Course number: **DATS 6450**

Course name: **Time series Modeling & Analysis**

Instructor: **Dr. Reza Jafari**

Lab number or title: **Final Project Report**

Submitted By: **Madhuri Yadav**

Date: **12/16/2020**

Table of Contents

Abstract:

This project aims on predicting The Metro Interstate Traffic Volume between Minneapolis-St Paul, MN using various time series prediction techniques learnt in the DATS 6450 Time series Modeling & Analysis course. We also work on EDA techniques such as ADF test, etc. as well as validate our results with techniques such as Chi-Square test etc.

Introduction:

It is important to study the traffic volume to help in some of the following points:

1. Increase the efficiency and life of roads
2. Reduces traffic volume at a particular section
3. Provide better means for development of infrastructures
4. Provide better means to utilize other roads in case of special events in the city
5. Provide estimate of no vehicles against no of persons
6. During the given pandemic situations where social distancing has higher priority.

The timeseries processes learnt in the course help us to estimate the traffic and to validate our predictions.

About Dataset:

Metro Interstate Traffic Volume Data Set is used for this project. The dataset was obtained from UCI Machine Learning Repository. The dataset contains 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. For metadata refer to the Appendix A. The shape of the dataset before preprocessing is (433845, 9).

Data Preprocessing:

The compressed data is read and save as csv file. Then the None values are replaced with the nan. "date_time" column is converted to datetime format. Since our data is hourly bases for prediction, we aggregate the data based on the date and acquire the daily data. During aggregation mean of the values are taken for numerical data and the first values are taken for categorical data types. Holiday data column is then converted to two values Yes and No. To reduce the number of the categories in the weather column following changes are done Drizzle, Haze, Thunderstorm, Fog are changed to Rain and Mist accordingly.

After Preprocessing the data set looks as follows:

Dataset Information:

```
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   holiday              48204 non-null  object
 1   temp                 48204 non-null  float64
 2   rain_1h              48204 non-null  float64
 3   snow_1h              48204 non-null  float64
 4   clouds_all           48204 non-null  int64
 5   weather_main         48204 non-null  object
 6   weather_description  48204 non-null  object
 7   date_time            48204 non-null  object
 8   traffic_volume       48204 non-null  int64
dtypes: float64(3), int64(2), object(4)
```

Dataset Description:

```
              temp      rain_1h      snow_1h   clouds_all  traffic_volume
count  1067.000000  1067.000000  1067.000000  1067.000000     1067.000000
mean    281.612894     0.426383     0.000399    44.925311     3271.916087
std      11.940657    12.041157     0.006412    27.637918      576.295399
min     249.040000     0.000000     0.000000     0.000000      192.942857
25%     272.820797     0.000000     0.000000    20.328431     2870.834359
50%     282.936429     0.000000     0.000000    43.259259     3422.458333
75%     292.337633     0.000000     0.000000    68.365385     3722.250000
max     302.587500   393.272400     0.156667    90.227273     4555.170213
```

Column values for "weather_main", "holiday"

```
weather_main  :
Clear     489
```

```
Clouds     225
Rain       175
Mist       103
Snow        75
Name: weather_main, dtype: int64

holiday  :
No      1036
Yes       31
Name: holiday, dtype: int64

The Shape of the dataset after preprocessing is : (1067, 7)
```

Data analysis:

The initial analysis was performed on the dataset. It is as follows.

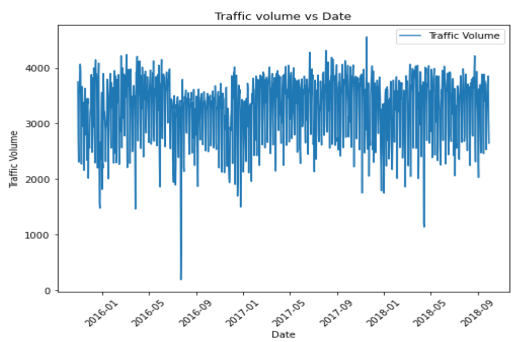


Figure 1: Traffic Volume vs Date

Figure 1 shows the distribution of the dependent variable vs date. Since it is difficult to get a clear look of the dataset with this graph. We plot only first 100 and the only 30 values as shown in the figure 2 and figure 3 respectively.

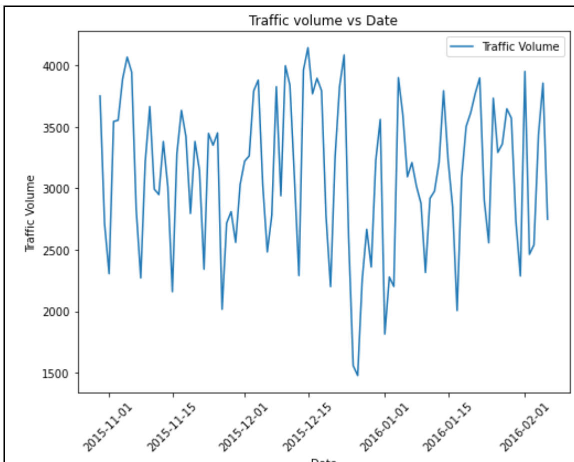


Figure 2

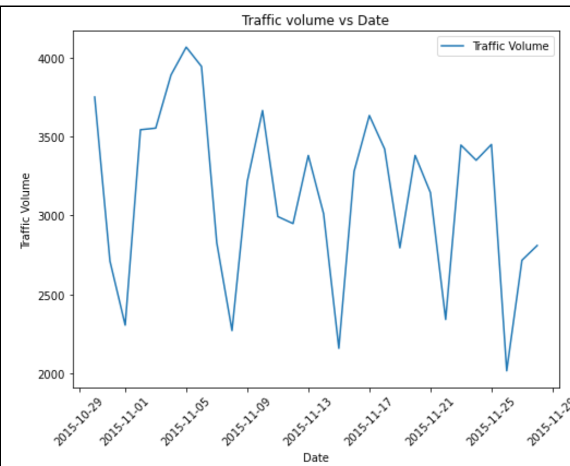


Figure 3

From figure 3 we clearly see the seasonality.

Figure 4 below shows the count of Holiday “Yes” and “No” values. We see that the values are imbalanced.

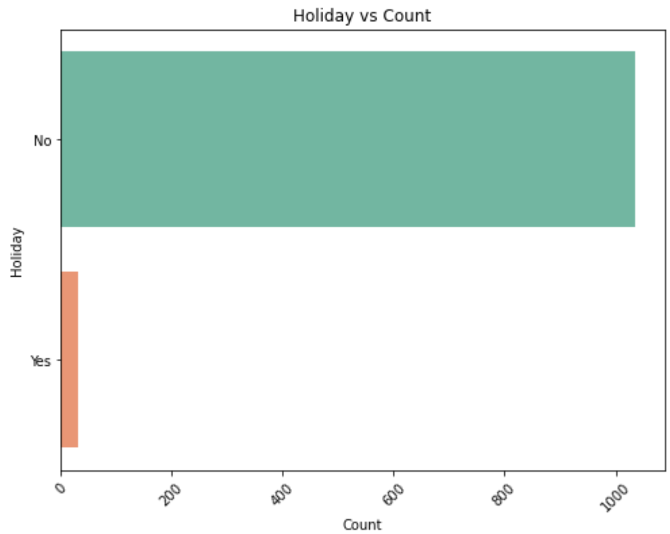


Figure 4

Figure 5 below shows the count of each categories in Weather column.

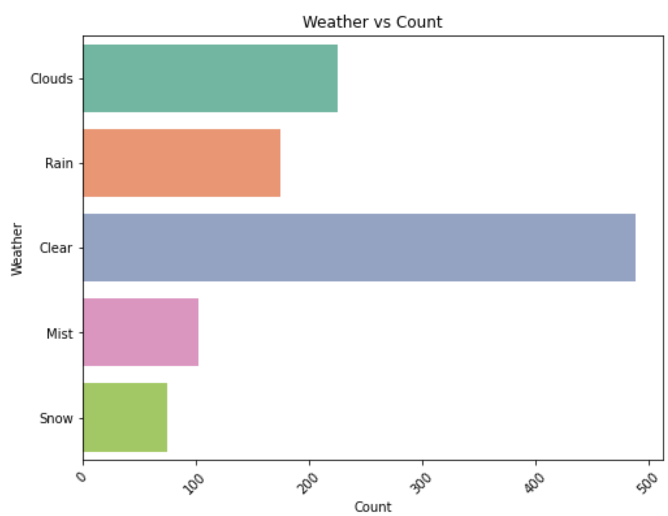


Figure 5

Figure 5 and Figure 6 below shows the distribution of the temperature and clouds data and we see that the mean is at around 283 and 45 respectively.

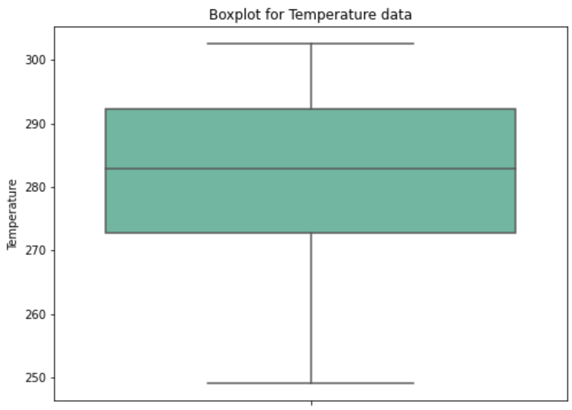


Figure 6

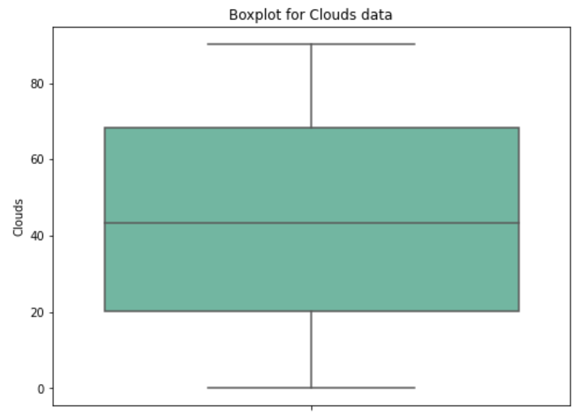


Figure 7

Figure 8 and Figure 9 shows the histogram of Rain and snow data and it is observed to be mostly 0 and the number of days of snow and rain are less.

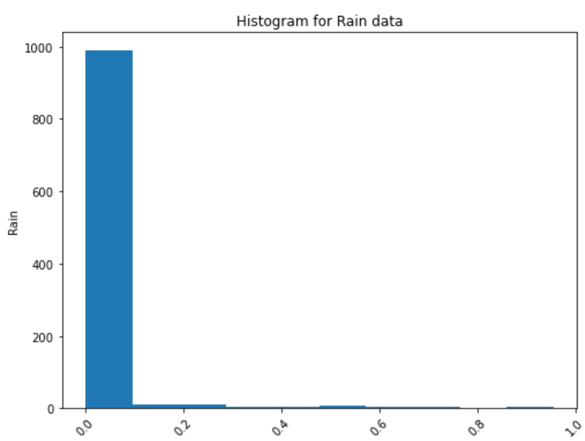


Figure 8

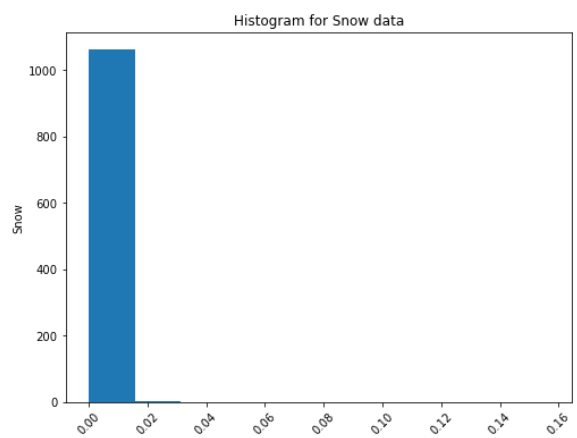


Figure 9

Figure 10 and figure 11 below show the ACF plot and Correlation-Coefficients for the dataset, respectively. Figure 10 shows that there is seasonality clearly and the data is repeated weekly. The Corr heatmap shows that there is no strong correlation between the dependent and the independent variables.

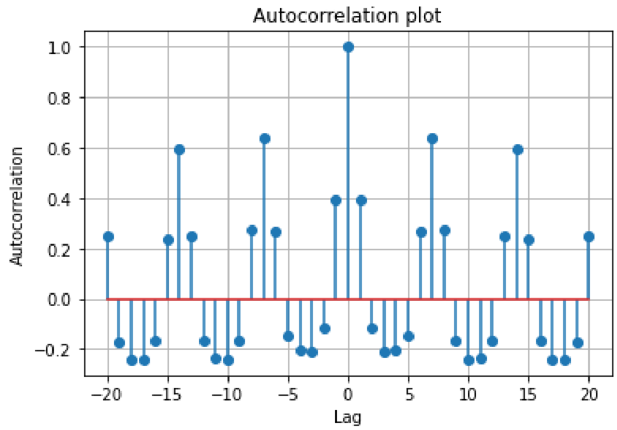


Figure 10

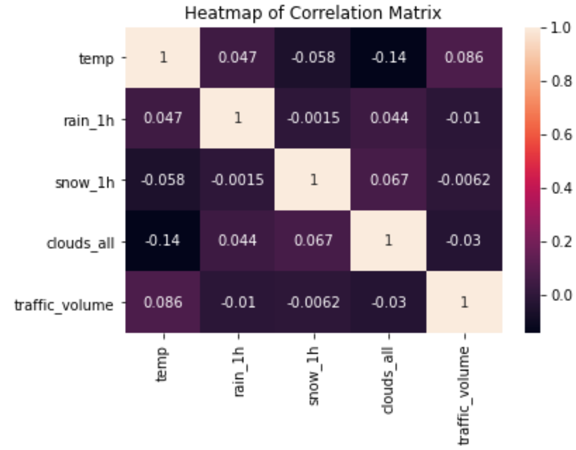


Figure 11

Train-Test Split:

Following are the shapes of the original dataset train set and the test set.

```
Shape of the entire dataset:  (1067, 7)
Shape of the train set :  (853, 7)
Shape of the test set :  (214, 7)
```

Stationarity:

```
ADF test for traffic_volume
ADF Statistic: -4.711791
p-value: 0.000080
Critical Values:
        1%: -3.437
        5%: -2.864
        10%: -2.568
P-value is 0.000078 i.e. less than 0.05(95% or more confidence interval),
hence we reject Null Hypothesis and conclude that the dependent variable is
stationary.
```

However, we observe the seasonality in our dataset hence we perform the first and second order differencing for ARMA process.

Time series Decomposition:

Since the data seems to have high seasonality, we perform decomposition on the dataset the results are as follows as shown in Figure 12.

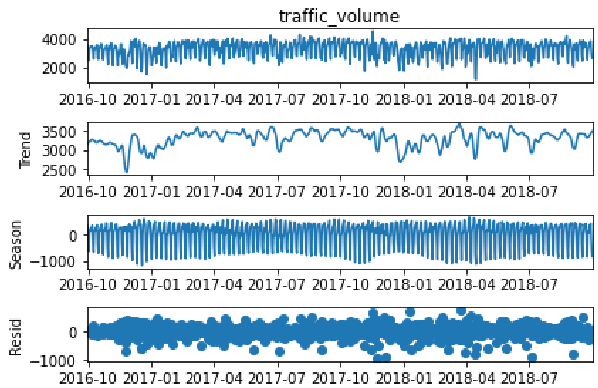


Figure 12

Figure 13 and figure 14 below shows the seasonality components of the dataset.

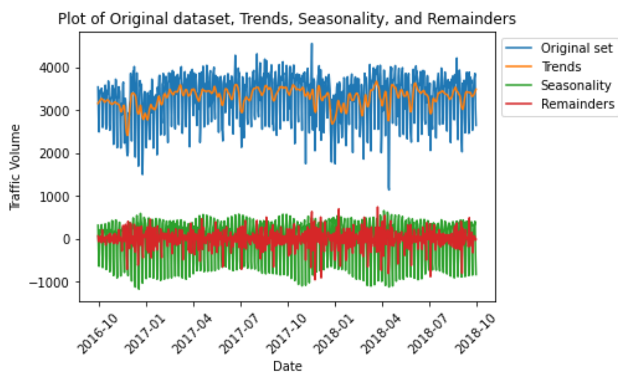


Figure 13

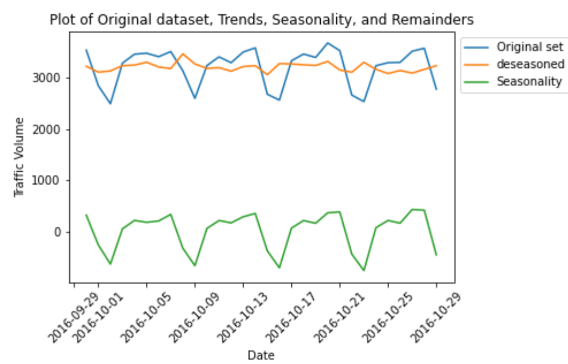


Figure 14

```
The strength of trend for this data set is: 0.54
The strength of seasonality for this data set is: 0.83
```

Feature selection:

For feature selection the results of linear regression are as follows.

```
=====================================================================================
                  The summary of the model with all variables.
=====================================================================================
                            OLS Regression Results
==============================================================================
Dep. Variable:         traffic_volume   R-squared:                       0.091
Model:                            OLS   Adj. R-squared:                  0.078
Method:                 Least Squares   F-statistic:                     7.188
Date:                Wed, 16 Dec 2020   Prob (F-statistic):           4.24e-09
Time:                        19:47:31   Log-Likelihood:                -4505.9
No. Observations:                 584   AIC:                             9030.
Df Residuals:                     575   BIC:                             9069.
Df Model:                           8
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
const              1674.3986     61.484     27.233      0.000    1553.637    1795.160
clouds_all          -75.7049     84.686     -0.894      0.372    -242.036      90.626
holiday_No         1137.2141     53.680     21.185      0.000    1031.782    1242.646
```

```
holiday_Yes           537.1845     87.329      6.151      0.000     365.662     708.707
rain_1h               180.3886    514.983      0.350      0.726    -831.090    1191.867
snow_1h              7.909e-15   5.18e-14      0.153      0.879   -9.39e-14     1.1e-13
temp                  266.4854    110.723      2.407      0.016      49.014     483.956
weather_main_Clear    422.6572     39.752     10.632      0.000     344.580     500.734
weather_main_Clouds   475.7453     49.653      9.581      0.000     378.222     573.269
weather_main_Mist     356.1330     68.642      5.188      0.000     221.313     490.953
weather_main_Rain     297.0305     60.932      4.875      0.000     177.354     416.707
weather_main_Snow     122.8326     67.757      1.813      0.070     -10.249     255.914
==============================================================================
Omnibus:                       38.796   Durbin-Watson:                   1.297
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               44.812
Skew:                          -0.668   Prob(JB):                     1.86e-10
Kurtosis:                       2.757   Cond. No.                     1.26e+18
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.04e-33. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Since the p values is more for some columns "snow_1h","rain_1h","clouds_all","weather_main_Snow" are removed one by one.

And following are the results after feature elimination.

```
=======================================================================================
              The summary of the model after features elimination.
=======================================================================================
                            OLS Regression Results
==============================================================================
Dep. Variable:         traffic_volume   R-squared:                       0.089
Model:                            OLS   Adj. R-squared:                  0.080
Method:                 Least Squares   F-statistic:                     9.443
Date:                Wed, 16 Dec 2020   Prob (F-statistic):           6.63e-10
Time:                        19:47:31   Log-Likelihood:                -4506.3
No. Observations:                 584   AIC:                             9027.
Df Residuals:                     577   BIC:                             9057.
Df Model:                           6
Covariance Type:            nonrobust
=======================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------------
const                1717.2161     67.487     25.445      0.000    1584.665    1849.767
holiday_No           1161.6981     53.340     21.779      0.000    1056.934    1266.462
holiday_Yes           555.5181     89.513      6.206      0.000     379.708     731.329
temp                  273.4530    110.391      2.477      0.014      56.636     490.270
weather_main_Clear    326.1443     83.552      3.903      0.000     162.041     490.248
weather_main_Clouds   362.0308     92.468      3.915      0.000     180.415     543.646
weather_main_Mist     241.5281    110.285      2.190      0.029      24.919     458.137
weather_main_Rain     174.5027    101.121      1.726      0.085     -24.108     373.113
==============================================================================
Omnibus:                       39.070   Durbin-Watson:                   1.296
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               45.334
Skew:                          -0.673   Prob(JB):                     1.43e-10
Kurtosis:                       2.772   Cond. No.                     8.19e+15
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.25e-29. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

We also observe the significant improvement in AIC values.

Multiple Linear Regression:

Also Following are the Coefficients of the LSE model.

```
The Coefficients for the LSE model are:
 const                  33017.199475
holiday_No             -13654.444346
holiday_Yes             -8688.635940
temp                      273.452969
weather_main_Clear        326.144272
weather_main_Clouds       362.030830
weather_main_Mist         241.528093
weather_main_Rain         174.502723
Name: traffic_volume, dtype: float64
-2451489.231002282 -6076.655145079008
```

ARMA Model:

Data Differencing:

The figure 15 below shoes the dataset after 1^{st} and 2^{nd} order differencing.

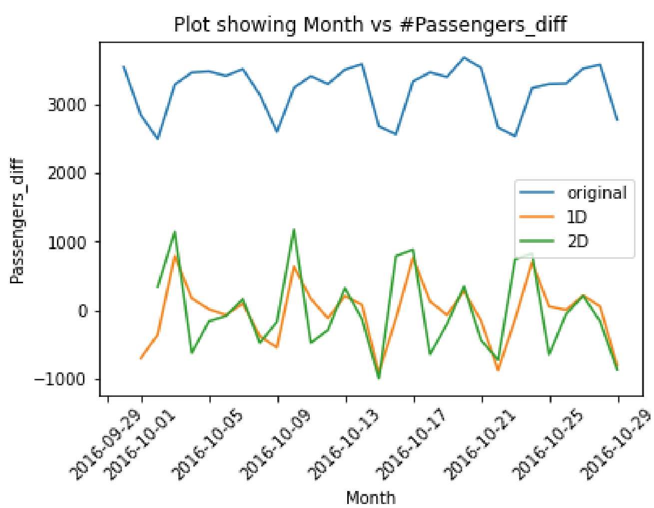
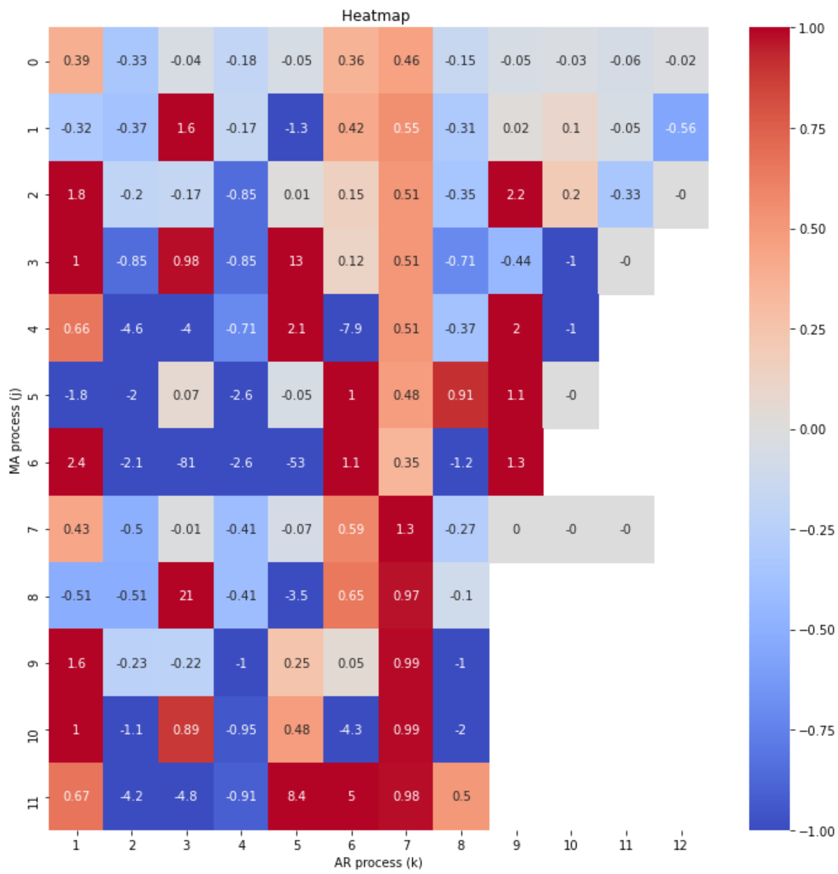


Figure 15

GPAC:

Below is the GPAC table for the dataset.



From the gapc table the observed orders are as follows.

oberved_orders = [(2,0),(2,2),(4,0),(4,2),(4,7),(7,11),(8,1),(8,8),(9,0),(9,5),(9,7),(10,1),(10,3),(11,2)]

Levenberg Marquardt Algorithm:

ARMA is performed using LM algorithm and the coefficients are used for the prediction it is observed that none of the above orders pass the CHI-SQURE test. Hence, we perform Brute force method and observe that the following ARMA (3,6), ARMA (5,5), ARMA (5,7) perform better than other models.

Following is the model summary:

```
The Model Summary is as Follows: ARMA  (3,6)
Final parameters are:  [-2.25  2.25 -1.   -1.81  1.41  0.03 -0.69  0.38 -0.1 ]
[-2.2475869708739005, 2.2484706904368155, -1.0008095440306755, 0, 0, 0] [-1.8107999196270776,
1.4134001746938882, 0.02646795751095804, -0.6864335286859218, 0.3841354061559829, -
0.09606344681977062]
Confidence Interval are:
-2.241 < a1 < -2.254
 2.259 < a2 <  2.238
-0.995 < a3 < -1.007
-1.727 < b1 < -1.894
 1.583 < b2 <  1.244
 0.225 < b3 < -0.172
-0.488 < b4 < -0.885
 0.554 < b5 <  0.214
-0.013 < b6 < -0.180
```

We Observe that the coefficients b3 to b6 are not significant since they include 0 in their interval. The predicted and forecasted values for the model are plotted as follows.

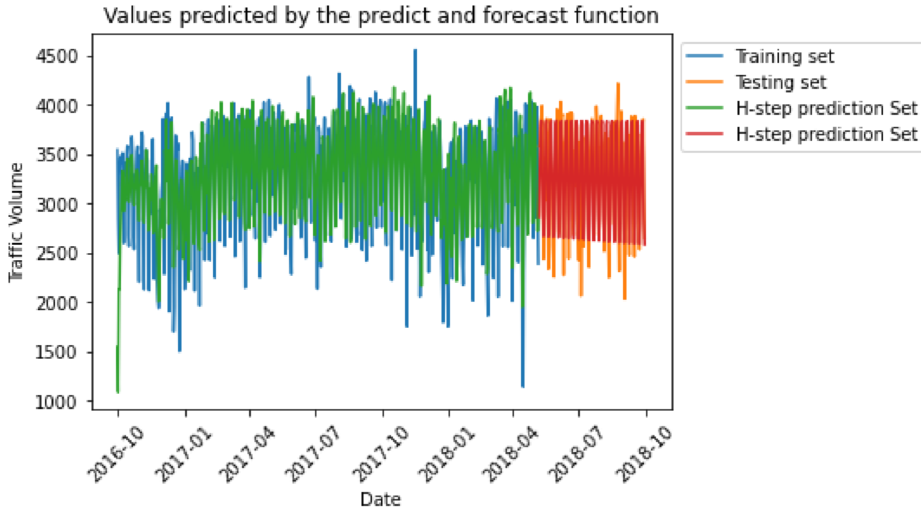
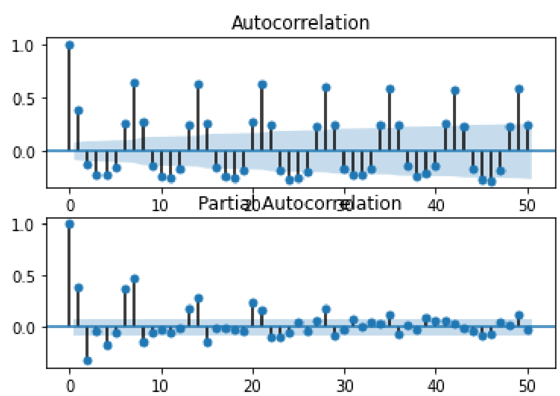


Figure 16

SARIMA Model:

Since we observe strong seasonality, we perform SARIMA model on the dataset.

The Acf and PCAF are as shows below:



Since there is no clear cutoff or trail off it is difficult to find out the order of the SARIMA model hence we perform Auto SARIMA.

Following is the order obtained:

```
Best model:  ARIMA(0,0,3)(0,1,1)[7] intercept
Total fit time: 36.500 seconds
```

Following is the summary of the SARIMAX Model:

SARIMAX Results			
Dep. Variable:	y	**No. Observations:**	584
Model:	SARIMAX(0, 0, 3)x(0, 1, [1], 7)	**Log Likelihood**	-4181.383
Date:	Wed, 16 Dec 2020	**AIC**	8374.766
Time:	19:52:43	**BIC**	8400.913

Sample:	0	HQIC	8384.962
	- 584		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	2.8898	2.519	1.147	0.251	-2.047	7.827
ma.L1	0.3758	0.035	10.729	0.000	0.307	0.444
ma.L2	0.1366	0.054	2.538	0.011	0.031	0.242
ma.L3	0.0641	0.047	1.355	0.175	-0.029	0.157
ma.S.L7	-0.9209	0.017	-54.909	0.000	-0.954	-0.888
sigma2	1.111e+05	4535.412	24.501	0.000	1.02e+05	1.2e+05

Ljung-Box (Q):	64.43	Jarque-Bera (JB):	782.25
Prob(Q):	0.01	Prob(JB):	0.00
Heteroskedasticity (H):	1.79	Skew:	-1.56
Prob(H) (two-sided):	0.00	Kurtosis:	7.78

Base-Models:

We also perform base models Average, Naïve, Drift, SES, Holt-Linear and Holt-Winter Methods foe with results are discussed in the next section.

Final Model selection:

The results for all the models performed on the dataset are as follows.

Methods/Values	Q-Value	R-value	MSE Prediction	MSE forecast	Mean Prediction Err	Mean Forecast Err	Var of Prediction Err	Var of Forecast Err
Average	893.88	1	327442.22	282937.54	58.03	55.48	324074.73	279859.84
Naive	644.57	1	393938.74	1191763.08	-1.98	954.94	393934.82	279859.84
Drift	641.83	0.99	398446.03	1497497.77	2.92	1101.43	398437.49	284349.23
SES	993.41	1	377483.02	492899.34	-2.27	461.56	377477.89	279859.84
Holt-Linear	937.24	1	316945.19	280808.81	8.99	30.81	316864.41	279859.84
Holt-Winter	84.35	0.41	124792.18	106484.61	-5.19	-117.35	124765.22	92714.55
SARIMA	296.25	0.56	222608.94	88359.25	93.23	62.67	213917.96	84431.72
ARMA(3,6)	62.5	0.68	184160.18	143525.41	8.68	8.95	184084.86	143445.32
ARMA(5,5)	14.32	0.43	148895.8	98287.35	5.85	-23.9	148861.59	97716.32
ARMA(5,7)	11.85	0.46	143495.5	95854.1	6.97	5.9	143446.93	95819.24

From the summary of all the models above we select SARIMA Method. Since the Model has Average Best of all the stats models and least RMSE value. The autocorrelation values and predicted values are as shown below in the plots Figure 17 to Figure 20

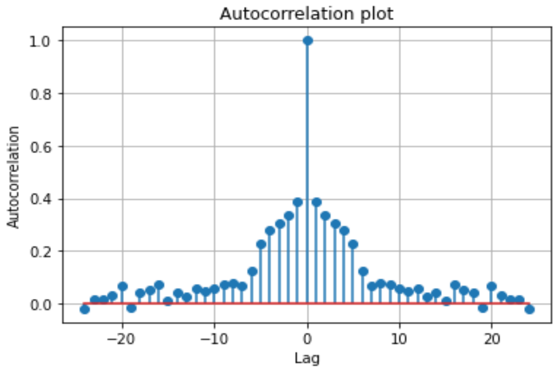


Figure 17

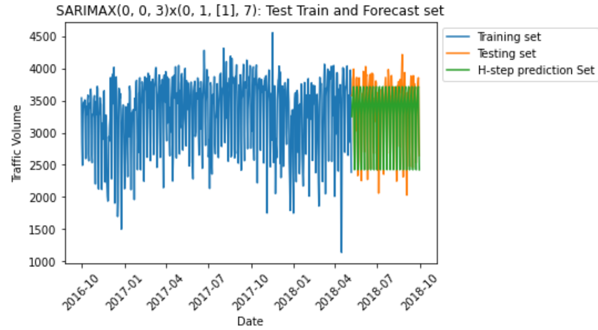


Figure 18

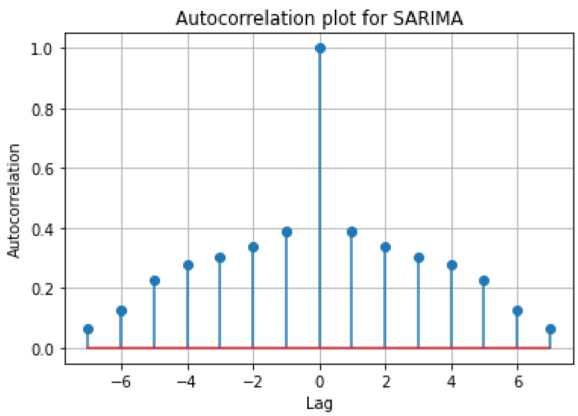


Figure 19

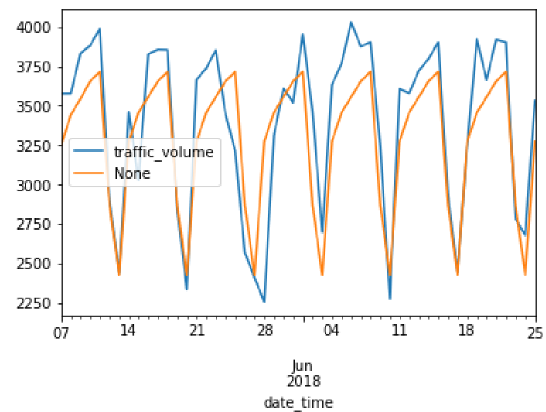
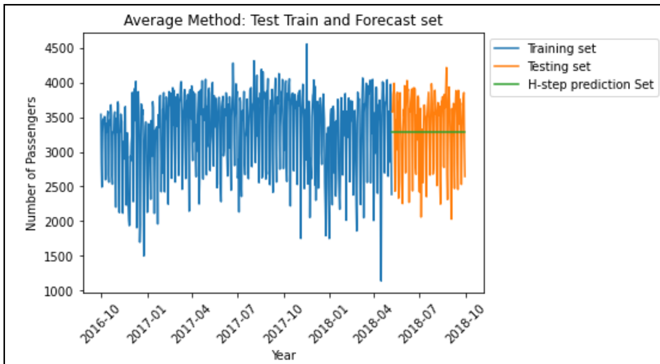
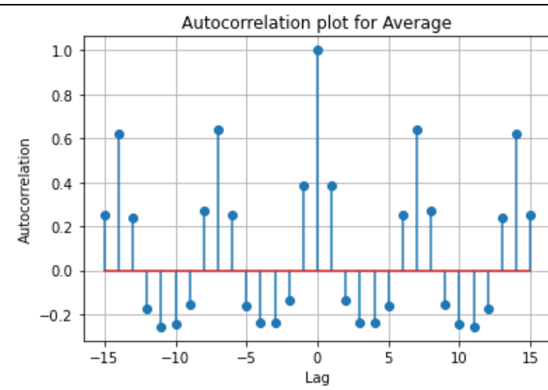
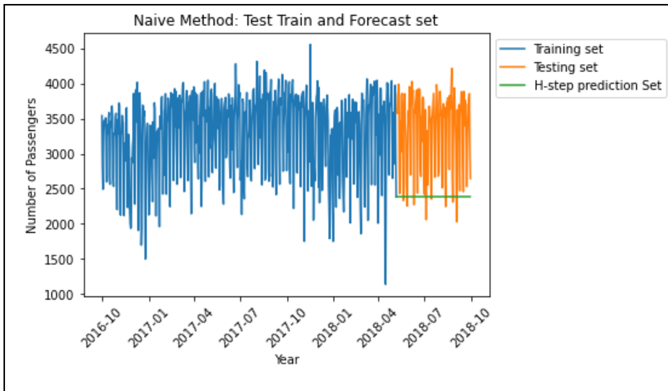


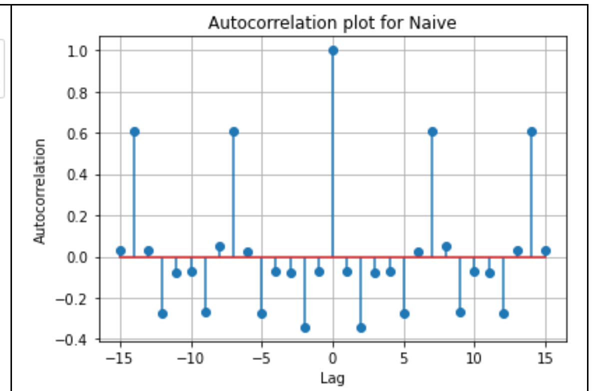
Figure 20

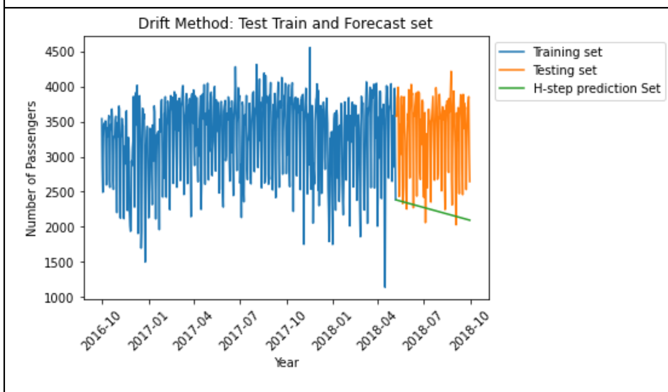
Below are the ACF and prediction plots for all the models.

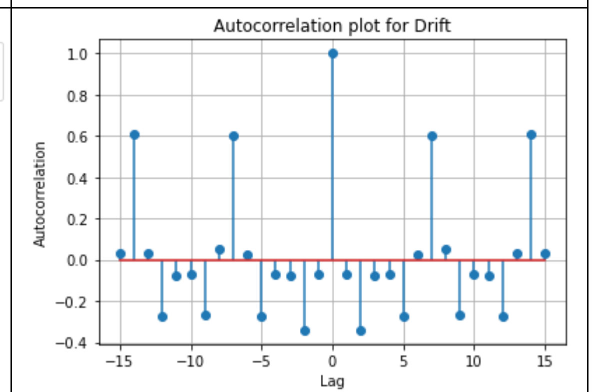


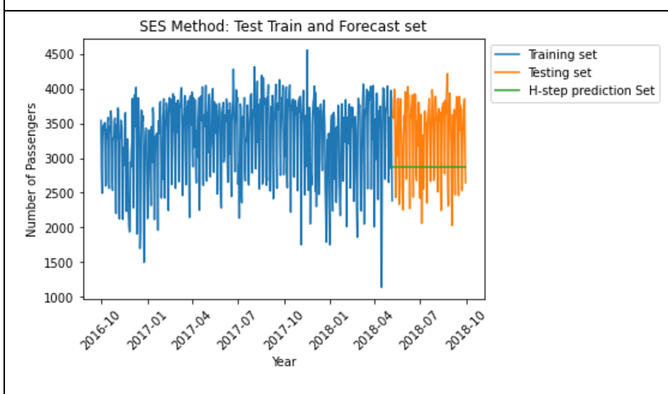


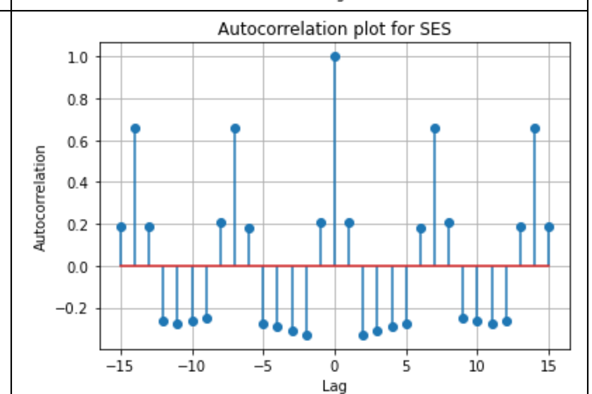

Naive Method: Test Train and Forecast set
Number of Passengers
Year
Training set
Testing set
H-step prediction Set

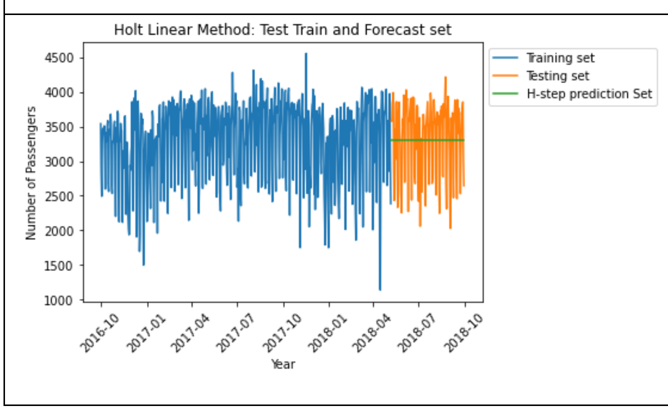

Autocorrelation plot for Naive
Autocorrelation
Lag

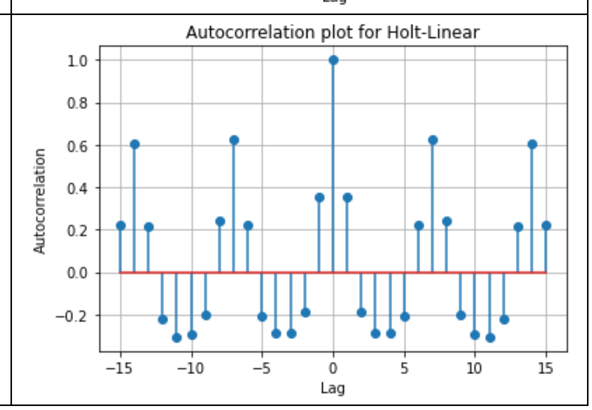

Drift Method: Test Train and Forecast set
Number of Passengers
Year
Training set
Testing set
H-step prediction Set

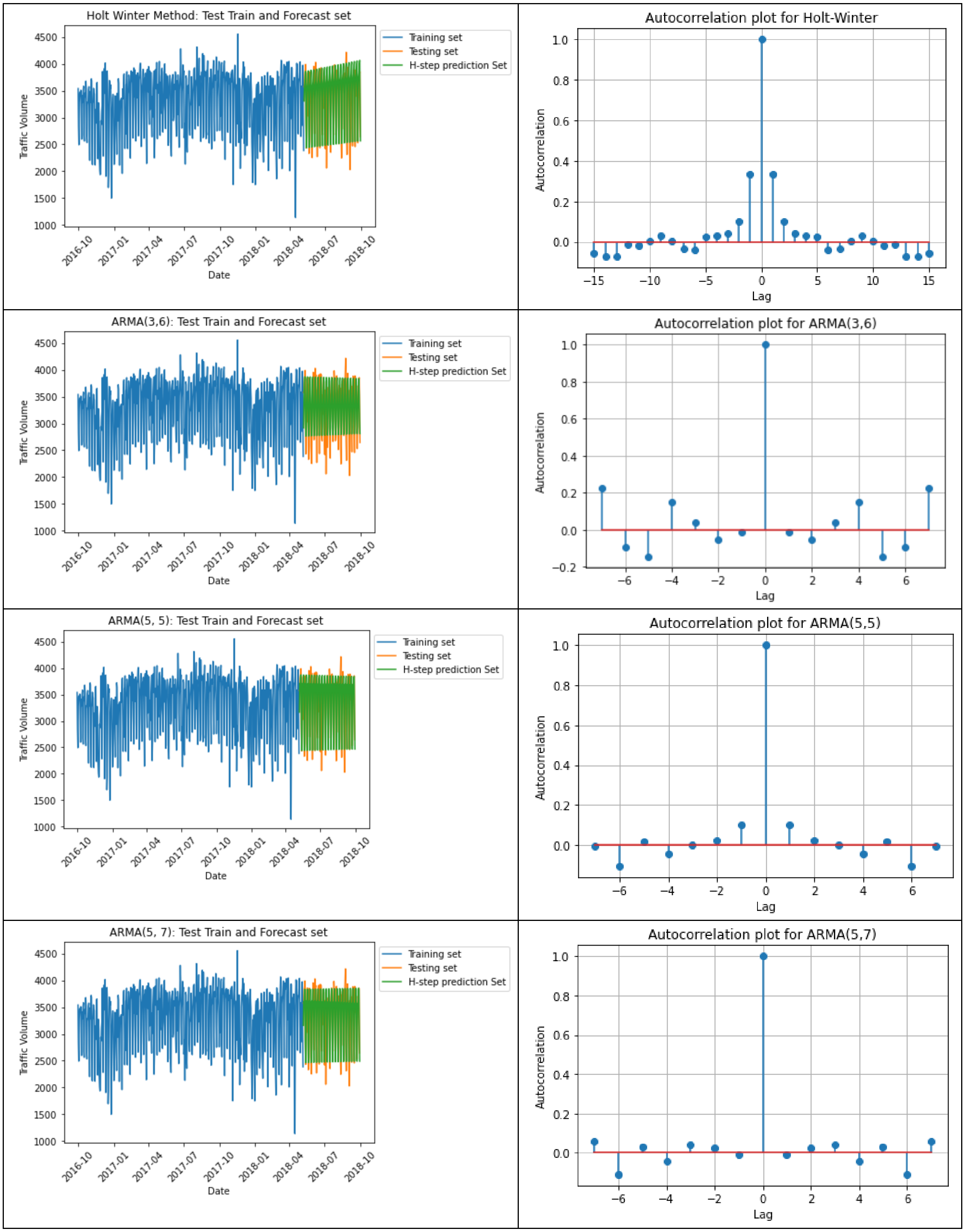

Autocorrelation plot for Drift
Autocorrelation
Lag

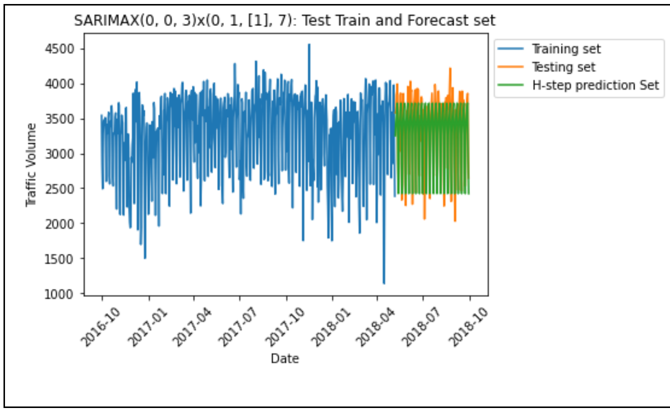

SES Method: Test Train and Forecast set
Number of Passengers
Year
Training set
Testing set
H-step prediction Set

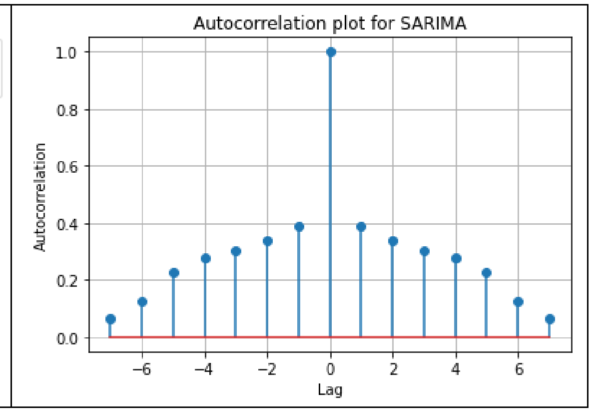

Autocorrelation plot for SES
Autocorrelation
Lag


Holt Linear Method: Test Train and Forecast set
Number of Passengers
Year
Training set
Testing set
H-step prediction Set


Autocorrelation plot for Holt-Linear
Autocorrelation
Lag


Holt Winter Method: Test Train and Forecast set
Training set
Testing set
H-step prediction Set
Traffic Volume
Date
Autocorrelation plot for Holt-Winter
Autocorrelation
Lag
ARMA(3,6): Test Train and Forecast set
Training set
Testing set
H-step prediction Set
Traffic Volume
Date
Autocorrelation plot for ARMA(3,6)
Autocorrelation
Lag
ARMA(5, 5): Test Train and Forecast set
Training set
Testing set
H-step prediction Set
Traffic Volume
Date
Autocorrelation plot for ARMA(5,5)
Autocorrelation
Lag
ARMA(5, 7): Test Train and Forecast set
Training set
Testing set
H-step prediction Set
Traffic Volume
Date
Autocorrelation plot for ARMA(5,7)
Autocorrelation
Lag





Summary and conclusion:

Since our data has strong seasonality of 0.83 as well as performed SARIMAX model gives the best results of all the other models we choose SARIMAX as our best model.

We could also perform further analysis of our model and implement SARIMA model to get the better results for prediction of our dataset. Also future work would include to work with prediction and forecast function for SARIMA.

Reference:

- https://www.aboutcivil.org/traffic-volume-study.html
- https://archive.ics.uci.edu/ml/datasets/Metro+Interstate+Traffic+Volume#
- https://www.geeksforgeeks.org/python-arima-model-for-time-series-forecasting/

Appendix A: Attribute Information

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

Appendix B: Code

```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from sklearn.model_selection import train_test_split
import numpy as np
```

```python
import math
import seaborn as sns
from statsmodels.tsa.seasonal import STL
from statsmodels.tsa.seasonal import seasonal_decompose
import statsmodels.tsa.holtwinters as ets
from sklearn.preprocessing import MinMaxScaler
from scipy.stats import chi2
from scipy.signal import dlsim
from pmdarima import auto_arima
import statsmodels.api as sm
import warnings
warnings.filterwarnings("ignore")



SEED = 42
np.set_printoptions(suppress=True)

# Functions
# Method calculates ACF
def acf_(y, k):
    m = np.mean(y)
    num, den = 0, 0
    for i in range(len(y)):
        den = den + ((y[i] - m) ** 2)
    for t in range(k, len(y)):
        num = num + ((y[t] - m) * (y[t - k] - m))
    return num / den


# function for plotting acf vs lag
def acf_plot(acf_list, l):
    lag = np.arange(int(-1 * (len(acf_list) / 2)), int(1 + len(acf_list) /
2))
    plt.stem(lag, acf_list, use_line_collection=True)
    if l:
        plt.title('Autocorrelation plot for ' + l)
    else:
        plt.title('Autocorrelation plot')
    plt.xlabel('Lag')
    plt.ylabel('Autocorrelation')
    plt.grid(True)
    plt.show()


# Method calculates ACF for given lags
def acf(data, lag, l, plot):
    i = 'a'
    if lag > len(data):
        print("lag is more than ", len(data) - 1)
        return
    acf_list = []
    for j in range(-1 * (lag), lag + 1):
        if j < 0:
            # since acf of -lag = acf of +lag
            i = j * (-1)
        else:
```

```
            i = j
        acf_list.append(acf_(data, i))
    if plot:
        acf_plot(acf_list, l)
    else:
        return acf_list

def phi(ry, j, k):
    num = np.zeros((k, k))
    den = np.zeros((k, k))
    for q in range(k):
        for p in range(k):
            if q != k - 1:
                num[p][q] = ry[np.abs(j + p - q)]
                den[p][q] = ry[np.abs(j + p - q)]
            else:
                num[p][q] = ry[np.abs(j + 1 + p)]
                den[p][q] = ry[np.abs(j - k + p + 1)]
    num = np.round(np.linalg.det(num), 5)
    den = np.round(np.linalg.det(den), 5)
    if den == 0.0:
        return math.inf
    return num / den


def cal_gpac(ry, rows, cols):
    gpac = np.zeros((rows, cols))
    for p in range(rows):
        for q in range(1, cols + 1):
            gpac[p][q - 1] = np.round(phi(ry, p, q), 2)
    gpac_df = pd.DataFrame(gpac)
    col_names = np.arange(1, cols + 1)
    gpac_df.columns = col_names
    return gpac_df


def plot_gpac(df, lable):
    plt.figure(figsize=(12,12))
    sns.heatmap(df, annot=True, vmin = -1, vmax = 1, cmap='coolwarm')
    plt.xlabel('AR process (k)')
    plt.ylabel('MA process (j)')
    lable = '' + lable
    plt.title(lable)
    plt.show()

def mean(y):
    return sum(y)/len(y)

def corr_cal(x,y):
    x_bar = mean(x)
    y_bar = mean(y)
    num = 0
    denx,deny = 0,0
    if len(x)!=len(y):
        print("Corr could not be calculated with operands of shapes
",len(x)," ",len(y))
    if len(x)==len(y):
```

```
        for t in range(len(x)):
            num = num + ((x[t]-x_bar)*(y[t]-y_bar))
            denx = denx + ((x[t]-x_bar)**2)
            deny = deny + ((y[t]-y_bar)**2)
        return np.round(num/(((denx)*(deny))**(0.5)),2)

def acf_(y,k):
    m = mean(y)
    num,den=0,0
    for i in range(len(y)):
        den=den+((y[i]-m)**2)
    for t in range(k,len(y)):
        num=num+((y[t]-m)*(y[t-k]-m))
    return num/den

#function for plotting acf vs lag
def acf_plot(acf_list,l):
    lag=np.arange(int(-1*(len(acf_list)/2)),int(1+len(acf_list)/2))
    plt.stem(lag,acf_list, use_line_collection = True)
    if l:
        plt.title('Autocorrelation plot for '+l)
    else: plt.title('Autocorrelation plot')
    plt.xlabel('Lag')
    plt.ylabel('Autocorrelation')
    plt.grid(True)
    plt.show()
    return

def acf(data,lag,l,plot):
    i = 'a'
    if lag>len(data):
        print("lag is more than ",len(data)-1)
        return
    acf_list =[]
    for j in range(-1*(lag),lag+1):
        if j<0:
            #since acf of -lag = acf of +lag
            i = j*(-1)
        else: i = j
        acf_list.append(acf_(data,i))
    if plot:
        acf_plot(acf_list,l)
    return acf_list

def avg(yt):
    return mean(yt)

def avg_method(yt,y_fr_len):
    y_pr = []
    for i in range(1,len(yt)):
        y_pr.append(avg(yt[:i]))

    y_fr = []
    for i in range(y_fr_len):
        y_fr.append(avg(yt))
    return y_pr,y_fr
```

```python
def naive(yt):
    return yt[-1]

def naive_method(yt,y_fr_len):
    y_pr = []
    for i in range(1,len(yt)):
        y_pr.append(naive(yt[:i]))

    y_fr = []
    for i in range(y_fr_len):
        y_fr.append(naive(yt))
    return y_pr,y_fr

def drift(yt,h):
    if len(yt)==1:
        return yt[0]
    return yt[-1]+h*((yt[-1]-yt[0])/(len(yt)-1))

def drift_method(yt,y_fr_len):
    y_pr = []
    for i in range(1,len(yt)):
        y_pr.append(drift(yt[:i],1))

    y_fr = []
    for i in range(y_fr_len):
        y_fr.append(drift(yt,i+1))
    return y_pr,y_fr


def ses(yt,alpha):
    if len(yt)<1:
        print("yt<1")
        return 0
    elif len(yt)==1:
        return yt[0]
    return (alpha*yt[-1]) + ((1-alpha)*ses(yt[:-1],alpha))

def ses_method(yt,y_fr_len,alpha):
    y_pr = []
    for i in range(1,len(yt)):
        y_pr.append(ses(yt[:i],alpha))

    y_fr = []
    for i in range(y_fr_len):
        y_fr.append(ses(yt,alpha))
    return y_pr,y_fr

def holt_linear(yt,l):
    model = ets.SimpleExpSmoothing(yt)
    fit = model.fit()
    y_pr = fit.fittedvalues
    y_fr = fit.forecast(l)
    return y_pr,y_fr

def plot(x1,x2,y1,y2,y3,xl,yl,l):
    plt.figure()
    plt.plot(x1,y1, label='Training set')
```

```
    plt.plot(x2,y2, label='Testing set')
    plt.plot(x2,y3, label='H-step prediction Set')
    plt.xlabel(xl)
    plt.ylabel(yl)
    plt.title(l)
    plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
    plt.xticks(rotation = 45)
    plt.show()

def err(t,a):
    return (t-a)

def mean_err(t,a):
    return np.round(mean(err(t,a)),2)

def mse_calc(y, y_pred):
    return np.round(sum(err(y,y_pred)**2)/len(y),2)

def var_calc(y):
    return np.round(sum((y-mean(y))**2)/(len(y)),2)

def box_pierce_test(y,y_pred,k):
    avg_acf = np.array(acf(err(y,y_pred),lag = k,l = "",plot = False))
    return np.round(len(y) * np.sum((avg_acf[:k])**2),2)

def plotall(x1,x2,y1,y2,y3,y4,y5,y6,y7,y8,xl,yl,l):
    plt.figure()
    plt.plot(x1,y1, label='Training set')
    plt.plot(x2,y2, label='Testing set')
    plt.plot(x2,y3, label='H-step prediction Average')
    plt.plot(x2,y4, label='H-step prediction Naive')
    plt.plot(x2,y5, label='H-step prediction Drift')
    plt.plot(x2,y6, label='H-step prediction SES')
    plt.plot(x2,y7, label='H-step prediction Holt-Linear')
    plt.plot(x2,y8, label='H-step prediction Holt-Winalet Seasonal')
    plt.xlabel(xl)
    plt.ylabel(yl)
    plt.title(l)
    plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
    plt.xticks(rotation = 45)
    plt.show()

#Initializing lag
k=15

val_table = {"headers":['method\values','Q-Value','R-value','MSE
Prediction','MSE forecast', 'Mean Prediction Err', 'Mean Forecast Err', 'Var
of Prediction Err','Var of Forecast Err']}
def table_data(method, p_train, p_pr, p_test, p_fr):
    if method in ['Average','Naive','Drift','SES']:
        p_train = p_train[1:]
    l = []
    l.append(method)
    l.append(box_pierce_test(p_train,p_pr,k))
    l.append(corr_cal(err(p_test,p_fr),p_test))
    l.append(mse_calc(p_train,p_pr))
    l.append(mse_calc(p_test,p_fr))
```

```python
    l.append(mean_err(p_train,p_pr))
    l.append(mean_err(p_test,p_fr))
    l.append(var_calc(err(p_train,p_pr)))
    l.append(var_calc(err(p_test,p_fr)))
    val_table[method] = l
    acf_avg = acf(err(p_train,p_pr),lag = k,l = method,plot = True)
    return

#8 Holt-Winters method: Using the Holt-Winters method try to find the best
# fit using the train dataset and make a prediction using the test set.
def holt_winter_seasonl(yt,l,t,s,period):
    model = ets.ExponentialSmoothing(yt, trend=t, seasonal=s,
seasonal_periods=period)
    fit = model.fit()
    y_pr = fit.fittedvalues
    y_fr = fit.forecast(l)
    return y_pr, y_fr

#9 Feature selection

def lm_fit(features, label, intercept=True):
    features = sm.add_constant(features)
    return sm.OLS(label, features).fit()


def lm_predict(model, test):
    test = sm.add_constant(test, has_constant='add')
    return model.predict(test)


def LSE_fit(x_train, y_train):
    x_train = sm.add_constant(x_train)
    x_trans = np.transpose(x_train)
    return
np.matmul(np.matmul(np.linalg.inv(x_trans.dot(x_train)),x_trans),y_train)


def LSE_predict(model, test):
    test = test.to_numpy()
    test = sm.add_constant(test, has_constant='add')
    return np.matmul(test,model)

def err(t,a):
    return (t-a)

def order_checker(orders, y_train, lags, start, end):
    results = []
    i,j = 0,1
    while i<9:
        while j<9:
            a, b = i,j
            model = sm.tsa.ARMA(y_train, (a,b)).fit(trend="nc", disp=0)
            y_hat_pred = model.predict(start=start, end=end)
            e = err(y_train, y_hat_pred)
            acf(e, lags, "", True)
            re = acf(e, lags, "", False)
            #acf(e, lags, "Residual Rrror", True)
```

```
            Q = len(y_train)*np.sum(np.square(re[:lags]))
            DOF = lags - a - b
            alfa = 0.01
            chi_critical = chi2.isf(alfa, df=DOF)
            if Q < chi_critical:
                print("The Residuals are White")
                results.append((a, b))
            else:
                print("Residuals ",a,b," are not White")
            print("Q: ", Q, " chi_critical: ", chi_critical)
            j += 1
        i+= 1
        j = 0
    return results

#12 Levenberg Marquardt algorithm

# Functions
def e_calculator(y, na, nb, theta):
    den = np.r_[1, theta[:na]]
    num = np.r_[1, theta[na:]]
    if na < nb:
        den = np.concatenate([den, np.zeros(nb - na)])
    elif na > nb:
        num = np.concatenate([num, np.zeros(na - nb)])
    sys = (den, num, 1)
    _, e = dlsim(sys, y)
    return e

def LM_step0(na, nb):
    return np.zeros(na + nb).flatten()


def LM_step1(y, na, nb, theta, delta):
    e = e_calculator(y, na, nb, theta)
    sse = np.matmul(e.T, e)
    X = []
    for i in range(len(theta)):
        theta_new = theta.copy()
        theta_new[i] = theta[i] + delta
        e_new = e_calculator(y, na, nb, theta_new)
        xi = (e - e_new) / delta
        X.append(xi)
    X = np.column_stack(X)
    A = np.matmul(X.T, X)
    g = np.matmul(X.T, e)
    return sse, A, g


def LM_step2(y, na, nb, theta, A, g, mu):
    I = np.identity(na + nb)
    muI = mu * I
    delta_theta = np.matmul(np.linalg.inv(A + muI), g).flatten()
    theta_new = theta + delta_theta
    e_new = e_calculator(y, na, nb, theta_new)
    sse_new = np.matmul(e_new.T, e_new)
    return delta_theta, theta_new, sse_new
```

```
def LM_step3(y, na, nb, theta, delta, A, g, mu, delta_theta, theta_new, sse,
sse_new, MAX_ITER, MAX_MU):
    iter, iter_list, sse_list = 0, [0], [sse[0][0]]
    while iter < MAX_ITER:
        print("Iteration: ", iter, " SSE: ", sse_new[0][0])
        iter_list.append(iter+1)
        sse_list.append(np.round((sse_new[0][0]), 3))
        if sse_new < sse:
            if np.linalg.norm(delta_theta) < 10 ** -3:
                theta = theta_new
                err_var = sse_new / (len(y) - (na + nb))
                cov_theta = np.multiply(err_var, np.linalg.inv(A))
                return theta, err_var, cov_theta, iter_list, sse_list
            else:
                theta = theta_new
                mu /= 10

        while sse_new >= sse:
            mu *= 10
            if mu > MAX_MU:
                print("mu exceeded mu max value")
                return None, None, None, None, None
            delta_theta, theta_new, sse_new = LM_step2(y, na, nb, theta, A,
g, mu)

        iter += 1
        if iter > MAX_ITER:
            print("iter exceeded iter max value")
            return None, None, None, None, None

        theta = theta_new
        sse, A, g = LM_step1(y, na, nb, theta, delta)
        delta_theta, theta_new, sse_new = LM_step2(y, na, nb, theta, A, g,
mu)


def LM_parameter_estimator(y, na, nb):
    # step 0
    theta = LM_step0(na, nb)

    # step 1
    delta = 10 ** -6
    sse, A, g = LM_step1(y, na, nb, theta, delta)

    # step 2
    mu = 0.01
    delta_theta, theta_new, sse_new = LM_step2(y, na, nb, theta, A, g, mu)

    # step 3
    MAX_ITER = 100
    MAX_MU = 10 ** 27
    return LM_step3(y, na, nb, theta, delta, A, g, mu, delta_theta,
theta_new, sse, sse_new, MAX_ITER, MAX_MU)
```

```python
def confidence(theta, cov_mat, n):
    confi_list = []
    for i in range(n):
        upper = theta[i] + 2 * np.sqrt(cov_mat[i][i])
        lower = theta[i] - 2 * np.sqrt(cov_mat[i][i])
        confi_list.append([upper, lower])
    return confi_list


def h_step_prediction(na, nb, train, y_hat_1, length):
    y_hat_h = []
    for h in range(length):
        y_term = 0
        for j in range(len(na)):
            if h < len(na) and (-(j + 1) + h) < 0:
                y_term += na[j] * train[-(j + 1) + h]
            else:
                y_term += na[j] * y_hat_h[h - (j + 1)]
        e_term = 0
        if h < len(nb):
            for j in range(len(nb)):
                if (-(j + 1) + h) < 0:
                    e_term += nb[j] * (train[-(j + 1) + h] - y_hat_1[-(j + 1)
+ h - 1])
        y_hat_h.append(np.round(e_term - y_term, 3))
    return y_hat_h


def one_step_prediction(na, nb, y_train):
    y_hat_1 = [0]
    for i in range(len(y_train)):
        min_a = min(i, len(na) - 1)
        y_part = 0
        for j in range(min_a + 1):
            y_part += na[j] * y_train[i - j]
        min_b = min(i, len(nb) - 1)
        e_part = 0
        for j in range(min_b + 1):
            e_part += nb[j] * (y_train[i - j] - y_hat_1[i - j])
        y_hat_1.append(np.round((e_part - y_part), 3))
    return y_hat_1[1:]

def ar_ma_process(a,b,y_train, tr_start, tr_end, ts_start, ts_end):
    model = sm.tsa.ARMA(y_train, (a,b)).fit(trend="nc", disp=0)
    y_pr = model.predict(start=tr_start, end=tr_end)
    y_fr = model.predict(start=ts_start, end=ts_end)
    return y_pr,y_fr

data = pd.read_csv('Metro_Interstate_Traffic_Volume.csv.gz',
compression='gzip', error_bad_lines=False)
data.to_csv(r'C:\Users\yadav\Downloads\Time
series\Project\Metro_Interstate_Traffic_Volume.csv', index = False,
header=True)
raw_data = pd.read_csv('Metro_Interstate_Traffic_Volume.csv')

#Initial Data Analysis and visualization
print(raw_data.head(5))
```

```
print(raw_data.info())

df = raw_data
df = df.replace("None", np.nan)


#setting date time as an index
# object to datetime format
df["date_time"] = pd.to_datetime(df["date_time"], format="%Y-%m-%d")
df = df.set_index(df["date_time"])

df = df.loc['2015-10-30':'2018-09-30'].copy(deep=True)

#Since the project concentrates on the day wise data, in further steps data
is groped based on date.
df = df.groupby(pd.Grouper(freq="D")).aggregate(
        { "temp": "mean", "rain_1h": "mean", "snow_1h": "mean", "clouds_all":
"mean",
         "traffic_volume": "mean", "weather_main": "first", "holiday":
"first"})

#grouping together minor values

#Replace None by np.nan
df["holiday"] = df["holiday"].replace({np.nan: "No"})
df["holiday"] = df["holiday"].replace(["Independence Day", "Columbus Day",
"Christmas Day", "Veterans Day",
                                       "Memorial Day", "Thanksgiving Day",
"New Years Day", "Labor Day",
                                       "State Fair", "Washingtons Birthday",
"Martin Luther King Jr Day"], "Yes")

# weather_main: Clear-776, Clouds-524, Rain-205, Mist-166, Snow-104, Drizzle-
28, Haze-25, Thunderstorm-22, Fog-10
df["weather_main"] = df["weather_main"].replace({"Drizzle": "Rain",
"Haze":"Rain", "Thunderstorm": "Rain", "Fog":"Mist"})

#Viewing the metadata after preprocessing
print("\n============================================================
===================")
print(df.isnull().sum())
print(df[df.isna().any(axis=1)])
print(df.head(5))
print(df.info())
print(df.describe())
for column in ["weather_main", "holiday"]:
    print(column, " : ", df[column].value_counts())

print(df.shape)

# Data Visualization
plt.figure(figsize=(8,6))
plt.plot(df["traffic_volume"], label='Traffic Volume')
plt.legend()
plt.xticks(rotation = 45)
plt.xlabel('Date')
plt.ylabel('Traffic Volume')
```

```
plt.title("Traffic volume vs Date")
plt.show()

plt.figure(figsize=(8,6))
sns.countplot(y='holiday', data= df, palette="Set2")
plt.xticks(rotation = 45)
plt.xlabel('Count')
plt.ylabel('Holiday')
plt.title("Holiday vs Count")
plt.show()

plt.figure(figsize=(8,6))
sns.countplot(y="weather_main", data= df,  palette="Set2")
plt.xticks(rotation = 45)
plt.xlabel('Count')
plt.ylabel('Weather')
plt.title("Weather vs Count")
plt.show()

plt.show()
plt.figure(figsize=(8,6))
sns.boxplot(y="temp", data= df,  palette="Set2")
plt.xticks(rotation = 45)
plt.xlabel('')
plt.ylabel('Temperature')
plt.title("Boxplot for Temperature data")
plt.show()

plt.show()
plt.figure(figsize=(8,6))
plt.hist(df.rain_1h.loc[df.rain_1h<1])
plt.xticks(rotation = 45)
plt.xlabel('')
plt.ylabel('Rain')
plt.title("Histogram for Rain data")
plt.show()

plt.show()
plt.figure(figsize=(8,6))
plt.hist(df["snow_1h"])
plt.xticks(rotation = 45)
plt.xlabel('')
plt.ylabel('Snow')
plt.title("Histogram for Snow data")
plt.show()

plt.show()
plt.figure(figsize=(8,6))
sns.boxplot(y="clouds_all", data= df,  palette="Set2")
plt.xticks(rotation = 45)
plt.xlabel('')
plt.ylabel('Clouds')
plt.title("Boxplot for Clouds data")
plt.show()


# Saving the clean dataset to CSV
```

```
df.to_csv(r'cleaned_dataset.csv')


#5a. Plot of the dependent variable versus time.
plt.figure(figsize=(8,6))
plt.plot(df["traffic_volume"], label='Traffic Volume')
plt.legend()
plt.xticks(rotation = 45)
plt.xlabel('Date')
plt.ylabel('Traffic Volume')
plt.title("Traffic volume vs Date")
plt.show()

#5a. Plotting only first 100 values
plt.figure(figsize=(8,6))
plt.plot(df["traffic_volume"][:100], label='Traffic Volume')
plt.legend()
plt.xticks(rotation = 45)
plt.xlabel('Date')
plt.ylabel('Traffic Volume')
plt.title("Traffic volume vs Date")
plt.show()


#5a. Plotting only first 100 values
plt.figure(figsize=(8,6))
plt.plot(df["traffic_volume"][:30], label='Traffic Volume')
plt.legend()
plt.xticks(rotation = 45)
plt.xlabel('Date')
plt.ylabel('Traffic Volume')
plt.title("Traffic volume vs Date")
plt.show()

#5b. ACF of the dependent variable.
acf(df["traffic_volume"], 20, "", True)

#5c. Correlation Matrix with seaborn heatmap and Pearson’s correlation
coefficient.
corr = df.corr()
sns.heatmap(corr, annot=True)
plt.title("Heatmap of Correlation Matrix")
plt.show()

#5e. Split the dataset into train set (80%) and test set (20%).
train, test = train_test_split(df, shuffle=False, test_size=0.2,
random_state=SEED)

print("Shape of the entire dataset: ", df.shape, "\nShape of the train set :
",
      train.shape, "\nShape of the test set : ", test.shape)


# Stationarity: Check for a need to make the dependent variable stationary.

print("ADF test for traffic_volume")
result = adfuller(df["traffic_volume"])
```

```
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))

print("P-value is 0.000078 i.e. less than 0.05(95% or more confidence
interval), hence we reject Null Hypothesis and conclude that the dependent
variable is stationary.")


df = df.loc['2016-09-30':'2018-09-30'].copy(deep=True)

# 5e. Split the dataset into train set (80%) and test set (20%).
train, test = train_test_split(df, shuffle=False, test_size=0.2,
random_state=SEED)

print("Shape of the entire dataset: ", df.shape, "\nShape of the train set :
",
      train.shape, "\nShape of the test set : ", test.shape)

# 7 model{“additive”, “multiplicative”}
decomposed_data = seasonal_decompose(df["traffic_volume"], "additive",
period=None )
decomposed_data.plot()
plt.title("title")
plt.show()

decomposed_data = seasonal_decompose(df["traffic_volume"], "multiplicative",
period=None )
decomposed_data.plot()
plt.title("title")
plt.show()


decomposed_data = STL(df["traffic_volume"])
res = decomposed_data.fit()
fig = res.plot()
plt.show()

t = res.trend
s = res.seasonal
r = res.resid

plt.figure()
plt.plot(df["traffic_volume"],label='Original set')
plt.plot(t,label='Trends')
plt.plot(s,label='Seasonality')
plt.plot(r,label='Remainders')
plt.title("Plot of Original dataset, Trends, Seasonality, and Remainders")
plt.xlabel("Date")
plt.ylabel("Traffic Volume")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.xticks(rotation = 45)
plt.show()
```

```python
print("The strength of trend for this data set is:",np.round(max(0,(1-
np.var(r)/np.var(t+r))),2))
print("The strength of seasonality for this data set is:",np.round(max(0,(1-
np.var(r)/np.var(s+r))),2))

deseasoned = df["traffic_volume"]-s

plt.figure()
plt.plot(df["traffic_volume"][:30],label='Original set')
plt.plot(deseasoned[:30],label='deseasoned')
plt.plot(s[:30],label='Seasonality')
plt.title("Plot of Original dataset, Trends, Seasonality, and Remainders")
plt.xlabel("Date")
plt.ylabel("Traffic Volume")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.xticks(rotation = 45)
plt.show()


p_train = np.array(train["traffic_volume"])
p_test = np.array(test["traffic_volume"])
year = pd.date_range(start='2016-09-30', end='2018-09-30', freq = 'D')
year_pr = year[:len(train)]
year_fr = year[len(train):]


p_pr_avg, p_fr_avg = avg_method(p_train,len(p_test))
plot(year_pr,year_fr,p_train,p_test,p_fr_avg,"Year","Number of
Passengers","Average Method: Test Train and Forecast set")

p_pr_naive, p_fr_naive = naive_method(p_train,len(p_test))
plot(year_pr,year_fr,p_train,p_test,p_fr_naive,"Year","Number of
Passengers","Naive Method: Test Train and Forecast set")

p_pr_drift, p_fr_drift = drift_method(p_train,len(p_test))
plot(year_pr,year_fr,p_train,p_test,p_fr_drift,"Year","Number of
Passengers","Drift Method: Test Train and Forecast set")

p_pr_ses, p_fr_ses = ses_method(p_train,len(p_test),0.5)
plot(year_pr,year_fr,p_train,p_test,p_fr_ses,"Year","Number of
Passengers","SES Method: Test Train and Forecast set")

p_pr_hlinear,p_fr_hlinear = holt_linear(p_train,len(p_test))
plot(year_pr,year_fr,p_train,p_test,p_fr_hlinear,"Year","Number of
Passengers","Holt Linear Method: Test Train and Forecast set")

p_pr_holtseasonal,p_fr_holtseasonal =
holt_winter_seasonl(train["traffic_volume"],len(test),'mul','mul',None)
plot(year_pr,year_fr,train["traffic_volume"],test["traffic_volume"],p_fr_holt
seasonal,"Date","Traffic Volume","Holt Winter Method: Test Train and Forecast
set")

plotall(year_pr,year_fr,p_train,p_test,p_fr_avg,p_fr_naive,p_fr_drift,p_fr_se
s,p_fr_hlinear,p_fr_holtseasonal,"Year","Number of Passengers","Test Train
and Forecast set")

table_data('Average', p_train, p_pr_avg, p_test, p_fr_avg)
```

```
table_data('Naive', p_train, p_pr_naive, p_test, p_fr_naive)
table_data('Drift', p_train, p_pr_drift, p_test, p_fr_drift)
table_data('SES', p_train, p_pr_ses, p_test, p_fr_ses)
table_data('Holt-Linear', p_train, p_pr_hlinear, p_test, p_fr_hlinear)
table_data('Holt-Winter', p_train, p_pr_holtseasonal, p_test,
p_fr_holtseasonal)

# To begin with feature selection using OLS model we need to reformat our
data to scaled numeric datset.

# convert categorical data to numeric usinng one hot encoding.
df = pd.get_dummies(df)

# Specify the name of the target
target = 'traffic_volume'

# separate train and test data
lm_train, lm_test = train_test_split(df, shuffle=False, test_size=0.2,
random_state=SEED)

# Scaling numeric values MinMax Scaler exclude target.

mm = MinMaxScaler()

# Scaling the training data
lm_train_mm =
pd.DataFrame(mm.fit_transform(lm_train[np.setdiff1d(lm_train.columns,
[target])]),
    columns=np.setdiff1d(lm_train.columns, [target]))
lm_train_mm.set_index(lm_train.index, inplace=True)
lm_train_mm[target] = lm_train[target]

# Scaling the testing data
lm_test_mm = pd.DataFrame(mm.transform(lm_test[np.setdiff1d(lm_test.columns,
[target])]),
    columns=np.setdiff1d(lm_test.columns, [target]))
lm_test_mm.set_index(lm_test.index, inplace=True)
lm_test_mm[target] = lm_test[target]

# linear model using all variables
lm_model = lm_fit(lm_train_mm[np.setdiff1d(lm_train_mm.columns,
target)],lm_train_mm[target])

print("\n======================================================================
===================")
print("                  The summary of the model with all variables.")
print("======================================================================
==================")
print(lm_model.summary())

# linear model after feature selection
lm_model = lm_fit(lm_train_mm[np.setdiff1d(lm_train_mm.columns,
[target,"snow_1h","rain_1h","clouds_all","weather_main_Snow"])],lm_train_mm[t
arget])

print("\n======================================================================
===================")
```

```
print("              The summary of the model after features elimination.")
print("============================================================================
=================")
print(lm_model.summary())

#10 LSE

lse_model = LSE_fit(lm_train_mm[np.setdiff1d(lm_train_mm.columns,
[target,"snow_1h","rain_1h","clouds_all","weather_main_Snow"])],lm_train_mm[t
arget])
print("The Coefficients for the LSE model are:\n",lse_model)


y_hat_LSE = LSE_predict(lse_model,
lm_test_mm[np.setdiff1d(lm_test_mm.columns,
[target,"snow_1h","rain_1h","clouds_all","weather_main_Snow"])])
e_LSE = lm_test_mm[target]-y_hat_LSE

y_hat_OLS = lm_predict(lm_model, lm_test_mm[np.setdiff1d(lm_test_mm.columns,
[target,"snow_1h","rain_1h","clouds_all","weather_main_Snow"])])
e_OLS = lm_test_mm[target]-y_hat_OLS

print(sum(e_LSE), sum(e_OLS))

df['1diff'] = df[target] - df[target].shift(1)
df['2diff'] = df[target] - 2 * (df[target].shift(1)) + df[target].shift(2)
df['log'] = np.log(df[target])
df['log_2diff'] = df['log'] - 2 * (df['log'].shift(1)) + df['log'].shift(2)

train['1diff'] = train[target] - train[target].shift(1)
train['2diff'] = train[target] - 2 * (train[target].shift(1)) +
train[target].shift(2)
train['log'] = np.log(train[target])
train['log_2diff'] = train['log'] - 2 * (train['log'].shift(1)) +
train['log'].shift(2)


test['log'] = np.log(test[target])
test['log_2diff'] = test['log'] - 2 * (test['log'].shift(1)) +
test['log'].shift(2)
test['1diff'] = test[target] - test[target].shift(1)
test['2diff'] = test[target] - 2 * (test[target].shift(1)) +
test[target].shift(2)

# Plot showing Month vs #Passengers_diff
plt.figure()
plt.plot(df[target][:30], label='original')
plt.plot(df['1diff'][:30], label='1D')
plt.plot(df['2diff'][:30], label='2D')
# plt.plot(df['log_2diff'][:30], label='Log 2 D')
plt.legend()
plt.xlabel("Month")
plt.xticks(rotation=45)
plt.ylabel("Passengers_diff")
plt.title("Plot showing Month vs #Passengers_diff")
plt.show()
```

```
# 11
j = 12
k = 12
lags = j + k


y = train["traffic_volume"]
actual_output = test["traffic_volume"]


# autocorrelation of traffic volume
ry_acf = acf(y, lags, "", False)


g_df = cal_gpac(ry_acf[lags:], 12, 12)
label = ("Heatmap ")
plot_gpac(g_df, label)

oberved_orders =
[(2,0),(2,2),(4,0),(4,2),(4,7),(7,11),(8,1),(8,8),(9,0),(9,5),(9,7),(10,1),(1
0,3),(11,2)]

start = '2016-09-30'
end = '2018-05-06'

oberved_orders = [(4,6)]


a,b=3, 6
theta, err_var, cov_theta, iter_list, sse_list =
LM_parameter_estimator(train["traffic_volume"].to_numpy(), a, b)
model = sm.tsa.ARMA(y, (a, b)).fit(trend="nc", disp=0)
max_ab = max(a, b)  # theta[:a], theta[a:]
an = [0] * max_ab
bn = [0] * max_ab
for i in range(a):
    an[i] = theta[i]
for i in range(b):
    bn[i] = theta[i+a]

print("The Model Summary is as Follows:")
print("Final parameters are: ", np.round(theta,2))
print(an, bn)

confidence_interval = confidence(theta, cov_theta, a + b)
j, k = 1, 1
print("Confidence Interval are: ")
for i in range(a + b):
    if i < a:
        print("% 5.3f < a%d < % 5.3f" % (confidence_interval[i][0], j,
confidence_interval[i][1]))
        j += 1
    else:
        print("% 5.3f < b%d < % 5.3f" % (confidence_interval[i][0], k,
confidence_interval[i][1]))
        k += 1
```

```
# ARMA (3, 6)
a,b = 3, 6
theta, err_var, cov_theta, iter_list, sse_list =
LM_parameter_estimator(train["traffic_volume"].to_numpy(), a, b)
model = sm.tsa.ARMA(y, (a, b)).fit(trend="nc", disp=0)
max_ab = max(a, b)  #theta[:a], theta[a:]
an = [0] * max_ab
bn = [0] * max_ab
for i in range(a):
    an[i] = theta[i]
for i in range(b):
    bn[i] = theta[i+a]

print("The Model Summary is as Follows:")
print("Final parameters are: ", np.round(theta,2))
print(an, bn)

confidence_interval = confidence(theta, cov_theta, a + b)
j, k = 1, 1
print("Confidence Interval are: ")
for i in range(a + b):
    if i < a:
        print("% 5.3f < a%d < % 5.3f" % (confidence_interval[i][0], j,
confidence_interval[i][1]))
        j += 1
    else:
        print("% 5.3f < b%d < % 5.3f" % (confidence_interval[i][0], k,
confidence_interval[i][1]))
        k += 1

y_train = train["traffic_volume"]
y_hat_1 = one_step_prediction(an, bn, y_train)
y_hat_h = h_step_prediction(an, bn, y_train, y_hat_1, len(test))

year_pr = year[:len(train)]
year_fr = year[len(train):]

print(len(train["traffic_volume"]),len(test["traffic_volume"]), len(y_hat_1),
len(y_hat_h))
plt.figure()
plt.plot(year_pr,train["traffic_volume"], label='Training set')
plt.plot(year_fr,test["traffic_volume"], label='Testing set')
plt.plot(year_pr,y_hat_1, label='H-step prediction Set')
plt.plot(year_fr,y_hat_h, label='H-step prediction Set')
plt.xlabel("Date")
plt.ylabel("Traffic Volume")
plt.title("Values predicted by the predict and forecast function")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.xticks(rotation = 45)
plt.show()

trs = '2016-09-30'
tre = '2018-05-06'
tss = '2018-05-07'
tse = '2018-09-30'
```

```
p_pr_arma36,p_fr_arma36 =
ar_ma_process(3,6,train["traffic_volume"],trs,tre,tss,tse)
plot(year_pr,year_fr,train["traffic_volume"],test["traffic_volume"],p_fr_arma
36,"Date","Traffic Volume","ARMA(3,6): Test Train and Forecast set")

p_pr_arma55,p_fr_arma55 =
ar_ma_process(5,5,train["traffic_volume"],trs,tre,tss,tse)
plot(year_pr,year_fr,train["traffic_volume"],test["traffic_volume"],p_fr_arma
55,"Date","Traffic Volume","ARMA(5, 5): Test Train and Forecast set")

p_pr_arma57,p_fr_arma57 =
ar_ma_process(5,7,train["traffic_volume"],trs,tre,tss,tse)
plot(year_pr,year_fr,train["traffic_volume"],test["traffic_volume"],p_fr_arma
57,"Date","Traffic Volume","ARMA(5, 7): Test Train and Forecast set")


table_data('ARMA(3,6)', p_train, p_pr_arma36, p_test, p_fr_arma36)
table_data('ARMA(5,5)',  p_train, p_pr_arma55, p_test, p_fr_arma55)
table_data('ARMA(5,7)',  p_train, p_pr_arma57, p_test, p_fr_arma57)

for key, v in val_table.items():
    a, b, c, d, e, f, g, h, i= v
    print ("{:<17} {:<11} {:<11} {:<18} {:<17} {:<22} {:<22} {:<25}
{:<25}".format(a,b,c,d,e,f,g,h,i))

fig, ax = plt.subplots(2,1)
fig = sm.graphics.tsa.plot_acf(train["traffic_volume"], lags=50, ax=ax[0])
fig = sm.graphics.tsa.plot_pacf(train["traffic_volume"], lags=50, ax=ax[1])
plt.show()


stepwise_fit = auto_arima(train["traffic_volume"], start_p = 1, start_q = 1,
                          max_p = 9, max_q = 9, m = 7,
                          start_P = 0, seasonal = True,
                          d = None, D = 1, trace = True,
                          error_action ='ignore',
                          stepwise = True)

# To print the summary
stepwise_fit.summary()


# Fit a SARIMAX(0, 0, 3)x(0, 1, [1], 7) on the training set

from statsmodels.tsa.statespace.sarimax import SARIMAX

sarima = SARIMAX(train["traffic_volume"],
                order = (0, 0, 3),
                seasonal_order =(0, 1, [1], 7))

result = sarima.fit()
result.summary()

trs = '2016-09-30'
tre = '2018-05-06'
tss = '2018-05-07'
tse = '2018-09-30'
```

```python
p_pr_sarima = result.predict(trs, tre)
p_fr_sarima = result.predict(tss, tse)

e = err(train["traffic_volume"], p_pr_sarima)
acf(e, lags, "", True)
re = acf(e, lags, "", False)

Q = len(train["traffic_volume"])*np.sum(np.square(re[:lags]))
a = 0
b = 3
DOF = lags - a - b
alfa = 0.01
chi_critical = chi2.ppf(1-alfa, DOF)

if Q < chi_critical:
    print("The Residuals are White")
else:
    print("Residuals are not White")

print("Q: ", Q, " chi_critical: ", chi_critical)

plot(train["traffic_volume"],test["traffic_volume"],p_fr_sarima,"Date","Traff
ic Volume","SARIMAX(0, 0, 3)x(0, 1, [1], 7): Test Train and Forecast set")
table_data('SARIMA', p_train, p_pr_sarima, p_test, p_fr_sarima)

print(train["traffic_volume"])

for key, v in val_table.items():
    a, b, c, d, e, f, g, h, i= v
    print ("{:<17} {:<11} {:<11} {:<18} {:<17} {:<22} {:<22} {:<25}
{:<25}".format(a,b,c,d,e,f,g,h,i))

# plot predictions and actual values
test["traffic_volume"][:50].plot(legend = True)
p_fr_sarima[:50].plot(legend = True)
```