjusted vs Original Dataset')
plt.legend()
plt.show()

# Adjusted trend
adj_trend= df_copy[target].values - T.values # additive decomposition

plt.plot(datetime,df_copy[target],label='Original')
plt.plot(datetime,adj_trend,label='Trend Adjusted')
plt.xlabel('Time')
plt.ylabel('Values')
plt.title('Trend Adjusted vs Original Dataset')
plt.legend()
plt.show()


# GPAC
gpac(10,10,y=df_copy['traffic_volume'],lags=40)

# Empty Dataframe
check =
pd.DataFrame(columns=['Q','MSE_pred','MSE_f','var_pred_er','var_forecast_e
r'])

# Holt-Winter Method of forecasting
# This considers the trend and seasonality
```

```
fitted, forcast,mse_pred, mse_f, var_p,var_f,er1,er2=
holt_trend_s(df_train['traffic_volume'],

df_test['traffic_volume'],'add',True,'add',seasonal_periods=24) #TES #er1-
residual error er2-forecast error
print(f'Train MSE: {mse_pred}')
print(f'Test MSE: {mse_f}')
# There is an overfitting problem with Holt-Winter

print(f'Variance: Prediction variance: {var_p} Forecast variance:
{var_f}')
print(f'Ratio between variance of prediction & forecast: {var_p/var_f}')
# The ratio is very far from 1. This is a bad model for this dataset

# acf= acf_raw(er1,100)
# q= qvalue(acf,
100,len(df_train['traffic_volume'])+len(df_test['traffic_volume']))
print('Ljung-Box test:')
Q= sm.stats.acorr_ljungbox(er1, lags=[100],return_df=True)
print(Q)
# null hypothesis: The residual is white noise
# alternate hypothesis: The residual is not white noise
# The residual error is not white noise with p-value less than 0.05
check.loc["Holt-Winters mtd"]= [Q.lb_stat.ravel()[0],mse_pred,mse_f,
round(np.var(er1),2),round(np.var(er2),2)]


# Graphing forecast and ACF of errors
#=====
def plot_pred(train,fitted,label):
    plt.plot(train, label='Train')
    plt.plot(fitted, label='Predicted')
    plt.title(label)
    plt.xlabel('Hour')
    plt.ylabel('Traffic Volume')
    plt.legend()
    plt.show()
#====
graphit(df_train['traffic_volume'], df_test['traffic_volume'], forcast,
"Forecast Holt-Winters Method")
plot_pred(df_train['traffic_volume'],fitted,"Prediction with Holt-Winters
Method")
ACF(er1,50,'Holt-Winter Method on Traffic Volume')
# e=df_train['traffic_volume'][1:]-fitted[:-1]
# Residual ACF plot is not a white noise


#   Feature Selection w/Backward stepwise regression

# SVD and Condition Number without categorical variables
x= df_train[['temp','rain_1h','snow_1h','clouds_all']]
cols= x.columns
X= x.values
H= np.matmul(X.T,X)
```

```
#SVD
_,d,_=np.linalg.svd(H)
res=pd.DataFrame(d,index=cols, columns=['Singular Values'])
print(res)
print(f'Condition number for X is {np.linalg.cond(x)}')
# The least singular value 3.2 is closer to zero compared to the other
values.
# This indicate presence of co-linear feature(s).
# The condition number indicates severe degree of co-linearity with
condition number greater than 1000.

# OLS with statsmodel without cat variables
x= sm.add_constant(x) # added constant 1 vector
y= df_train['traffic_volume'].values
model= sm.OLS(y,x).fit()
print(model.summary())

# Remove feature const
model= sm.OLS(y,x.drop(columns=['const'])).fit()
print(model.summary())

# Remove feature snow_all with p>|t| 0.912
model= sm.OLS(y,x.drop(columns=['const','snow_1h'])).fit()
print(model.summary())

# Remove feature rain_1h with p>|t| 0.409
final_model= sm.OLS(y,x.drop(columns=['const','snow_1h','rain_1h'])).fit()
print(final_model.summary())
# The final model has the best F-statistics with p-value and a significant
# t-value for the two features left (temp & clouds_all)
xnew= df_train[['temp','clouds_all']]
print(f'Condition number after Feature Selection for X is
{np.linalg.cond(xnew)}')

# Base Models

# Average Forecast
train_f, test_f, mse_train, mse_test,er1,er2=
average_forecast(df_train['traffic_volume'],df_test['traffic_volume'])
print(f'Train MSE: {mse_train}')
print(f'Test MSE: {mse_test}')
print('Ljung-Box test:')
Q= sm.stats.acorr_ljungbox(er1, lags=[100],return_df=True)
print(Q)
check.loc["Average mtd"]= [Q.lb_stat.ravel()[0],mse_train,mse_test,
round(np.var(er1),2),round(np.var(er2),2)]
graphit(df_train['traffic_volume'], df_test['traffic_volume'], test_f,
"Average Forecast Method")
plot_pred(df_train['traffic_volume'].values,train_f.ravel(),"Prediction
with Average Method")
ACF(er1,50,'Average Method on Traffic Volume')

# Naive Forecast Method
train_f, test_f, mse_train, mse_test,er1,er2=
```

```
naive_forecast(df_train['traffic_volume'],df_test['traffic_volume'])
print(f'Train MSE: {mse_train}')
print(f'Test MSE: {mse_test}')
print('Ljung-Box test:')
Q= sm.stats.acorr_ljungbox(er1, lags=[100],return_df=True)
print(Q)
check.loc["Naive mtd"]= [Q.lb_stat.ravel()[0],mse_train, mse_test,
round(np.var(er1),2),round(np.var(er2),2)]
graphit(df_train['traffic_volume'], df_test['traffic_volume'], test_f,
"Naive Forecast Method")
plot_pred(df_train['traffic_volume'].values,train_f.ravel(),"Prediction
with Naive Method")
ACF(er1,50,'Naive Method on Traffic Volume')

# Drift Forecast Method
train_f, test_f, mse_train, mse_test,er1,er2=
drift_forecast(df_train['traffic_volume'],df_test['traffic_volume'])
print(f'Train MSE: {mse_train}')
print(f'Test MSE: {mse_test}')
print('Ljung-Box test:')
Q= sm.stats.acorr_ljungbox(er1, lags=[100],return_df=True)
print(Q)
check.loc["Drift mtd"]= [Q.lb_stat.ravel()[0],mse_train, mse_test,
round(np.var(er1),2),round(np.var(er2),2)]
graphit(df_train['traffic_volume'], df_test['traffic_volume'], test_f,
"Drift Forecast Method")
plot_pred(df_train['traffic_volume'].values,train_f.ravel(),"Prediction
with Drift Method")
ACF(er1,50,'Drift Method on Traffic Volume')


# Simple Exponential Smoothing
train_f, test_f, mse_train, mse_test,er1,er2=
SES(df_train['traffic_volume'],df_test['traffic_volume'],alpha=0.5,

initial_condition=df_train['traffic_volume'][0])
print(f'Train MSE: {mse_train}')
print(f'Test MSE: {mse_test}')
print('Ljung-Box test:')
Q= sm.stats.acorr_ljungbox(er1, lags=[100],return_df=True)
print(Q)
check.loc["SES mtd"]= [Q.lb_stat.ravel()[0],mse_train, mse_test,
round(np.var(er1),2),round(np.var(er2),2)]
graphit(df_train['traffic_volume'], df_test['traffic_volume'], test_f,
"Simple Exponential Smoothing")
plot_pred(df_train['traffic_volume'].values,train_f.ravel(),"Simple
Exponential Smoothing")
ACF(er1,50,'Simple Exponential Smoothing Method on \nTraffic Volume')


# Multiple linear Regression Using Least Square
x= df_train[['temp','clouds_all']]
cols= x.columns
X= x.values
```

```
y= df_train['traffic_volume'].values
#check for collinearity
H= np.matmul(X.T,X)
#SVD
_,d,_=np.linalg.svd(H)
res=pd.DataFrame(d,index=cols, columns=['Singular Values'])
print(res)
print(f'Condition number for X is {np.linalg.cond(x)}')

#Splitting into train and test sets
xtrain,xtest,ytrain,ytest= train_test_split(x, y,
test_size=0.2,shuffle=False)

# regression model
reg_model= sm.OLS(ytrain,xtrain).fit()
print(reg_model.summary())
# Regression equation
reg_fitted= reg_model.predict(xtrain)
reg_forecast= reg_model.predict(xtest)

residual= ytrain[1:] - reg_fitted[:-1]
forecast_error= ytest[2:] - reg_forecast[:-2]
ACF(residual,50,'Residual on Multiple Linear Regression')
print('Ljung-Box test:')
Q= sm.stats.acorr_ljungbox(residual, lags=[100],return_df=True)
print(Q)
mse_pred=mean_squared_error(ytrain,reg_fitted)
print(f'MSE of prediction: {mse_pred}')
from sklearn.metrics import r2_score
r2=r2_score(ytrain,reg_fitted)
print(f' R Squared {reg_model.rsquared}')
print(f'Adjusted R Squared {reg_model.rsquared_adj}')
plot_pred(ytrain,reg_fitted,"Prediction with Mulitiple Linear Regression")
graphit(ytrain, ytest, reg_forecast, "Regression Forecast")
#MSE=np.square(np.subtract(ytrain,reg_fitted)).mean()
mse_forc= mean_squared_error(ytest,reg_forecast)
check.loc["Multiple LR"]= [Q.lb_stat.ravel()[0],mse_pred, mse_forc,
round(np.var(residual),2),round(np.var(forecast_error),2)]
print(check)
# Check variance ratio
check2=check.copy(deep=True)
check2['variance ratio']=check.iloc[:,-2]/check.iloc[:,-1]
print(check2)

# ARMA process
# Parameter estimation with LM Algorithm
y=df_train['traffic_volume'].values
# global y,N
N=len(df_train)
coef=LM_algorithm(y=y,na=2,nb=0) #LM Algorithm
print(f'Parameters from LM Algorithm:\n {coef}')
covariance_matrix1=cov_teta
print('Covariance Matrix\n',covariance_matrix1)
c1,c2= coef[0]-(2*np.sqrt(covariance_matrix1[0,0])),
```

```
coef[0]+(2*np.sqrt(covariance_matrix1[0,0]))
c3,c4= coef[1]-(2*np.sqrt(covariance_matrix1[1,1])),
coef[1]+(2*np.sqrt(covariance_matrix1[1,1]))
print(f'Confidence interval:\n {c1},{c2}')
print(f'Confidence interval:\n {c3},{c4}')
zero_pole(na=2,nb=0,teta=coef.ravel())

#+=============================statsmodels version
0.10.2=================================
# Param estimation with statsmodels version==0.10.2
# model_coef= sm.tsa.ARMA(np.array(y),order=(2,0)).fit(trend='nc',disp=0)
# print(model_coef.summary())
# print_coef(2,0,model_coef)
#+===========================================================================
===============

# ARMA(2,0)
# predictgion function with known parameters of ARMA(2,0)

prediction= []
teta= coef.ravel()
teta=np.round(teta,4)
y=df_train['traffic_volume'].ravel()
for i in range(len(df_train)):
    if i==0:
        prediction.append(-teta[0]*y[i])
    else:
        prediction.append((-teta[0])*y[i]-teta[1]*y[i-1])
prediction=np.abs(prediction)
residuals= y[1:]-np.array(prediction[:-1])
plot_pred(y,prediction,"Prediction with ARMA(2,0)") #plot of train and
prediction
ACF(residuals,50,'Residuals from ARMA(2,0)')
print('Ljung-Box test:')
Q= sm.stats.acorr_ljungbox(residuals, lags=[50],return_df=True)
print(Q)
ACF_PACF(residuals,50)
mse_arma=mean_squared_error(y[1:], np.array(prediction[:-1]))
print(f'MSE Prediction ARMA: {mse_arma}')
# forecast for ARMA
forecast= []
y=df_test['traffic_volume'].ravel()
for i in range(len(df_test)):
    if i==0:
        forecast.append((-teta[0])*y[-1]-teta[1]*y[-2])
    elif i==1:
        forecast.append((-teta[0])*prediction[-1]-teta[1]*y[-1])
    elif i==2:
        forecast.append((-teta[0]) * forecast[i-1] - teta[1] *
prediction[-1])
    else:
        forecast.append((-teta[0]) * forecast[i - 1] - teta[1] *
forecast[i-2])
forecast=np.abs(forecast)
```

```
forecast_error= y[2:]-np.array(forecast[:-2])
print(f"Variance ratio for ARMA test & forecast:
{np.var(df_test['traffic_volume'].ravel())/np.var(forecast)}")
mse_arma_f=mean_squared_error(df_test['traffic_volume'].ravel()[1:],
np.array(forecast[:-1]))
print(f'MSE Forecast ARMA: {mse_arma_f}')
graphit(df_train['traffic_volume'].values,
df_test['traffic_volume'].values, forecast, "ARMA(2,0) Forecast")
#========================

check.loc["ARMA(200)"]= [Q.lb_stat.ravel()[0],mse_arma, mse_arma_f,
round(np.var(residuals),2),round(np.var(forecast_error),2)]
print(check)

#===========SARIMA==========
# Trying seasonal differencing of 24
y_dif_24= dif_s(df_train['traffic_volume'],24)
y_dif_24= np.array(y_dif_24)
cal_rolling_mean_var(y_dif_24)
ADF_Cal(y_dif_24)
kpss_test(y_dif_24)
gpac(10,10,y_dif_24,50)


import statsmodels.api as sm
# SARIMA with python package
#SARIMA= (0,1,1,24)
y=df_train['traffic_volume'].ravel()
sarima=
sm.tsa.statespace.SARIMAX(y,order=(0,0,0),seasonal_order=(0,1,1,24),
                                    enforce_stationarity=False,
                                    enforce_invertibility=False)
results=sarima.fit()
print(results.summary())
sarima_train=results.get_prediction(start=0,
end=len(df_train['traffic_volume'].ravel()), dynamic=False)
Sarima_pred=sarima_train.predicted_mean
Sarima_residual= df_train['traffic_volume'].ravel()[1:]-Sarima_pred[1:-1]
plot_pred(df_train['traffic_volume'].ravel(),np.abs(Sarima_pred),"Predicti
on with SARIMA(0,1,1)24") #plot of train and prediction
ACF(Sarima_residual,50,'Prediction with SARIMA(0,1,1)24')
print('Ljung-Box test:')
Q= sm.stats.acorr_ljungbox(Sarima_residual, lags=[50],return_df=True)
print(Q)
ACF_PACF(Sarima_residual,50)
mse_sarima=mean_squared_error(df_train['traffic_volume'].ravel()[1:],
Sarima_pred[1:-1])
print(f'MSE Prediction SARIMA: {mse_sarima}')


#forecast
sarima_forecast=results.predict(start=len(df_train), end=(len(df_copy2)))
sarima_f_error=df_test['traffic_volume'].values[2:]-sarima_forecast[1:-2]
# ACF(sarima_f_error,50,'Forecast with ARIMA(2,0,0)xSARIMA(0,1,1)24')
```

```
graphit(df_train['traffic_volume'].values,
df_test['traffic_volume'].values, np.abs(sarima_forecast[1:]), "SARIMA
Forecast")
mse_sarima_f=mean_squared_error(df_test['traffic_volume'].ravel()[2:],
sarima_forecast[1:-2])
print(f'MSE forecast SARIMA: {mse_sarima_f}')
check.loc["SARIMA(0,1,1,24)"]= [Q.lb_stat.ravel()[0],mse_sarima,
mse_sarima_f,
round(np.var(Sarima_residual),2),round(np.var(sarima_f_error),2)]

#code is seperate in an ipython notebook ran on google colab
check.loc["LSTM"]= [28721.48,1251498.2, 6719125.78, 2001939,6627908]
print(check)

# Check variance ratio
check2=check.copy(deep=True)
check2['variance ratio']=check.iloc[:,-2]/check.iloc[:,-1]
check2['RMSE_Forecast']=np.sqrt(check2.iloc[:,2])
check2.sort_values(by=['RMSE_Forecast','MSE_f','variance
ratio'],ascending=True,inplace=True)
print(check2)
```