Appliances Energy

Time Series Analysis & Modeling – Final Project

Sarah Gates
DATS 6450 11

Table of Contents

1. Abstract

This is the final project of the DATS 6450 course section 11 with Dr. Reza Jafari for the Fall 2020 semester. The purpose of this project is to apply time series analytical and modeling techniques to evaluate and preprocess a multivariate time-dependent dataset with the ultimate goal of creating a strong predictive model. Many modeling techniques will be used in the course of this project with various techniques leveraged to assess performance. Based on these measurements, a final model will be selected and presented that generates the best predictions.

2. Introduction

Predicting the appliances energy that a house uses has many implications for energy allocation at the individual user planning level as well as the government planning level. Studying this for an individual home can provide valuable granular information and provide insight into how appliance energy is utilized throughout a given day.

The goal of my project is to build a model using the algorithms learned this semester to predict energy usage in Kilowatt Hour (Wh) of appliances of a single low energy house. The metrics logged are temperature and humidity in various rooms of the house and outdoor conditions. This house is located in Belgium near the Chèvres Airport weather station where outdoor conditions are also recorded. The original metrics were logged every 10 minutes (totaling 19,735 data points). In order to make the data more workable, I refactored the data to 30-minute intervals (totaling, 6579 data points). I took the sum of appliance usage and the average of metrics logged.

3. Description of Dataset

3.1 About the Data

A description of each field in the dataset is summarized in figure 3.1.1:

Figure 3.1.1

Variable	Description
date	date of recording
half_hour	half hour interval of recording
Sum of Appliances	energy use in Wh
Sum of lights	energy use of light fixtures in the house in Wh
Average of T1	Temperature in kitchen area, in Celsius
Average of RH_1	Humidity in kitchen area, in %
Average of T2	emperature in living room area, in Celsius
Average of RH_2	Humidity in living room area, in %
Average of T3	Temperature in laundry room area
Average of RH_3	Humidity in laundry room area, in %
Average of T4	Temperature in office room, in Celsius
Average of RH_4	Humidity in office room, in %
Average of T5	Temperature in bathroom, in Celsius
Average of RH_5	Humidity in bathroom, in %
Average of T6	Temperature outside the building (north side), in Celsius
Average of RH_6	Humidity outside the building (north side), in %
Average of T7	Temperature in ironing room , in Celsius
Average of RH_7	Humidity in ironing room, in %
Average of T8	Temperature in teenager room 2, in Celsius
Average of RH_8	Humidity in teenager room 2, in %
Average of T9	Temperature in parents room, in Celsius
Average of RH_9	Humidity in parents room, in %
Average of T_out	Temperature outside (from Chievres weather station), in Celsius
Average of Press_mm_hg	Humidity outside (from Chievres weather station), in %
Average of RH_out	Humidity outside (from Chievres weather station), in %
Average of Windspeed	Wind speed (from Chievres weather station), in m/s
Average of Visibility	Visibility (from Chievres weather station), in km
Average of Tdewpoint	Tdewpoint (from Chievres weather station)

Appliances energy usage and lights energy usage were two possible target variables, but appliances yielded the higher usage. Figure 3.1.2 shows the first five rows of the dataframe. The “datetime” variable was created by combining the “date” field and “half_hour” field to create a unique time series index variable.

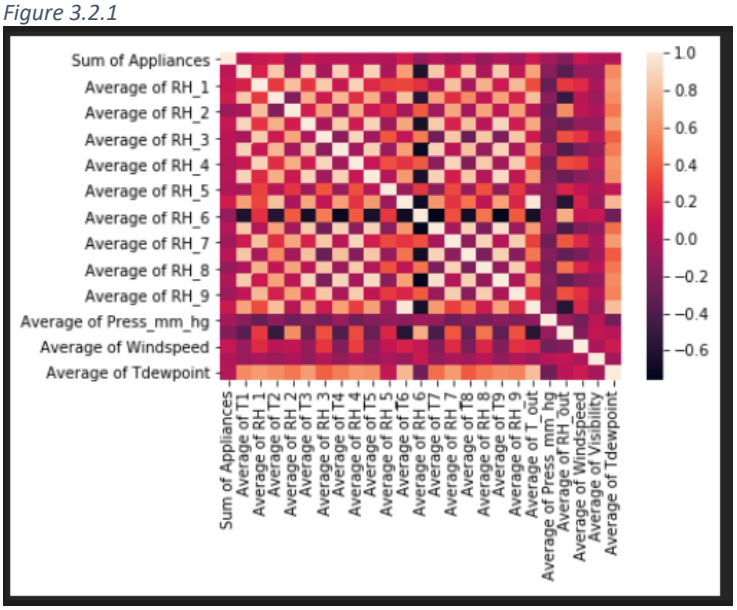
Figure 3.1.2

	date	half_hour	Sum of Appliances	Average of T1	Average of RH_1	Average of T2	Average of RH_2	Average of T3	Average of RH_3	Average of T4	...	Average of RH_8	Average of T9	Average of RH_9	Average of T_out	Average of Press_mm_hg	Average of RH_out	Average of Windspeed	Average of Visibility	Average of Tdewpoint	datetime
0	1/11/2016	5:00:00 PM	170	19.890000	46.863333	19.200000	44.713056	19.790000	44.817778	18.975556	...	48.831111	17.033333	45.530000	6.483333	733.600000	92.000000	6.666667	59.166667	5.200000	2016-01-11 17:00:00
1	1/11/2016	5:30:00 PM	160	19.890000	46.142222	19.200000	44.540000	19.790000	44.977778	18.890000	...	48.590000	17.000000	45.363333	6.133333	733.900000	92.000000	5.666667	47.666667	4.900000	2016-01-11 17:30:00
2	1/11/2016	6:00:00 PM	180	19.845556	45.641389	19.200000	44.477778	19.750000	44.863333	18.890000	...	48.590000	17.000000	45.290000	5.916667	734.166667	91.833333	5.166667	40.000000	4.683333	2016-01-11 18:00:00
3	1/11/2016	6:30:00 PM	880	19.950000	46.116667	19.337778	44.400000	19.790000	44.863333	18.926667	...	48.604444	16.963333	45.290000	5.966667	734.366667	91.333333	5.666667	40.000000	4.633333	2016-01-11 18:30:00
4	1/11/2016	7:00:00 PM	780	20.273333	52.206667	19.717778	45.111111	19.937778	45.973333	19.000000	...	48.806667	16.914444	45.320556	6.000000	734.616667	90.500000	6.000000	40.000000	4.516667	2016-01-11 19:00:00

3.2 Correlation

The next step in the exploratory data analysis of the data was to examine a correlation plot between all variables, displayed in figure 3.2.1. It appears as if a meaningful amount of correlation exists between Appliances and the variables in the dataset.

Figure 3.2.1

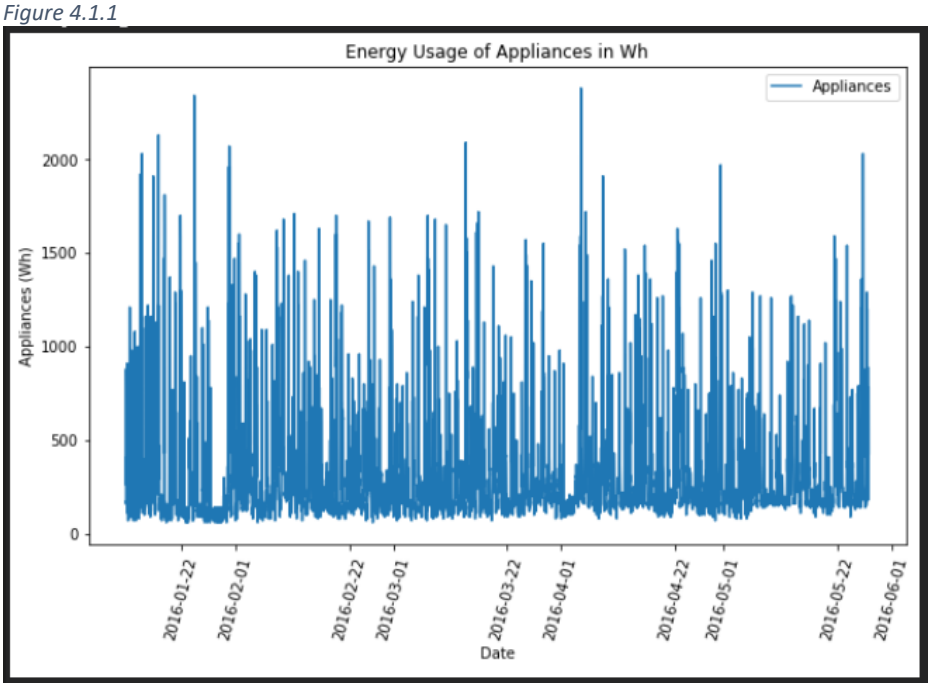


4. Time Series Exploratory Data Analysis

4.1 Stationary Check

It is critical in time series modeling to evaluate if the target variable is stationary, meaning that the mean and variance does not change significantly over time. The appliances variable is plotted in figure 4.1.1. It appears as if the mean does not change and the variance appears stable as well.

Figure 4.1.1



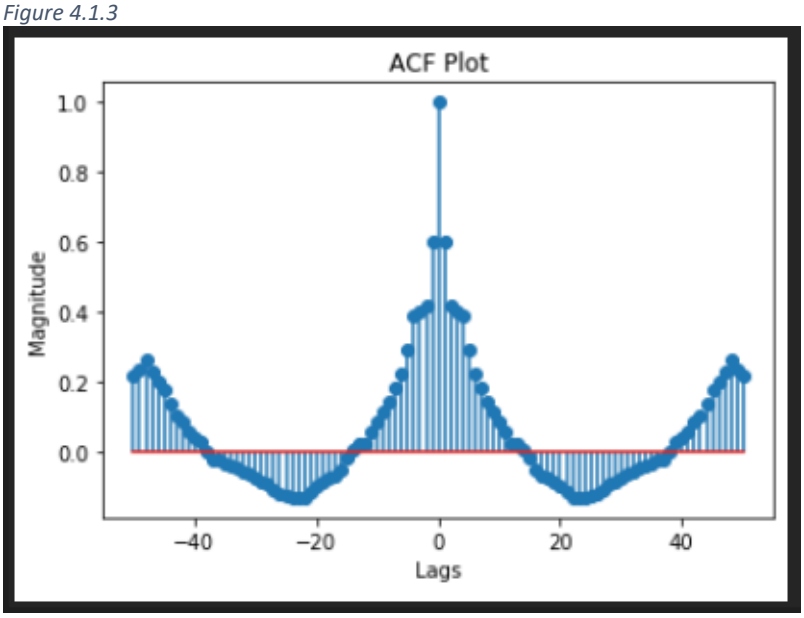
The ADF test allows us to test the stationary nature of the data in a quantifiable way. The ADF statistic in figure 4.1.2 is lower than the critical value at the 1% level, meaning that the null hypothesis can be rejected in favor of the data being stationary. The p value is also extremely close to zero, further supporting this conclusion.

Figure 4.1.2

```
ADF Statistic: -18.473443
p-value: 0.000000
Critical Values:
        1%: -3.431
        5%: -2.862
        10%: -2.567
```

The ACF plot of the target variable at lag value 50 is displayed in figure 4.1.3. Its relatively slow descent after lag 1 indicates that it does not display characteristics of white noise and can indeed be modelled.

Figure 4.1.3



4.2 Time Series Decomposition

We will now perform time series decomposition on the target variable by evaluating the moving average trend compared to the original target variable and the detrended data. Figure 4.2.1 displays moving average window of three. I wanted to start here in order to get a better sense of the trend. There is not much to conclude here since the MA trend data is difficult to analyze.

Figure 4.2.1

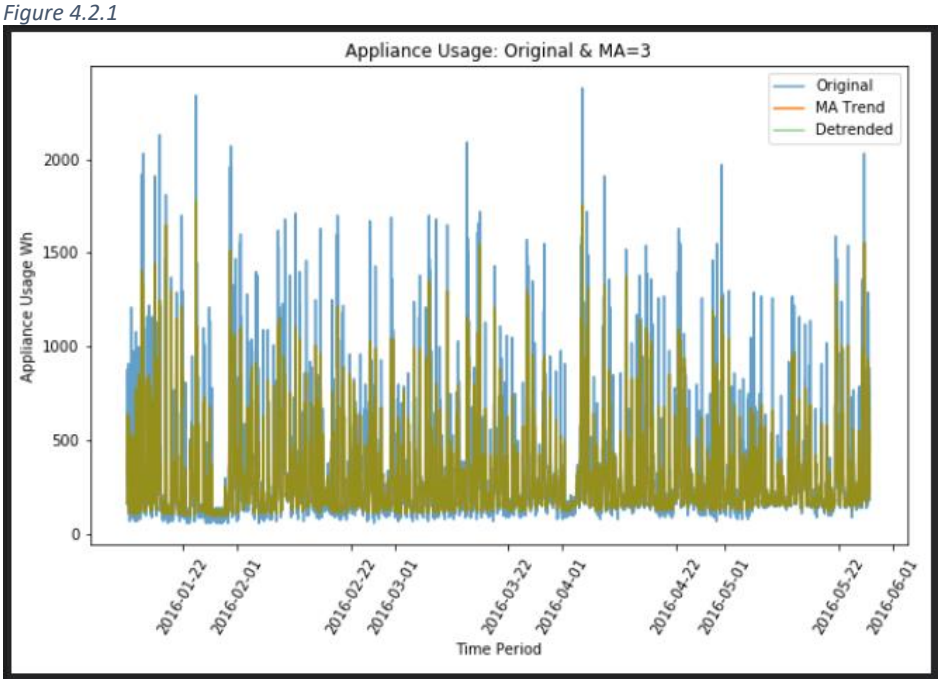


Figure 4.2.2 shows window size 24 and a folding order of 2 (since the window size is even). This gives us a better sense of the trend and possible patterns. I chose 24 since there are 24 hours in a day (and the real cycle of 48 seemed too large). The MA trend and detrended data appears to display a regular cycle.

Figure 4.2.2

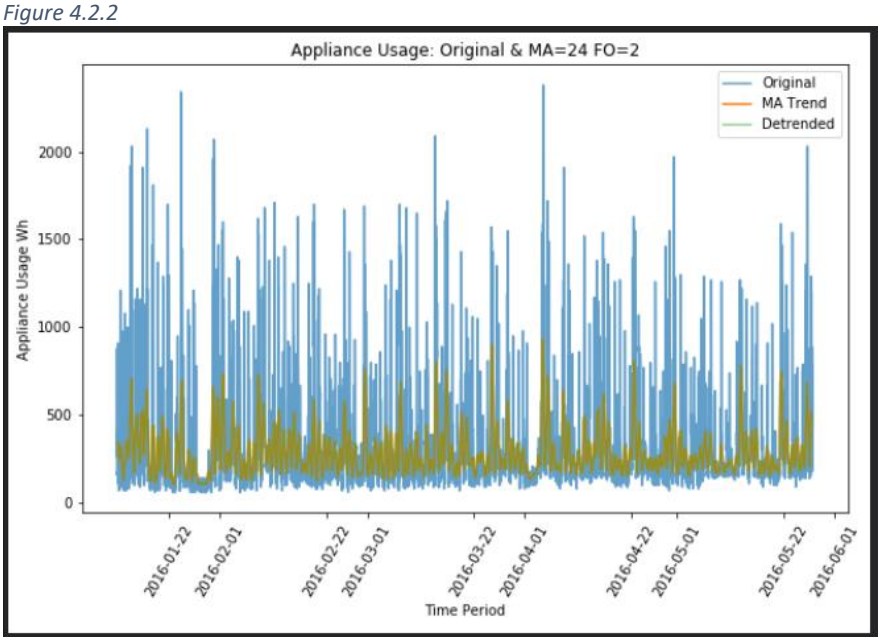


Figure 4.2.3 gives us a full breakdown of the seasonal-trend decomposition (STL).

Figure 4.2.3

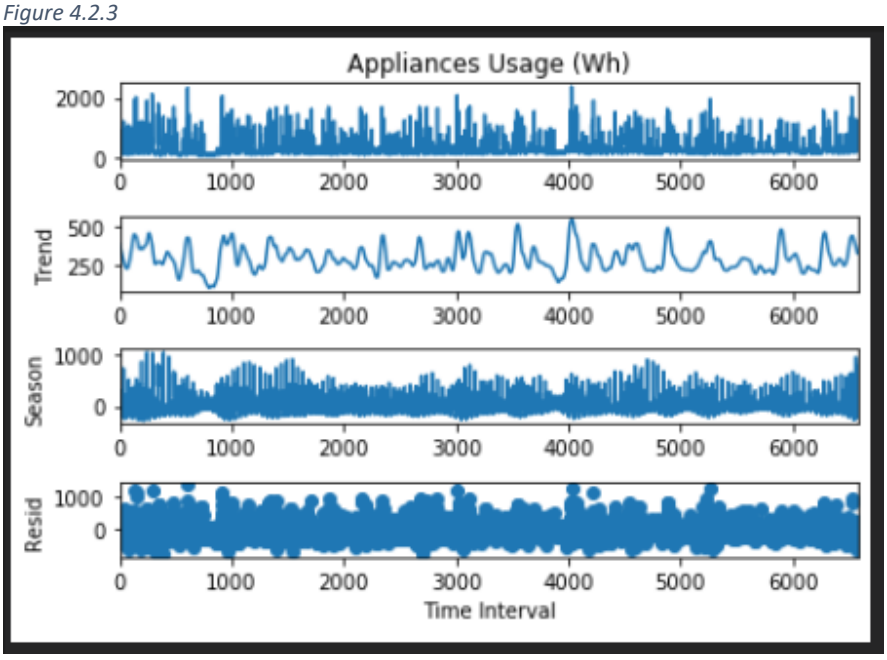
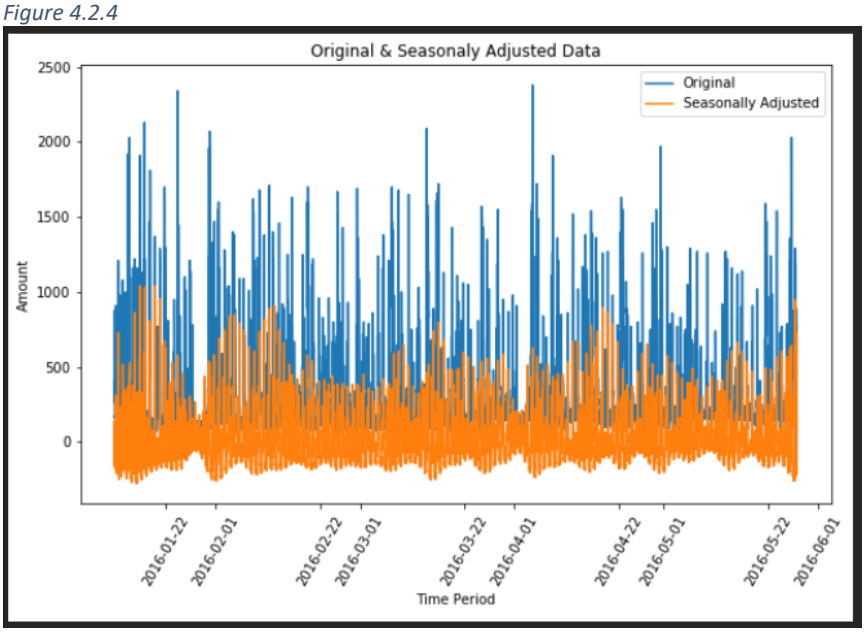


Figure 4.2.4 plots only the seasonally adjusted data along with the original data – it shows a slightly detectable relatively regular seasonal pattern.

Figure 4.2.4



The calculated strength of the trend and seasonality is displayed in figure 4.2.5. As detected earlier in the ADF test, the strength of the trend is very low. The strength of seasonality is neutral at close to .5, but still detectable.

Figure 4.2.5

```
The strength of trend is:  0.23096279468971 67
The strength of seasonality is:  0.5395603380024478
```

5. Model Creation

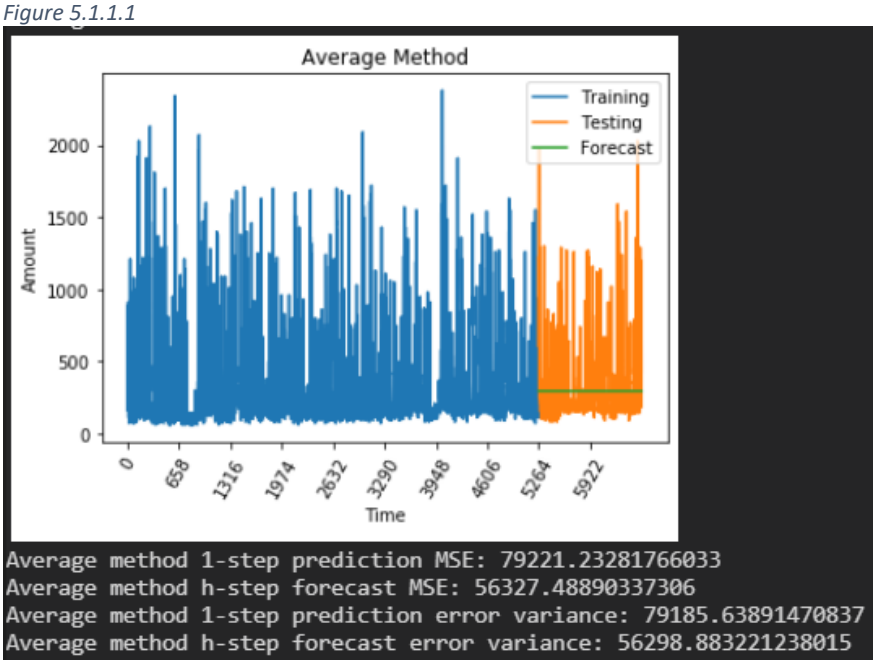
Now that the exploratory data analysis is complete, we will move on to the model creation step. I will use the models studied in class this semester, starting with the base models: average, naïve, drift, and simple exponential smoothing (SES). Then I will implement the Holt-Winters method. Then utilize the Multiple Linear Regression method by utilizing the other variables in the dataset and determining a model using the best combination of predictor variables. Finally I will explore the Autoregressive-moving-average (ARMA) model, the most complex model covered in class. I will perform model performance diagnostics along the way.

5.1 Base Models

5.1.1 Average method

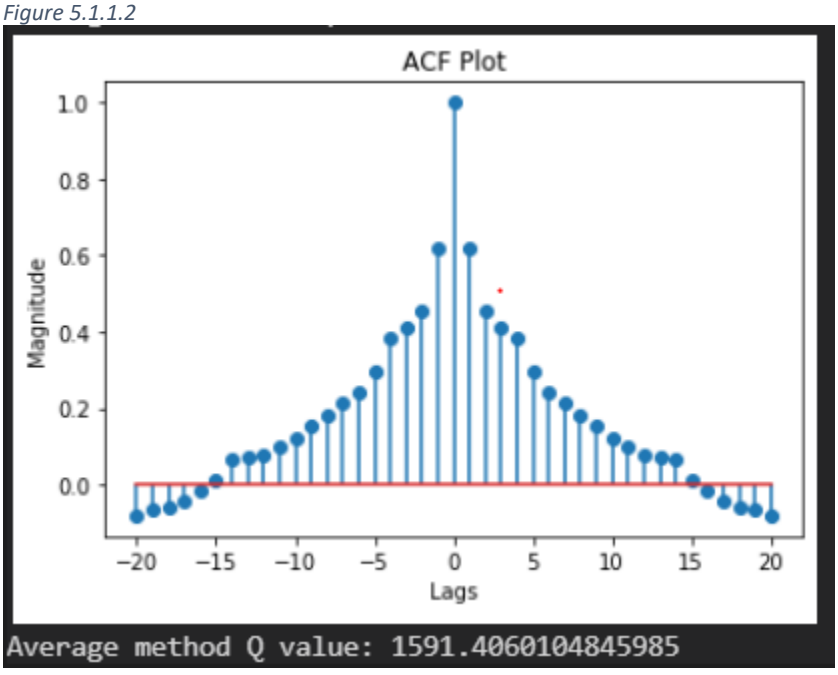
Breaking the data into 80% training and 20% testing, the average method is displayed in figure 5.1.1.1 along with the 1-step prediction MSE, h-step forecast MSE, 1-step prediction error variance, and h-step forecast error variance. All metrics are fairly poor for model evaluation.

Figure 5.1.1.1



The ACF plot of the residuals is displayed in figure 5.1.1.2. It does not display characteristics of white noise, meaning that this is not a good model. The Q value of 1591.4 is also displayed in the figure – the critical value of the chi square test at degrees of freedom = 5263 (training set size) – 2 – 1 = 5260 and probability level 0.05 is 18,014.4. Since the Q value is less than the critical value, we can fail to reject the null and say that the model can be accepted, although the other metrics do not agree.

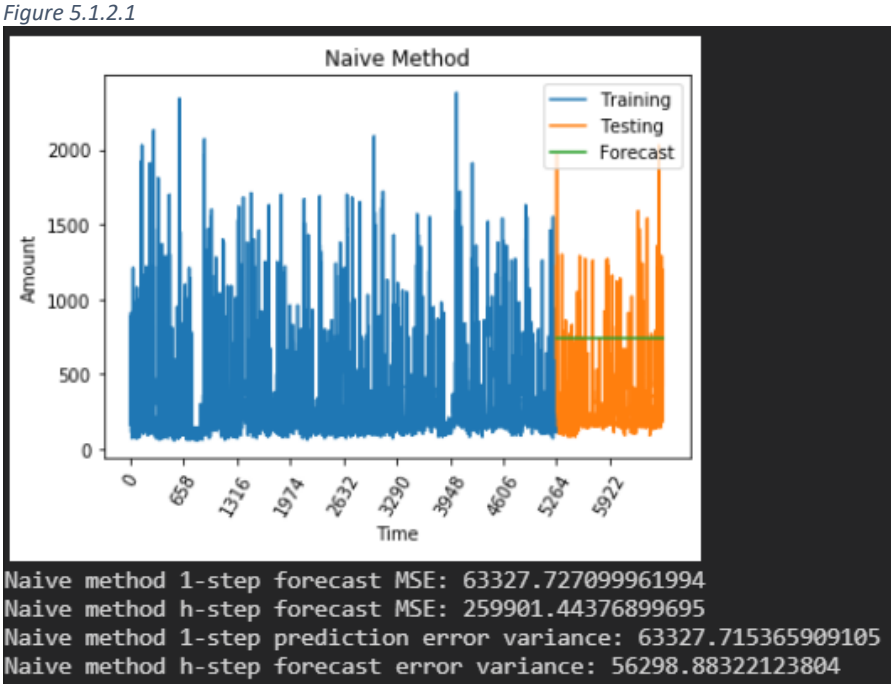
Figure 5.1.1.2



5.1.2 Naïve method

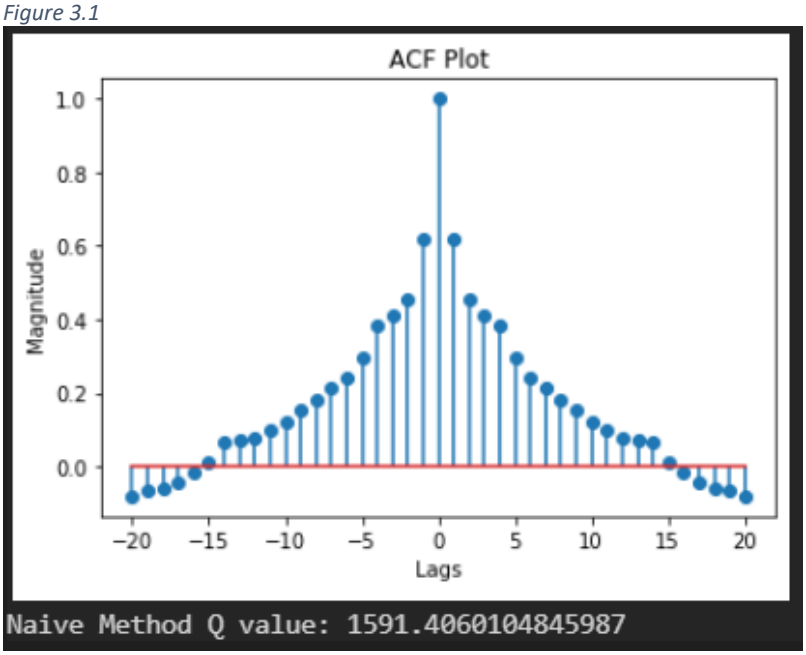
The naïve method is displayed in figure 5.1.2.1 along with the 1-step prediction MSE, h-step forecast MSE, 1-step prediction error variance, and h-step forecast error variance. Again, all metrics are fairly poor for model evaluation.

Figure 5.1.2.1



The ACF plot of the residuals is displayed in figure 5.1.2.2. It also does not display characteristics of white noise, meaning that this is not a good model. The Q value of 1591.4 is also displayed in the figure. It is very close to the average method but not exactly the same. Since the Q value is less than the critical value, we can again fail to reject the null and say that the model can be accepted, although the other metrics do not agree.

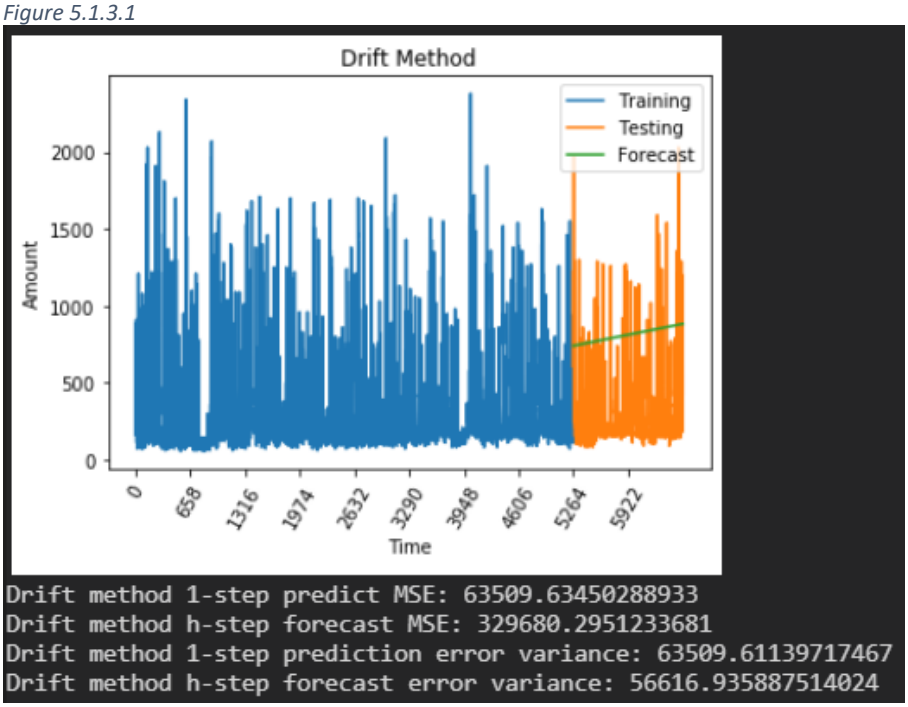
Figure 3.1



5.1.3 Drift method

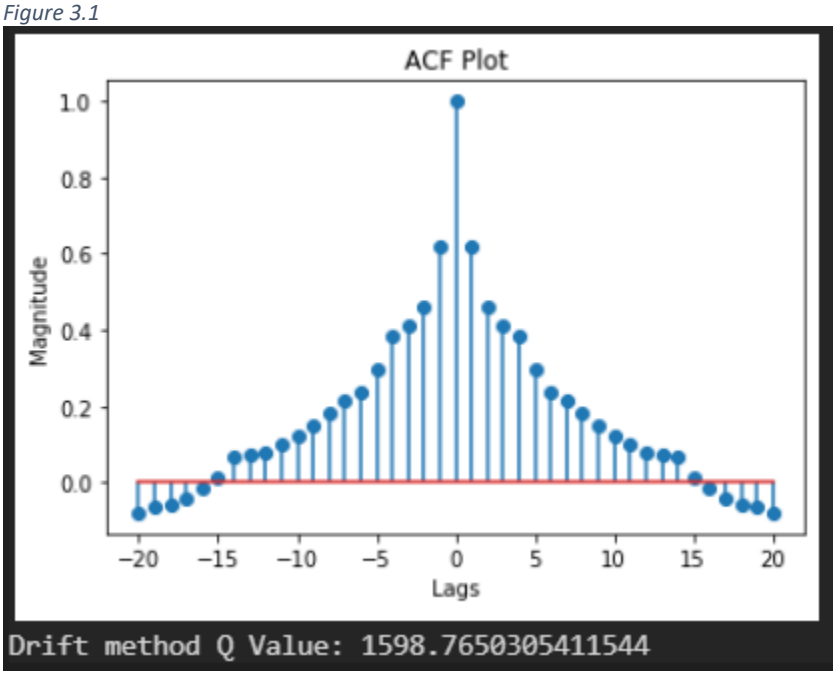
The drift method is displayed in figure 5.1.3.1 along with the 1-step prediction MSE, h-step forecast MSE, 1-step prediction error variance, and h-step forecast error variance. Again, all metrics are fairly poor for model evaluation.

Figure 5.1.3.1



The ACF plot of the residuals is displayed in figure 5.1.3.2. It also does not display characteristics of white noise, meaning that this is not a good model. The Q value of 1598.8 is also displayed in the figure. Since the Q value is less than the critical value (but higher than the other two), we can again fail to reject the null and say that the model can be accepted, although the other metrics do not agree.

Figure 3.1

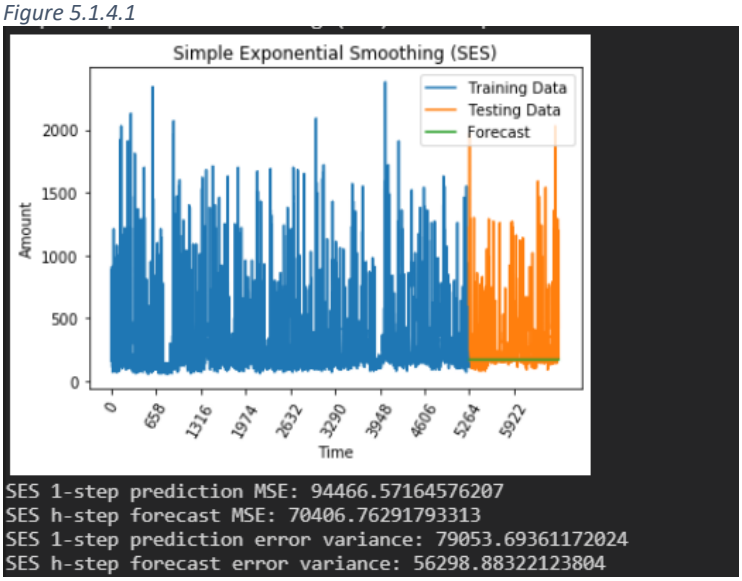


5.1.4 SES method

(alpha=0)

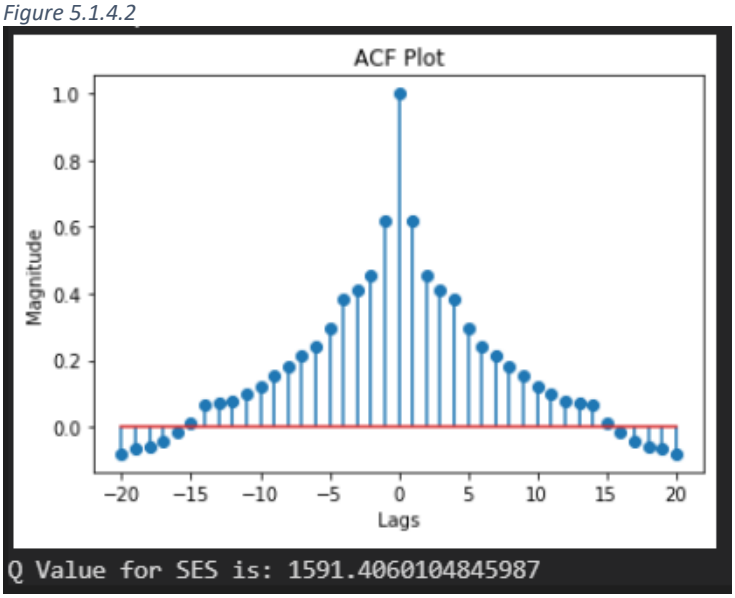
The SES method for alpha=0 is displayed in figure 5.1.4.1 along with the 1-step prediction MSE, h-step forecast MSE, 1-step prediction error variance, and h-step forecast error variance. Again, all metrics are poor for model evaluation.

Figure 5.1.4.1



The ACF plot of the residuals is displayed in figure 5.1.4.2. It also does not display characteristics of white noise, meaning that this is not a good model. The Q value of 1591.4 is also displayed in the figure. Since the Q value is less than the critical value, we can again fail to reject the null and say that the model can be accepted, although the other metrics do not agree.

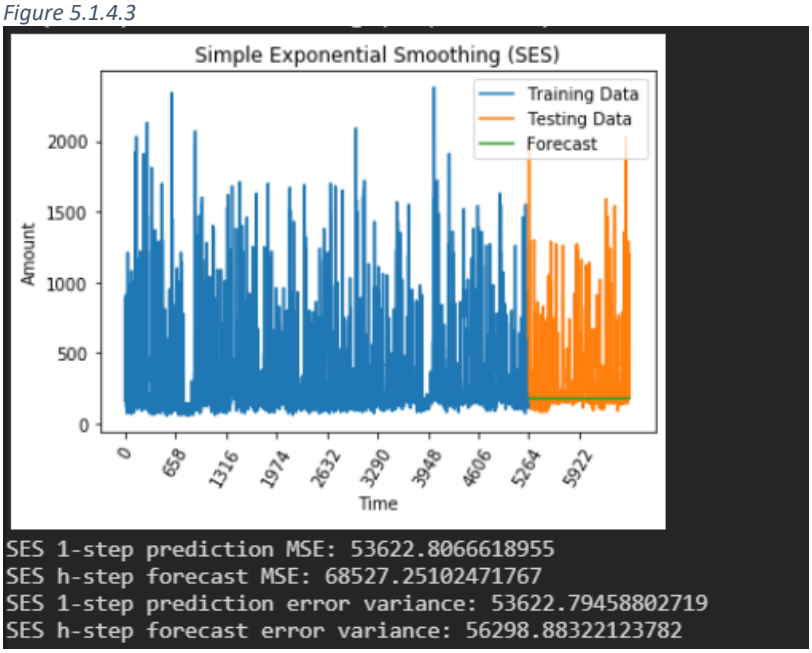
Figure 5.1.4.2



(alpha=0.5)

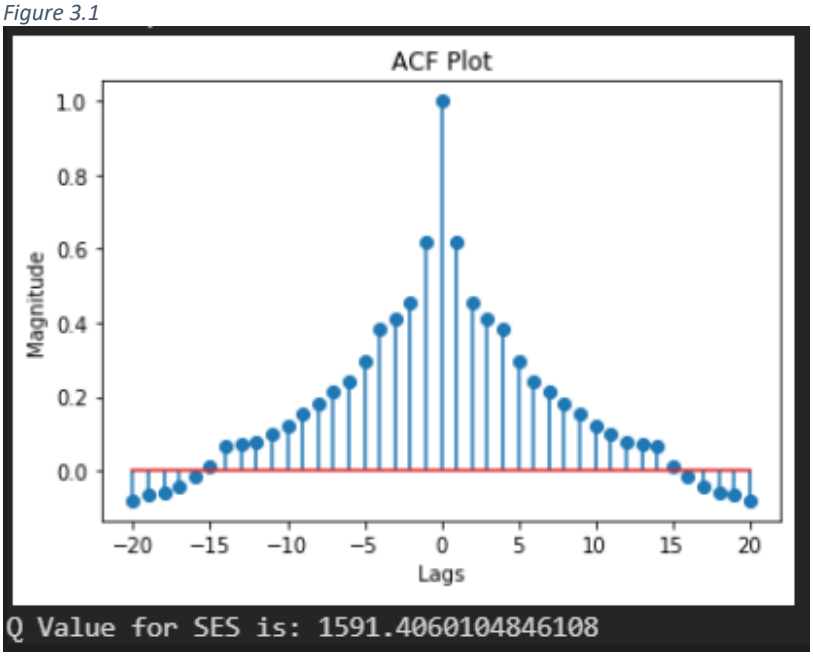
The SES method for alpha=0.5 is displayed in figure 5.1.4.3 along with the 1-step prediction MSE, h-step forecast MSE, 1-step prediction error variance, and h-step forecast error variance. Again, all metrics are subpar for model evaluation.

Figure 5.1.4.3



The ACF plot of the residuals is displayed in figure 5.1.4.4. It also does not display characteristics of white noise, meaning that this is not a good model. The Q value of 1591.4 is also displayed in the figure. It is extremely close to the Q value at alpha=0. Since the Q value is less than the critical value, we can again fail to reject the null and say that the model can be accepted, although the other metrics do not agree.

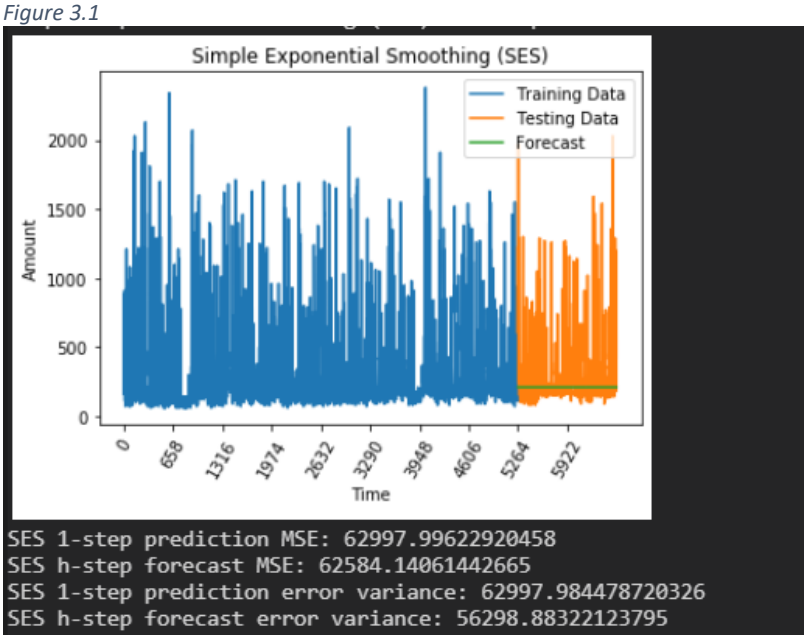
Figure 3.1



(alpha=0.99)

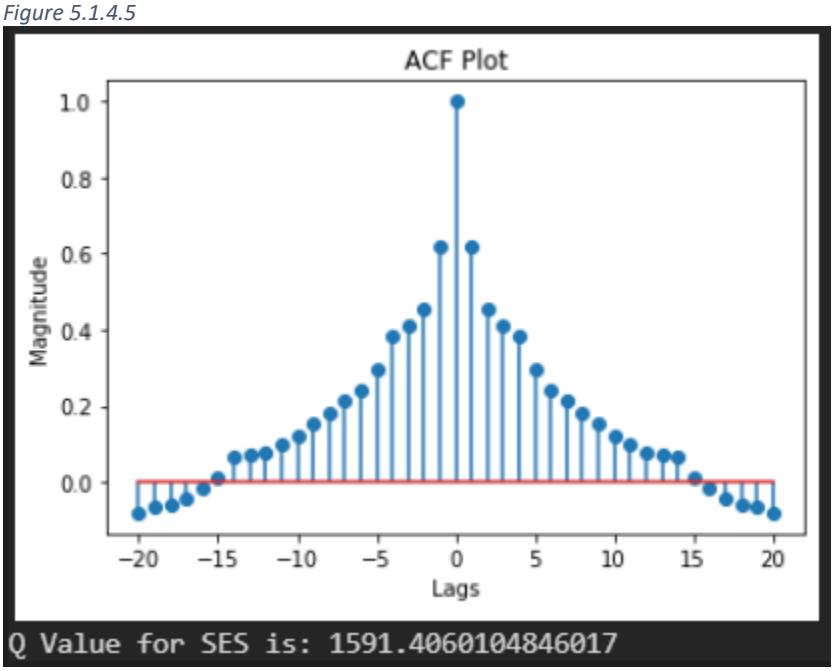
The SES method for alpha=0.99 is displayed in figure 5.1.4.5 along with the 1-step prediction MSE, h-step forecast MSE, 1-step prediction error variance, and h-step forecast error variance. Again, all metrics are subpar for model evaluation.

Figure 3.1



The ACF plot of the residuals is displayed in figure 5.1.4.5. It also does not display characteristics of white noise, meaning that this is not a good model. The Q value of 1591.4 is also displayed in the figure. It is extremely close to the Q value at alpha=0 and 0.5. Since the Q value is less than the critical value, we can again fail to reject the null and say that the model can be accepted, although the other metrics do not agree.

Figure 5.1.4.5



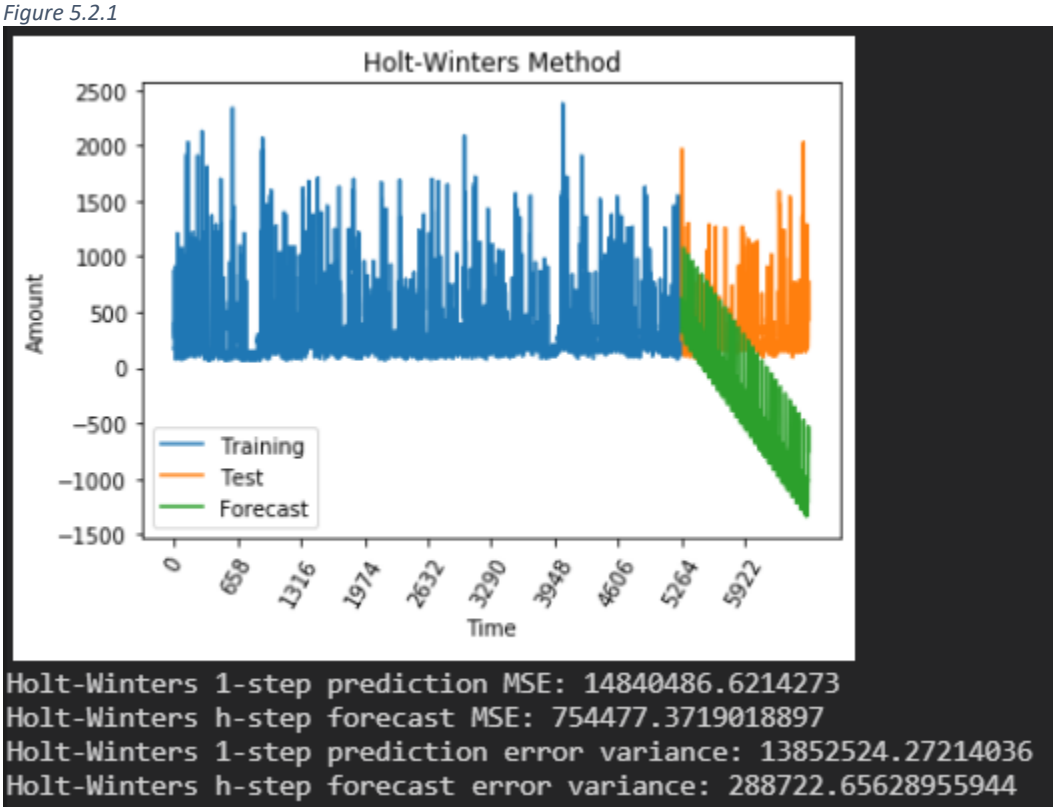
5.2 Holt-Winters

We will now move on to the Holt-Winters model at various levels of seasonal length and damped trend.

Seasonal = 48, damped_trend = False

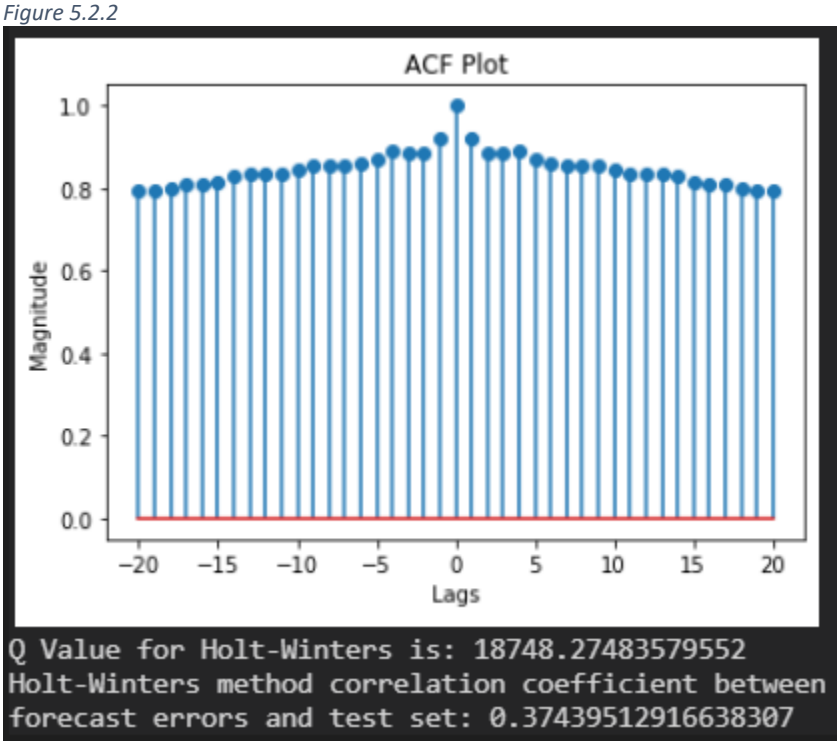
Figure 5.2.1 displays the original data and the generated forecast along with the 1-step prediction MSE, h-step forecast MSE, 1-step prediction error variance, and h-step forecast error variance. Although the seasonal trends can now be modeled, the prediction is very poor. Seasonal 48 is selected here since the time series is at 30-minute intervals in a 24 hour day (meaning there are 48 datapoints per “season”). Perhaps dampening the trend will help model performance.

Figure 5.2.1



The ACF plot of the residuals is displayed in figure 5.2.2. It does not at all display characteristics of white noise, meaning that this is not a good model at all. The Q value of 18,748.3 is also displayed in the figure. The Q value is higher than the critical value, so we reject the null and say this model cannot be used. The correlation coefficient between forecast errors and test set is relatively low.

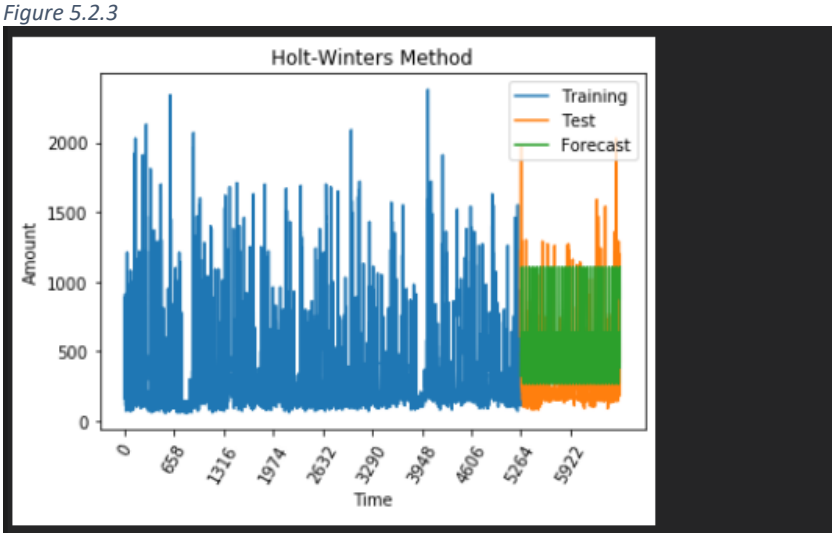
Figure 5.2.2



Seasonal = 48, damped_trend = True

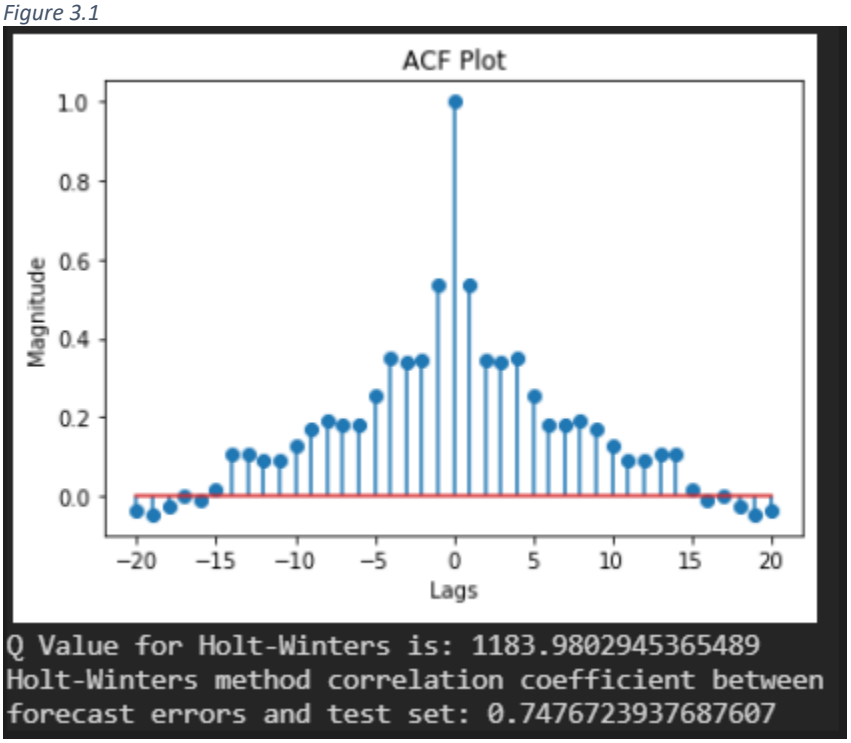
Figure 5.2.3 displays the original data and the generated forecast along with the 1-step prediction MSE, h-step forecast MSE, 1-step prediction error variance, and h-step forecast error variance. Seasonal 48 is selected again here with a damped trend. Although the trend follows the data this time, the seasonal fluctuations of the predictions do not at all capture the data. All metrics are very bad for model performance.

Figure 5.2.3



The ACF plot of the residuals is displayed in figure 5.2.4. It displays the characteristics of white noise more than the previous model. The Q value of 1184 is also displayed in the figure. The Q value is less than the critical value, so we fail to reject the null and say this model can be used, although the other performance metrics do not agree. The correlation coefficient between forecast errors and test set is relatively high.

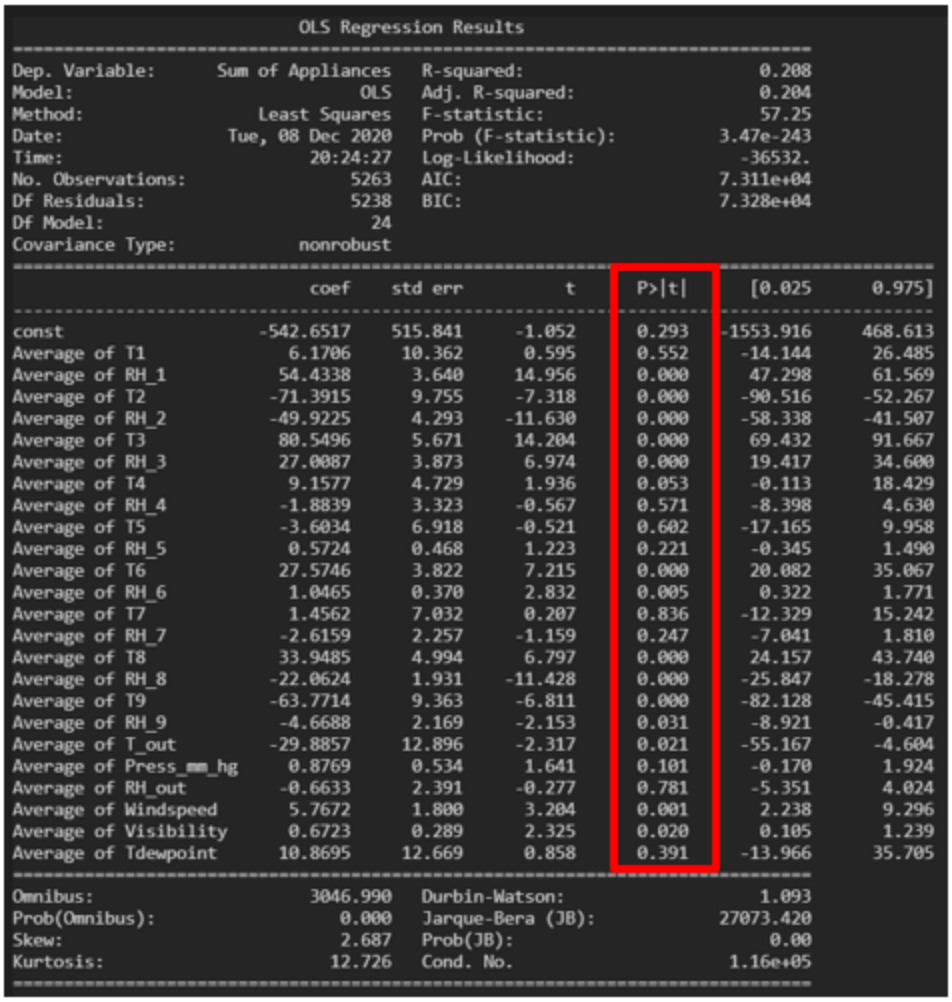
Figure 3.1



5.3 Multiple Linear Regression

Moving on to Multiple Linear Regression, I first created a model with all of the variables to evaluate performance of each in relation to the target. I created a stepwise regression function that creates all combinations of variables possible and logs the performance metrics: adjusted r squared, r squared, AIC, and BIC. The 20 variables in the dataset however were too many for the function to run, so I removed all variables with a p value less than .2, displayed in figure 5.3.1.

Figure 5.3.1



OLS Regression Results

Dep. Variable:	Sum of Appliances	R-squared:	0.208
Model:	OLS	Adj. R-squared:	0.204
Method:	Least Squares	F-statistic:	57.25
Date:	Tue, 08 Dec 2020	Prob (F-statistic):	3.47e-243
Time:	20:24:27	Log-Likelihood:	-36532.
No. Observations:	5263	AIC:	7.311e+04
Df Residuals:	5238	BIC:	7.328e+04
Df Model:	24		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-542.6517	515.841	-1.052	0.293	-1553.916	468.613
Average of T1	6.1706	10.362	0.595	0.552	-14.144	26.485
Average of RH_1	54.4338	3.640	14.956	0.000	47.298	61.569
Average of T2	-71.3915	9.755	-7.318	0.000	-90.516	-52.267
Average of RH_2	-49.9225	4.293	-11.630	0.000	-58.338	-41.507
Average of T3	80.5496	5.671	14.204	0.000	69.432	91.667
Average of RH_3	27.0087	3.873	6.974	0.000	19.417	34.600
Average of T4	9.1577	4.729	1.936	0.053	-0.113	18.429
Average of RH_4	-1.8839	3.323	-0.567	0.571	-8.398	4.630
Average of T5	-3.6034	6.918	-0.521	0.602	-17.165	9.958
Average of RH_5	0.5724	0.468	1.223	0.221	-0.345	1.490
Average of T6	27.5746	3.822	7.215	0.000	20.082	35.067
Average of RH_6	1.0465	0.370	2.832	0.005	0.322	1.771
Average of T7	1.4562	7.032	0.207	0.836	-12.329	15.242
Average of RH_7	-2.6159	2.257	-1.159	0.247	-7.041	1.810
Average of T8	33.9485	4.994	6.797	0.000	24.157	43.740
Average of RH_8	-22.0624	1.931	-11.428	0.000	-25.847	-18.278
Average of T9	-63.7714	9.363	-6.811	0.000	-82.128	-45.415
Average of RH_9	-4.6688	2.169	-2.153	0.031	-8.921	-0.417
Average of T_out	-29.8857	12.896	-2.317	0.021	-55.167	-4.604
Average of Press_mm_hg	0.8769	0.534	1.641	0.101	-0.170	1.924
Average of RH_out	-0.6633	2.391	-0.277	0.781	-5.351	4.024
Average of Windspeed	5.7672	1.800	3.204	0.001	2.238	9.296
Average of Visibility	0.6723	0.289	2.325	0.020	0.105	1.239
Average of Tdewpoint	10.8695	12.669	0.858	0.391	-13.966	35.705

Omnibus:	3046.990	Durbin-Watson:	1.093
Prob(Omnibus):	0.000	Jarque-Bera (JB):	27073.420
Skew:	2.687	Prob(JB):	0.00
Kurtosis:	12.726	Cond. No.	1.16e+05

There were 16 variables remaining, which allowed me to run my function. Figure 5.3.2 displays the output of the function, which ranks performance by adjusted r squared.

Figure 5.3.2

	Variables	AIC	BIC	Adj. R-squared	R-squared
65520	[[Average of RH_1, Average of T2, Average of R...	73106.836113	73205.362960	0.619794	0.620878
65534	[[Average of RH_1, Average of T2, Average of R...	73108.443250	73213.538554	0.619750	0.620906
65406	[[Average of RH_1, Average of T2, Average of R...	73111.095824	73203.054215	0.619414	0.620427
65445	[[Average of RH_1, Average of T2, Average of R...	73111.303587	73203.261977	0.619399	0.620412
65528	[[Average of RH_1, Average of T2, Average of R...	73112.347851	73210.874699	0.619396	0.620481
...	...	...	...	...	...
14	[[Average of Windspeed]]	75272.058834	75278.627291	0.424768	0.424877
12	[[Average of T_out]]	75418.568722	75425.137179	0.408530	0.408642
96	[[Average of T6, Average of T_out]]	75420.082641	75433.219554	0.408472	0.408697
7	[[Average of RH_6]]	75445.825555	75452.394011	0.405459	0.405572
6	[[Average of T6]]	75482.731192	75489.299649	0.401275	0.401389

65535 rows × 5 columns

The best model (tested combination 65520 out of 65535) contains the variables listed in figure 5.3.3.

Figure 5.3.3

```
The best model includes these variables: ['Average of RH_1', 'Average of T2', 'Average of RH_2', 'Average of T3',
'Average of RH_3', 'Average of T4', 'Average of T6', 'Average of RH_6', 'Average of T8', 'Average of RH_8', 'Aver
age of T9', 'Average of RH_9', 'Average of T_out', 'Average of Windspeed', 'Average of Visibility']
```

Figure 5.3.4 displays the summary of the best model. All of the coefficients have a significant p value. The F statistic of the model is 573 (with a p value very close to 0) which is relatively high. This is a good sign for model performance.

Figure 5.3.4

OLS Regression Results

Dep. Variable:	Sum of Appliances	R-squared (uncentered):	0.621
Model:	OLS	Adj. R-squared (uncentered):	0.620
Method:	Least Squares	F-statistic:	573.0
Date:	Wed, 09 Dec 2020	Prob (F-statistic):	0.00
Time:	12:49:16	Log-Likelihood:	-36538.
No. Observations:	5263	AIC:	7.311e+04
Df Residuals:	5248	BIC:	7.321e+04
Df Model:	15		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Average of RH_1	52.4506	2.996	17.506	0.000	46.577	58.324
Average of T2	-68.4863	5.710	-11.995	0.000	-79.679	-57.293
Average of RH_2	-47.3194	3.132	-15.109	0.000	-53.459	-41.180
Average of T3	81.9814	4.875	16.818	0.000	72.425	91.538
Average of RH_3	23.7243	3.302	7.184	0.000	17.250	30.198
Average of T4	10.7720	4.240	2.540	0.011	2.459	19.085
Average of T6	27.4949	3.450	7.969	0.000	20.731	34.259
Average of RH_6	1.2964	0.313	4.143	0.000	0.683	1.910
Average of T8	35.3418	4.010	8.813	0.000	27.480	43.203
Average of RH_8	-22.5086	1.465	-15.363	0.000	-25.381	-19.636
Average of T9	-65.2364	6.969	-9.361	0.000	-78.898	-51.575
Average of RH_9	-4.9097	1.965	-2.499	0.012	-8.761	-1.058
Average of T_out	-21.8921	3.753	-5.833	0.000	-29.249	-14.535
Average of Windspeed	4.6664	1.636	2.852	0.004	1.459	7.874
Average of Visibility	0.7276	0.283	2.568	0.010	0.172	1.283

Omnibus:	3044.480	Durbin-Watson:	1.087
Prob(Omnibus):	0.000	Jarque-Bera (JB):	26993.766
Skew:	2.684	Prob(JB):	0.00
Kurtosis:	12.709	Cond. No.	290.

Next, I used the generated model to run on the training data to create a prediction set and the test data to create a forecast set. This is displayed in figure 5.3.5. The original data appears to have a higher variance that the model does not capture, but the seasonal pattern appears to be followed.

Figure 5.3.5

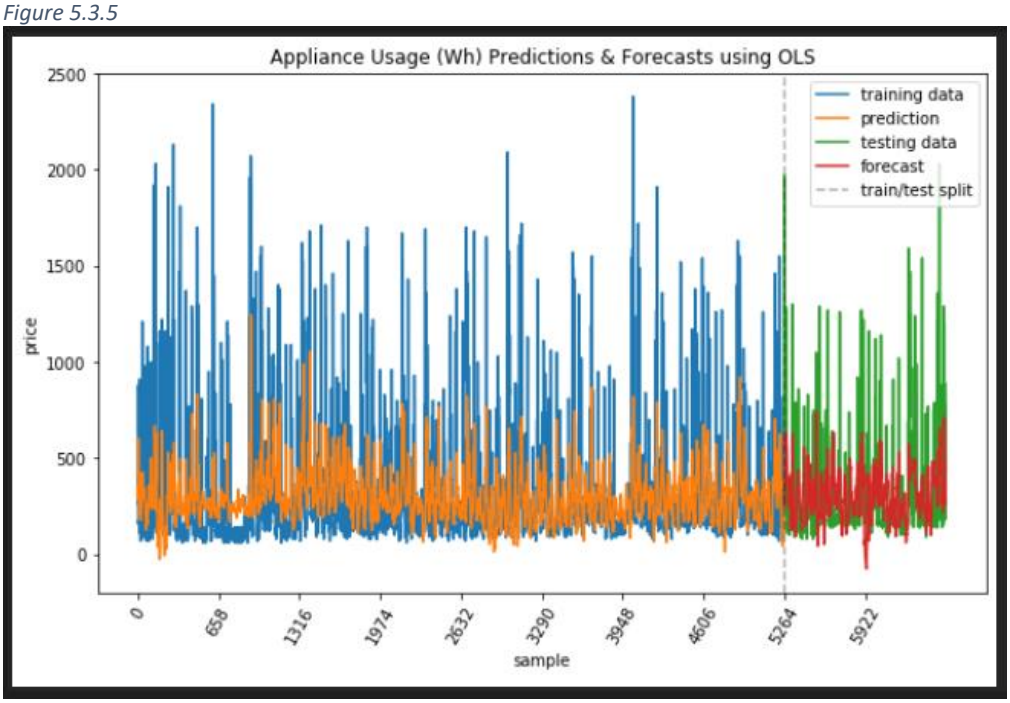
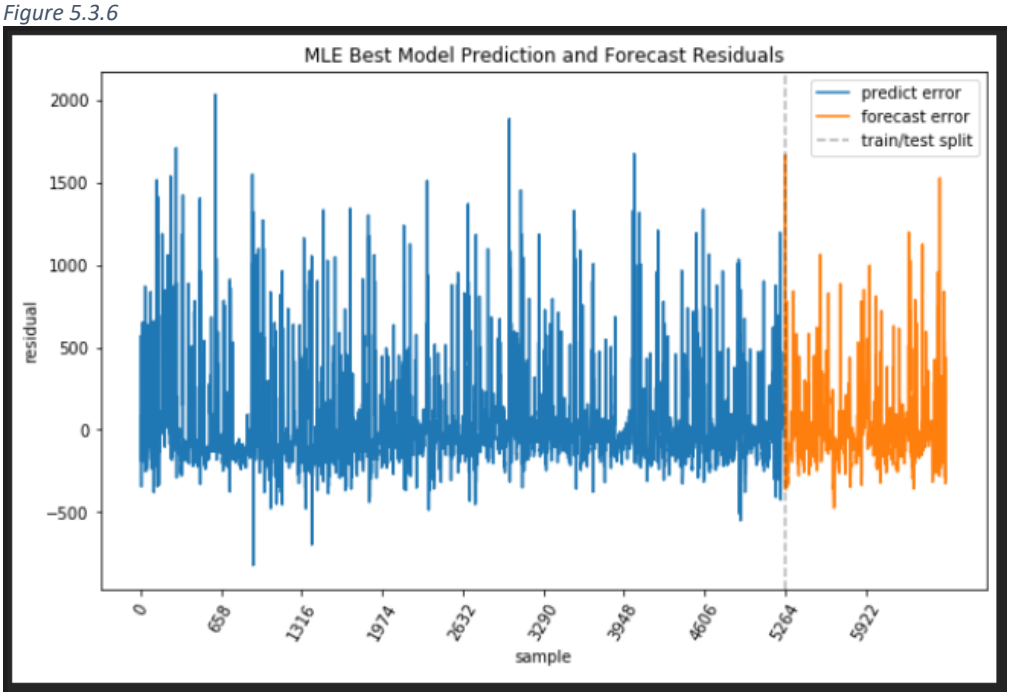


Figure 5.3.6 displays the residuals of the predictions and the forecasts. The prediction error seems to closely match the forecast error.

Figure 5.3.6



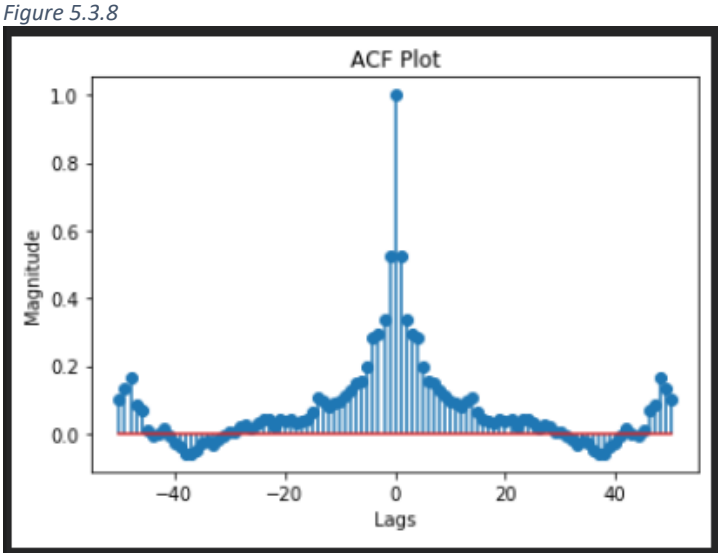
The Q value and MLE are displayed in figure 5.3.7. The Q value is 957.4, which is the lowest we've seen so far. It is lower than the critical value so we can fail to reject the null and say this model can be used. The MLE is also the lowest recorded so far.

Figure 5.3.7

```
MLE Q value: 957.3814657604295
MLE h-step MSE:  48225.69938450677
```

The ACF plot of residuals is displayed in figure 5.3.8. It displays stronger characteristics of white noise, meaning that this is a stronger model.

Figure 5.3.8



The 1-step prediction and h-step forecast error variance and standard deviations are displayed in figure 5.3.9.

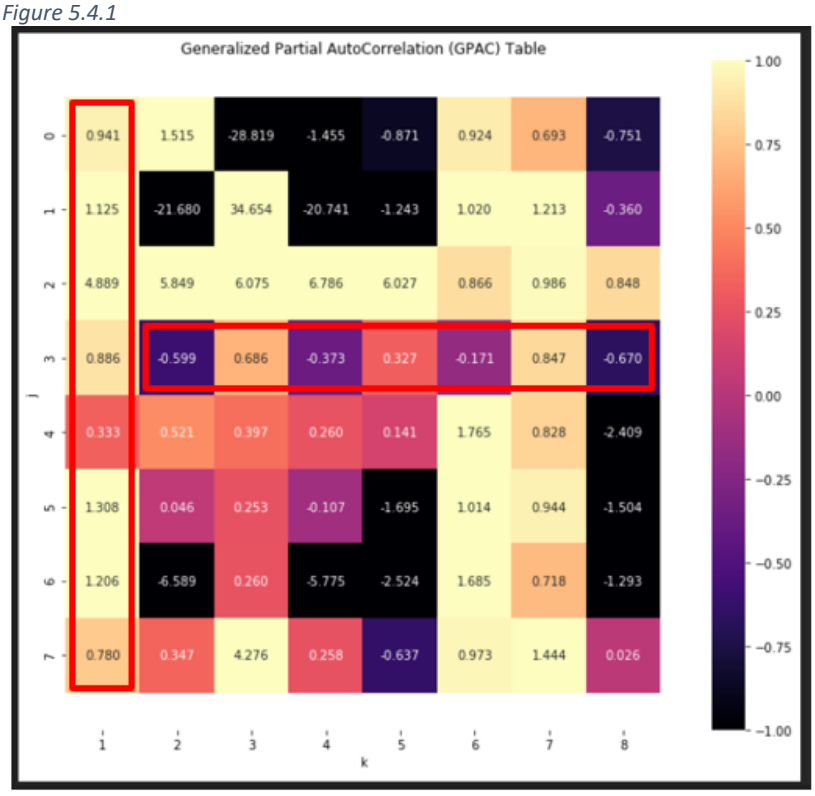
Figure 5.3.9

```
Best MLE 1-step prediction error variance: 62764.08780218717
Best MLE 1-step prediction error standard deviation: 250.5276188410914
Best MLE h-step forecast error variance: 47877.52842560792
Best MLE h-step forecast error standard deviation: 218.8093426378497
```

5.4 Generalized Partial Autocorrelation

We will now move on to the Generalized Partial Autocorrelation method for determining order of autoregressive and order of moving average in order to implement the ARMA model. The generated GPAC(8,8) table for the Appliances energy data is displayed in figure 5.4.1. The method for choosing an AR order and MA order is to first look at the column k that is consistently close to zero. At k=1, the values are all relatively close to 0. However, they're not consistently ALL close, so I will test k=1 and k=0. Then to select the MA order, we look at rows of j after the selected k. At j=3, the values are relatively close to 0. The two ARMA models I will test are ARMA(1,3) and ARMA(0,3).

Figure 5.4.1



5.5 ARMA

5.5.1 ARMA(1,3)

Figure 5.5.1.1 provides the summary for the output of the first generated ARMA model using the ARMA method from statsmodels.tsa.arima_model. The coefficients are listed along with the 95% two-tailed confidence intervals for each outlined in the red box. The model appears to be unbiased because the confidence intervals of the coefficients do not contain 0 so there is no zero/pole cancelation.

Figure 5.5.1.1

ARMA Model Results

Dep. Variable:	Sum of Appliances	No. Observations:	5263
Model:	ARMA(1, 3)	Log Likelihood	-35849.662
Method:	css-mle	S.D. of innovations	219.784
Date:	Tue, 08 Dec 2020	AIC	71711.324
Time:	20:45:56	BIC	71750.735
Sample:	0	HQIC	71725.101

	coef	std err	z	P>|z|	[0.025	0.975]
const	294.3816	11.221	26.236	0.000	272.390	316.374
ar.L1.Sum of Appliances	0.8486	0.014	62.479	0.000	0.822	0.875
ma.L1.Sum of Appliances	-0.3425	0.018	-18.553	0.000	-0.379	-0.306
ma.L2.Sum of Appliances	-0.1721	0.015	-11.663	0.000	-0.201	-0.143
ma.L3.Sum of Appliances	0.0760	0.016	4.851	0.000	0.045	0.107

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	1.1785	+0.0000j	1.1785	0.0000
MA.1	-2.2714	-0.0000j	2.2714	-0.5000
MA.2	2.2681	-0.8067j	2.4073	-0.0544
MA.3	2.2681	+0.8067j	2.4073	0.0544

The training and testing data along with the prediction and forecast of the ARMA(1,3) model are displayed in figure 5.5.1.2.

Figure 5.5.1.2

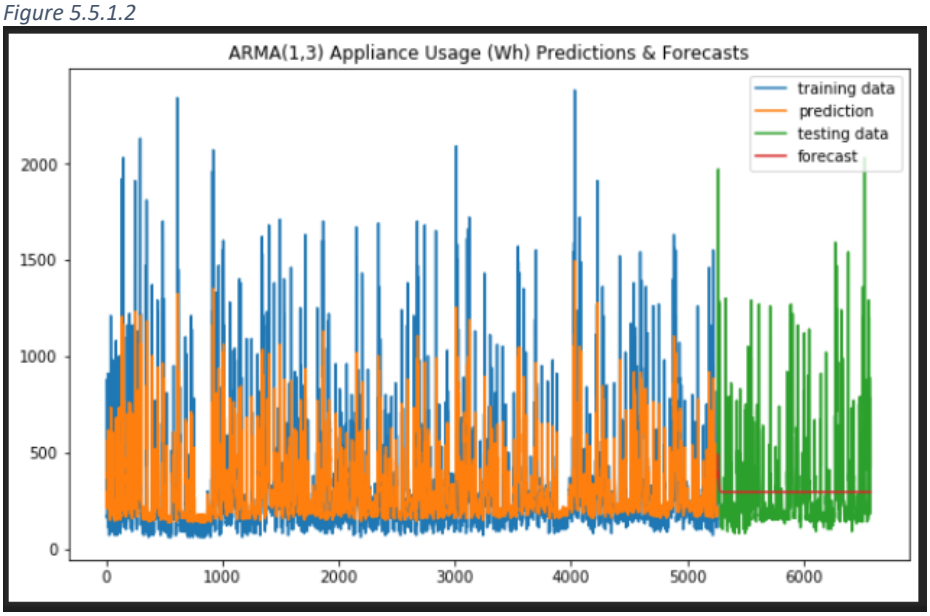


Figure 5.5.1.3 displays just the test and forecast data. The fluctuations do not appear to be reflected in the forecast as they are in the prediction. This will result in a high MSE for this model.

Figure 5.5.1.3

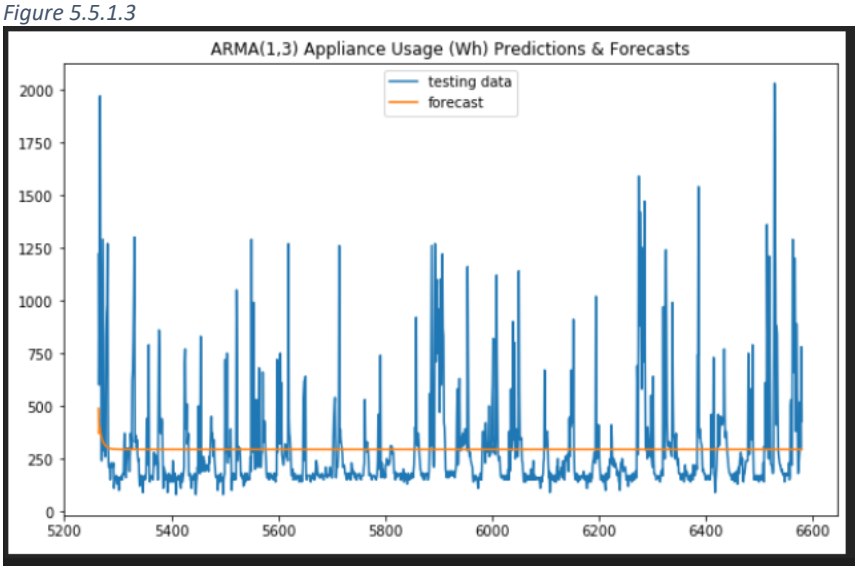


Figure 5.5.1.4 displays the residuals of the predictions.

Figure 5.5.1.4

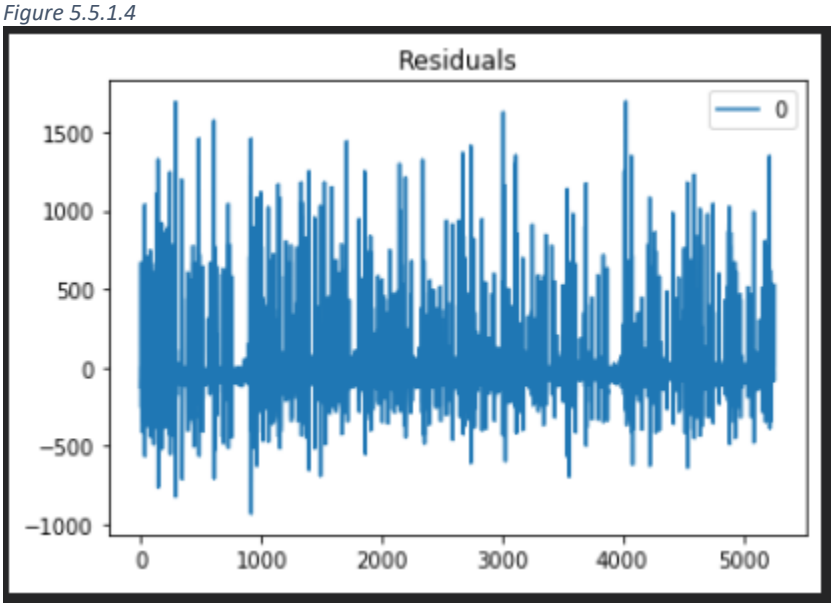


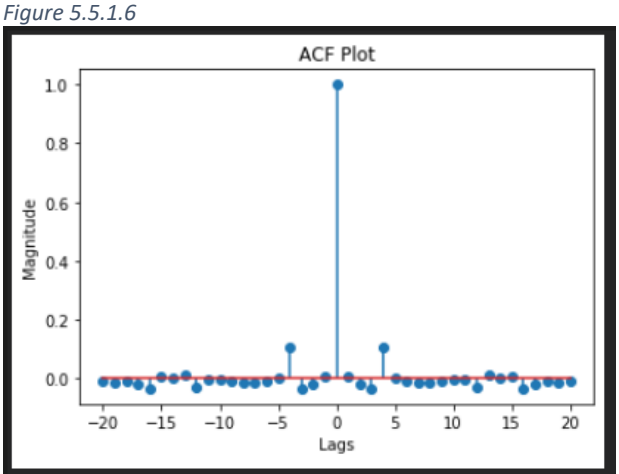
Figure 5.5.1.5 displays the roots/poles of the coefficients. Again, there doesn't appear to be any cancellations between AR and MA.

Figure 5.5.1.5

```
AR roots:  [1.17846872]
MA roots:  [-2.27135577-0.j          2.26812744-0.80669424j  2.26812744+0.80669424j]
```

Figure 5.5.1.6 displays the ACF of the residuals of the prediction error as well as the ACF of the residuals of the forecast error. The residuals of the prediction error display characteristics of white noise. However, the residuals of the forecast error do not. Q values are also displayed, both are below their respective thresholds and can therefore be accepted.

Figure 5.5.1.6



The covariance matrix of the ARMA(1,3) coefficients can be seen in figure 5.5.1.7.

Figure 5.5.1.7

	const	ar.L1.Sum of Appliances	ma.L1.Sum of Appliances	ma.L2.Sum of Appliances	ma.L3.Sum of Appliances
const	125.901951	-0.000019	0.000034	-0.000010	-0.000061
ar.L1.Sum of Appliances	-0.000019	0.000184	-0.000160	-0.000094	-0.000079
ma.L1.Sum of Appliances	0.000034	-0.000160	0.000341	0.000026	-0.000002
ma.L2.Sum of Appliances	-0.000010	-0.000094	0.000026	0.000218	0.000003
ma.L3.Sum of Appliances	-0.000061	-0.000079	-0.000002	0.000003	0.000245

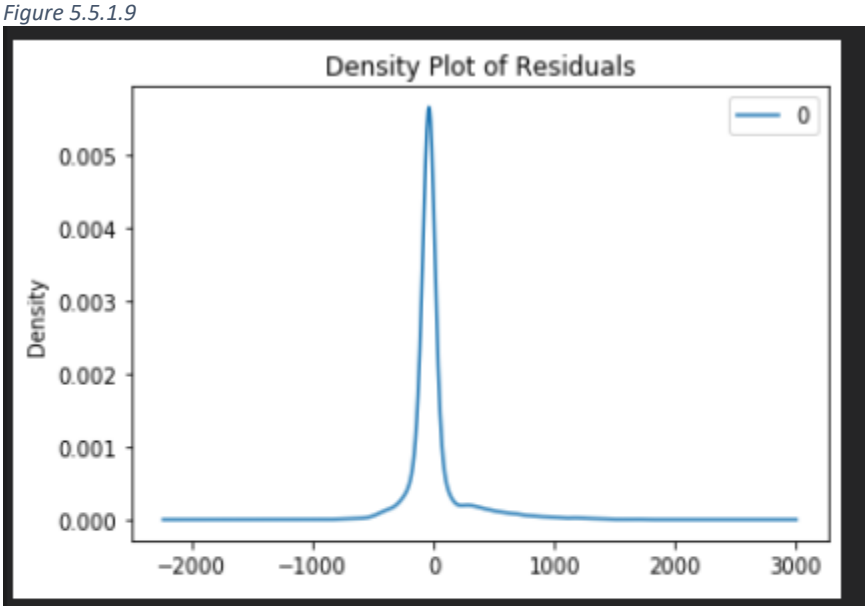
The variance of the prediction error and forecast error along with the h-step MSE are displayed in figure 5.5.1.8. The h-step MSE is the best indicator of model performance since it applies the model to new data. It is not as low as the MLE model so this model will not be labeled best in the final comparison.

Figure 5.5.1.8

```
Variance of prediction error:  48315.728072226164
Variance of forecast error:  55510.23026050083
ARMA(1,3) h-step MSE:  55504.94998339806
```

It is also interesting to observe the density plot of prediction residuals of the model as seen in figure 5.5.1.9. Most exist very close to 0, which means that this ARMA(1,3) model matches very closely to the training data.

Figure 5.5.1.9



	0
count	5263.000000
mean	0.040698
std	219.808393
min	-930.528043
25%	-70.090904
50%	-37.169525
75%	-4.149097
max	1693.840485

5.5.2 ARMA(0,3)

Figure 5.5.2.1 provides the summary for the output of the second generated ARMA model. The coefficients are listed along with the 95% two-tailed confidence intervals for each outlined in the red box. The model appears to be unbiased because the confidence intervals of the coefficients do not contain 0 so there is no zero/pole cancelation.

Figure 5.5.2.1

ARMA Model Results

Dep. Variable:	Sum of Appliances	No. Observations:	5263
Model:	ARMA(0, 3)	Log Likelihood	-36084.151
Method:	css-mle	S.D. of innovations	229.803
Date:	Tue, 08 Dec 2020	AIC	72178.302
Time:	21:36:16	BIC	72211.144
Sample:	0	HQIC	72189.783

	coef	std err	z	P>|z|	[0.025	0.975]
const	294.1861	6.091	48.301	0.000	282.249	306.124
ma.L1.Sum of Appliances	0.5588	0.015	38.409	0.000	0.530	0.587
ma.L2.Sum of Appliances	0.2445	0.013	18.995	0.000	0.219	0.270
ma.L3.Sum of Appliances	0.1197	0.013	9.242	0.000	0.094	0.145

Roots

	Real	Imaginary	Modulus	Frequency
MA.1	-1.8997	-0.0000j	1.8997	-0.5000
MA.2	-0.0713	-2.0957j	2.0969	-0.2554
MA.3	-0.0713	+2.0957j	2.0969	0.2554

The training and testing data along with the prediction and forecast of the ARMA(0,3) model are displayed in figure 5.5.2.2.

Figure 5.5.2.2

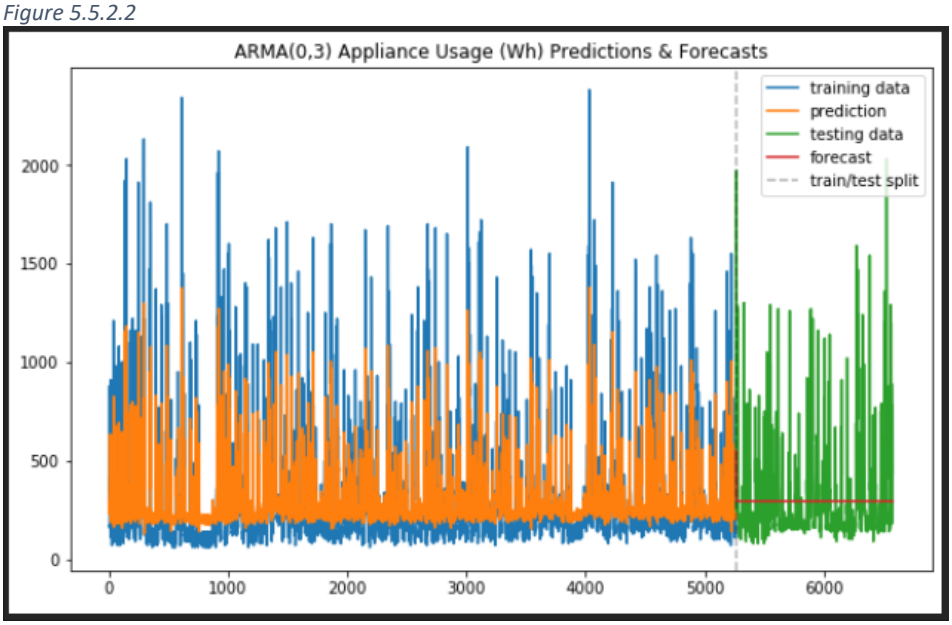


Figure 5.5.2.3 displays just the test and forecast data. Again, the fluctuations do not appear to be reflected in the forecast as they are in the prediction. This will result in a high MSE for this model as well.

Figure 5.5.2.3

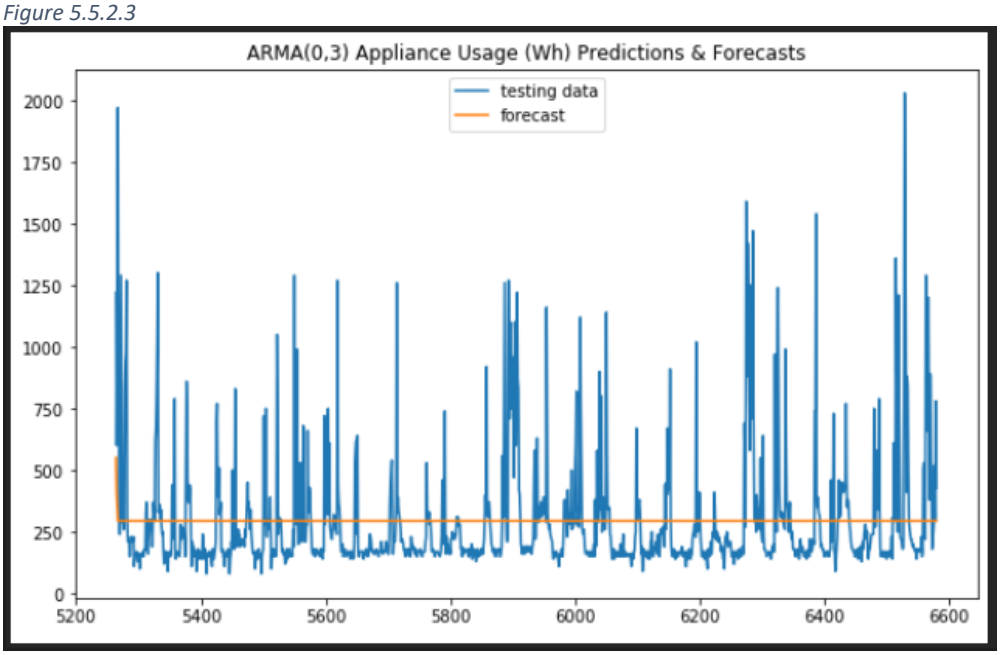


Figure 5.5.2.4 displays the residuals of the predictions.

Figure 5.5.2.4

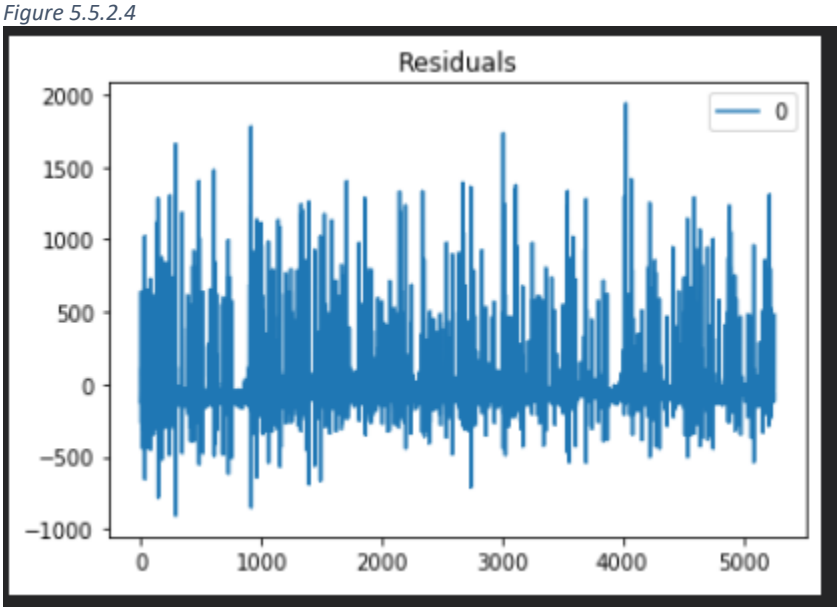


Figure 5.5.2.5 displays the roots/poles of the coefficients. Since the AR order is 0, there aren't any cancellations between AR and MA.

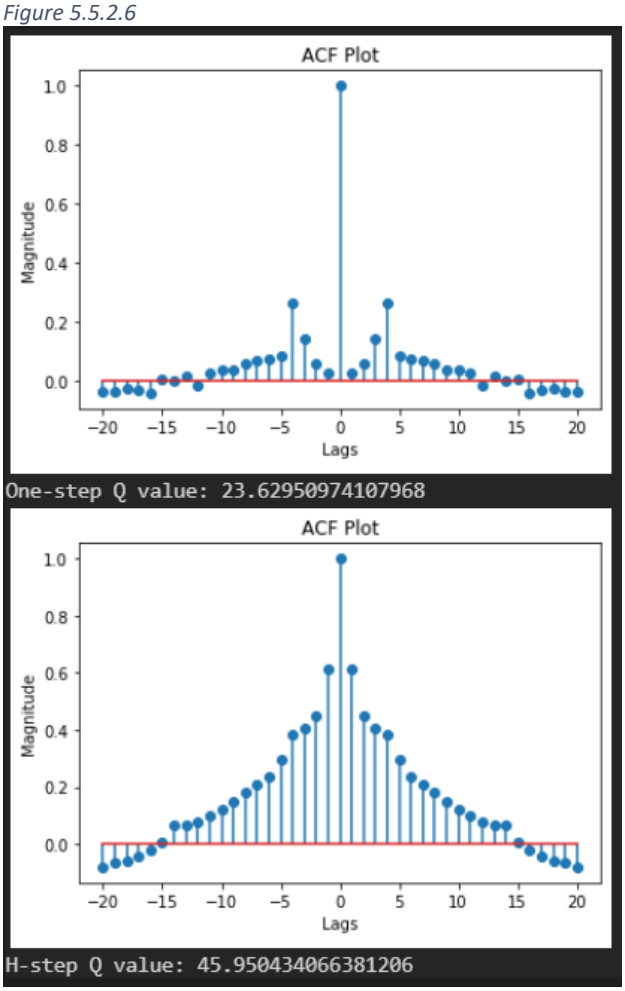
Figure 5.5.2.5

```
AR roots:  []
MA roots:  [-1.89967919-0.j         -0.07126426-2.09570971j -0.07126426+2.09570971j]
```

Figure 5.5.2.6 displays the ACF of the residuals of the prediction error as well as the ACF of the residuals of the forecast error. Again, the residuals of the prediction error display characteristics of white noise.

However, the residuals of the forecast error do not. Q values are also displayed, both are below their respective thresholds and can therefore be accepted.

Figure 5.5.2.6



The covariance matrix of the ARMA(1,3) coefficients can be seen in figure 5.5.2.7.

Figure 5.5.2.7

	const	ma.L1.Sum of Appliances	ma.L2.Sum of Appliances	ma.L3.Sum of Appliances
const	37.096003	0.000035	0.000033	-0.000007
ma.L1.Sum of Appliances	0.000035	0.000212	0.000091	-0.000039
ma.L2.Sum of Appliances	0.000033	0.000091	0.000166	0.000033
ma.L3.Sum of Appliances	-0.000007	-0.000039	0.000033	0.000168

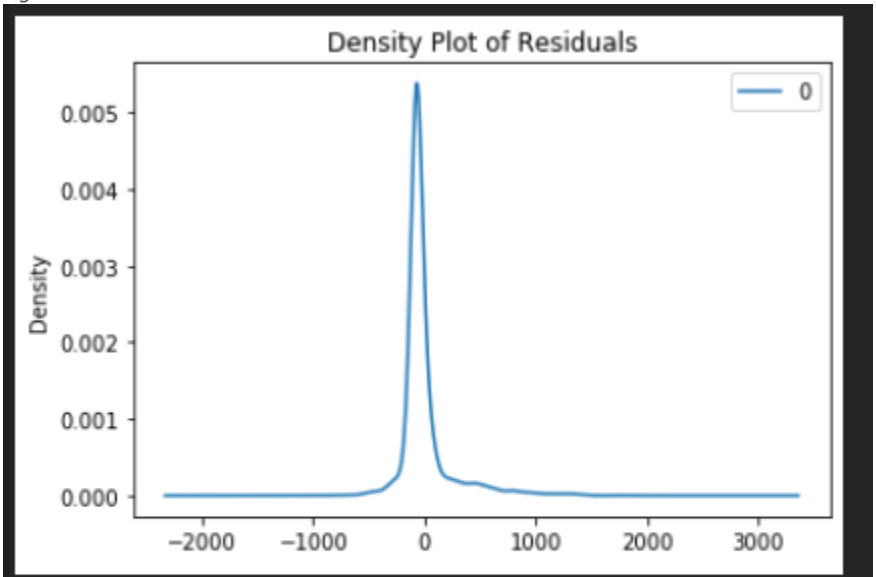
The variance of the prediction error and forecast error along with the h-step MSE are displayed in figure 5.5.2.8. The h-step MSE is the best indicator of model performance since it applies the model to new data. It is not as low as the MLE model so this model will not be labeled best in the final comparison.

Figure 5.5.2.8

```
Variance of prediction error:  52820.62205516074
Variance of forecast error:  55510.23026050083
ARMA(0,3) h-step MSE:  55922.817983088746
```

Again, we observe the density plot of prediction residuals of the model as seen in figure 5.5.2.9. Most exist very close to 0, which means that this ARMA(0,3) model matches very closely to the training data.

Figure 5.5.2.9



	0
count	5263.000000
mean	0.009641
std	229.827374
min	-908.189185
25%	-88.236059
50%	-59.571134
75%	-3.502188
max	1939.797963

6. Final Model Selection

The metric I am using for final model selection is h-step (forecast) MSE because it measures the error of the forecast. This is the best indicator of model performance because it shows how the model performs on new data. Figure 6.0.1 displays the forecast MSE for all models created in ascending order. MLE has the lowest MSE and is therefore the final model selected.

Figure 6.0.1

Model	Forecast MSE
MLE Best	48,225.7
ARMA(1,3)	55,509.3
ARMA(0,3)	55,922.8
Average	56,327.5
SES (alpha=0.99)	62,584.1
SES (alpha=0.5)	68,527.3
Holt-Winters (seasonal=48, damped=True)	69,737.6
SES (alpha=0)	70,406.8
Naïve	259,901.4
Drift	329,680.3
Holt-Winters (seasonal=48, damped=False)	754,477.4
Holt-Winters (seasonal=24, damped=False)	1,674,610.5
Holt-Winters (seasonal=12, damped=False)	7,284,517.9

6.1 Forecast Function

The final forecast function is displayed in figure 6.1.1. Given all 15 of these variables, one should be able to determine a prediction for Appliance usage in Wh.

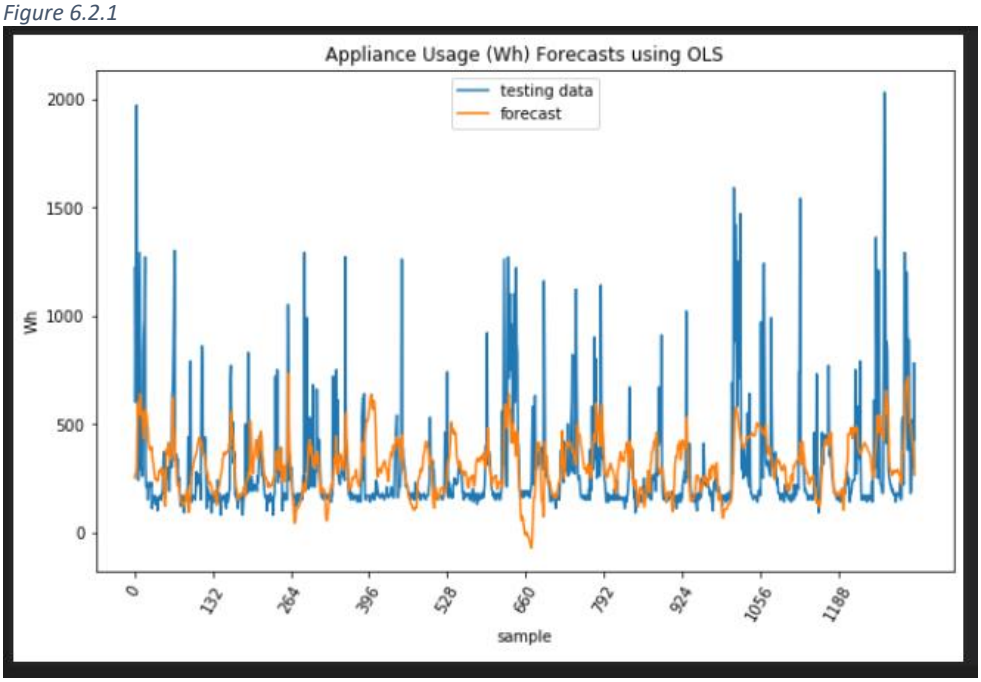
Figure 6.1.1

Y = Average of RH_1*(52.4506) + Average of T2*(-68.4863) + Average of RH_2*(-47.3194) + Average of T3*(81.9814) + Average of RH_3*(23.7243) + Average of T4*(10.772) + Average of T6*(27.4949) + Average of RH_6*(1.2964) + Average of T8*(35.3418) + Average of RH_8*(-22.5086) + Average of T9*(-65.2364) + Average of RH_9*(-4.9097) + Average of T_out*(-21.8921) + Average of Windspeed*(4.6664) + Average of Visibility*(.7276)

6.2 H-step Ahead Prediction

The h-step ahead prediction of the MLE model is displayed in figure 6.2.1. It plots the original test data along with the predictions using the best developed model in figure 6.1.1. The predictions look to follow the test data and follow the patterns detected in the training set.

Figure 6.2.1



6.3 Summary & Conclusion

After testing at least 13 different models and variations of different models, the final one selected was the multiple linear regression model. This was the only model that utilized variables other than the target variable. Given the nature of the data, this makes a certain amount of sense. The amount of energy used for appliances may be largely attributed to fluctuations in temperature and humidity as the inhabitants attempt to maintain a consistent level of comfort throughout the days. Outside temperature didn't seem to have as much of an affect on appliance usage as I had originally thought. This may be due to the structure of the house itself – perhaps the materials used contribute to maintaining a level of homeostasis since it is classified as a "sustainable house".

Overall, the error of the MSE of multiple linear regression is decently low enough to be considered a good model. The adjusted R squared of .619 is not as strong as I would have liked, but perhaps a broader set of variables (like number of family members or appliances) collected besides temperature and humidity would yield a better model.

7. Appendix

Developed Code

```
# Appliance Energy Usage
#%%
# import libraries
import statistics
import random
import math
import datetime as dt
import statsmodels.api as sm
import numpy as np
import pandas as pd
from pandas.plotting import lag_plot
from pandas import concat
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import itertools
import seaborn as sns
from pandas.plotting import register_matplotlib_converters
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import STL
import statsmodels.tsa.holtwinters as ets
from statsmodels.tsa.ar_model import AutoReg
from statsmodels.tsa.arima_process import ArmaProcess
from scipy import signal
from scipy import stats, linalg
from lifelines import KaplanMeierFitter
from functions import *
import warnings
warnings.filterwarnings('ignore')

#%%
# 1- Load the energy
df = pd.read_csv('C:/Users/sjg27/OneDrive/Documents/GWU Data Science/Fall 20/Time Series/project/data/ener
gydata_complete_halfhourly.csv', header=0)
df['datetime'] = pd.to_datetime(df['date'] + ' ' + df['half_hour'])
df.info()
df.head()

# plot dependent variable 'Sum of Appliances'
plt.figure(figsize=(10, 6))
plt.plot(df['datetime'], df['Sum of Appliances'], label = 'Appliances')
plt.legend(loc = 0)
plt.xlabel('Date')
plt.ylabel('Appliances (Wh)')
plt.title('Energy Usage of Appliances in Wh')
plt.xticks(rotation=75)
# plt.xticks(np.arange(0, len(df['datetime']), 10), rotation=75)
plt.show()

# ADF - stationary data check
appliances = df['Sum of Appliances']
appliances_list = appliances.tolist()
ADF_Cal(appliances_list)
# appliances is a stationary variable

# ACF analysis
appliances_acf = fun_acf(appliances_list,50)
plot_acf(appliances_acf,50)

#%%
# correlation matrix
sns.heatmap(df.corr());

#%%
# decomposition

# 3-MA plot (moving average window size 3)
window = 3
ma_3 = cal_MA_wFO(appliances, window)
k_3=int((window-1)/2)
detrended_ma_3 = np.array(ma_3) - np.array(appliances[k_3:len(appliances)-k_3])
datetime = df['datetime']

plt.figure(figsize=(10, 6))
plt.plot(datetime, appliances, label='Original', alpha=.7)
plt.plot(datetime[k_3:len(appliances)-k_3], ma_3, label='MA Trend')
plt.plot(datetime[k_3:len(appliances)-k_3], ma_3, label='Detrended', alpha=.5)
plt.xticks(rotation=60)
plt.xlabel('Time Period')
plt.ylabel('Appliance Usage Wh')
plt.title('Appliance Usage: Original & MA=3')
plt.legend()
```

```
plt.show()

# 2x24-MA plot (moving average window size 24 and folding order 2)
window = 24
foldingorder = 2
ma_2x24 = cal_MA_wFO(appliances, window, foldingorder)
k_2x24=int(((window-2)/2)+(foldingorder/2))
detrended_ma_2x24 = np.array(ma_2x24) - np.array(appliances[k_2x24:len(appliances)-k_2x24])
datetime = df['datetime']

plt.figure(figsize=(10, 6))
plt.plot(datetime, appliances, label='Original', alpha=.7)
plt.plot(datetime[k_2x24:len(appliances)-k_2x24], ma_2x24, label='MA Trend')
plt.plot(datetime[k_2x24:len(appliances)-k_2x24], ma_2x24, label='Detrended', alpha=.5)
plt.xticks(rotation=60)
plt.xlabel('Time Period')
plt.ylabel('Appliance Usage Wh')
plt.title('Appliance Usage: Original & MA=24 FO=2')
plt.legend()
plt.show()
#%%
# STL decomposition
df_STL = pd.Series(appliances, name = 'Appliances Usage (Wh)')

res = STL(df_STL, period=48).fit()
res.plot()
plt.xlabel('Time Interval')
plt.show()

# seasonally adjusted data vs original data
plt.figure(figsize=(10, 6))
plt.plot(datetime, appliances, label='Original')
plt.plot(datetime, res.seasonal, label='Seasonally Adjusted')
plt.xticks(rotation=60)
plt.xlabel('Time Period')
plt.ylabel('Amount')
plt.title('Original & Seasonaly Adjusted Data')
plt.legend()
plt.show()

# trend strength calc (0 to 1 scale - 1 highly trended)
var_r = res.resid.var()
var_tr = (res.trend + res.resid).var()
strength_trend = max(0, 1-(var_r / var_tr))
print('The strength of trend is: ', strength_trend)

# seasonal strength calc (0 to 1 scale - 1 highly seasonal)
var_sr = (res.seasonal + res.resid).var()
strength_seasonality = max(0, 1-(var_r / var_sr))
print('The strength of seasonality is: ', strength_seasonality)

#%%
# Multiple Linear Regression
# feature Selection - define variables from dataframe
df_var_name = np.asarray(df.columns)
df_var_name = df_var_name[2:-1]
df_vars = df[df_var_name]

# train/test split
train, test = train_test_split(df_vars, train_size=0.8, shuffle=False)

df_preds = df_var_name[1:]
y_train = train[['Sum of Appliances']]
y_test = test[['Sum of Appliances']]
X_train = train[df_preds]
X_test = test[df_preds]

# too many variables to run the stepwise function - must eliminate some variables
X_train_all = train[df_preds]
```

```
X_train_all = sm.add_constant(X_train)
OLS_all = sm.OLS(y_train, X_train_all).fit()
print(OLS_all.summary())

# after reviewing summary - removing variables with a higher p value than .2
df_preds_new = ['Average of RH_1', 'Average of T2', 'Average of RH_2', 'Average of T3', 'Average of RH_3',
 'Average of T4', 'Average of T6', 'Average of RH_6', 'Average of T8', 'Average of RH_8', 'Average of T9',
 'Average of RH_9', 'Average of T_out', 'Average of Press_mm_hg', 'Average of Windspeed', 'Average of Visi
bility']

X_train = train[df_preds_new]
X_test = test[df_preds_new]

#%%
# DON'T RERUN IN TESTING (COMMENT OUT)
# stepwise regression - forward step - takes a long time to run with many variables! - about 20 minutes wi
th 16 variables
stepwise_reg = stepwise_fun_base(X_train, y_train)
stepwise_reg_top = stepwise_reg.sort_values('Adj. R-squared',ascending = False)
stepwise_reg_top.head()
stepwise_reg_top_vars = stepwise_reg_top['Variables'].iloc[0][0].tolist()
print('The best model includes these variables:', stepwise_reg_top_vars)
#%%
# t-test and F-test on the best final model
# t-
test: stat for each coefficient - look at p value - want small for each (below .05 confidence threshold)
# F-test for entire model - look at p value (Prob (F-
statistic)) - want small (below .05 confidence threshold)

# stepwise_reg_top_vars = ['Average of RH_1', 'Average of T2', 'Average of RH_2', 'Average of T3', 'Averag
e of RH_3', 'Average of T4', 'Average of T6', 'Average of RH_6', 'Average of T8', 'Average of RH_8', 'Aver
age of T9', 'Average of RH_9', 'Average of T_out', 'Average of Windspeed', 'Average of Visibility'] # FOR
TESTING
X_train_best = train[stepwise_reg_top_vars]
X_test_best = test[stepwise_reg_top_vars]
OLS_model_best = sm.OLS(y_train, X_train_best).fit()
OLS_model_predict_best = OLS_model_best.predict(X_train_best)
OLS_model_details_best = OLS_model_best.summary()
print(OLS_model_details_best)
print('\n')
print('The F-statistic for the best model is:', OLS_model_best.fvalue.round(4))
print('The p-value of the F-statistic for the best model is:', OLS_model_best.f_pvalue)

# %%
# plot of one-step predictions and h-step forecasts
OLS_model_forecast_best = OLS_model_best.predict(X_test_best)

length = []
for i in range(len(df_vars)):
    length.append(i+1)

limit = len(X_train_best)
length_train = length[:limit]
length_test = length[limit:]

# plot of the train, test and predicted values in one graph
plt.figure(figsize=(10, 6))
plt.plot(length_train, y_train, label='training data')
plt.plot(length_train, OLS_model_predict_best, label='prediction')
plt.plot(length_test, y_test, label='testing data')
plt.plot(length_test, OLS_model_forecast_best, label='forecast')
plt.axvline(limit, linestyle='dashed', color='black', alpha=.3, label='train/test split')
plt.xlabel('sample')
plt.xticks(np.arange(0,len(length),round(len(length)/10)), rotation=60)
plt.ylabel('Wh')
plt.title('Appliance Usage (Wh) Predictions & Forecasts using OLS')
plt.legend()
plt.show()
```

```
#%%
# MLE residuals
app_train, app_test = train_test_split(appliances, train_size=0.8, shuffle=False)
mle_predict_error = []
for i in range(0, len(app_train)):
    mle_predict_error.append(app_train[i] - OLS_model_predict_best[i])

index_train = len(app_train)
mle_forecast_error = []
for i in range(0, len(app_test)):
    mle_forecast_error.append(app_test[i+index_train] - OLS_model_forecast_best[i+index_train])

# plot of residuals
plt.figure(figsize=(10, 6))
plt.plot(length_train, mle_predict_error, label='predict error')
plt.plot(length_test, mle_forecast_error, label='forecast error')
plt.axvline(limit, linestyle='dashed', color='black', alpha=.3, label='train/test split')
plt.xlabel('sample')
plt.xticks(np.arange(0,len(length),round(len(length)/10)), rotation=60)
plt.ylabel('residual')
plt.legend()
plt.title('MLE Best Model Prediction and Forecast Residuals')
plt.show()

# ACF of predict error
#mle_pred_acf = fun_acf(mle_predict_error,50)
#plot_acf(mle_pred_acf,50)

# ACF of forecast errir
mle_fcst_acf = fun_acf(mle_forecast_error,20)
plot_acf(mle_fcst_acf,20)

# var, std dev of predict error & forecast error
mle_predict_error_mean = sum(mle_predict_error) / len(mle_predict_error)
mle_predict_var1 = []
for i in range(len(mle_predict_error)):
    mle_predict_var1.append((mle_predict_error[i] - mle_predict_error_mean) ** 2)
mle_predict_var = sum(mle_predict_var1) / len(mle_predict_var1)
print('Best MLE 1-step prediction error variance:',mle_predict_var)
mle_predict_std = math.sqrt(mle_predict_var)
print('Best MLE 1-step prediction error standard deviation:', mle_predict_std)

mle_forecast_error_mean = sum(mle_forecast_error) / len(mle_forecast_error)
mle_forecast_var1 = []
for i in range(len(mle_forecast_error)):
    mle_forecast_var1.append((mle_forecast_error[i] - mle_forecast_error_mean) ** 2)
mle_forecast_var = sum(mle_forecast_var1) / len(mle_forecast_var1)
print('Best MLE h-step forecast error variance:',mle_forecast_var)
mle_forecast_std = math.sqrt(mle_forecast_var)
print('Best MLE h-step forecast error standard deviation:', mle_forecast_std)
#%%
q_mle = fun_acf(mle_forecast_error, 20)
q_mle = q_mle[1:]

q_mle_total = 0
for i in q_mle:
    q_mle_total += (i ** 2)

q_mle_total = q_mle_total * len(mle_forecast_error)
print('MLE Q value:', q_mle_total)
print('MLE h-step MSE: ', mean_squared_error(y_test,OLS_model_forecast_best))

# %%
# GPAC
target = appliances
acf_list = list_acf(target)
Cal_GPAC(acf_list, 8, 8)

# %%
```

```
# ARMA - statsmodels
from statsmodels.tsa.arima_model import ARIMA as ARMA
from statsmodels.graphics.api import qqplot

# ARMA (1,3)
app_train, app_test = train_test_split(appliances, train_size=0.8, shuffle=False)
arma_mod = ARMA(app_train, order=(1, 0, 3)).fit()
print(arma_mod.params)

resid = arma_mod.resid
stats.normaltest(resid)

fig = plt.figure(figsize=(10,8))
ax = fig.add_subplot(111)
fig = qqplot(resid, line='q', ax=ax, fit=True)

print(arma_mod.summary())

# plot residual errors
residuals = pd.DataFrame(arma_mod.resid)
residuals.plot(title='Residuals', legend=False)
plt.legend()
plt.show()
residuals.plot(kind='kde', title='Density Plot of Residuals')
plt.show()
print(residuals.describe())

print('AR roots: ', arma_mod.arroots)
print('MA roots: ', arma_mod.maroots)

# generate predicted and forecasted values for arma_mod
date_periods = []
for i in range(0,len(datetime)):
    date_periods.append(i+1)

limit = len(app_train)
length_train = date_periods[:limit]
length_test = date_periods[limit:]

arma_mod_predict = arma_mod.fittedvalues
arma_mod_forecast = arma_mod.predict(limit, 6579, dynamic=True)

# plot of the train, test and predicted values in one graph
plt.figure(figsize=(10, 6))
plt.plot(length_train, app_train, label='training data')
plt.plot(length_train, arma_mod_predict, label='prediction')
plt.plot(length_test, app_test, label='testing data')
plt.plot(length_test, arma_mod_forecast[:-1], label='forecast') ##
plt.axvline(limit, linestyle='dashed', color='black', alpha=.3, label='train/test split')
plt.xlabel('sample')
# plt.xticks(np.arange(0,len(length),round(len(length)/10)), rotation=60)
plt.ylabel('Appliance Usage (Wh)')
plt.title('ARMA(1,3) Appliance Usage (Wh) Predictions & Forecasts')
plt.legend()
plt.show()

#%%
# residual acf

# one-step
one_step_acf = fun_acf(residuals[0].tolist())
plot_acf(one_step_acf)

q_one_step_total = 0
for i in one_step_acf:
    q_one_step_total += (i ** 2)

q_one_step_total = q_one_step_total * len(one_step_acf)
print('One-step Q value:', q_one_step_total)
```

```
# h-step
h_step_error = []
for i in range(0,len(app_test)):
    ind = i + 5263
    diff = app_test[ind] - arma_mod_forecast[ind]
    h_step_error.append(diff)

h_step_acf = fun_acf(h_step_error)
plot_acf(h_step_acf)

q_h_step_total = 0
for i in h_step_acf:
    q_h_step_total += (i ** 2)

q_h_step_total = q_h_step_total * len(h_step_acf)
print('H-step Q value:', q_h_step_total)

#%%
print('Variance of prediction error: ', statistics.variance(residuals[0].tolist()))
print('Variance of forecast error: ', statistics.variance(h_step_error))
print('ARMA(1,3) h-step MSE: ', mean_squared_error(y_test,arma_mod_forecast[:-1]))
#%%
# covariance
covar = pd.DataFrame(arma_mod.cov_params())
covar

#%%
# ARMA (0,3)
arma_mod2 = ARMA(app_train, order=(0, 0, 3)).fit()
print(arma_mod2.params)

resid2 = arma_mod2.resid
stats.normaltest(resid2)

fig = plt.figure(figsize=(10,8))
ax = fig.add_subplot(111)
fig = qqplot(resid2, line='q', ax=ax, fit=True)

print(arma_mod2.summary())
# plot residual errors
residuals = pd.DataFrame(arma_mod2.resid)
residuals.plot(title='Residuals')
plt.show()
residuals.plot(kind='kde', title='Density Plot of Residuals')
plt.show()
print(residuals.describe())

print('AR roots: ', arma_mod2.arroots)
print('MA roots: ', arma_mod2.maroots)

arma_mod2_predict = arma_mod2.fittedvalues
arma_mod2_forecast = arma_mod2.predict(limit, 6579, dynamic=True)

# plot of the train, test and predicted values in one graph
plt.figure(figsize=(10, 6))
plt.plot(length_train, app_train, label='training data')
plt.plot(length_train, arma_mod2_predict, label='prediction')
plt.plot(length_test, app_test, label='testing data')
plt.plot(length_test, arma_mod2_forecast[:-1], label='forecast') ##
plt.axvline(limit, linestyle='dashed', color='black', alpha=.3, label='train/test split')
plt.xlabel('sample')
# plt.xticks(np.arange(0,len(length),round(len(length)/10)), rotation=60)
plt.ylabel('Appliance Usage (Wh)')
plt.title('ARMA(0,3) Appliance Usage (Wh) Predictions & Forecasts')
plt.legend()
plt.show()

# residual acf
```

```
# one-step
one_step_acf = fun_acf(residuals[0].tolist())
plot_acf(one_step_acf)

q_one_step_total = 0
for i in one_step_acf:
    q_one_step_total += (i ** 2)

q_one_step_total = q_one_step_total * len(one_step_acf)
print('One-step Q value:', q_one_step_total)

# h-step
h_step_error = []
for i in range(0,len(app_test)):
    ind = i + 5263
    diff = app_test[ind] - arma_mod_forecast[ind]
    h_step_error.append(diff)

h_step_acf = fun_acf(h_step_error)
plot_acf(h_step_acf)

q_h_step_total = 0
for i in h_step_acf:
    q_h_step_total += (i ** 2)

q_h_step_total = q_h_step_total * len(h_step_acf)
print('H-step Q value:', q_h_step_total)

#%%
print('Variance of prediction error: ', statistics.variance(residuals[0].tolist()))
print('Variance of forecast error: ', statistics.variance(h_step_error))
print('ARMA(0,3) h-step MSE: ', mean_squared_error(y_test,arma_mod2_forecast[:-1]))
#%%
# covariance
covar2 = pd.DataFrame(arma_mod2.cov_params())
covar2

#%%
# Base models
app_train_arr = np.asarray(app_train)
app_test_arr = np.asarray(app_test)
date_periods_arr = np.asarray(date_periods)

#%%
# Average Method
print("Average Method:")
average_method(app_train_arr, app_test_arr, date_periods_arr)

# %%
# Naive Method
print("Naive Method:")
naive_method(app_train_arr, app_test_arr, date_periods_arr)

# %%
# Drift Method
print("Drift Method:")
drift_method(app_train_arr, app_test_arr, date_periods_arr)

# %%
# Simple Exponential Smoothing (SES) Method
print("Simple Exponential Smoothing (SES) with alpha=0:")
ses_method(app_train_arr, app_test_arr, date_periods_arr, 0)
# %%
print("Simple Exponential Smoothing (SES) with alpha=0.5:")
ses_method(app_train_arr, app_test_arr, date_periods_arr, 0.5)
# %%
print("Simple Exponential Smoothing (SES) with alpha=0.99:")
ses_method(app_train_arr, app_test_arr, date_periods_arr, 0.99)
```

```
# %%
# Holt-Winters
# use additive method because seasonal variations are roughly constant throughout the data series
print("Holt-Winters:")
holt_winters_method(app_train_arr, app_test_arr, date_periods_arr, season_num=48)
# %%
print("Holt-Winters:")
holt_winters_method(app_train_arr, app_test_arr, date_periods_arr, season_num=12)
# %%
print("Holt-Winters:")
holt_winters_method(app_train_arr, app_test_arr, date_periods_arr, season_num=24)

# %%
# best model plot - MLE forecast vs test
plt.figure(figsize=(10, 6))
plt.plot(length_test, y_test, label='testing data')
plt.plot(length_test, OLS_model_forecast_best, label='forecast')
plt.xlabel('sample')
plt.ylabel('Wh')
plt.title('Appliance Usage (Wh) Forecasts: Best OLS Model')
plt.legend()
plt.show()
# %%
```

Functions File

```
#%%
# import libraries
import matplotlib.pyplot as plt
import math
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.ar_model import AutoReg
import statsmodels.tsa.holtwinters as ets
from statsmodels.tsa.api import Holt
import itertools
import pandas as pd
import numpy as np
import seaborn as sns

# ACF calculation function
def fun_acf(x, lags=20):
    T = len(x)
    mean = sum(x)/len(x)
    k = []
    for i in range(len(x)):
        if len(k)<=lags:
            k.append(i)
    tau = [1]

    # calculate denominator
    den=0
    for i in x:
        den+=((i-mean)**2)

    # calculate all numerators
    nums = []
    if T == 1:
        return tau # corner case
    else:
        for j in k[1:]:
            num = []
            for i in range(1, len(x)):
                t = x[i]
                t_lag = x[i-j]
```

```
                if (i-j) >= 0:
                    num.append((t - mean)*(t_lag - mean))
            nums.append(num)

    # append to tau
    for i in nums:
        tau.append(sum(i)/den)

    return tau

# ACF list
def list_acf(x):
    T_list = []
    for i in x:
        if len(T_list)<=50: # max lag
            T_list.append(i)

    T_stem = []
    for i in reversed(T_list[1:]):
        T_stem.append(i)
    for i in T_list:
        T_stem.append(i)

    return(T_stem)

# ACF plot function
def plot_acf(x, lags=20):
    '''
    x: output of fun_acf()
    This function is the ACF plot
    '''
    k_list = []
    for i in x:
        if len(k_list)<=lags: # default lag=20
            k_list.append(x.index(i))

    k_stem = []
    for i in reversed(k_list[1:]):
        k_stem.append(-1*i)
    for i in k_list:
        k_stem.append(i)
    #print(k_stem)

    T_list = []
    for i in x:
        if len(T_list)<=50: # max lag 50
            T_list.append(i)

    T_stem = []
    for i in reversed(T_list[1:]):
        T_stem.append(i)
    for i in T_list:
        T_stem.append(i)
    #print(T_stem)

    plt.stem(k_stem, T_stem, use_line_collection = True)
    plt.xlabel("Lags")
    plt.ylabel("Magnitude")
    plt.title("ACF Plot")
    plt.show()

# ADF test function - tests if input variable is stationary
def ADF_Cal(x):
    result = adfuller(x)
    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))
```

```
# Correlation coefficient
def correlation_coefficient_cal(x, y):
    x_bar = sum(x)/len(x)
    y_bar = sum(y)/len(y)
    num = []
    den_1 = []
    den_2 = []
    for i in range(len(x)) :
        a = (x[i] - x_bar)*(y[i] - y_bar)
        num.append(a)
        b = (x[i] - x_bar)**2
        den_1.append(b)
        c = (y[i] - y_bar)**2
        den_2.append(c)
    r = sum(num)/math.sqrt((sum(den_1) * sum(den_2)))
    return r

# Stepwise function - forward selection - for choosing best variables in Multiple Linear Regression/LSE
def stepwise_fun(X,y):
    '''
    X: dataframe without a constant column added
    y: target variable
    '''
    all_combinations = []
    for r in range(len(X.columns.tolist()) + 1):
        combinations_object = itertools.combinations(X.columns.tolist(), r)
        combinations_list = list(combinations_object)
        all_combinations += combinations_list

    stepwise_summary = pd.DataFrame({'Variables': [], 'AIC': [], 'BIC': [], 'Adj. R-squared': []})
    for i in range(1,len(all_combinations)):
        vars_i = np.asarray(all_combinations[i])
        X_i = X[vars_i]
        X_i = sm.add_constant(X_i)
        OLS_i = sm.OLS(y,X_i).fit()
        metrics_i = {'AIC': OLS_i.aic, 'Adj. R-
squared': OLS_i.rsquared_adj, 'BIC': OLS_i.bic, 'Variables': [vars_i], 'R-squared': OLS_i.rsquared}
        stepwise_summary = stepwise_summary.append(metrics_i, ignore_index=True)
    return stepwise_summary

# no constant
def stepwise_fun_base(X,y):
    '''
    X: dataframe without a constant column added
    y: target variable
    '''
    all_combinations = []
    for r in range(len(X.columns.tolist()) + 1):
        combinations_object = itertools.combinations(X.columns.tolist(), r)
        combinations_list = list(combinations_object)
        all_combinations += combinations_list

    stepwise_summary = pd.DataFrame({'Variables': [], 'AIC': [], 'BIC': [], 'Adj. R-squared': []})
    for i in range(1,len(all_combinations)):
        vars_i = np.asarray(all_combinations[i])
        X_i = X[vars_i]
        # X_i = sm.add_constant(X_i)
        OLS_i = sm.OLS(y,X_i).fit()
        metrics_i = {'AIC': OLS_i.aic, 'Adj. R-
squared': OLS_i.rsquared_adj, 'BIC': OLS_i.bic, 'Variables': [vars_i], 'R-squared': OLS_i.rsquared}
        stepwise_summary = stepwise_summary.append(metrics_i, ignore_index=True)
    return stepwise_summary

# Moving average functions
# manual: enter window size and folding order when running function
def cal_MA_wFO_manual(x, window_size=0, folding_order=0):
    print('Enter window size:')
    window_size=int(input())
```

```
    print('Enter folding order (enter 0 if none):')
    folding_order=int(input())
    # corner cases
    if window_size <= 2:
        return("Window size must be greater than 2")
    if (window_size % 2 == 0):
        if (folding_order > 0):
            if (folding_order % 2 != 0):
                return("Window size and folding order must be either both even or both odd")
    if (window_size % 2 != 0):
        if (folding_order > 0):
            if (folding_order % 2 == 0):
                return("Window size and folding order must be either both even or both odd")

    # moving average: no folding order
    i = 0
    moving_averages = []
    while i < len(x) - window_size + 1:
        window = x[i : i + window_size]
        window_average = sum(window) / window_size
        moving_averages.append(window_average)
        i += 1

    if (folding_order < 1):
        return(moving_averages)
    # moving average: folding order
    else:
        ma_ma = []
        j = 0
        while j < len(moving_averages) - folding_order + 1:
            window_j = moving_averages[j : j + folding_order]
            window_average_j = sum(window_j) / folding_order
            ma_ma.append(window_average_j)
            j += 1
        return(ma_ma)
# not manual:  window size (and folding order) are inputs
def cal_MA_wFO(x, window_size, folding_order=0):
    # corner cases
    if window_size <= 2:
        return("Window size must be greater than 2")
    if (window_size % 2 == 0):
        if (folding_order > 0):
            if (folding_order % 2 != 0):
                return("Window size and folding order must be either both even or both odd")
    if (window_size % 2 != 0):
        if (folding_order > 0):
            if (folding_order % 2 == 0):
                return("Window size and folding order must be either both even or both odd")

    # moving average: no folding order
    i = 0
    moving_averages = []
    while i < len(x) - window_size + 1:
        window = x[i : i + window_size]
        window_average = sum(window) / window_size
        moving_averages.append(window_average)
        i += 1

    if (folding_order < 1):
        return(moving_averages)
    # moving average: folding order
    else:
        ma_ma = []
        j = 0
        while j < len(moving_averages) - folding_order + 1:
            window_j = moving_averages[j : j + folding_order]
            window_average_j = sum(window_j) / folding_order
            ma_ma.append(window_average_j)
            j += 1
```

```
        return(ma_ma)

# Estimate AR coefficients - enter # samples, AR order, list of coefficients - generates simulation based
on inputs and uses AutoReg to estimate coeffs
def AR_est():
    print('Enter the number of samples:')
    T_AR = int(input())
    print('Enter the order number:')
    order = int(input())
    print('Enter the parameters (with a space in between each)')
    params = list(map(float, input("Enter a multiple value: ").split()))

    e_AR = np.random.normal(1, np.sqrt(2), size = T_AR) # WN mean and standard dev
    y_AR = np.zeros(len(e_AR))

    for i in range(len(e_AR)):
        if i == 0:
            y_AR[0] = e_AR[0]
        elif i == 1:
            y_AR[1] = params[0]*y_AR[0] + e_AR[1]
        else:
            if order > 1:
                order_len = order * 1
                order_list = [1]
                while order_len > 1:
                    order_list.append(order_list[-1]+1)
                    order_len -=1
            y_AR[i] = sum(params[j-1]*y_AR[i-j] for j in order_list) + e_AR[i]

    model_AR = AutoReg(y_AR, lags=order)
    model_AR_fit = model_AR.fit()
    print('Estimated Coefficients: %s' % model_AR_fit.params)
    true_coeff = [1]
    for i in params:
        true_coeff.append(i)
    print('True Coefficients: %s' % true_coeff)

    return y_AR

# GPAC function - input ACF process from statsmodel library
def Cal_GPAC(acf, cols, rows):
    GPAC_table = np.zeros((cols, rows))
    mid = int(len(acf) / 2)
    for j in range(rows):
        for k in range(1, cols + 1):
            num = np.zeros((k, k))
            den = np.zeros((k, k))

            acf_counter = mid + j

            for c in range(k):
                k_counter = 0
                for r in range(k):
                    den[r, c] = acf[acf_counter + k_counter]
                    k_counter += 1
                acf_counter -= 1

            num[:, :-1] = den[:, :-1]

            acf_counter = mid + j
            for r in range(k):
                num[r, -1] = acf[acf_counter + 1]
                acf_counter += 1

            num_det = np.linalg.det(num)
            den_det = np.linalg.det(den)

            gpac_value = num_det / den_det
            GPAC_table[j, k-1] = gpac_value
```

```
    xticks = np.arange(1,k+1,1)

    plt.subplots(figsize=(15,10))
    ax = sns.heatmap(GPAC_table, vmin=-
1, vmax=1, center=0, square=True, cmap='magma', annot=True, fmt='.3f')
    bottom, top = ax.get_ylim()
    ax.set_ylim(bottom + 0.5, top - 0.5)
    ax.set_xticklabels(xticks, horizontalalignment='center')
    ax.set(xlabel='k', ylabel='j')
    plt.title('Generalized Partial AutoCorrelation (GPAC) Table')
    plt.show()

# BASE MODELS - Average, Naive, Drift, Simple Exponential Smoothing (SES)
# Average Method
def average_method(train, test, times):
    avg_predict = []
    avg_forecast = []
    tot = 0
    count = 0

    for i in range(len(train)-1):
        tot += train[i]
        count += 1
        running_avg = tot / count
        avg_predict.append(running_avg)
    avg_predict = avg_predict[1:]

    avg_forecast_1 = np.mean(train)
    avg_forecast = np.ones(len(test)) * avg_forecast_1
    avg_forecast = avg_forecast.tolist()

    limit = len(train)
    times_train = times[:limit]
    times_test = times[limit:]

    plt.figure()
    plt.plot(times_train, train, label='Training')
    plt.plot(times_test, test, label='Testing')
    plt.plot(times_test, avg_forecast, label='Forecast')
    plt.xlabel('Time')
    plt.xticks(np.arange(0,len(times),round(len(times)/10)), rotation=60)
    plt.ylabel('Amount')
    plt.title('Average Method')
    plt.legend()
    plt.show()

    train_avg_error = train[2:]
    avg_predict_error = train_avg_error - avg_predict
    avg_predict_error_sq = avg_predict_error ** 2
    avg_predict_MSE = sum(avg_predict_error_sq) / len(avg_predict_error_sq)
    print('Average method 1-step prediction MSE:',avg_predict_MSE)

    avg_forecast_error = test - avg_forecast
    avg_forecast_error_sq = avg_forecast_error ** 2
    avg_forecast_MSE = sum(avg_forecast_error_sq) / len(avg_forecast_error_sq)
    print('Average method h-step forecast MSE:',avg_forecast_MSE)

    avg_predict_error_mean = sum(avg_predict_error) / len(avg_predict_error)
    avg_predict_var1 = []
    for i in range(len(avg_predict_error)):
        avg_predict_var1.append((avg_predict_error[i] - avg_predict_error_mean) ** 2)
    avg_predict_var = sum(avg_predict_var1) / len(avg_predict_var1)
    print('Average method 1-step prediction error variance:',avg_predict_var)

    avg_forecast_error_mean = sum(avg_forecast_error) / len(avg_forecast_error)
    avg_forecast_var1 = []
    for i in range(len(avg_forecast_error)):
        avg_forecast_var1.append((avg_forecast_error[i] - avg_forecast_error_mean) ** 2)
```

```
    avg_forecast_var = sum(avg_forecast_var1) / len(avg_forecast_var1)
    print('Average method h-step forecast error variance:',avg_forecast_var)

    q_avg = fun_acf(avg_forecast_error, 20)
    plot_acf(q_avg)
    q_avg = q_avg[1:]

    q_avg_total = 0
    for i in q_avg:
        q_avg_total += (i ** 2)

    q_avg_total = q_avg_total * len(avg_forecast_error)
    print('Average method Q value:', q_avg_total)


# Naive Method
def naive_method(train, test, times):
    naive_predict = []
    for i in range(1, len(train)):
        naive_predict.append(train[(i - 1)])

    naive_forecast_1 = train[-1]
    naive_forecast = np.ones(len(test)) * naive_forecast_1
    naive_forecast = naive_forecast.tolist()

    limit = len(train)
    times_train = times[:limit]
    times_test = times[limit:]

    plt.figure()
    plt.plot(times_train, train, label='Training')
    plt.plot(times_test, test, label='Testing')
    plt.plot(times_test, naive_forecast, label='Forecast')
    plt.xlabel('Time')
    plt.xticks(np.arange(0,len(times),round(len(times)/10)), rotation=60)
    plt.ylabel('Amount')
    plt.title('Naive Method')
    plt.legend()
    plt.show()

    train_naive_error = train[1:]
    naive_predict_error = train_naive_error - naive_predict
    naive_predict_error_sq = naive_predict_error ** 2
    naive_predict_MSE = sum(naive_predict_error_sq) / len(naive_predict_error_sq)
    print('Naive method 1-step forecast MSE:', naive_predict_MSE)

    naive_forecast_error = test - naive_forecast
    naive_forecast_error_sq = naive_forecast_error ** 2
    naive_forecast_MSE = sum(naive_forecast_error_sq) / len(naive_forecast_error_sq)
    print('Naive method h-step forecast MSE:', naive_forecast_MSE)

    naive_predict_error_mean = sum(naive_predict_error) / len(naive_predict_error)
    naive_predict_var1 = []
    for i in range(len(naive_predict_error)):
        naive_predict_var1.append((naive_predict_error[i] - naive_predict_error_mean) ** 2)
    naive_predict_var = sum(naive_predict_var1) / len(naive_predict_var1)
    print('Naive method 1-step prediction error variance:', naive_predict_var)

    naive_forecast_error_mean = sum(naive_forecast_error) / len(naive_forecast_error)
    naive_forecast_var1 = []
    for i in range(len(naive_forecast_error)):
        naive_forecast_var1.append((naive_forecast_error[i] - naive_forecast_error_mean) ** 2)
    naive_forecast_var = sum(naive_forecast_var1) / len(naive_forecast_var1)
    print('Naive method h-step forecast error variance:',naive_forecast_var)

    q_naive = fun_acf(naive_forecast_error, 20)
    plot_acf(q_naive)
    q_naive = q_naive[1:]
```

```
    q_naive_total = 0
    for i in q_naive:
        q_naive_total += (i ** 2)

    q_naive_total = q_naive_total * len(naive_forecast_error)
    print('Naive Method Q value:', q_naive_total)


# Drift Method
def drift_method(train, test, times):
    drift_predict = []
    drift_forecast = []

    for i in range(1, len(train)-1):
        slope = ((train[i]-train[0])/(i))
        intercept = train[0] - slope
        drift_predict.append(intercept + ((i+2) * slope))

    m = (train[-1] - train[0]) / (len(train)-1)
    for i in range(len(test)):
        drift_forecast.append(train[-1] + ((i+1) * m))

    limit = len(train)
    times_train = times[:limit]
    times_test = times[limit:]

    plt.figure()
    plt.plot(times_train, train, label='Training')
    plt.plot(times_test, test, label='Testing')
    plt.plot(times_test, drift_forecast, label='Forecast')
    plt.xlabel('Time')
    plt.xticks(np.arange(0,len(times),round(len(times)/10)), rotation=60)
    plt.ylabel('Amount')
    plt.title('Drift Method')
    plt.legend()
    plt.show()

    train_drift_error = train[2:]
    drift_predict_error = train_drift_error - drift_predict
    drift_predict_error_sq = drift_predict_error ** 2
    drift_predict_MSE = sum(drift_predict_error_sq) / len(drift_predict_error_sq)
    print('Drift method 1-step predict MSE:', drift_predict_MSE)

    drift_forecast_error = test - drift_forecast
    drift_forecast_error_sq = drift_forecast_error ** 2
    drift_forecast_MSE = sum(drift_forecast_error_sq) / len(drift_forecast_error_sq)
    print('Drift method h-step forecast MSE:', drift_forecast_MSE)

    drift_predict_error_mean = sum(drift_predict_error) / len(drift_predict_error)
    drift_predict_var1 = []

    for i in range(len(drift_predict_error)):
        drift_predict_var1.append((drift_predict_error[i] - drift_predict_error_mean) ** 2)
    drift_predict_var = sum(drift_predict_var1) / len(drift_predict_var1)
    print('Drift method 1-step prediction error variance:', drift_predict_var)

    drift_forecast_error_mean = sum(drift_forecast_error) / len(drift_forecast_error)
    drift_forecast_var1 = []
    for i in range(len(drift_forecast_error)):
        drift_forecast_var1.append((drift_forecast_error[i] - drift_forecast_error_mean) ** 2)
    drift_forecast_var = sum(drift_forecast_var1) / len(drift_forecast_var1)
    print('Drift method h-step forecast error variance:', drift_forecast_var)

    q_drift = fun_acf(drift_forecast_error, 20)
    plot_acf(q_drift)
    q_drift = q_drift[1:]
```

```
    q_drift_total = 0
    for i in q_drift:
        q_drift_total += (i ** 2)

    q_drift_total = q_drift_total * len(drift_forecast_error)
    print('Drift method Q Value:', q_drift_total)


# Simple Exponential Smoothing (SES)
def ses_method(train, test, times, alpha):
    ses_predict = []
    ses_forecast = []
    ses_one = train[0]

    for i in range(1, len(train)):
        ses_value = (alpha * train[i - 1]) + ((1 - alpha) * ses_one)
        ses_one = ses_value
        ses_predict.append(ses_value)

    for i in range(len(test)):
        ses_forecast.append(ses_predict[-1])

    limit = len(train)
    times_train = times[:limit]
    times_test = times[limit:]

    plt.figure()
    plt.plot(times_train, train, label='Training Data')
    plt.plot(times_test, test, label='Testing Data')
    plt.plot(times_test, ses_forecast, label='Forecast')
    plt.xlabel('Time')
    plt.xticks(np.arange(0,len(times),round(len(times)/10)), rotation=60)
    plt.ylabel('Amount')
    plt.title('Simple Exponential Smoothing (SES)')
    plt.legend()
    plt.show()

    train_ses_error = train[1:]
    ses_predict_error = train_ses_error - ses_predict
    ses_predict_error_sq = ses_predict_error ** 2
    ses_predict_MSE = sum(ses_predict_error_sq) / len(ses_predict_error_sq)
    print('SES 1-step prediction MSE:', ses_predict_MSE)

    ses_forecast_error = test - ses_forecast
    ses_forecast_error_sq = ses_forecast_error ** 2
    ses_forecast_MSE = sum(ses_forecast_error_sq) / len(ses_forecast_error_sq)
    print('SES h-step forecast MSE:', ses_forecast_MSE)

    ses_predict_error_mean = sum(ses_predict_error) / len(ses_predict_error)
    ses_predict_var1 = []
    for i in range(len(ses_predict_error)):
        ses_predict_var1.append((ses_predict_error[i] - ses_predict_error_mean) ** 2)
    ses_predict_var = sum(ses_predict_var1) / len(ses_predict_var1)
    print('SES 1-step prediction error variance:', ses_predict_var)

    ses_forecast_error_mean = sum(ses_forecast_error) / len(ses_forecast_error)
    ses_forecast_var1 = []
    for i in range(len(ses_forecast_error)):
        ses_forecast_var1.append((ses_forecast_error[i] - ses_forecast_error_mean) ** 2)
    ses_forecast_var = sum(ses_forecast_var1) / len(ses_forecast_var1)
    print('SES h-step forecast error variance:', ses_forecast_var)

    q_ses = fun_acf(ses_forecast_error, 20)
    plot_acf(q_ses)
    q_ses = q_ses[1:]

    q_ses_total = 0
    for i in q_ses:
```

```
        q_ses_total += (i ** 2)

    q_ses_total = q_ses_total * len(ses_forecast_error)
    print('Q Value for SES is:', q_ses_total)


# HOLT-WINTER'S
def holt_winters_method(train, test, times, season_num=2, trend_hw='add', seasonal_hw='add'):
    model = ets.ExponentialSmoothing(train, trend=trend_hw, seasonal =seasonal_hw, seasonal_periods=season
_num, damped_trend=True)

    fit_forecast = model.fit()
    holtwin_forecast = fit_forecast.forecast(steps=len(test))

    # cross validation
    holtwin_predict = []
    for i in range(season_num,len(train)):
        window = train[:i+season_num]
        model = ets.ExponentialSmoothing(window, trend=trend_hw, seasonal=seasonal_hw,seasonal_periods=sea
son_num)
        fit_i = model.fit()
        holtwin_predict_i = fit_i.forecast(steps=len(train[:i+season_num]))
        holtwin_predict_list = holtwin_predict_i.tolist()
        holtwin_predict.append(holtwin_predict_list[-1])

    limit = len(train)
    times_train = times[:limit]
    times_test = times[limit:]

    plt.figure()
    plt.plot(times_train, train, label = 'Training')
    plt.plot(times_test, test, label = 'Test')
    plt.plot(times_test, holtwin_forecast, label = 'Forecast')
    plt.legend(loc = 0)
    plt. title("Holt-Winters Method")
    plt.ylabel('Amount')
    plt.xlabel('Time')
    plt.xticks(np.arange(0,len(times),round(len(times)/10)), rotation=60)
    plt.show()

    # calc holt win predict error
    train_holtwin_error = train[(len(train)-len(holtwin_predict)):]
    holtwin_predict_error = []
    for i in range(len(train_holtwin_error)):
        predict_error = train_holtwin_error[i] - holtwin_predict[i]
        holtwin_predict_error.append(predict_error)

    holtwin_predict_error_sq = []
    for i in range(len(holtwin_predict_error)):
        holtwin_predict_error_sq.append(holtwin_predict_error[i] ** 2)
    holtwin_predict_MSE = np.sum(holtwin_predict_error_sq) / len(holtwin_predict_error_sq)
    print('Holt-Winters 1-step prediction MSE:', holtwin_predict_MSE)

    holtwin_forecast_error = test - holtwin_forecast

    holtwin_forecast_error_sq = holtwin_forecast_error ** 2

    holtwin_forecast_MSE = sum(holtwin_forecast_error_sq) / len(holtwin_forecast_error_sq)
    print('Holt-Winters h-step forecast MSE:', holtwin_forecast_MSE)

    holtwin_predict_error_mean = sum(holtwin_predict_error) / len(holtwin_predict_error)
    holtwin_predict_variance = []
    for i in range(len(holtwin_predict_error)):
        holtwin_predict_vari = (holtwin_predict_error[i] - holtwin_predict_error_mean) ** 2
        holtwin_predict_variance.append(holtwin_predict_vari)
    holtwin_predict_var = np.sum(holtwin_predict_variance) / len(holtwin_predict_variance)
    print('Holt-Winters 1-step prediction error variance:', holtwin_predict_var)
```

```
    holtwin_forecast_error_mean = sum(holtwin_forecast_error) / len(holtwin_forecast_error)
    holtwin_forecast_variance = []
    for i in range(len(holtwin_forecast_error)):
        holtwin_forecast_vari = (holtwin_forecast_error[i] - holtwin_forecast_error_mean) ** 2
        holtwin_forecast_variance.append(holtwin_forecast_vari)
    holtwin_forecast_var = sum(holtwin_forecast_variance) / len(holtwin_forecast_variance)
    print('Holt-Winters h-step forecast error variance:', holtwin_forecast_var)

    q_holtwin = fun_acf(holtwin_forecast_error)
    plot_acf(q_holtwin)
    q_holtwin = q_holtwin[1:]

    q_holtwin_total = 0
    for i in q_holtwin:
        q_holtwin_total += (i ** 2)

    q_holtwin_total = q_holtwin_total * len(holtwin_forecast_error)
    print('Q Value for Holt-Winters is:', q_holtwin_total)

    holtwin_cor = correlation_coefficient_cal(holtwin_forecast_error, test)
    print('Holt-
Winters method correlation coefficient between forecast errors and test set:', holtwin_cor)
```

8. References

- Data source: Luis Candanedo, luismiguel.candanedoibarra '@' umons.ac.be, University of Mons (UMONS)