Household Power Consumption

DATS 6313 Final Project Report of Time Series - Spring 2023

HaeLee Kim

2. Content

3. Figures and Tables

CHAPTER	FIGURES AND TABLES
6.	6-b. Global Active Power between 2007 and 2010 6-c. ACF/PACF 6-d. Correlation Matrix
7.	7-a. ADF-test 7-b. KPSS-test 7-c. Rolling Mean/Variance
8.	8. STL Decomposition
9.	9. y test and Holt-Winters method prediction
10.	Feature selection/elimination
11.	11-a. Training set, Testing set, and H-step Forecast with Average Forecast Method 11-b. Training set, Testing set, and H-step Forecast with Naive Method 11-c. Training set, Testing set, and H-step Forecast with Drift Method 11-d. Training set, Testing set, and H-step Forecast with SES Method (alpha=0.5)
12.	12-a-1. Training set, Testing set, and 1-step ahead prediction with Multiple Linear Regression 12-a-2. Training set, Testing set, and h-step ahead prediction with Multiple Linear Regression
13.	13-a. Generalized Partial Autocorrelation(GPAC) Table: ARMA 13-b. ACF/PACF of the data
14.	14-a. Estimation of ARMA(4,0) model summary 14-b. Estimation of ARMA(4,4) model summary 14-c. Estimation of ARMA(8,0) model summary 14-d. Estimation of ARMA(8,4) model summary
15.	15-e. EX4. SARIMAX (4,0,4) x (0,0,0,0) model summary 15-e. EX4. ACF of Prediction Errors 15-e. EX5. SARIMAX (4,0,4) x (1,0,0,28) model summary 15-e. EX5. ACF of Prediction Errors
16.	16. Console Capture of Deep Learning Model Result
17.	17. Final model selection Table
19.	19. h-Step Ahead Prediction with the Final Model 19. y test and h-Step Ahead Prediction with the Final Model

4. Abstract

This project focuses on the application of Time Series models to analyze and predict individual household electric power consumption using a dataset obtained from the UCI Machine Learning Repository. The dataset covers power consumption measurements from 2006 to 2010 and was preprocessed by handling missing values, resampling to a 6-hourly interval, and converting the data format to CSV.

The report outlines the implementation of various Time Series models, including Average, Naive, Drift, SES, Multiple Linear Regression, ARMA, SARIMA, and LSTM. The performance of the models was evaluated using MSE, AIC, and BIC metrics, and the SARIMA (4,0,4) x (1,0,1,28) model was identified as providing the best fit to the data with an MSE of 0.6063.

The report also discusses the limitations of the final model and suggests other types of models that could potentially improve performance. The project's findings have implications for energy management and demand forecasting and demonstrate the effectiveness of Time Series models in this domain. The report provides an overview of the time series analysis and modeling process, which could be useful for similar applications in the future.

5. Introduction

This project will explore various time series models to accurately predict household power consumption using the Individual Household Electric Power Consumption dataset. The aim of this project is to provide a comprehensive understanding of the time series analysis and modeling process and its practical applications in forecasting household power consumption.

I will begin by discussing the methodology used in this analysis and provide an overview of the time series modeling process. I will then describe the different models used in this study, including Average, Naive, Drift, SES, Multiple Linear Regression, ARMA, SARIMA, and LSTM.

The performance of each model will be evaluated using various metrics, including the mean squared error (MSE), Akaike Information Criterion (AIC), and Bayesian Information Criterion (BIC). Based on this evaluation, I will identify the best-fit model and provide insights into the performance of other models.

The results and findings of this project may be useful in various applications, such as energy management and demand forecasting.

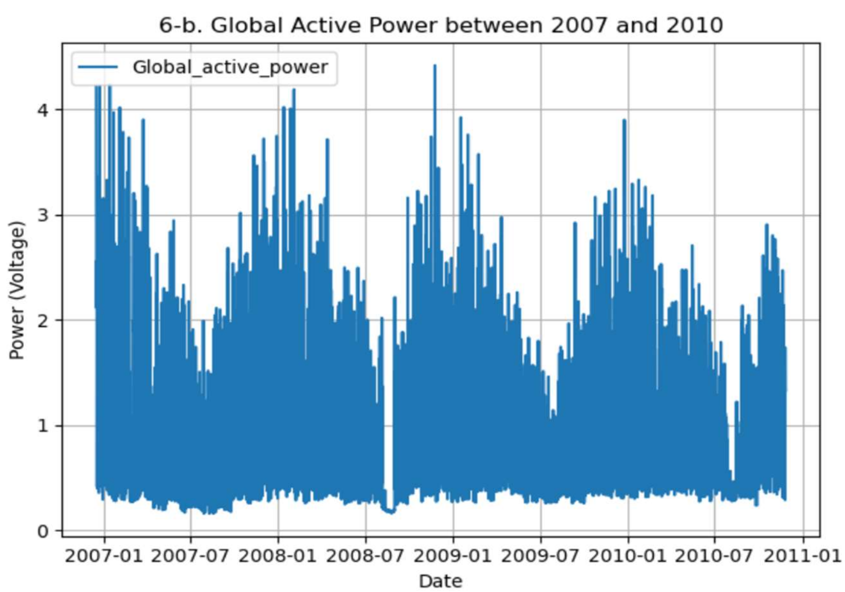
6. Description of the dataset [1][2]

The raw dataset of individual household electric power consumption contains 2,075,259 data points with 7 features. The data was collected in a local area, Sceaux (7km of Paris), in France between December 2006 and November 2010, with a data point recorded every minute. The features in the dataset include global active power, global reactive power, voltage, global intensity, sub-metering 1, sub-metering 2, and sub-metering 3. In this project, "Global Active Power" variable will be used as a dependent variable. It represents the active energy consumed by the household and is measured in kilowatts (kW). Therefore, my goal is to accurately predict the power consumption of a household based on the other variables in the dataset.

6-a. Pre-processing dataset

First, for dataset cleaning, the missing values were filled with the mean of that specific hour.
Second, to simplify the data, the raw dataset was resampled hourly and then resampled every 6 hours, resulting in one day having 4 data points. The resampling was done by taking the mean of the hourly and 6-hourly data points.
As a result, the final dataset contains 5766 data points with 7 features, including the dependent variable and independent variables.

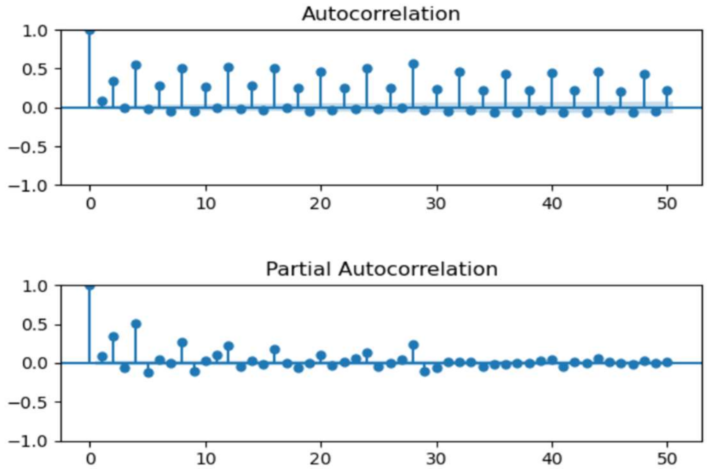
6-b. Plot of the dependent variable versus time



The plot of the dependent variable over time shows the fluctuation of the energy consumption over the course of the four-year period. The energy consumption follows a cyclical pattern with periodic dips and spikes, which may be related to daily or seasonal factors.

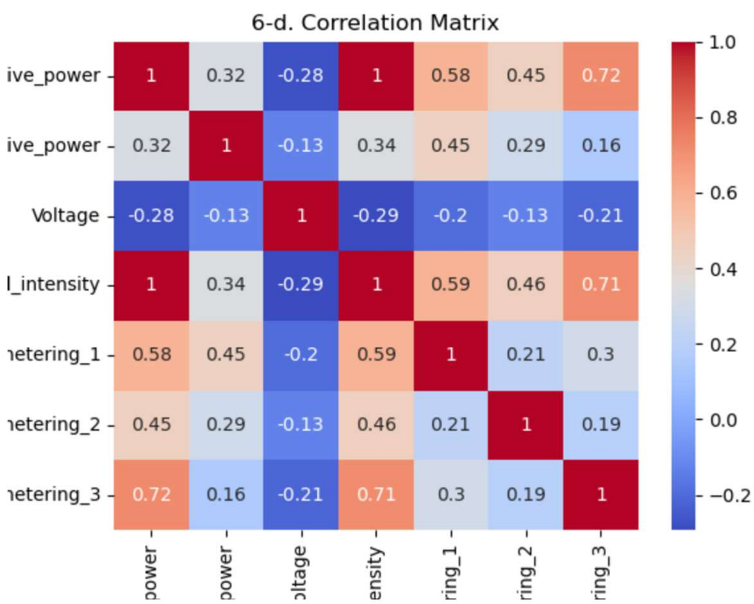
[1] Data Source: https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption
[2] To make the original dataset more accessible for analysis, I converted it from a text file format to a more user-friendly CSV format. The resulting CSV file that I attached in blackboard contains a 6-hourly resampled dataset that is easier for you to work with for this project.

6-c. ACF/PACF of the dependent variable



The ACF and PACF both showed a tail off pattern. However, they also showed a repeating pattern every 4 lags, which suggests the presence of seasonality in the data.

6-d. Correlation Matrix with seaborn heatmap with the Pearson's correlation coefficient



The high correlation between 'Global_intensity' and 'Global_active_power' is because they are both related to the amount of electricity being used in the household. 'Global_active_power' is calculated by multiplying 'Global_intensity' and 'Voltage'. Therefore, it is expected that these two variables will have a strong correlation.

6-e. Split the dataset

The dataset was divided into a training set consisting of 80% of the data and a test set consisting of the remaining 20%.

7. Stationarity

7-a. ADF-test

```
ADF Statistic: -5.844490
p-value: 0.000000
Critical Values:
    1%: -3.431
    5%: -2.862
    10%: -2.567
```

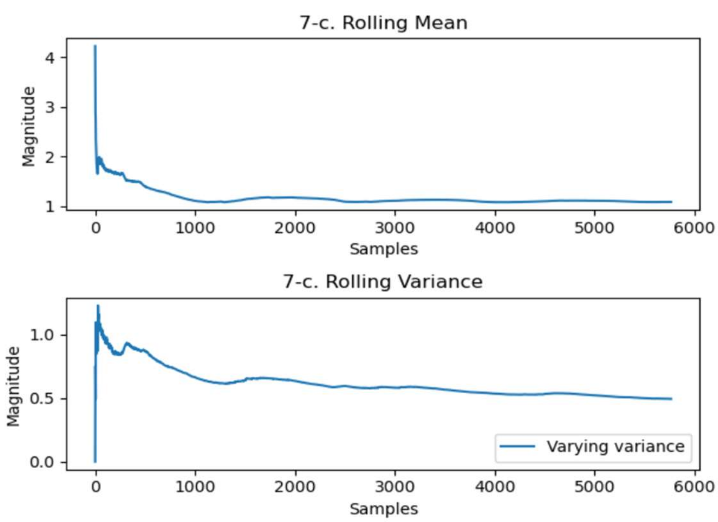
The ADF result shows that p-value is 0.00 below a threshold (5%) and it suggests we reject the null hypothesis. It means the dependent variable, Global Active Power is stationary.

7-b. KPSS-test

```
Results of KPSS Test:
Test Statistic             0.443840
p-value                    0.058259
Lags Used                 39.000000
Critical Value (10%)       0.347000
Critical Value (5%)        0.463000
Critical Value (2.5%)      0.574000
Critical Value (1%)        0.739000
```

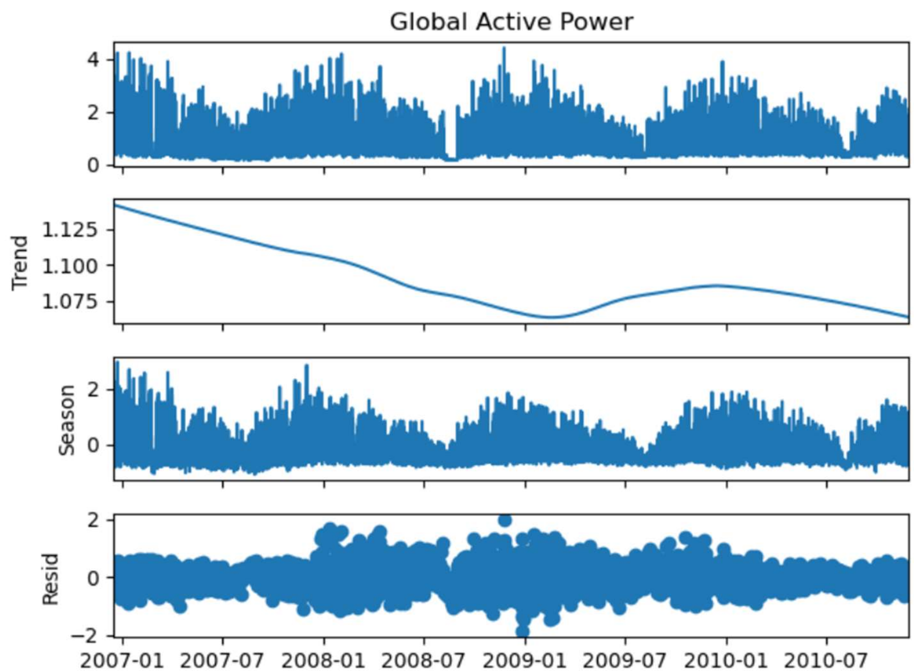
The result shows that p-value is 0.058 above a threshold (5%) and it suggests we cannot reject the null hypothesis. It further supports that the dependent variable, Global Active Power is stationary.

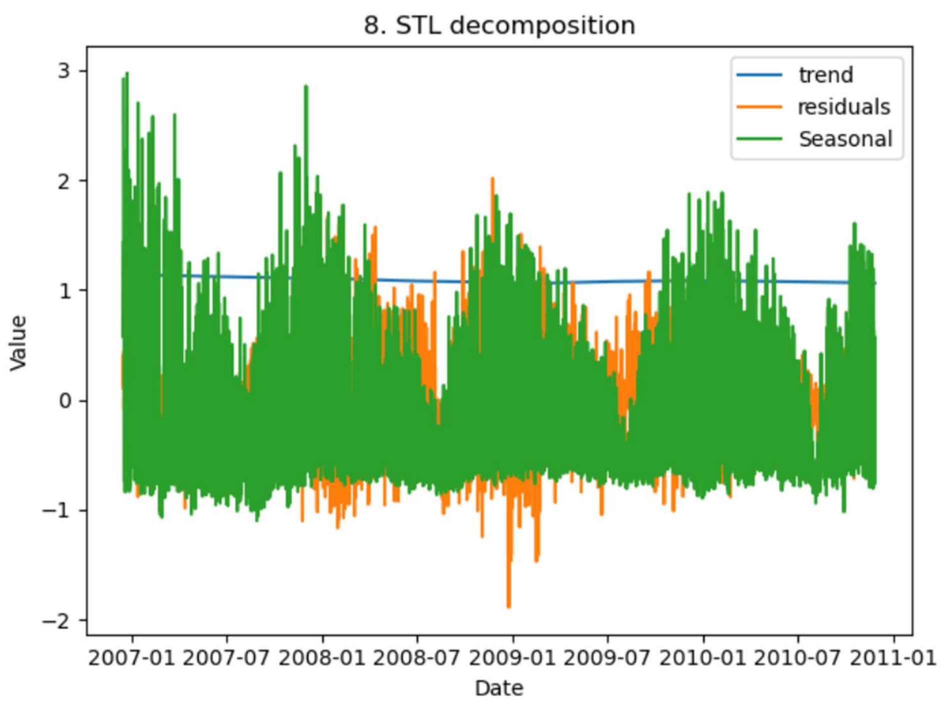
7-c. Rolling Mean/Variance



The plot demonstrated that the rolling mean and variance became constant over time, indicating that the data is stationary.

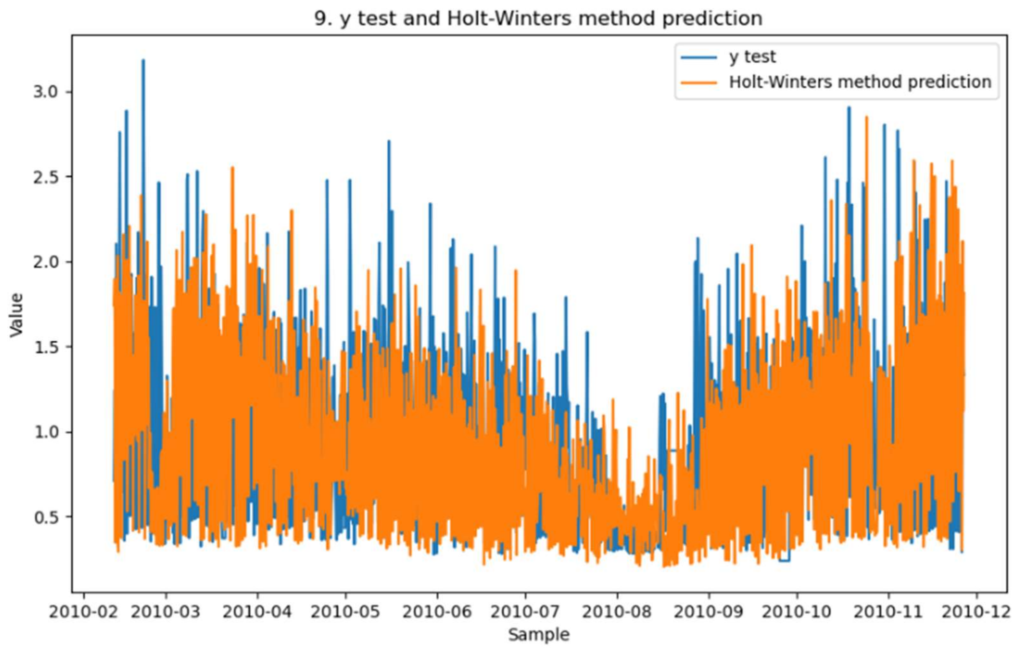
8. Time series Decomposition





I performed STL decomposition and found that there is a strong yearly seasonal pattern present in the data. This was observed in the seasonality component of the decomposition, with a strength of 0.7998. Additionally, the trend component of the decomposition had a strength of 0.005, indicating a relatively weak trend in the data.

9. Holt-Winters method



The Holt-Winters method, a forecasting technique that considers both trend and seasonality, was applied to the dataset, resulting in a Mean Squared Error (MSE) of 0.2565. This indicates that the method performed well in predicting the values of the dataset.

10. Feature selection/elimination

To check and remove the collinearity issue of the data, I have performed the following steps.

Step 1.

a. Variance Inflation Factor (VIF) was calculated for each independent variable. The results showed that Global_intensity had the highest VIF value of 13.16, followed by Voltage with a VIF value of 10.43 and Global_reactive_power with a VIF value of 10.17, indicating a high level of multicollinearity between these variables.
b. Singular values for each variable were also calculated, with the highest singular value being 3.34e+08 for the first variable. If singular values are close to zero, the corresponding features are highly correlated. It would be necessary to remove some of them to avoid collinearity in regression model.
c. The condition number was calculated to be 5813.70. When the condition number is above 1000, it indicates that there is severe multicollinearity in the data.
d. To prevent collinearity it might be necessary to eliminate some variables. To address this, I excluded the feature with the highest VIF value, which was Global_intensity and performed the Singular Values, Conditional Number, and VIF test again in step 2.

Step 2.

a. Variance Inflation Factor (VIF) was calculated for each independent variable. The results showed that Global_reactive_power had the highest VIF value of 10.17, followed by Voltage with a VIF value of 9.31.
b. Singular values for each variable were also calculated, with the highest singular value being 3.34e+08 for the first variable.
c. The condition number was still high as 5811.26 over 1000.
d. Again, I removed the feature with the highest VIF value, which was Global_reactive_power and checked the multicollinearity.

Step 3.

a. Variance Inflation Factor (VIF) showed that all variables had values below 5, indicating that there was no significant multicollinearity among the variables. Therefore, I retained all the independent variables for the regression model. VIF = [('Voltage', 2.53), ('Sub_metering_1', 1.44), ('Sub_metering_2', 1.33), ('Sub_metering_3', 2.73)]
b. The Singular values also increased.
   Singular Values = [3.34791475e+08 1.82725530e+05 3.99918749e+04 2.29013472e+04]
c. The condition number is now 123.55, which is acceptable.

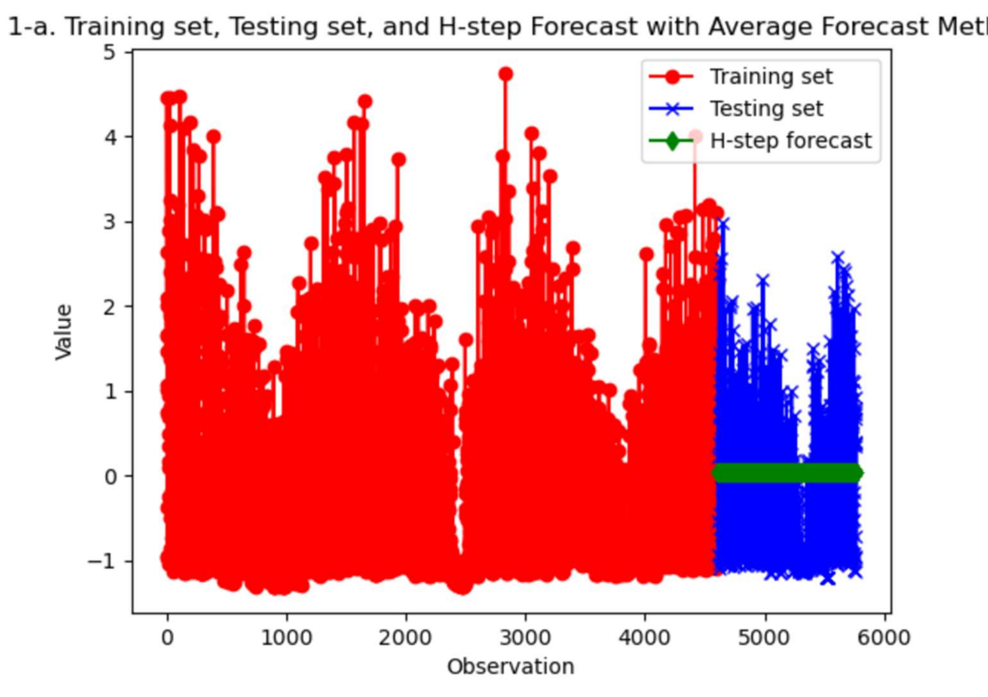
Step 4.

a. I standardized the dataset and performed the backward stepwise regression.
b. Final features to keep are 'Voltage', 'Sub_metering_1', 'Sub_metering_2', 'Sub_metering_3'.

11. Base-models

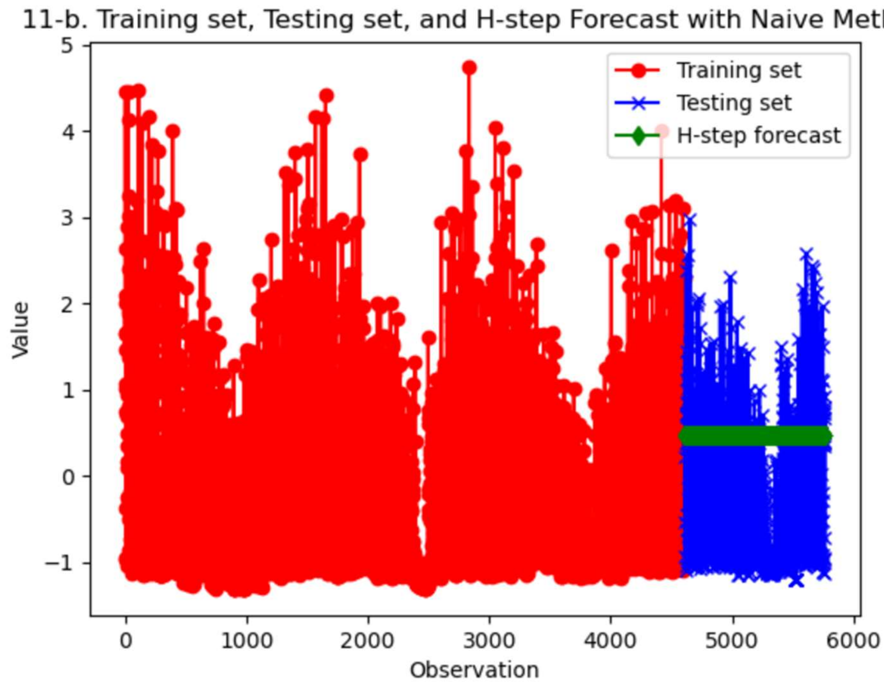
11-a. Average Forecast Method

The average mean squared error (MSE) of prediction errors was 1.0927, while the average MSE of forecast errors was 0.6481.



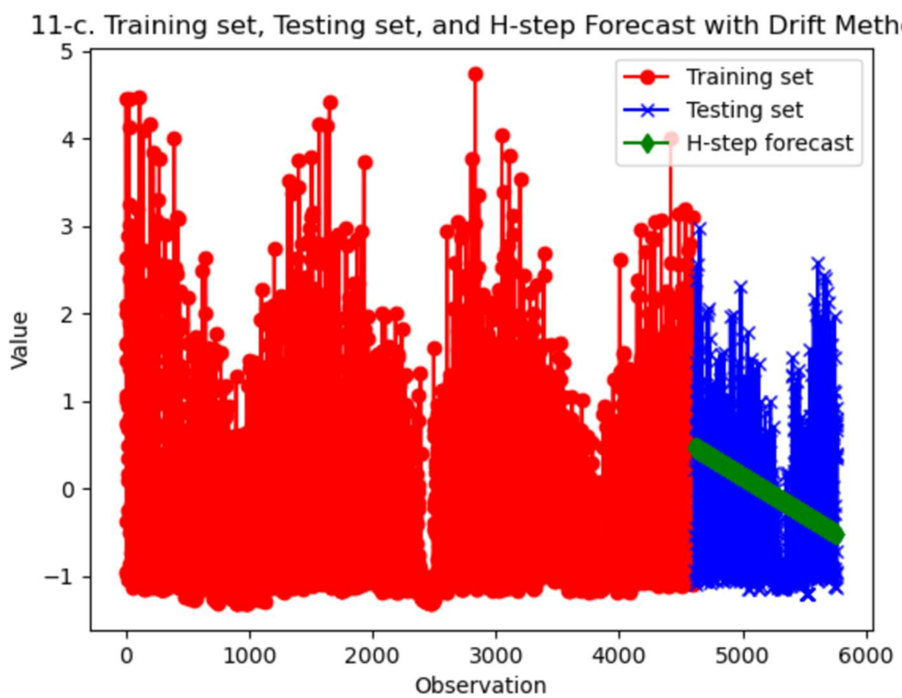
11-b. Naïve Method

The average mean squared error (MSE) of prediction errors was 1.9809, while the average MSE of forecast errors was 0.9933.



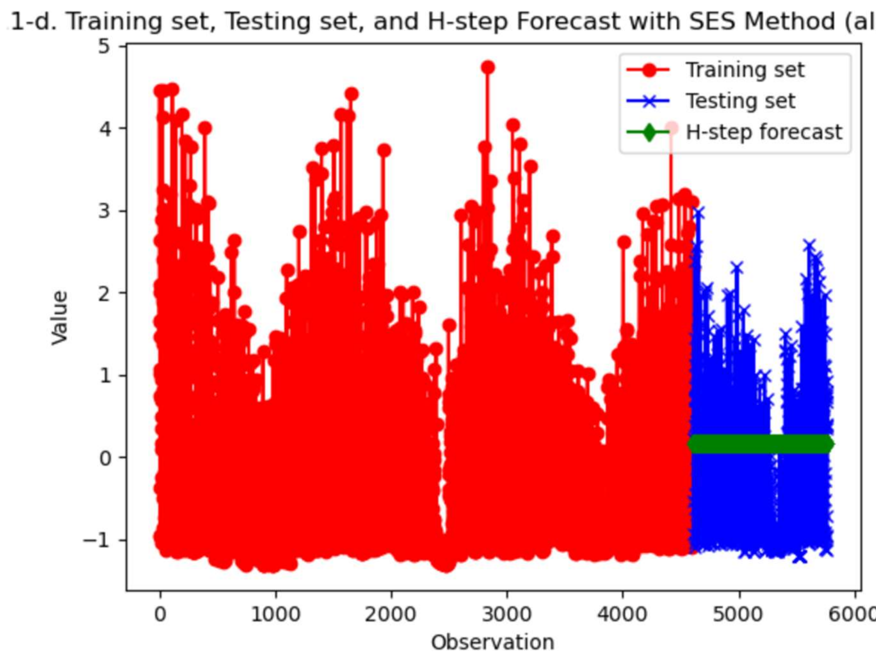
11-c. Drift Method

The average mean squared error (MSE) of prediction errors was 1.9859, while the average MSE of forecast errors was 0.6838.



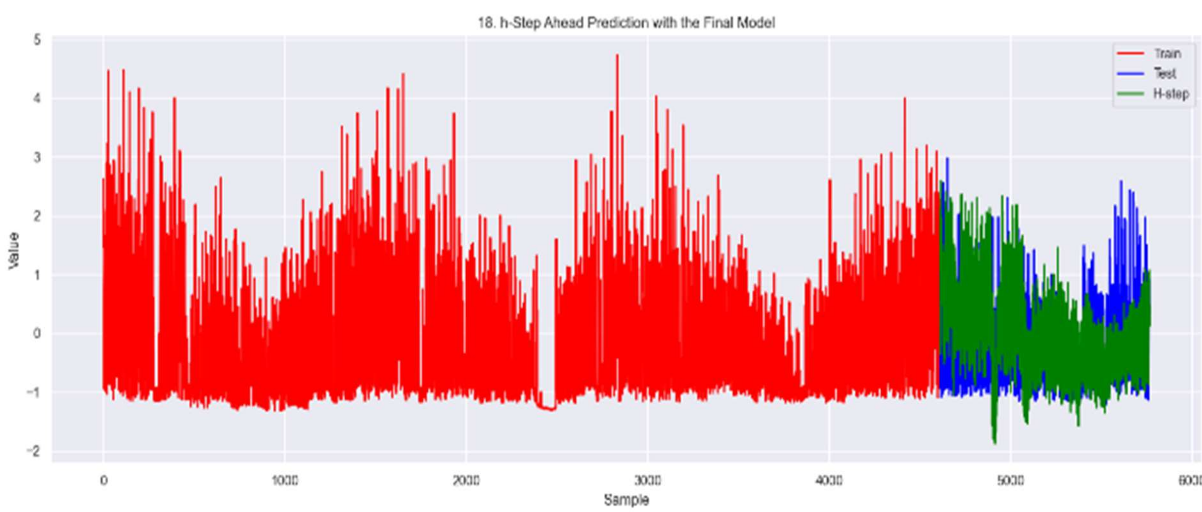
11-d. Simple and Exponential Smoothing (alpha=0.5) Method

The average mean squared error (MSE) of prediction errors was 1.2008, while the average MSE of forecast errors was 0.7074.



11-e. SARIMA Model

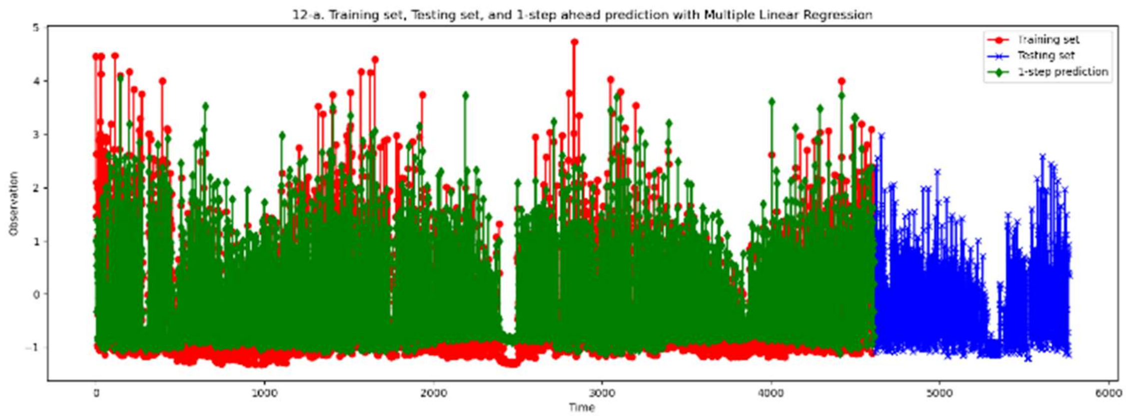
The final selected SARIMA Model (will be discussed in 17, 18, and 19 again) shows the MSE of as 0.7074.

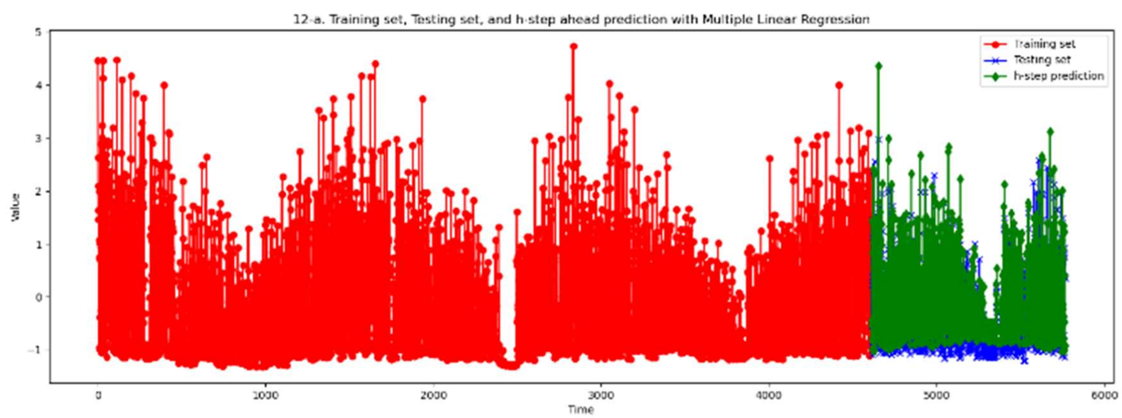


12. Multiple linear regression model

12-a. one-step ahead prediction and comparison of the performance versus the test set

The average mean squared error (MSE) of one-step ahead prediction was 0.289, while the average MSE of forecast errors was 1.650.





12-b. Hypothesis tests analysis: F-test, t-test

12-c. AIC, BIC, RMSE, R-squared and Adjusted R-squared

The regression model's goodness of fit statistics are as follows: the AIC value is 7386.812, the BIC value is 7418.994, the RMSE value is 0.538, and the adjusted R-squared value is 0.733.

12-e. Q-value

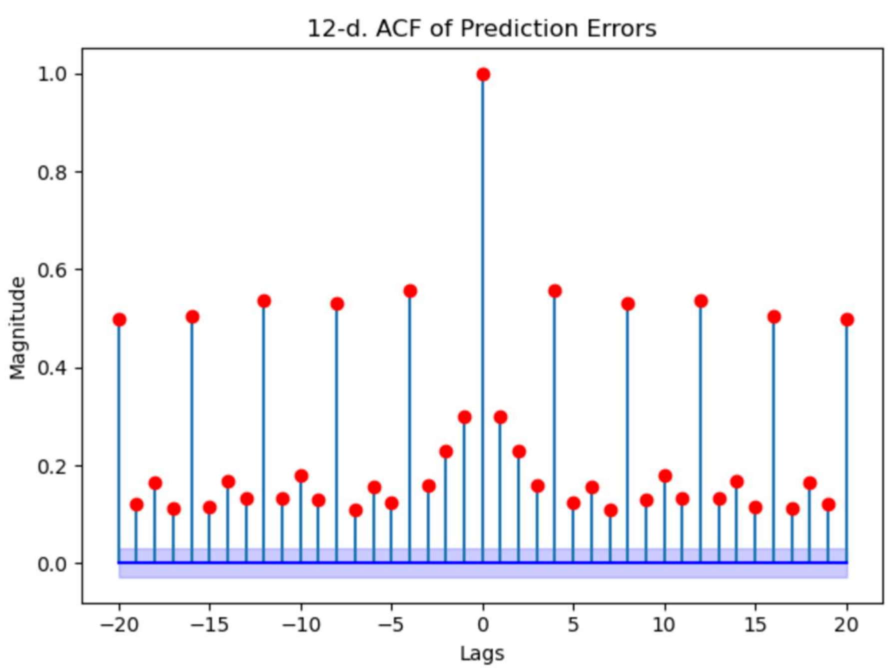
Upon analyzing the model, the Q value was found to be 8325.847.

12-f. Variance and mean of the residuals

The mean of the residuals was 5.145 and the variance was 0.289, indicating that the model has some room for improvement.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      0   R-squared:                       0.734
Model:                            OLS   Adj. R-squared:                  0.734
Method:                 Least Squares   F-statistic:                     3178.
Date:                Wed, 10 May 2023   Prob (F-statistic):               0.00
Time:                        17:17:21   Log-Likelihood:                -3688.4
No. Observations:                4612   AIC:                             7387.
Df Residuals:                    4607   BIC:                             7419.
Df Model:                           4
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const              0.0237      0.008      2.979      0.003       0.008       0.039
Voltage           -0.0457      0.008     -5.818      0.000      -0.061      -0.030
Sub_metering_1     0.3568      0.008     42.999      0.000       0.341       0.373
Sub_metering_2     0.2596      0.008     32.915      0.000       0.244       0.275
Sub_metering_3     0.5756      0.008     67.832      0.000       0.559       0.592
==============================================================================
Omnibus:                     1789.002   Durbin-Watson:                   1.393
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             7697.182
Skew:                           1.876   Prob(JB):                         0.00
Kurtosis:                       8.097   Cond. No.                         1.56
==============================================================================
```

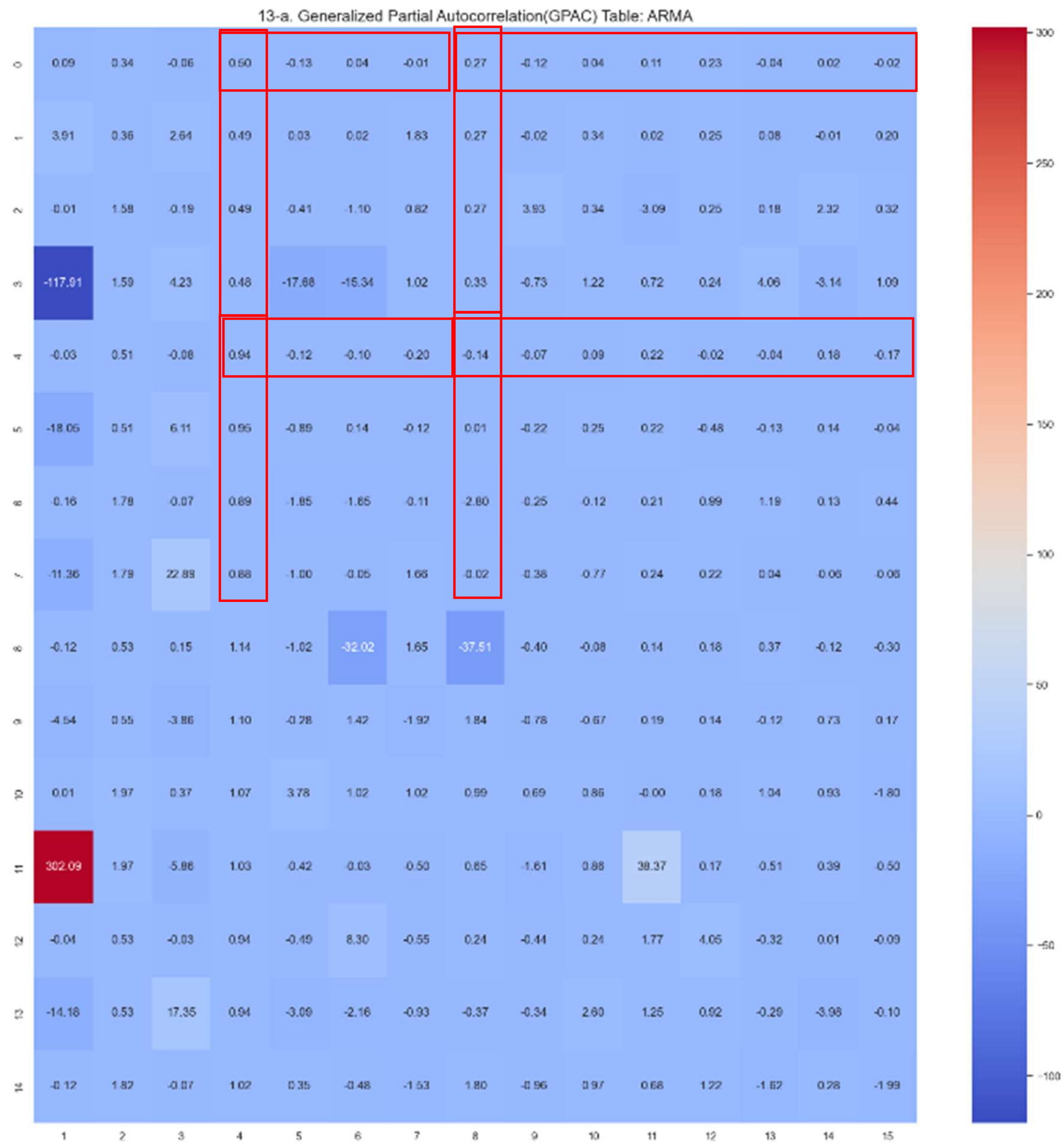
12-d. ACF of residuals



The pattern in the ACF every 4 lags may suggest a daily pattern in the data, as there are 4 observations per day. Further exploring this daily pattern could potentially improve the performance of the model.

13. ARMA and ARIMA and SARIMA model order determination

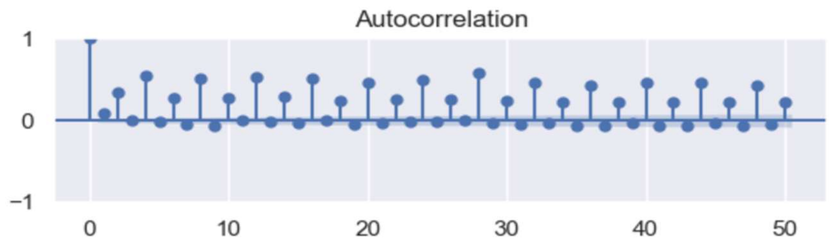
13-a.c. Preliminary ARMA model order (highlighted) determination with GPAC table

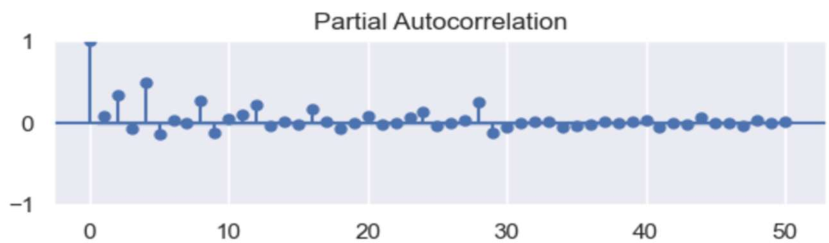


During the preliminary model development, I checked the Generalized Partial Autocorrelation (GPAC) table to determine the order of the ARMA model. Based on the analysis, the suitable model orders for my dataset were identified as ARMA(4,0), ARMA(4,4), ARMA(8,0), and ARMA(8,8).

13-b. Plot of the autocorrelation function

The Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) exhibited tail-off patterns, indicating the presence of correlation in the data.





14. Levenberg Marquardt algorithm.

14-a. Estimation of ARMA(4,0) model parameters

After estimating the parameters for ARMA(4,0) model, it was observed that the estimated AR coefficients were statistically significant, which was confirmed by both p-values and confidence intervals.

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      0   No. Observations:                 4612
Model:                 ARIMA(4, 0, 0)   Log Likelihood               -5747.136
Date:                Wed, 10 May 2023   AIC                          11506.271
Time:                        17:17:24   BIC                          11544.890
Sample:                             0   HQIC                         11519.862
                               - 4612
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0394      0.050      0.786      0.432      -0.059       0.138
ar.L1          0.1133      0.012      9.786      0.000       0.091       0.136
ar.L2          0.1731      0.012     14.821      0.000       0.150       0.196
ar.L3         -0.1029      0.011     -9.449      0.000      -0.124      -0.082
ar.L4          0.5054      0.012     41.327      0.000       0.481       0.529
sigma2         0.7075      0.013     53.056      0.000       0.681       0.734
===================================================================================
Ljung-Box (L1) (Q):                  19.05   Jarque-Bera (JB):              1176.85
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.67   Skew:                             0.89
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.72
===================================================================================
```

14-b. Estimation of ARMA(4,4) model parameters

After estimating the parameters for ARMA(4,4) model, it was observed that the estimated AR and MA coefficients were statistically significant, which was confirmed by both p-values and confidence intervals.

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      0   No. Observations:                 4612
Model:                 ARIMA(4, 0, 4)   Log Likelihood               -5186.447
Date:                Wed, 10 May 2023   AIC                          10392.894
Time:                        17:17:30   BIC                          10457.258
Sample:                             0   HQIC                         10415.545
                               - 4612
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0662      0.074      0.894      0.371      -0.079       0.211
ar.L1         -0.0087      0.002     -3.827      0.000      -0.013      -0.004
ar.L2         -0.0064      0.002     -2.856      0.004      -0.011      -0.002
ar.L3         -0.0084      0.002     -3.588      0.000      -0.013      -0.004
ar.L4          0.9890      0.002    412.210      0.000       0.984       0.994
ma.L1          0.1149      0.008     13.982      0.000       0.099       0.131
ma.L2          0.1291      0.009     14.925      0.000       0.112       0.146
ma.L3          0.0947      0.008     11.752      0.000       0.079       0.110
ma.L4         -0.8180      0.008    -97.656      0.000      -0.834      -0.802
sigma2         0.5545      0.009     64.535      0.000       0.538       0.571
===================================================================================
Ljung-Box (L1) (Q):                  74.83   Jarque-Bera (JB):              1321.23
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.61   Skew:                             0.79
Prob(H) (two-sided):                  0.00   Kurtosis:                         5.09
===================================================================================
```

14-c. Estimation of ARMA(8,0) model parameters

After estimating the parameters for the ARMA(8,0) model, it was observed that not all of the estimated AR and MA coefficients were statistically significant. As a result, it was decided that this model would not be used for further analysis.

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      0   No. Observations:                 4612
Model:                 ARIMA(8, 0, 0)   Log Likelihood               -5522.053
Date:                Wed, 10 May 2023   AIC                          11064.106
Time:                        17:17:32   BIC                          11128.470
Sample:                             0   HQIC                         11086.757
                               - 4612
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0419      0.062      0.681      0.496      -0.079       0.163
ar.L1          0.1871      0.012     15.223      0.000       0.163       0.211
ar.L2          0.1224      0.014      8.914      0.000       0.095       0.149
ar.L3         -0.0357      0.013     -2.734      0.006      -0.061      -0.010
ar.L4          0.3715      0.013     28.257      0.000       0.346       0.397
ar.L5         -0.1169      0.013     -8.657      0.000      -0.143      -0.090
ar.L6          0.0091      0.015      0.609      0.543      -0.020       0.038
ar.L7         -0.0573      0.014     -4.223      0.000      -0.084      -0.031
ar.L8          0.2756      0.012     22.260      0.000       0.251       0.300
sigma2         0.6416      0.012     52.639      0.000       0.618       0.666
===================================================================================
Ljung-Box (L1) (Q):                   4.65   Jarque-Bera (JB):              1147.56
Prob(Q):                              0.03   Prob(JB):                         0.00
Heteroskedasticity (H):               0.67   Skew:                             0.88
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.70
===================================================================================
```

14-b. Estimation of ARMA(8,4) model parameters

After estimating the parameters for the ARMA(8,4) model, it was observed that not all of the estimated AR and MA coefficients were statistically significant. As a result, it was decided that this model would not be used for further analysis.

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      0   No. Observations:                 4612
Model:                 ARIMA(8, 0, 4)   Log Likelihood               -5114.247
Date:                Wed, 10 May 2023   AIC                          10256.495
Time:                        17:17:43   BIC                          10346.605
Sample:                             0   HQIC                         10288.206
                               - 4612
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0834      0.162      0.516      0.606      -0.233       0.400
ar.L1          0.2308      0.015     15.268      0.000       0.201       0.260
ar.L2          0.1560      0.017      9.336      0.000       0.123       0.189
ar.L3         -0.0340      0.017     -2.017      0.044      -0.067      -0.001
ar.L4          1.0653      0.015     69.419      0.000       1.035       1.095
ar.L5         -0.2284      0.015    -15.512      0.000      -0.257      -0.200
ar.L6         -0.1479      0.016     -9.239      0.000      -0.179      -0.117
ar.L7          0.0288      0.016      1.785      0.074      -0.003       0.060
ar.L8         -0.0767      0.014     -5.315      0.000      -0.105      -0.048
ma.L1         -0.0092      0.009     -1.036      0.300      -0.027       0.008
ma.L2         -0.0188      0.010     -1.972      0.049      -0.037      -0.000
ma.L3          0.0186      0.009      2.033      0.042       0.001       0.037
ma.L4         -0.9127      0.009   -103.654      0.000      -0.930      -0.895
sigma2         0.5437      0.009     57.913      0.000       0.525       0.562
===================================================================================
Ljung-Box (L1) (Q):                   0.10   Jarque-Bera (JB):              1074.18
Prob(Q):                              0.75   Prob(JB):                         0.00
Heteroskedasticity (H):               0.63   Skew:                             0.77
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.79
===================================================================================
```

14-e.

I compared the performance of various ARMA models using the LM algorithm, and calculated the train and test mean squared errors (MSE) for each model. The results were as follows:

ARMA(4, 0): Train MSE (Prediction) = 0.71, Test MSE (Forecasting) = 0.64

ARMA(4, 4): Train MSE (Prediction) = 0.56, Test MSE (Forecasting) = 0.60

ARMA(8, 0): Train MSE (Prediction) = 0.64, Test MSE (Forecasting) = 0.64

ARMA(8, 4): Train MSE (Prediction) = 0.54, Test MSE (Forecasting) = 0.67

Overall, these results suggest that the ARMA(4,4) model is the best fit for the given dataset, and can be used for future forecasting and analysis.

15. Diagnostic Analysis

15-a. Diagnostic tests on ARMA(4,4)

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      0   No. Observations:                 4612
Model:                 ARIMA(4, 0, 4)   Log Likelihood               -5186.447
Date:                Wed, 10 May 2023   AIC                          10392.894
Time:                        17:17:30   BIC                          10457.258
Sample:                             0   HQIC                         10415.545
                               - 4612
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0662      0.074      0.894      0.371      -0.079       0.211
ar.L1         -0.0087      0.002     -3.827      0.000      -0.013      -0.004
ar.L2         -0.0064      0.002     -2.856      0.004      -0.011      -0.002
ar.L3         -0.0084      0.002     -3.588      0.000      -0.013      -0.004
ar.L4          0.9890      0.002    412.210      0.000       0.984       0.994
ma.L1          0.1149      0.008     13.982      0.000       0.099       0.131
ma.L2          0.1291      0.009     14.925      0.000       0.112       0.146
ma.L3          0.0947      0.008     11.752      0.000       0.079       0.110
ma.L4         -0.8180      0.008    -97.656      0.000      -0.834      -0.802
sigma2         0.5545      0.009     64.535      0.000       0.538       0.571
===================================================================================
Ljung-Box (L1) (Q):                  74.83   Jarque-Bera (JB):              1321.23
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.61   Skew:                             0.79
Prob(H) (two-sided):                  0.00   Kurtosis:                         5.09
===================================================================================
```

The coefficients are statistically significant, based on both the p-values and the confidence intervals. Furthermore, the Ljung-Box test has provided a p-value of 0.00, which indicates that there is no evidence of residual autocorrelation.

Additionally, there is no zero/pole cancellation in the model. The poles of the model are [-9.99935888e-01+0.j, -6.77291587e-05+0.99885101j, -6.77291587e-05-0.99885101j, 9.91381616e-01+0.j], and the zeros are [-0.97227237+0.j, -0.00465345+0.98527024j, -0.00465345-0.98527024j, 0.86666145+0.j].

These results indicate that the ARMA(4,4) model is a good fit for the data, and it can be used to make accurate predictions of household power consumption.

15-b. Display the estimated variance of the error and the estimated covariance of the estimated parameters.

```
15-b. Estimated variance of the error for ARMA(4,4): 1.0
15-b. Estimated covariance of the estimated parameters for ARMA(4,4):
 [[ 0.005  0.     0.     0.    -0.    -0.    -0.    -0.    -0.    -0.    ]
 [ 0.     0.     0.     0.     0.    -0.    -0.    -0.    -0.    -0.    ]
 [ 0.     0.     0.     0.     0.    -0.    -0.    -0.    -0.     0.    ]
 [ 0.     0.     0.     0.     0.    -0.    -0.    -0.    -0.     0.    ]
 [-0.     0.     0.     0.     0.    -0.    -0.    -0.    -0.     0.    ]
 [-0.    -0.    -0.    -0.    -0.     0.     0.     0.     0.    -0.    ]
 [-0.    -0.    -0.    -0.    -0.     0.     0.     0.     0.    -0.    ]
 [-0.    -0.    -0.    -0.    -0.     0.     0.     0.     0.    -0.    ]
 [-0.    -0.    -0.    -0.    -0.     0.     0.     0.     0.     0.    ]
 [-0.    -0.     0.     0.     0.    -0.    -0.    -0.     0.     0.    ]]
```

15-c. Model Bias Check

Based on the analysis conducted, the ARMA(4,4) model appears to be unbiased. The constant coefficient is close to zero and the confidence interval includes zero, which is a sign of unbiasedness. This indicates that the mean of the errors is not significantly different from zero, and there is no systematic over or under-prediction in the model.

15-d. Variance of the residual errors versus Variance of the forecast errors

```
15-d. Variance of the residual errors is 0.5636
15-d. Variance of the forecast errors is 0.4322
```

15-e. Check for the better ARIMA or SARIMA model

Step 1. SARIMA model without weekly seasonality check

In the first step of my analysis, I tested four different seasonal autoregressive integrated moving average (SARIMA) models on the dataset.

The models tested were SARIMA(0,0,0)x(1,0,0,4), SARIMA(0,0,0)x(1,0,1,4), SARIMA(4,0,0)x(0,0,0,0), and SARIMA(4,0,4)x(0,0,0,0).

Ex1. SARIMA (0,0,0) x (1,0,0,4): MSE = 0.6337

Ex2. SARIMA (0,0,0) x (1,0,1,4): MSE = 0.6312

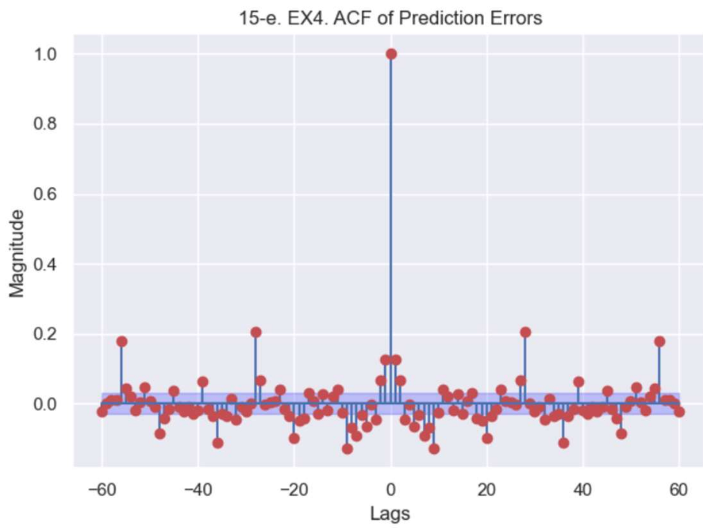
Ex3. SARIMA (4,0,0) x (0,0,0,0): MSE = 0.6340

Ex4. SARIMA (4,0,4) x (0,0,0,0): MSE = 0.5730

Among these models, SARIMA(4,0,4)x(0,0,0,0) had the lowest mean squared error (MSE) of 0.5730, indicating that it produced the most accurate predictions.

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      0   No. Observations:                 4612
Model:               SARIMAX(4, 0, 4)   Log Likelihood               -5184.631
Date:                Wed, 10 May 2023   AIC                          10387.262
Time:                        19:40:16   BIC                          10445.190
Sample:                             0   HQIC                         10407.648
                               - 4612
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.0109      0.002     -4.609      0.000      -0.016      -0.006
ar.L2         -0.0084      0.002     -3.565      0.000      -0.013      -0.004
ar.L3         -0.0102      0.003     -4.038      0.000      -0.015      -0.005
ar.L4          0.9870      0.003    393.601      0.000       0.982       0.992
ma.L1          0.1118      0.008     13.626      0.000       0.096       0.128
ma.L2          0.1344      0.009     15.406      0.000       0.117       0.152
ma.L3          0.0887      0.008     10.589      0.000       0.072       0.105
ma.L4         -0.8139      0.009    -94.941      0.000      -0.831      -0.797
sigma2         0.5537      0.008     66.038      0.000       0.537       0.570
===================================================================================
Ljung-Box (L1) (Q):                  80.53   Jarque-Bera (JB):              1387.38
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.61   Skew:                             0.81
Prob(H) (two-sided):                  0.00   Kurtosis:                         5.14
===================================================================================
```

However, after analyzing the autocorrelation function (ACF) plot, it was observed that the plot showed patterns at every 28th data point, suggesting the presence of weekly seasonality in the dataset. Further investigation into seasonality is needed to confirm this observation. It is worth noting that the data is recorded every 6 hours, resulting in a total of 4 observations per day, which translates to 28 observations per week.



Step 2. SARIMA model with weekly seasonality check

The initial step in developing a SARIMA model was to fit the model without considering the presence of weekly seasonality. However, in the second step, I tested two different SARIMA models.

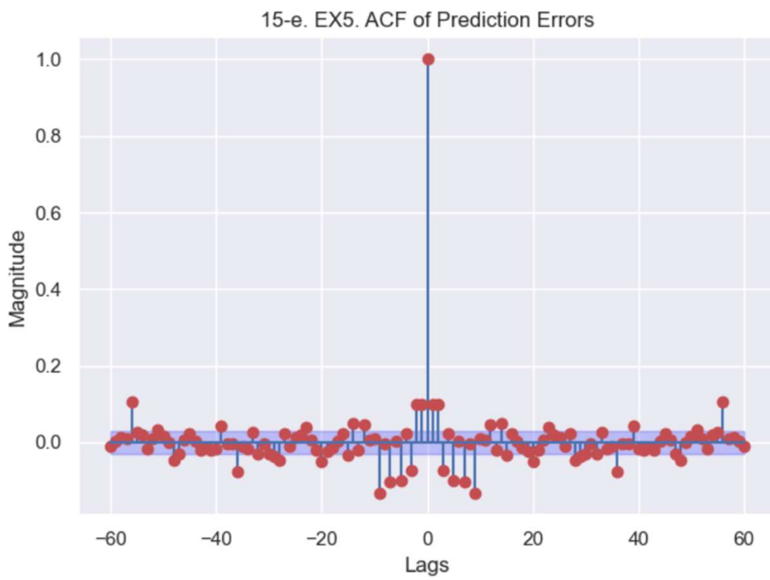
The models tested were SARIMA(4,0,4)x(1,0,0,28) and SARIMA(4,0,4)x(1,0,1,28)

Ex5. SARIMA (4,0,4) x (1,0,0,28): MSE = 0.5544

Ex6. SARIMA (4,0,4) x (1,0,1,28): MSE = 0.7777

```
                                      SARIMAX Results
===========================================================================================
Dep. Variable:                                  0   No. Observations:                  4612
Model:             SARIMAX(4, 0, 4)x(1, 0, [1], 28)   Log Likelihood               -4829.440
Date:                            Tue, 09 May 2023   AIC                           9680.879
Time:                                    18:50:46   BIC                           9751.680
Sample:                                         0   HQIC                          9705.796
                                           - 4612
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.1402      0.017     -8.129      0.000      -0.174      -0.106
ar.L2         -0.1004      0.019     -5.354      0.000      -0.137      -0.064
ar.L3         -0.1242      0.019     -6.444      0.000      -0.162      -0.086
ar.L4          0.8012      0.017     47.262      0.000       0.768       0.834
ma.L1          0.3580      0.023     15.292      0.000       0.312       0.404
ma.L2          0.3382      0.027     12.671      0.000       0.286       0.391
ma.L3          0.2913      0.026     11.310      0.000       0.241       0.342
ma.L4         -0.5031      0.024    -21.105      0.000      -0.550      -0.456
ar.S.L28       0.9697      0.004    218.533      0.000       0.961       0.978
ma.S.L28      -0.8283      0.011    -73.737      0.000      -0.850      -0.806
sigma2         0.4744      0.007     68.398      0.000       0.461       0.488
===================================================================================
Ljung-Box (L1) (Q):                   4.11   Jarque-Bera (JB):              1508.06
Prob(Q):                              0.04   Prob(JB):                         0.00
Heteroskedasticity (H):               0.63   Skew:                             0.74
Prob(H) (two-sided):                  0.00   Kurtosis:                         5.38
===================================================================================
```

After examining various SARIMAX models with different seasonal orders, I found that the SARIMAX model with a seasonal order of (1, 0, 1, 28) provided the best fit based on the lowest MSE value. This model was able to capture the weekly seasonality and ARMA patterns in the data, resulting in a low residual error and a high degree of statistical significance for all coefficients. Furthermore, the ACF of the residuals for this model exhibited a white noise pattern, indicating a good model fit.



16. Deep Learning Model

```
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
16. Mean squared error (MSE) of LSTM (Test: Forecasting): 1.8444
```

I utilized a multivariate LSTM model to fit the dataset and performed h-step predictions. The model yielded a mean squared error (MSE) of 1.8444.

17. Final Model selection

MODELS	MSE	AIC	BIC
AVERAGE	0.6480		
NAIVE	0.9933		
DRIFT	0.6838		
SES	0.7074		
MULTIPLE LINEAR REGRESSION	1.6502		
ARIMA (4,0,4)	0.6032	10256.495	10346.605
SARIMA (4,0,4) X (1,0,1,28)	0.6063	9680.879	9751.680
LSTM	1.5568		

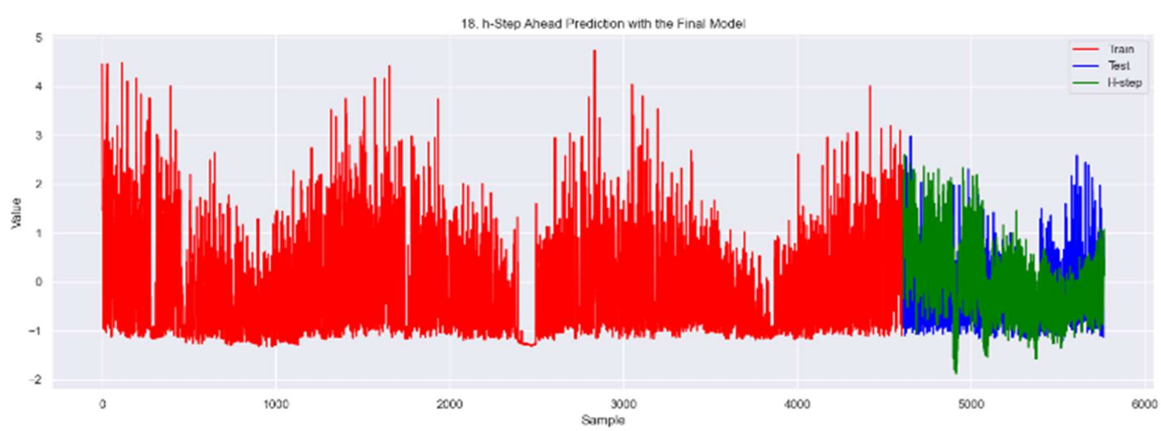
The SARIMA(1,0,1,28) model seems to be the most suitable for the dataset, with an MSE of 0.6063 and lower AIC and BIC values when compared to the ARIMA(4,0,4) model. This model captures both the seasonal and non-seasonal components effectively. The ARIMA (4,0,4) model covers the non-seasonal component with the inclusion of autoregressive (AR), differencing (I), and moving average (MA) terms. The SARIMA (1,0,1,28) model with a weekly seasonality of 28, captures the seasonal component.

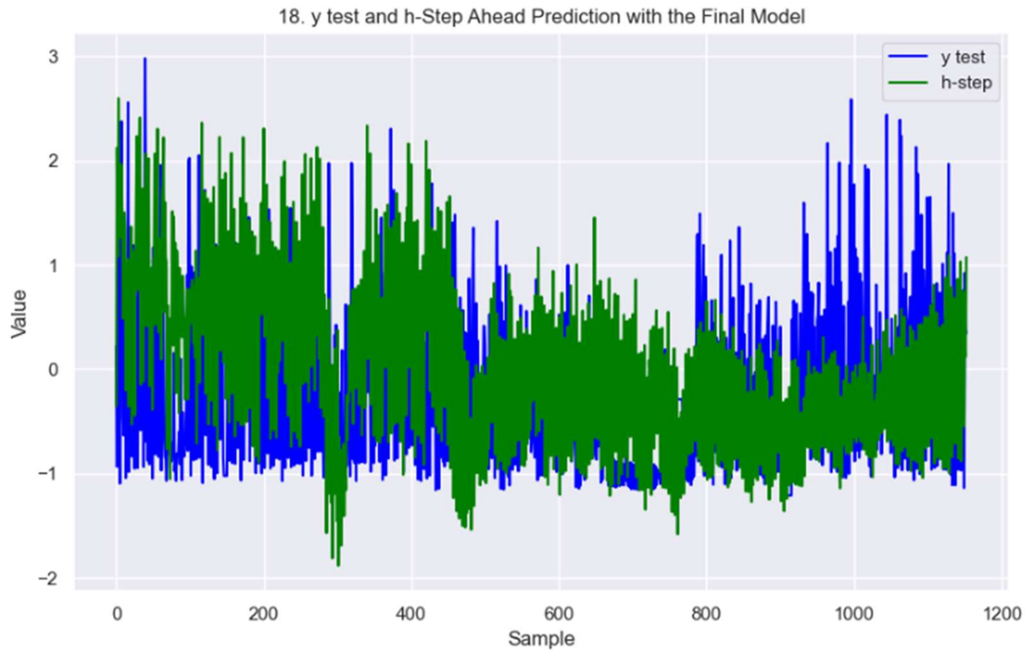
18. Forecast function

I selected the SARIMA (4,0,4) x (1,0,1,28) model and developed a function for h-step ahead prediction. The forecast results for this function can be found in section 19.

19. h-step ahead Predictions

To assess the performance of the SARIMA(1,0,1,28) model, I generated a multi-step forecast for the test dataset and compared the predicted values against the actual values by plotting them on the same graph. The plot clearly shows that the model accurately captures the underlying patterns in the data and provides a good fit to the test set. Overall, the SARIMA model exhibits good performance in forecasting multiple time steps.





20. Summary and conclusion

In this project, various time series models were employed to forecast household power consumption, including Average, Naive, Drift, SES, Multiple Linear Regression, ARMA, SARIMA, and LSTM. The models were evaluated based on the Mean Squared Error (MSE), Akaike Information Criterion (AIC), and Bayesian Information Criterion (BIC), and the SARIMA (4,0,4) x (1,0,1,28) model was found to be the most accurate with an MSE of 0.6063. This model takes into account both non-seasonal and seasonal components of the data, providing a comprehensive and precise prediction. The use of multiple time series models allowed for a thorough evaluation of the performance of each method, providing insight into the strengths and weaknesses of each approach. Overall, the results of this project demonstrate the effectiveness of time series models for forecasting household power consumption.

However, there are some limitations to the SARIMA (4,0,4) x (1,0,1,28) model. Firstly, the model may not be suitable for forecasting long-term trends or sudden changes in the data, which can significantly affect the accuracy of the forecast. Additionally, the model may be influenced by external factors that are not considered as features in this dataset, such as weather conditions or social events.

To improve the performance of the model, it may be worthwhile exploring other types of models, such as neural network-based models, which can capture more complex patterns and relationships in the data. Advanced models like Gradient Boosting, XGBoost, or Random Forest may also be effective, especially if the dataset has many variables and interactions between them.

21. Appendix

```
# Final Project
# Spring 2023 Time-Series
# HaeLee Kim

import csv
# with open(r'C:\Users\haele\Desktop\23 Spring Time-Series\Final
Project\household_power_consumption.txt') as infile:
#     with open('TS_Final_Dataset.csv', 'w', newline='') as outfile:
#         writer = csv.writer(outfile, delimiter=',')
#         writer.writerow(['Date', 'Time', 'Global_active_power',
'Global_reactive_power', 'Voltage', 'Global_intensity', 'Sub_metering_1',
'Sub_metering_2', 'Sub_metering_3'])
#         for line in infile:
#             line = line.strip().replace(';', ',')
#             writer.writerow(line.split(','))
#
# import pandas as pd
# df = pd.read_csv(r'C:\Users\haele\Desktop\23 Spring Time-Series\Final
Project\TS_Final_Dataset.csv')
# df = df.drop(0)
# df = df.reset_index(drop=True)
# print(df.info())
# print(df.head())
# df = df.replace('?', pd.NA)
# if df.isna().any().any():
#     print("Yes, there are missing values in the DataFrame.")
#     # print(df.isna().sum())
# else:
#     print("No, there are no missing values in the DataFrame.")
#
# df['Date'] = pd.to_datetime(df['Date'],
format='%d/%m/%Y').dt.strftime('%Y-%m-%d')
# df["DateTime"] = pd.to_datetime(df["Date"] + " " + df["Time"])
# df = df.set_index("DateTime")
# df['Global_active_power'] = pd.to_numeric(df['Global_active_power'],
errors='coerce')
# df['Global_reactive_power'] = pd.to_numeric(df['Global_reactive_power'],
errors='coerce')
# df['Voltage'] = pd.to_numeric(df['Voltage'], errors='coerce')
# df['Global_intensity'] = pd.to_numeric(df['Global_intensity'],
errors='coerce')
# df['Sub_metering_1'] = pd.to_numeric(df['Sub_metering_1'],
errors='coerce')
# df['Sub_metering_2'] = pd.to_numeric(df['Sub_metering_2'],
errors='coerce')
# df['Sub_metering_3'] = pd.to_numeric(df['Sub_metering_3'],
errors='coerce')
#
# # Resample the data to hourly frequency and aggregate the values using
mean
# hourly_means = df.resample("H").mean(numeric_only=True)
# df = df.fillna(hourly_means.ffill())
```

```
# df_hourly = df.resample("H").mean(numeric_only=True)
# # print(df_hourly.head())
# # print(df_hourly.shape)
# # print(df_hourly.isna().sum())
# # df_hourly.to_csv('TS_Final_Dataset_Hourly.csv', index=True)
#
# # Resample the data by 6-hourly interval and calculate the mean for each
group
# df_six_hourly = df_hourly.resample("6H").mean(numeric_only=True)
# print(df_six_hourly.head())
# print(df_six_hourly.shape)
# print(df_six_hourly.isna().sum())
# df_six_hourly.to_csv('TS_Final_Dataset_Six_Hourly.csv', index=True)

# 6. Description of the dataset
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import signal
import math
import statsmodels.api as sm
import seaborn as sns
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

df = pd.read_csv(r'C:\Users\haele\Desktop\23 Spring Time-Series\Final
Project\TS_Final_Dataset_Six_Hourly.csv', parse_dates=['DateTime'],
index_col='DateTime')
y = df.index # Extract the datetime index as the y variable
print(df.head())
print(df.info())
print(df.isna().sum())
print(df.shape)

# 6-b. Plot of the dependent variable versus time
global_active = df['Global_active_power']
global_reactive = df['Global_reactive_power']
plt.figure()
plt.plot(y, global_active, label='Global_active_power')
plt.grid()
plt.title('6-b. Global Active Power between 2007 and 2010')
plt.legend(loc='upper left')
plt.xlabel('Date')
plt.ylabel('Power (Voltage)')
plt.tight_layout()
plt.show()

# 6-c. ACF/PACF of the dependent variable
def ACF_PACF_Plot(y,lags):
 acf = sm.tsa.stattools.acf(y, nlags=lags)
 pacf = sm.tsa.stattools.pacf(y, nlags=lags)
 fig = plt.figure()
 plt.subplot(211)
 plt.title(f'6-c. ACF/PACF of the dataset')
 plot_acf(y, ax=plt.gca(), lags=lags)
```

```
 plt.subplot(212)
 plot_pacf(y, ax=plt.gca(), lags=lags)
 fig.tight_layout(pad=3)
 plt.show()
ACF_PACF_Plot(df['Global_active_power'],50)

# 6-d. ACF/PACF of the dependent variable
corr_matrix = df.corr(method='pearson')
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('6-d. Correlation Matrix')
plt.show()

# 6-e. Split the dataset into train set (80%) and test set (20%)
X = df.drop('Global_active_power', axis=1)
y = df['Global_active_power']
train_size = int(len(df) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# 7. Stationarity
# 7-a. ADF-test
from statsmodels.tsa.stattools import adfuller
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))
ADF_Cal(df['Global_active_power'])
print('7-a. The ADF result shows that p-value is 0.00 below a threshold 
(5%) and it suggests we reject the null hypothesis. It means Global Active 
Power is stationary.')

# 7-b. KPSS-test
from statsmodels.tsa.stattools import kpss
def kpss_test(timeseries):
 print ('Results of KPSS Test:')
 kpsstest = kpss(timeseries, regression='c', nlags="auto")
 kpss_output = pd.Series(kpsstest[0:3], index=['Test Statistic','p-
value','Lags Used'])
 for key,value in kpsstest[3].items():
    kpss_output['Critical Value (%s)'%key] = value
 print (kpss_output)
kpss_test(df['Global_active_power'])
print('7-b. The result shows that p-value is 0.058 above a threshold (5%) 
and it suggests we cannot reject the null hypothesis. It means Global 
Active Power is stationary.')

# 7-c. rolling mean and variance
def Cal_rolling_mean_var(y, N):
    rolling_mean = np.zeros(N)
    rolling_var = np.zeros(N)
    for i in range(N):
```

```
        rolling_mean[i] = np.mean(y[:i+1])
        rolling_var[i] = np.var(y[:i+1], ddof=0)
    # print("Rolling Mean:\n", rolling_mean[-5:])
    # print("Rolling Variance:\n", rolling_var[-5:])

    fig, ax = plt.subplots(2,1)
    for j in range(2):
        if j==0:
            ax[j].plot(rolling_mean)
            ax[j].set_title('7-c. Rolling Mean')
        else:
            ax[j].plot(rolling_var, label='Varying variance')
            ax[j].set_title('7-c. Rolling Variance')
            ax[j].legend(loc='lower right')
        ax[j].set_xlabel('Samples')
        ax[j].set_ylabel('Magnitude')
    fig.tight_layout()
    plt.show()
Cal_rolling_mean_var(df['Global_active_power'], len(df))

# 8. Time series Decomposition
from statsmodels.tsa.seasonal import STL
Temp = pd.Series(df['Global_active_power'].values, index=df.index, name =
'Global Active Power')
STL = STL(Temp, period=365*4)
res = STL.fit()
fig = res.plot()
plt.show()
T = res.trend
S = res.seasonal
R = res.resid
plt.figure()
plt.plot(T.index, T.values, label = 'trend')
plt.plot(S.index, R.values, label = 'residuals')
plt.plot(R.index, S.values, label = 'Seasonal')
plt.title('8. STL decomposition')
plt.xlabel('Date')
plt.ylabel('Value')
plt.legend()
plt.tight_layout()
plt.show()
# Calculate the strength of trend
FT = max(0, 1 - np.var(R) / (np.var(T + R)))
print(f'8. The strength of trend for this data set is {FT}')
# Calculate the strength of seasonality
FS = max(0, 1 - np.var(R) / np.var(S + R))
print(f'8. The strength of Seasonality for this data set is {FS}')

# 9. Holt-Winters method
from statsmodels.tsa.holtwinters import ExponentialSmoothing
model = ExponentialSmoothing(y_train, seasonal_periods=365*4, trend='add',
seasonal='add')
model_fit = model.fit()
y_pred = model_fit.forecast(len(X_test))
```

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
print('9. Mean Squared Error (MSE) of Holt-Winters method:', mse)

plt.figure(figsize=(10, 6))
plt.plot(y_test, label='y test')
plt.plot(y_pred, label='Holt-Winters method prediction')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('9. y test and Holt-Winters method prediction')
plt.legend()
plt.show()

# 10. Feature selection/elimination
# 10-a. SVD analysis
H = X.T @ X
s, d, v = np.linalg.svd(H)
print("10-a. Step 1. SingularValues = ", d)

# 10-b  condition number
con_num = np.linalg.cond(X)
print("10-b. Step 1. ConditionNumber =", con_num)

# 10-c. VIF test and Check/Eliminate multicollinearity
from statsmodels.stats.outliers_influence import variance_inflation_factor
def calculate_vif(X):
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in
range(X.shape[1])]
    vif_tuples = [(vif.iloc[i, 0], vif.iloc[i, 1]) for i in
range(vif.shape[0])]
    return vif_tuples
X1 = sm.add_constant(X) # add intercept term
model1 = sm.OLS(y, X1).fit()
print(model1.summary())
vif_scores1 = calculate_vif(X)
print("step 1. VIF:", vif_scores1)
# step2
Xa = X.drop('Global_intensity', axis=1)
H = X.T @ Xa
s, d, v = np.linalg.svd(H)
print("10-a. step 2. SingularValues = ", d)
con_num = np.linalg.cond(Xa)
print("10-b. step 2. ConditionNumber =", con_num)
X2 = sm.add_constant(Xa)
model2 = sm.OLS(y, X2).fit()
print(model2.summary())
vif_scores2 = calculate_vif(Xa)
print("step 2. VIF:", vif_scores2)
# step3
Xb = Xa.drop('Global_reactive_power', axis=1)
H = X.T @ Xb
```

```
s, d, v = np.linalg.svd(H)
print("10-a. step 3. SingularValues = ", d)
con_num = np.linalg.cond(Xb)
print("10-b. step 3. ConditionNumber =", con_num)
X3 = sm.add_constant(Xb) # add intercept term
model3 = sm.OLS(y, X3).fit()
print(model3.summary())
vif_scores3 = calculate_vif(Xb)
print("step 3. VIF:", vif_scores3)


# 10-d. backward stepwise regression
# standardize the dataset
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_std = pd.DataFrame(scaler.fit_transform(Xb), columns=Xb.columns)
y_std = pd.DataFrame(scaler.fit_transform(y.values.reshape(-1, 1)))
# unknown coefficients/ statsmodels package and OLS function
train_size = int(len(df) * 0.8)
X_train_std, X_test_std= X_std[:train_size], X_std[train_size:]
y_train_std, y_test_std = y_std[:train_size], y_std[train_size:]
X_train1_std= sm.add_constant(X_train_std)
Y_train_std = np.array(y_train_std).reshape((-1, 1))
# backward stepwise regression
keep = ['Voltage', 'Sub_metering_1', 'Sub_metering_2', 'Sub_metering_3']
eliminate = []
OLS = sm.OLS(Y_train_std, sm.add_constant(X_train1_std)).fit()
pvalues = OLS.pvalues
max_pvalue = pvalues.drop('const').max()
while max_pvalue >= 0.05 and len(keep) > 1:
    k = pvalues.drop('const').idxmax()
    eliminate.append(k)
    keep.remove(k)
    X_train_sub = X_train_std[keep]
    OLS = sm.OLS(Y_train_std, sm.add_constant(X_train_sub)).fit()
    pvalues = OLS.pvalues
    max_pvalue = pvalues.drop('const').max()
print(OLS.summary())
print(f"10-d. Features to keep: {keep}")
print(f"10-d. Features to eliminate: {eliminate}")

# prediction for the test and plot
X_train_selected = sm.add_constant(X_train_std[keep])
new_OLS = sm.OLS(y_train_std, X_train_selected).fit()
y_train_pred = new_OLS.predict(X_train_selected)
X_test_selected = sm.add_constant(X_test_std[keep])
y_test_pred = new_OLS.predict(X_test_selected)
plt.plot(y_train_std, label='Train')
plt.plot(y_test_std, label='Test')
plt.plot(y_test_pred, label='Predicted Test')
plt.xlabel('Observation')
plt.ylabel('Value')
plt.title('10-d. Train, Test, and Predicted Values')
plt.legend()
```

```
plt.show()

# 11.Base-models: average, naïve, drift, simple and exponential smoothing.
# 11-a. Average Forecast Method
# 1-step ahead prediction (training set)
y = y_train_std.values
y_hat = np.zeros(len(y))
e = np.zeros(len(y))
for i in range(1, len(y)):
    y_hat[i] = np.mean(y[:i])
    e[i] = y[i] - y_hat[i]
e2 = np.square(e)
# h-step ahead forecast (testing set)
yh = y_test_std.values
yh_hat = np.mean(y)*np.ones(len(yh))
eh = np.zeros(len(yh))
for i in range(len(yh)):
    eh[i] = yh[i] - yh_hat[i]
eh2 = np.square(eh)
# Plot the test set, training set and the h-step forecast in one graph.
plt.plot(y, color='red', marker='o', label='Training set')
plt.plot(np.arange(len(y), len(y)+len(yh)), yh, color='blue', marker='x',
label='Testing set')
plt.plot(np.arange(len(y), len(y)+len(yh)), yh_hat, color='green',
marker='d', label='H-step forecast')
plt.title('11-a. Training set, Testing set, and H-step Forecast with
Average Forecast Method')
plt.xlabel('Observation')
plt.ylabel('Value')
plt.legend()
plt.show()
# Average: MSE of prediction errors and the forecast errors
mse_prediction = np.mean(e2[2:])
mse_forecast = np.mean(eh2)
table = [["MSE of prediction errors (Average)","MSE of forecast errors
(Average)"], [mse_prediction, mse_forecast]]
print(table)
rmse_prediction = np.sqrt(mse_prediction)
rmse_forecast = np.sqrt(mse_forecast)
table = [["RMSE of prediction errors (Average)","RMSE of forecast errors
(Average)"], [rmse_prediction, rmse_forecast]]
print(table)

# 11-b. Naive Method
# 1-step ahead prediction (training set)
y = y_train_std.values
y_hat = np.zeros(len(y))
e = np.zeros(len(y))
for i in range(1, len(y)):
    y_hat[i] = y[i-1]
    e[i] = y[i] - y_hat[i]
e2 = np.square(e)
# h-step ahead forecast (testing set)
yh = y_test_std.values
```

```
yh_hat = y[-1]*np.ones(len(yh))
eh = np.zeros(len(yh))
for i in range(len(yh)):
    eh[i] = yh[i] - y[-1]
eh2 = np.square(eh)
# Plot the test set, training set and the h-step forecast in one graph.
plt.plot(y, color='red', marker='o', label='Training set')
plt.plot(np.arange(len(y), len(y)+len(yh)), yh, color='blue', marker='x',
label='Testing set')
plt.plot(np.arange(len(y), len(y)+len(yh)), yh_hat, color='green',
marker='d', label='H-step forecast')
plt.title('11-b. Training set, Testing set, and H-step Forecast with Naive
Method')
plt.xlabel('Observation')
plt.ylabel('Value')
plt.legend()
plt.show()
# Naive MSE
mse_prediction = np.mean(e2[2:])
mse_forecast = np.mean(eh2)
table = [["MSE of prediction errors (Naive)","MSE of forecast errors
(Naive)"], [mse_prediction, mse_forecast]]
print(table)
rmse_prediction = np.sqrt(mse_prediction)
rmse_forecast = np.sqrt(mse_forecast)
table = [["RMSE of prediction errors (Naive)","RMSE of forecast errors
(Naive)"], [rmse_prediction, rmse_forecast]]
print(table)

# 11-c. Drift Method
# 1-step ahead prediction (training set)
y = y_train_std.values
y_hat = np.zeros(len(y))
e = np.zeros(len(y))
for i in range(2, len(y)):
    y_hat[i] = y[i-1]+1*(y[i-1]-y[0])/((i+1-1)-1)
    e[i] = y[i] - y_hat[i]
e2 = np.square(e)
# h-step ahead forecast (testing set)
yh = y_test_std.values
yh_hat = np.zeros(len(yh))
eh = np.zeros(len(yh))
for i in range(len(yh)):
    yh_hat[i] = y[-1] + (i+1)*(y[-1]-y[0])/(len(y)-1)
    eh[i] = yh[i] - yh_hat[i]
eh2 = np.square(eh)
# Plot the test set, training set and the h-step forecast in one graph.
plt.plot(y, color='red', marker='o', label='Training set')
plt.plot(np.arange(len(y), len(y)+len(yh)), yh, color='blue', marker='x',
label='Testing set')
plt.plot(np.arange(len(y), len(y)+len(yh)), yh_hat, color='green',
marker='d', label='H-step forecast')
plt.title('11-c. Training set, Testing set, and H-step Forecast with Drift
Method')
```

```
plt.xlabel('Observation')
plt.ylabel('Value')
plt.legend()
plt.show()
# Drift MSE
mse_prediction = np.mean(e2[2:])
mse_forecast = np.mean(eh2)
table = [["MSE of prediction errors (Drift)","MSE of forecast errors
(Drift)"], [mse_prediction, mse_forecast]]
print(table)
rmse_prediction = np.sqrt(mse_prediction)
rmse_forecast = np.sqrt(mse_forecast)
table = [["RMSE of prediction errors (Drift)","RMSE of forecast errors
(Drift)"], [rmse_prediction, rmse_forecast]]
print(table)

# 11-d Simple Exponential Method
# 1-step ahead prediction (training set)
y = y_train_std.values
y_hat = np.zeros(len(y))
e = np.zeros(len(y))
prehat= y[0]
alpha = 0.5
for i in range(1, len(y)):
    y_hat[i] = alpha*y[i-1] + (1-alpha)*prehat
    prehat = y_hat[i]
    e[i] = y[i] - y_hat[i]
e2 = np.square(e)
# h-step ahead forecast (testing set)
yh = y_test_std.values
yh_hat = np.zeros(len(yh))
eh = np.zeros(len(yh))
for i in range(len(yh)):
    yh_hat[i] = alpha*y[-1] + (1-alpha)*y_hat[-1]
    eh[i] = yh[i] - yh_hat[i]
eh2 = np.square(eh)
# Plot the test set, training set and the h-step forecast in one graph.
plt.plot(y, color='red', marker='o', label='Training set')
plt.plot(np.arange(len(y), len(y)+len(yh)), yh, color='blue', marker='x',
label='Testing set')
plt.plot(np.arange(len(y), len(y)+len(yh)), yh_hat, color='green',
marker='d', label='H-step forecast')
plt.title('11-d. Training set, Testing set, and H-step Forecast with SES
Method (alpha=0.5)')
plt.xlabel('Observation')
plt.ylabel('Value')
plt.legend()
plt.show()
## SES 0.5 MSE
mse_prediction = np.mean(e2[2:])
mse_forecast = np.mean(eh2)
table = [["MSE of prediction errors (SES alpha 0.5)","MSE of forecast
errors (SES alpha 0.5)"], [mse_prediction, mse_forecast]]
print(table)
```

```
rmse_prediction = np.sqrt(mse_prediction)
rmse_forecast = np.sqrt(mse_forecast)
table = [["RMSE of prediction errors (SES alpha 0.5)","RMSE of forecast
errors (SES alpha 0.5)"], [rmse_prediction, rmse_forecast]]
print(table)

# compare the SARIMA model performance with the base model predication
# # SARIMA model
# sarima_model = SARIMAX(y_train, order=(1, 1, 1), seasonal_order=(0, 1,
1, 12)).fit()
# sarima_pred = sarima_model.forecast(len(test))
# sarima_rmse = rmse(test, sarima_pred)
#
# # Compare the performance of the base models and SARIMA model
# # print('Naive model RMSE:', naive_rmse)
# # print('Simple average model RMSE:', avg_rmse)
# # print('Drift model RMSE:', drift_rmse)
# # print('Simple exponential smoothing model RMSE:', ses_rmse)
# # print("Holt's exponential smoothing model RMSE:", holt_rmse)
# print('SARIMA model RMSE:', sarima_rmse)

# 12. multiple linear regression
# Perform one-step ahead prediction and compare the performance versus the
test set
# 1-step ahead prediction (training set)
model = sm.OLS(y_train_std, sm.add_constant(X_train_std)).fit()
# 1-step ahead predictions for training set
y_pred_train = model.predict(sm.add_constant(X_train_std))
# 1-step ahead predictions for test set
y_pred_test = model.predict(sm.add_constant(X_test_std))
fig, ax = plt.subplots(figsize=(18, 6))
ax.plot(y_train_std, color='red', marker='o', label='Training set')
ax.plot(y_test_std, color='blue', marker='x', label='Testing set')
ax.plot(y_train_std.index, y_pred_train, color='green', marker='d',
label='1-step prediction')
ax.set_title('12-a-1. Training set, Testing set, and 1-step ahead
prediction with Multiple Linear Regression')
ax.set_xlabel('Time')
ax.set_ylabel('Observation')
ax.legend()
plt.show()

fig, ax = plt.subplots(figsize=(18, 6))
ax.plot(y_train_std, color='red', marker='o', label='Training set')
ax.plot(y_test_std, color='blue', marker='x', label='Testing set')
ax.plot(y_test_std.index, y_pred_test, color='green', marker='d',
label='h-step prediction')
ax.set_title('12-a-2. Training set, Testing set, and h-step ahead
prediction with Multiple Linear Regression')
ax.set_xlabel('Time')
ax.set_ylabel('Value')
ax.legend()
plt.show()
```

```
mse_train = mean_squared_error(y_train_std, y_pred_train)
print(f"12-a. MSE (One-step Ahead Prediction): {mse_train:.4f}")
y_pred_test = model_fit.predict(start=len(y_train_std),
end=len(y_train_std)+len(y_test_std)-1)
mse_test = mean_squared_error(y_test_std, y_pred_test)
print(f"12-a. MSE (Test: Forecasting): {mse_test:.4f}")


# 12-b. F-test, t-test
t_values = model.tvalues
p_values = model.pvalues
result_t = pd.DataFrame({'t-values': t_values, 'p-values': p_values})
print(f'12-b. {result_t}')
f_test = model.f_test(np.identity(len(model.params)))
f_value = f_test.fvalue
p_value = f_test.pvalue
print(f'12-b. f-values: {f_value}, p-values: {p_value}')

# 12-c. AIC, BIC, RMSE, and Adjusted R-squared
print(model.summary())
print("12-c. AIC: ", model.aic)
print("12-c. BIC: ", model.bic)
mse = mean_squared_error(y_train_std, y_pred_train)
rmse = np.sqrt(mse)
print("12-c. RMSE: ", rmse)
print("12-c. Adjusted R-squared: ", model.rsquared_adj)

# 12-d. ACF of residuals.
# prediction errors and plot the ACF of prediction errors
y_train_std1 = y_train_std.to_numpy().flatten()
y_pred_train1 = y_pred_train.to_numpy()
residuals = y_train_std1 - y_pred_train1
def acf_fuc(y, lag):
    y1 = y[lag:]
    y2 = y[:(len(y)-lag)]
    denominator = np.sum((y1 - np.mean(y)) * (y2 - np.mean(y)))
    numerator = (np.var(y)*len(y))
    ans = denominator/numerator
    return ans
acf_list = []
for i in range(21):
    acf_list.append(acf_fuc(residuals, i))
new_acf_list = acf_list[::-1] + acf_list[1:]
lag = 20
x = np.arange(-lag, lag + 1)
y = np.array(new_acf_list)
plt.stem(x, y, markerfmt='ro', basefmt='b')
conf_int = 0.05
upper_CI = 1.96 / math.sqrt(len(residuals))
lower_CI = -1.96 / math.sqrt(len(residuals))
plt.fill_between(x, lower_CI, upper_CI, color='blue', alpha=0.2)
plt.title('12-d. ACF of Prediction Errors')
plt.xlabel('Lags')
plt.ylabel('Magnitude')
```

```
plt.tight_layout()
plt.show()

# 12-e. Q-value
acf = np.zeros(lag+1)
for i in range(len(acf)):
    acf[i] = acf_fuc(residuals[2:],i)
acf2 = np.square(acf)
Q = len(residuals[2:])*np.sum(acf2[1:])
print("12-e. Q value is", Q)

# 12-f. Mean and variance of the residuals.
mean_residuals = np.mean(residuals)
var_residuals = np.var(residuals)
print("12-f. Mean of the residuals is", mean_residuals)
print("12-f. Variance of the residuals is", var_residuals)

# 13. ARMA and ARIMA and SARIMA model order determination
# 13-a.    Preliminary model development procedures and results (ARMA
model order determination).
from statsmodels.tsa.stattools import acf
# ry = acf(y_train_std, nlags=50)
ry = acf(y_train_std, nlags=50)
ry2 = np.concatenate((ry[::-1], ry[1:]))
c = len(ry2)//2
# display GPAC table for the default values of k and j
def compute_gpac_table(data,j_max=15, k_max=15):
    gpac_array = np.zeros((j_max, k_max))
    for j in range(0, j_max):
        for k in range(1, k_max+1):
            if k == 1:
                gpac_array[j, k-1] = ry2[c+j+1] / ry2[c+j]
            else:
                denom = np.zeros((k, k))
                for row_denom in range(k):
                    end = c - j + k - row_denom
                    start = c - j - row_denom
                    row = ry2[start:end]
                    denom[row_denom, :] = row
                numer = denom.copy()
                start = c + j + 1
                end = start + k
                numer[:, -1] = ry2[start:end]
                if np.linalg.det(denom) == 0:
                    pai = np.inf
                else:
                    pai = np.linalg.det(numer)/np.linalg.det(denom)
                gpac_array[j, k-1] = pai
    return gpac_array

gpac_array = compute_gpac_table(ry)
col_labels = [k for k in range(1, gpac_array.shape[1]+1)]
row_labels = [j for j in range(0, gpac_array.shape[0])]
sns.set(font_scale=1)
```

```
fig, ax = plt.subplots(figsize=(20, 20))
sns.heatmap(gpac_array, annot=True, fmt='.2f', cmap='coolwarm',
xticklabels=col_labels, yticklabels=row_labels, ax=ax)
ax.set_title('13-a. Generalized Partial Autocorrelation(GPAC) Table:
ARMA', fontsize=16)
plt.show()
print("Picked orders using GPAC table are (4,0), (4,1), (4,4), (8,0),
(8,1)")

# 13-b.    Should include discussion of the autocorrelation function and
the GPAC.
# Include a plot of the autocorrelation function and the GPAC table within
this section).
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
def ACF_PACF_Plot(y,lags):
 acf = sm.tsa.stattools.acf(y, nlags=lags)
 pacf = sm.tsa.stattools.pacf(y, nlags=lags)
 fig = plt.figure()
 plt.subplot(211)
 plt.title(f'13-b. ACF/PACF of the data)')
 plot_acf(y, ax=plt.gca(), lags=lags)
 plt.subplot(212)
 plot_pacf(y, ax=plt.gca(), lags=lags)
 fig.tight_layout(pad=3)
 plt.show()
ACF_PACF_Plot(y_train_std,50)

# 13-c. Include the GPAC table in your report and highlight the estimated
order.

# 14. Estimate ARMA model parameters using the Levenberg Marquardt
algorithm.
# Display the parameter estimates, the standard deviation of the parameter
estimates and confidence intervals.

orders = [(4, 0), (4, 4), (8, 0), (8, 4)]
mse_results = []
for order in orders:
    ar_order = order[0]
    ma_order = order[1]
    model = sm.tsa.ARIMA(y_train_std, order=(ar_order, 0, ma_order)).fit()
    coefficients = model.params.round(3)
    y_train_pred = model.predict()
    mse_train = mean_squared_error(y_train_std, y_train_pred)
    y_test_pred = model.predict(start=len(y_train_std),
end=len(y_train_std) + len(y_test_std)-1)
    mse_test = mean_squared_error(y_test_std, y_test_pred)
    mse_results.append((order, mse_train, mse_test))
    print(f"\n14. Estimated parameters for ARMA({ar_order},{ma_order}):")
    print(f"\n14. Estimated parameters for ARMA({ar_order},{ma_order}):")
    print("14. Estimated AR coefficients:\n", coefficients[1:ar_order +
1])
    print("14. Estimated MA coefficients:\n", coefficients[ar_order + 1:-
1])
```

```
    n = ar_order + ma_order
    CI = np.zeros((n, 2))
    std_err = np.zeros(n)
    for i in range(n):
        std_err[i] = np.sqrt(model.cov_params().iloc[i, i])
        CI[i, 0] = coefficients[i] - 2 * std_err[i]
        CI[i, 1] = coefficients[i] + 2 * std_err[i]
        print(f"14-{i}. {i}th standard deviation: {round(std_err[i], 3)}")
        print(f"14-{i}. CI of {i}th coefficient {coefficients[i]}:
[{round(CI[i, 0], 3)}, {round(CI[i, 1], 3)}]")
    print(model.summary())

print("14 & 17. MSE Results of ARMA:")
for result in mse_results:
    print(f"ARMA{result[0]}: Train MSE (Prediction) = {result[1]}, Test
MSE (Forecasting) = {result[2]}\n\n")
print("\n\n 14. Conclusion: ARMA(4,4) works well! ARMA(4,0) will be also
considered in SARIMA.")

# 15. Diagnostic Analysis
# 15-a. Diagnostic tests (confidence intervals, zero/pole cancellation,
chi-square test).
ar_order = 4
ma_order = 4
model = sm.tsa.ARIMA(y_train_std, order=(ar_order, 0, ma_order)).fit()
coefficients = model.params.round(3)
n = ar_order + ma_order
CI = np.zeros((n, 2))
std_err = np.zeros(n)
for i in range(n):
    std_err[i] = np.sqrt(model.cov_params().iloc[i, i])
    CI[i, 0] = coefficients[i] - 2 * std_err[i]
    CI[i, 1] = coefficients[i] + 2 * std_err[i]
    print(f"15-a. {i}th standard deviation: {round(std_err[i], 3)}")
    print(f"15-a. CI of {i}th coefficient {coefficients[i]}: [{round(CI[i,
0], 3)}, {round(CI[i, 1], 3)}]")
    if CI[i, 0] > 0 or CI[i, 1] < 0:
        print(f"15-a. {i}th coefficient is statistically important.")
    else:
        print(f"15-a. {i}th coefficient is statistically not important.")
# check for zero/pole cancellation
poles = np.roots(np.append(1, -model.arparams))
zeros = np.roots(np.append(1, model.maparams))
print("15-a. Poles - AR roots: ", poles)
print("15-a. Zeros - MA roots: ", zeros)
# chi-square test for residuals
from statsmodels.stats.diagnostic import acorr_ljungbox
residuals = model.resid
lags = min(10, len(residuals)-1)
lbvalue, pvalue = acorr_ljungbox(residuals, lags=lags, boxpierce=False)
# print("15-a. Ljung-Box (L%d) (Q): %.2f" % (lags, lbvalue[-1]))

# 15-b. the estimated variance of the error and the estimated covariance
of the estimated parameters.
```

```
error_var = np.round(model.scale, 3)
print(f"15-b. Estimated variance of the error for
ARMA({ar_order},{ma_order}): {error_var}")
# Estimated covariance of the estimated parameters
param_cov = np.round(model.cov_params().to_numpy(), 3)
print(f"15-b. Estimated covariance of the estimated parameters for
ARMA({ar_order},{ma_order}): \n {param_cov}")

# 15-c.
print(model.summary())
print("the constant of coefficent is closed to zero and CI includes zero,
which means this is unbiased.")

# 15-d.
var_residuals = np.var(residuals)
print("15-d. Variance of the residual errors is", var_residuals)
y_test_std = y_test_std.to_numpy().flatten()
y_pred_test = y_pred_test.to_numpy()
forecast_errors = y_test_std - y_pred_test
var_forecast_errors = np.var(forecast_errors)
print("15-d. Variance of the forecast errors is", var_forecast_errors)

# # 15-e. ARIMA or SARIMA model may better represents the dataset
# # 15-e. ex1.
# model = sm.tsa.SARIMAX(y_train_std, order=(0,0,0),
seasonal_order=(1,0,0,4))
# model_fit = model.fit()
# y_model_hat = model_fit.predict(start=1, end=len(y_train_std)-1)
# plt.figure(figsize=(10, 6))
# plt.plot(y_train_std[1:], label='y')
# plt.plot(y_model_hat, label='1-step')
# plt.xlabel('Sample')
# plt.ylabel('Value')
# plt.title('15-e. EX1. y and 1-Step Ahead Prediction with SARIMA')
# plt.legend()
# plt.show()
# print(model_fit.summary())
#
# y_pred_train = y_model_hat.to_numpy()
# residuals = y_train_std1[1:] - y_pred_train
# acf_list = []
# lag = 60
# for i in range(lag+1):
#     acf_list.append(acf_fuc(residuals, i))
# new_acf_list = acf_list[::-1] + acf_list[1:]
# x = np.arange(-lag, lag + 1)
# y = np.array(new_acf_list)
# plt.stem(x, y, markerfmt='ro', basefmt='b')
# conf_int = 0.05
# upper_CI = 1.96 / math.sqrt(len(residuals))
# lower_CI = -1.96 / math.sqrt(len(residuals))
# plt.fill_between(x, lower_CI, upper_CI, color='blue', alpha=0.2)
# plt.title('15-e. EX1. ACF of Prediction Errors')
# plt.xlabel('Lags')
```

```
# plt.ylabel('Magnitude')
# plt.tight_layout()
# plt.show()
#
# mse_train = mean_squared_error(y_train_std[1:], y_model_hat)
# print(f"15-a. EX5-a. MSE of the SARIMA model (Training: Prediction):
{mse_train:.4f}")
# y_pred_test = model_fit.predict(start=len(y_train_std),
end=len(y_train_std)+len(y_test_std)-1)
# mse_test = mean_squared_error(y_test_std, y_pred_test)
# print(f"15-e. EX5-b. MSE of the SARIMA model (Test: Forecasting):
{mse_test:.4f}")
#
# # 15-e. ex2.
# model = sm.tsa.SARIMAX(y_train_std, order=(0,0,0),
seasonal_order=(1,0,1,4))
# model_fit = model.fit()
# y_model_hat = model_fit.predict(start=1, end=len(y_train_std)-1)
# plt.figure(figsize=(10, 6))
# plt.plot(y_train_std[1:], label='y')
# plt.plot(y_model_hat, label='1-step')
# plt.xlabel('Sample')
# plt.ylabel('Value')
# plt.title('15-e. EX2. y and 1-Step Ahead Prediction with SARIMA')
# plt.legend()
# plt.show()
# print(model_fit.summary())
#
# y_pred_train = y_model_hat.to_numpy()
# residuals = y_train_std1[1:] - y_pred_train
# acf_list = []
# lag = 60
# for i in range(lag+1):
#     acf_list.append(acf_fuc(residuals, i))
# new_acf_list = acf_list[::-1] + acf_list[1:]
# x = np.arange(-lag, lag + 1)
# y = np.array(new_acf_list)
# plt.stem(x, y, markerfmt='ro', basefmt='b')
# conf_int = 0.05
# upper_CI = 1.96 / math.sqrt(len(residuals))
# lower_CI = -1.96 / math.sqrt(len(residuals))
# plt.fill_between(x, lower_CI, upper_CI, color='blue', alpha=0.2)
# plt.title('15-e. EX2. ACF of Prediction Errors')
# plt.xlabel('Lags')
# plt.ylabel('Magnitude')
# plt.tight_layout()
# plt.show()
#
# mse_train = mean_squared_error(y_train_std[1:], y_model_hat)
# print(f"15-e. EX5-a. MSE of the SARIMA model (Training: Prediction):
{mse_train:.4f}")
# y_pred_test = model_fit.predict(start=len(y_train_std),
end=len(y_train_std)+len(y_test_std)-1)
# mse_test = mean_squared_error(y_test_std, y_pred_test)
```

```
# print(f"15-e. EX5-b. MSE of the SARIMA model (Test: Forecasting):
{mse_test:.4f}")
#
# # 15-e. ex3.
# model = sm.tsa.SARIMAX(y_train_std, order=(4,0,0),
seasonal_order=(0,0,0,0))
# model_fit = model.fit()
# y_model_hat = model_fit.predict(start=1, end=len(y_train_std)-1)
# plt.figure(figsize=(10, 6))
# plt.plot(y_train_std[1:], label='y')
# plt.plot(y_model_hat, label='1-step')
# plt.xlabel('Sample')
# plt.ylabel('Value')
# plt.title('15-e. EX3. y and 1-Step Ahead Prediction with SARIMA')
# plt.legend()
# plt.show()
# print(model_fit.summary())
#
# y_pred_train = y_model_hat.to_numpy()
# residuals = y_train_std1[1:] - y_pred_train
# acf_list = []
# lag = 60
# for i in range(lag+1):
#     acf_list.append(acf_fuc(residuals, i))
# new_acf_list = acf_list[::-1] + acf_list[1:]
# x = np.arange(-lag, lag + 1)
# y = np.array(new_acf_list)
# plt.stem(x, y, markerfmt='ro', basefmt='b')
# conf_int = 0.05
# upper_CI = 1.96 / math.sqrt(len(residuals))
# lower_CI = -1.96 / math.sqrt(len(residuals))
# plt.fill_between(x, lower_CI, upper_CI, color='blue', alpha=0.2)
# plt.title('15-e. EX3. ACF of Prediction Errors')
# plt.xlabel('Lags')
# plt.ylabel('Magnitude')
# plt.tight_layout()
# plt.show()
#
# mse_train = mean_squared_error(y_train_std[1:], y_model_hat)
# print(f"15-e. EX5-a. MSE of the SARIMA model (Training: Prediction):
{mse_train:.4f}")
# y_pred_test = model_fit.predict(start=len(y_train_std),
end=len(y_train_std)+len(y_test_std)-1)
# mse_test = mean_squared_error(y_test_std, y_pred_test)
# print(f"15-e. EX5-b. MSE of the SARIMA model (Test: Forecasting):
{mse_test:.4f}")
#
# 15-e. ex4.
model = sm.tsa.SARIMAX(y_train_std, order=(4,0,4),
seasonal_order=(0,0,0,0))
model_fit = model.fit()
y_model_hat = model_fit.predict(start=1, end=len(y_train_std)-1)
plt.figure(figsize=(10, 6))
plt.plot(y_train_std[1:], label='y')
```

```
plt.plot(y_model_hat, label='1-step')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('15-e. EX4. y and 1-Step Ahead Prediction with SARIMA')
plt.legend()
plt.show()
print(model_fit.summary())

y_pred_train = y_model_hat.to_numpy()
residuals = y_train_std1[1:] - y_pred_train
acf_list = []
lag = 60
for i in range(lag+1):
    acf_list.append(acf_fuc(residuals, i))
new_acf_list = acf_list[::-1] + acf_list[1:]
x = np.arange(-lag, lag + 1)
y = np.array(new_acf_list)
plt.stem(x, y, markerfmt='ro', basefmt='b')
conf_int = 0.05
upper_CI = 1.96 / math.sqrt(len(residuals))
lower_CI = -1.96 / math.sqrt(len(residuals))
plt.fill_between(x, lower_CI, upper_CI, color='blue', alpha=0.2)
plt.title('15-e. EX4. ACF of Prediction Errors')
plt.xlabel('Lags')
plt.ylabel('Magnitude')
plt.tight_layout()
plt.show()

mse_train = mean_squared_error(y_train_std[1:], y_model_hat)
print(f"15-e. EX4-a. MSE of the SARIMA model (Training: Prediction):
{mse_train:.4f}")
y_pred_test = model_fit.predict(start=len(y_train_std),
end=len(y_train_std)+len(y_test_std)-1)
mse_test = mean_squared_error(y_test_std, y_pred_test)
print(f"15-e. EX4-b. MSE of the SARIMA model (Test: Forecasting):
{mse_test:.4f}")

# 15-e. ex5.
model = sm.tsa.SARIMAX(y_train_std, order=(4,0,4),
seasonal_order=(1,0,0,28))
model_fit = model.fit()
y_model_hat = model_fit.predict(start=1, end=len(y_train_std)-1)
plt.figure(figsize=(10, 6))
plt.plot(y_train_std[1:], label='y')
plt.plot(y_model_hat, label='1-step')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('15-e. EX5. y and 1-Step Ahead Prediction with SARIMA
(periodicity=28)')
plt.legend()
plt.show()
print(model_fit.summary())

y_pred_train = y_model_hat.to_numpy()
```

```
residuals = y_train_std1[1:] - y_pred_train
acf_list = []
lag = 60
for i in range(lag+1):
    acf_list.append(acf_fuc(residuals, i))
new_acf_list = acf_list[::-1] + acf_list[1:]
x = np.arange(-lag, lag + 1)
y = np.array(new_acf_list)
plt.stem(x, y, markerfmt='ro', basefmt='b')
conf_int = 0.05
upper_CI = 1.96 / math.sqrt(len(residuals))
lower_CI = -1.96 / math.sqrt(len(residuals))
plt.fill_between(x, lower_CI, upper_CI, color='blue', alpha=0.2)
plt.title('15-e. EX5. ACF of Prediction Errors')
plt.xlabel('Lags')
plt.ylabel('Magnitude')
plt.tight_layout()
plt.show()

mse_train = mean_squared_error(y_train_std[1:], y_model_hat)
print(f"15-e. EX5-a. MSE of the SARIMA model (Training: Prediction):
{mse_train:.4f}")
y_pred_test = model_fit.predict(start=len(y_train_std),
end=len(y_train_std)+len(y_test_std)-1)
mse_test = mean_squared_error(y_test_std, y_pred_test)
print(f"15-e. EX5-b. MSE of the SARIMA model (Test: Forecasting):
{mse_test:.4f}")
#
# # 15-e. ex6.
# model = sm.tsa.SARIMAX(y_train_std, order=(4,0,4),
seasonal_order=(1,1,1,28))
# model_fit = model.fit()
# y_model_hat = model_fit.predict(start=1, end=len(y_train_std)-1)
# plt.figure(figsize=(10, 6))
# plt.plot(y_train_std[1:], label='y')
# plt.plot(y_model_hat, label='1-step')
# plt.xlabel('Sample')
# plt.ylabel('Value')
# plt.title('15-e. EX6. y and 1-Step Ahead Prediction with SARIMA
(periodicity=28)')
# plt.legend()
# plt.show()
# print(model_fit.summary())
#
# y_pred_train = y_model_hat.to_numpy()
# residuals = y_train_std1[1:] - y_pred_train
# acf_list = []
# lag = 60
# for i in range(lag+1):
#     acf_list.append(acf_fuc(residuals, i))
# new_acf_list = acf_list[::-1] + acf_list[1:]
# x = np.arange(-lag, lag + 1)
# y = np.array(new_acf_list)
# plt.stem(x, y, markerfmt='ro', basefmt='b')
```

```
# conf_int = 0.05
# upper_CI = 1.96 / math.sqrt(len(residuals))
# lower_CI = -1.96 / math.sqrt(len(residuals))
# plt.fill_between(x, lower_CI, upper_CI, color='blue', alpha=0.2)
# plt.title('15-e. EX6. ACF of Prediction Errors')
# plt.xlabel('Lags')
# plt.ylabel('Magnitude')
# plt.tight_layout()
# plt.show()
#
# mse_train = mean_squared_error(y_train_std[1:], y_model_hat)
# print(f"15-e. EX6-a. MSE of the SARIMA model (Training: Prediction):
{mse_train:.4f}")
# y_pred_test = model_fit.predict(start=len(y_train_std),
end=len(y_train_std)+len(y_test_std)-1)
# mse_test = mean_squared_error(y_test_std, y_pred_test)
# print(f"15-e. EX6-b. MSE of the SARIMA model (Test: Forecasting):
{mse_test:.4f}")

# 17. Final Model selection
# 18. Forecast function & 19. h-step ahead Predictions
model = sm.tsa.SARIMAX(y_train_std, order=(4,0,4),
seasonal_order=(1,0,0,28))
model_fit = model.fit()
y_pred_train = model_fit.predict(start=1, end=len(y_train_std)-1)
plt.figure(figsize=(10, 6))
plt.plot(y_train_std[1:], color='red', label='y train')
plt.plot(y_pred_train, color='green', label='1-step')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('19. y and 1-Step Ahead Prediction with the Final Model')
plt.legend()
plt.show()
print(model_fit.summary())

y_pred_test = model_fit.predict(start=1, end=len(y_test_std)-1)
plt.figure(figsize=(10, 6))
plt.plot(y_test_std[1:], color='blue', label='y test')
plt.plot(y_pred_test.values.flatten(), color='green', label='h-step')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('19. y test and h-Step Ahead Prediction with the Final Model')
plt.legend()
plt.show()
print(model_fit.summary())

plt.figure(figsize=(18, 6))
plt.plot(y_train_std, color='red', label='Train')
plt.plot(np.arange(len(y_train_std),
len(y_train_std)+len(y_test_std[1:])),y_test_std[1:], color='blue',
label='Test')
plt.plot(np.arange(len(y_train_std),
len(y_train_std)+len(y_test_std[1:])),y_pred_test.values.flatten(),
color='green', label='H-step')
```

```
plt.xlabel('Sample')
plt.ylabel('Value')
plt.title('19. h-Step Ahead Prediction with the Final Model')
plt.legend()
plt.show()
print(model_fit.summary())

# 16. Deep Learning Model
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
from sklearn.metrics import mean_squared_error

X_train_std = X_train_std.values.reshape(X_train_std.shape[0],
X_train_std.shape[1], 1)
model = Sequential()
model.add(LSTM(50, input_shape=(X_train_std.shape[1], 1)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train_std, y_train_std, epochs=100, batch_size=32, verbose=0)

# define function to make h-step ahead predictions
def lstm_forecast(model, X_test_std, h):
    # create empty array to hold forecasted values
    forecast = []
    # initial input to the model is the last sequence in the training data
    last_sequence = X_test_std[0]
    for i in range(h):
        # make one-step prediction
        y_pred = model.predict(last_sequence.reshape(1,
X_test_std.shape[1], 1))
        # append predicted value to forecast array
        forecast.append(y_pred[0,0])
        # update last sequence with predicted value
        last_sequence = np.append(last_sequence[1:], y_pred)
    return forecast

# h-step predictions on test set
h = len(y_test_std)
X_test_std = X_test_std.values.reshape(X_test_std.shape[0],
X_test_std.shape[1], 1)
forecast = lstm_forecast(model, X_test_std, h)
mse_test = mean_squared_error(y_test_std, forecast)
print("16. Mean squared error (MSE) of LSTM (Test: Forecasting):
{:.4f}".format(mse_test))
```