Temperature Forecasting

DATS 6313: Time Series Analysis & Modeling

Fall 2023

Instructor: Dr. Reza Jafari

Final Term Project Report

Jadhav, Shubham
G30570862
Department of Computer Science
School of Engineering
George Washington University

GitHub Repository: https://github.com/shubhjadhav/Temperature-Forecasting

December 11, 2023

Abstract

This project presents a study leveraging an occupancy detection dataset derived from environmental sensor readings in an office building. The objective is to predict room temperature based on variables such as occupancy, humidity, light, CO2, and humidity ratio. The dataset, capturing minute-by-minute measurements over a week. The primary focus is on developing an accurate predictive model for temperature, crucial for applications in building energy management, occupancy detection, and indoor environmental quality control. Through a comparative analysis of different algorithms, with an emphasis on logistic regression and time series methodologies, the study aims to identify the most effective model for temperature prediction. Results highlight the superiority of the ARIMA model among the experimental models, showcasing its proficiency in capturing temporal dynamics inherent in the dataset. This finding emphasizes the potential of ARIMA in accurately predicting temperature levels, providing valuable insights for optimizing energy usage and ensuring occupant comfort in smart building applications. In conclusion, the success of the ARIMA model establishes a promising avenue for further exploration in smart building technologies and environmental monitoring systems.

Keywords—Models, Train, Test, Forecast, Prediction, Error

Contents

List of Figures

List of Tables

1 Introduction

Time series analysis is a method used to analyze data points arranged over time. It serves as a fundamental concept for understanding data points collected at successive time intervals, facilitating the identification of temporal patterns, trends, and interrelationships. Time series algorithms play a vital role in data mining, enabling data miners to extract meaningful insights from sequential data. This type of data, commonly found in fields like finance, economics, engineering, and environmental science, involves observations made in chronological order. In this report, we'll analyze the Occupancy Detection dataset from the UCI Machine Learning Repository. The dataset, collected from an office room with sensors tracking occupancy, temperature, humidity, and CO2 levels, will be used to predict temperature levels in the room based on other variables.

The report starts with exploring the dataset using visuals and summary stats. We then use different techniques for time series modeling, including the Naive Method, Simple Exponential Smoothing, Holt-Winter Model, ARIMA, and the drift method. Model performance is evaluated using metrics like Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and AIC and BIC values.

The culmination of the report includes the presentation of findings, drawing conclusions, and a thoughtful discussion on the strengths and limitations inherent in each model. By elucidating the intricacies of time series analysis and modeling, the report underscores the vital role of time series data in addressing real-world challenges and applications.

2 Data Exploration and Preprocessing

2.1 Data Description

The Occupancy Detection dataset is a time series dataset that contains data collected from an office room over a period of 7 days. The dataset was collected by using several sensors including temperature, humidity, CO2 levels, and light intensity, among others. The goal of the dataset is to predict room temperature. The dataset contains 9752 observations and 7 features including date and time, temperature, humidity, light intensity, CO2 levels, humidity ratio, and occupancy.

Columns	Description
Date	time year-month-day hour:minute:second
Temperature	Temperature in Celsius
Humidity	Relative Humidity %
Light	Light in Lux
CO2	Carbon-Di-Oxide in ppm
Humidity Ratio	Derived quantity from temperature
Occupancy	occupied status

Table 1: Dataset Variables

date	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
2015-02-11 14:48:00	21.76	31.1333	437.333	1029.67	0.00502101	1
2015-02-11 14:49:00	21.79	31	437.333	1000	0.00500858	1
2015-02-11 14:50:00	21.7675	31.1225	434	1003.75	0.00502157	1
2015-02-11 14:51:00	21.7675	31.1225	439	1009.5	0.00502157	1
2015-02-11 14:52:00	21.79	31.1333	437.333	1005.67	0.0050303	1

Table 2: Sample data

2.2 Data Pre-processing

The dataset is clean and does not any any missing values or NAN values. Therefore, there is no requirement for data cleaning procedure. There are no categorical variables.

	index	0
0	Temperature	0
1	Humidity	0
2	Light	0
3	CO2	0
4	HumidityRatio	0
5	Occupancy	0

Table 3: Null value statistics

	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
count	9752	9752	9752	9752	9752	9752
mean	21.0018	29.8919	123.068	753.225	0.00458878	0.210111
std	1.02069	3.95284	208.221	297.096	0.000530985	0.407408
min	19.5	21.865	0	484.667	0.00327476	0
25%	20.29	26.6421	0	542.312	0.00419631	0
50%	20.79	30.2	0	639	0.00459331	0
75%	21.5333	32.7	208.25	831.125	0.00499797	0
max	24.39	39.5	1581	2076.5	0.00576861	1

Table 4: Descriptive statistics

2.3 Time Series Plot of Dependent Variable

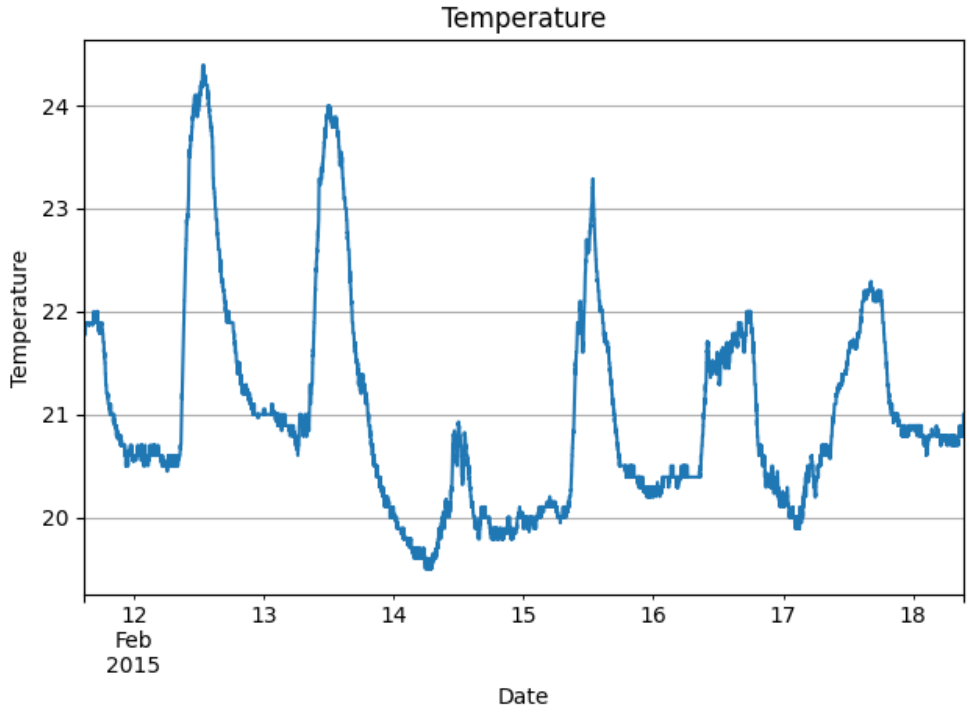

Figure 1: Dependent Variable against Time

Observation: We can observe from the plot that there is not stationary for the dependent/-target variable. i.e., Temperature. The variable exhibits seasonality as well; that is, it starts off cold at the beginning of the day, gradually rises during the day, and then drops again at the beginning of the following day. We can observe seasonality but cannot identity any conclusive trend. In next step, we will check for auto correlation and partial auto correlation for the target variable.

2.4 ACF/PACF of the dependent variable

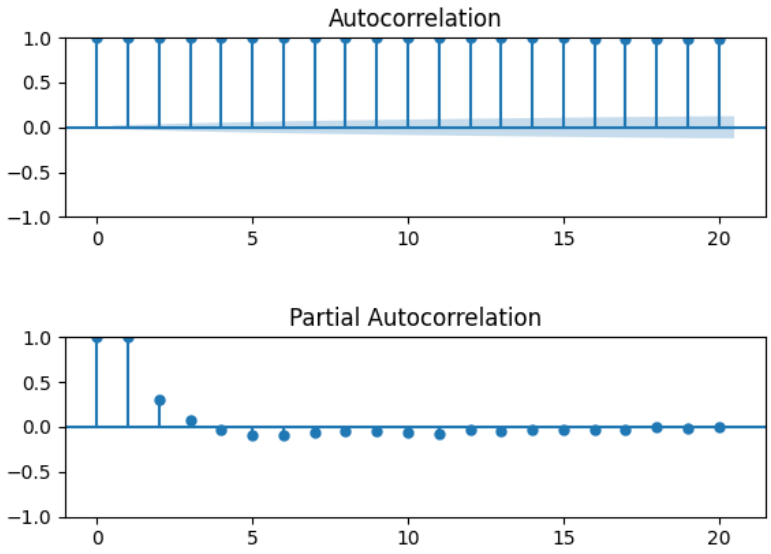

Figure 2: ACF/PACF of the dependent variable

Observation: We can observe from the ACF/PACF plot that there is not stationary for the dependent/target variable. i.e., Temperature.

2.5 Correlation Matrix

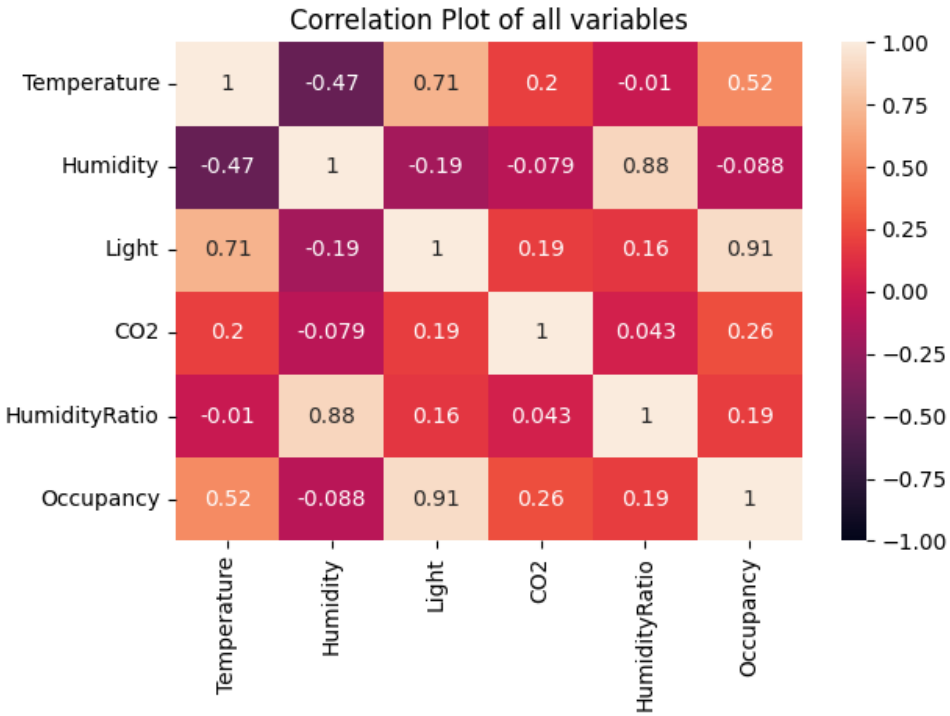

Figure 3: Correlation Matrix

Observation: The correlation matrix shows that there is a positive association between temperature and changeable light. In contrast, there is a negative correlation between the temperature and the variable humidity. Whereas, HumidityRatio and CO2 are not correlated to temperature.

2.6 Split the dataset

We will be split the dataset into train set with 80% data and test set with 20% data.

```
X_train, X_test, y_train, y_test = train_test_split(
    temp_df.drop(columns=[target]),
    temp_df[target],
    shuffle=False,
    test_size=0.2
)
```

Figure 4: Code snippet for data split

```
Training set size: 7801 rows and 6 columns
Testing set size: 1951 rows and 6 columns
```

Figure 5: Data split stats

3 Stationarity

Stationarity describes how a time series of changes will remain the same. In mathematical terms, a time series is stationary when its statistical properties are independent of time, i.e., it has constant mean, constant variance, and covariance is independent of time.

3.1 Raw Dataset

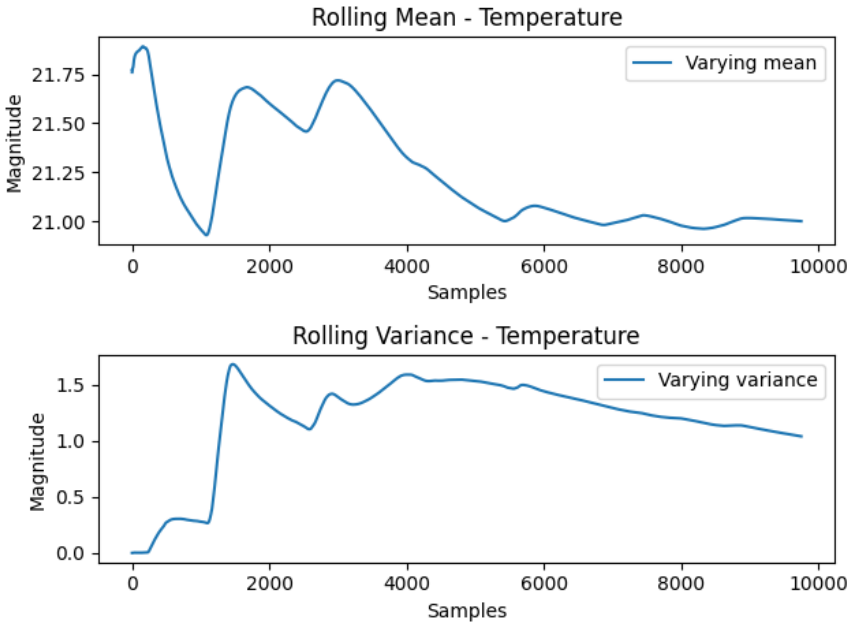

Figure 6: Rolling Mean and Variable of raw data

Observation: We can observe from the plot that rolling mean and variance is not stabilizing therefore the data is not stationary. Therefore in the following step we will perform differencing of the data to make the dataset stationary.

3.2 First Order Non-seasonal Differencing

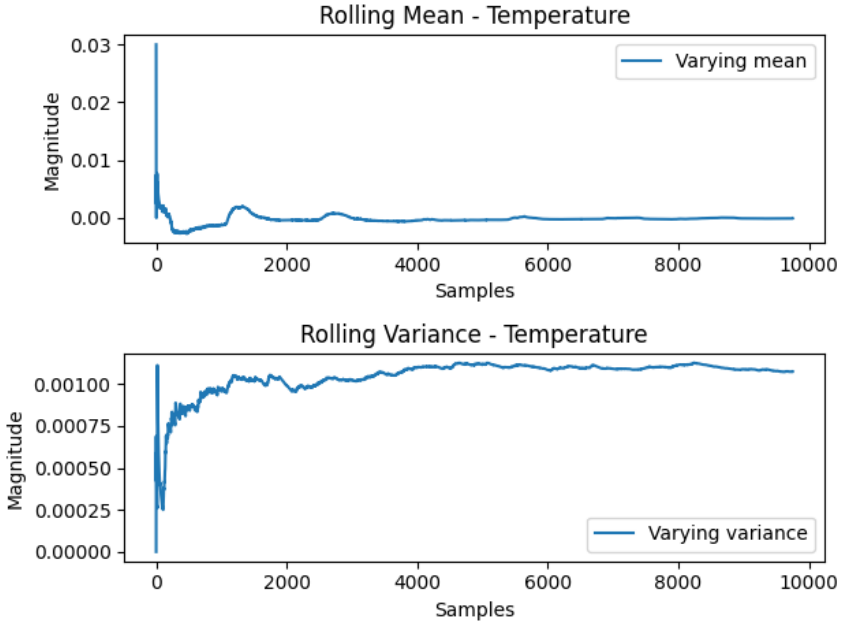

Figure 7: Rolling Mean and Variable of 1st order differenced data

Observation: We can observe from the plot that rolling mean and variance is stabilizing with time therefore we can say that the data is stationary. To conclude this observation we will plot ACF/PACF and perform ADF and KPSS statistical test.

3.3 ACF/PCF Plot

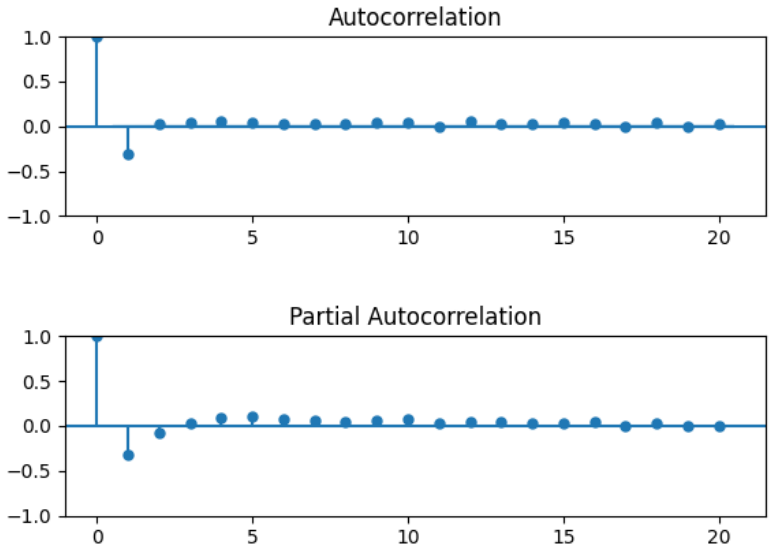

Figure 8: ACF/PACF plot after differencing

Observation: We can observe from the ACF/PACF plot that differenced data is stationary.

3.4 ACF/PCF Plot

```
ADF Statistic: -9.684629
p-value: 0.000000
Critical Values:
    1%: -3.431023
    5%: -2.861838
    10%: -2.566928
None
```

Figure 9: ADF Statistics Test

```
Results of KPSS Test:
Test Statistic            0.082141
p-value                   0.100000
Lags Used                32.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
dtype: float64
```

Figure 10: KPSS Statistics Test

Observation: From the ADF and KPSS test also we can see the p-value of ADF test is less than the 0.05 whereas the p-value of KPSS test is more than 0.05, so we can conclude that data is stationary.

3.5 Time Series Plot of Difference data

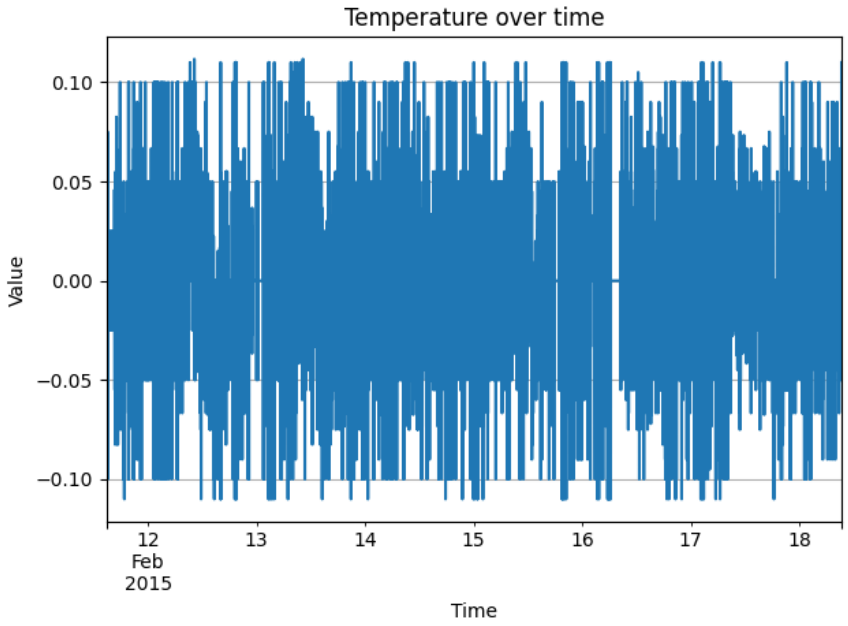

Figure 11: Plot of difference data against Time

4 Time series Decomposition

There are two ways to decompose a time series:

1. **Multiplicative:** The multiplicative way of decomposition is most appropriate when the series' fluctuations are on a relative scale.

$$y(t) = T(t) * S(t) * R(t)$$

2. **Additive:** The additive way of decomposition is most appropriate when the size of the series' variations are on a consistent numerical scale.

$$y(t) = T(t) + S(t) + R(t)$$

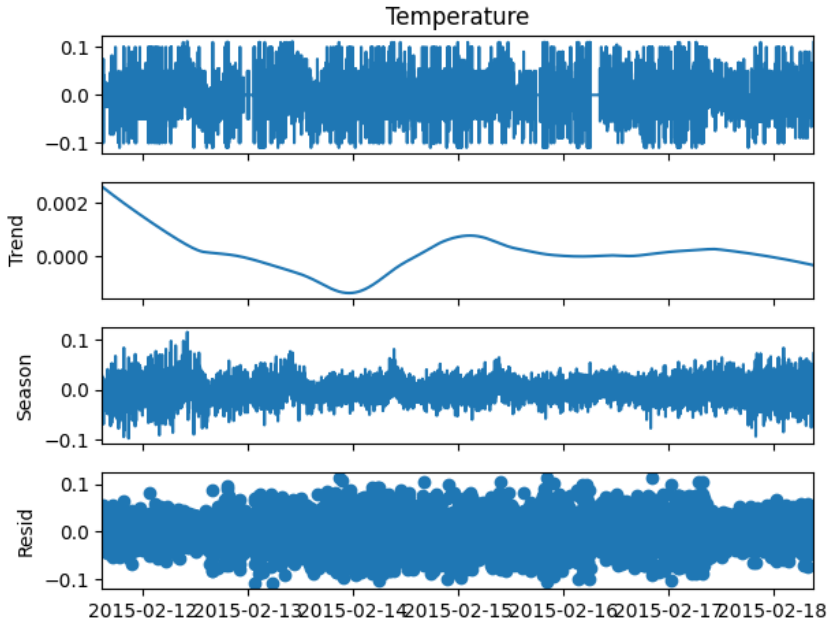

Figure 12: STL decomposition

```
The strength of trend for this data set is 0.0012 or 0.12%

The strength of seasonality for this data set is 0.4107 or 41.07%
```

Figure 13: Strength of trend and seasonality

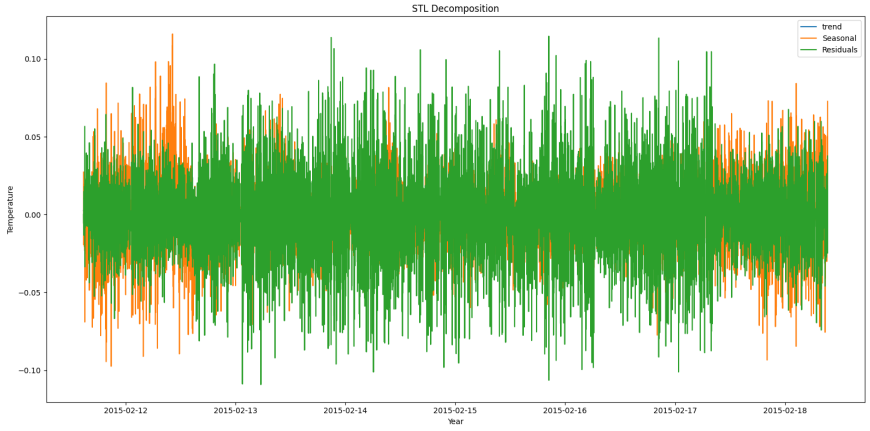

Figure 14: Plot of decomposed difference data against Time

5 Holt-Winter's Method

The Holt-Winters model, also known as triple exponential smoothing, is a forecasting technique designed for time series data that displays a seasonal pattern. This model utilizes

exponential smoothing on different components of the time series, incorporating a seasonal aspect to predict future values. By assigning more weight to recent observations and less weight to older ones, the model effectively captures trends and seasonality. This method proves especially valuable for time series data featuring both trend and seasonality, such as sales data or stock prices. Additionally, it can be adapted to handle time series data with only a trend, only seasonality, or neither.

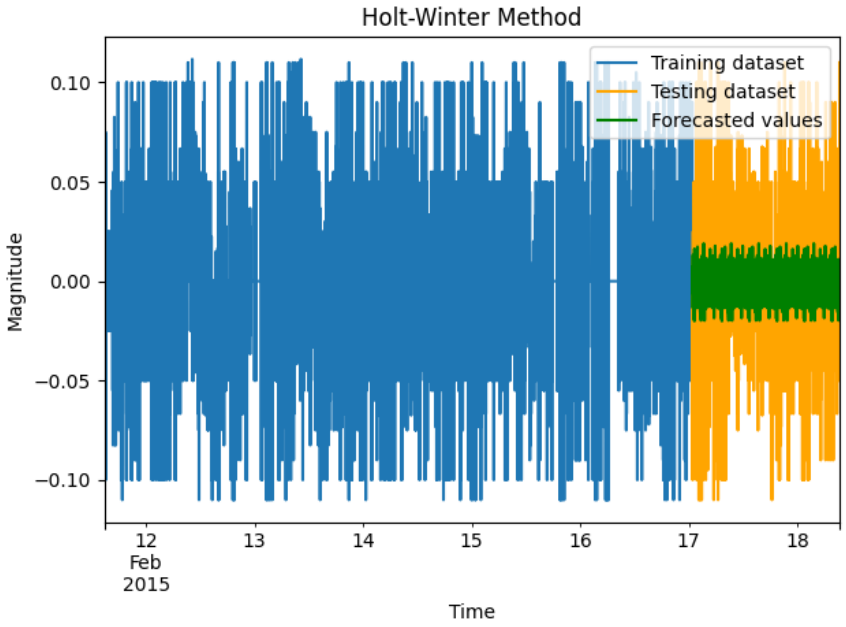

Figure 15: Train/Test/Forecast Plot from Holt-Winter Method

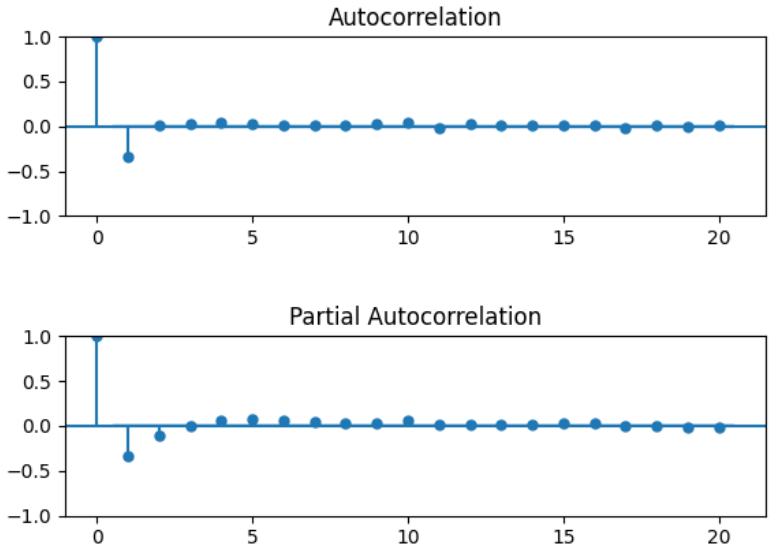

Figure 16: ACF/PCF Plot of Residuals

	Model	Q-Value	Critical-Value	White-Residual	m_res	mse_res	var_res	m_pred	mse_pred	var_pred
0	Holt-Winter	957.818	36.1909	No	-0	0.001	0.001	0.0016	0.001	0.001

Table 5: Statistical values of Holt-Winter Method

Table Description

- M_RES = Mean of Residual Errors
- MSE_RES = Mean Squared of Residual Errors
- VAR_RES = Variance of Residual Errors
- M_PRED = Mean of Forecast Errors
- MSE_PRED = Mean Squared of Forecast Errors
- VAR_PRED = Variance of Forecast Errors

6 Base Models

6.1 Average Method

The average method is a straightforward time series forecasting approach that entails computing the average of past observations to predict future values. This method operates on the assumption that upcoming values will resemble the historical average. It is characterized by simplicity, requiring no complex mathematical calculations for implementation. However, the average method is suitable only for time series data that is relatively stable and lacks significant trends or seasonal patterns. It may not be effective in handling data with pronounced fluctuations or distinct patterns. Additionally, the method can be sensitive to outliers or extreme values in the dataset.

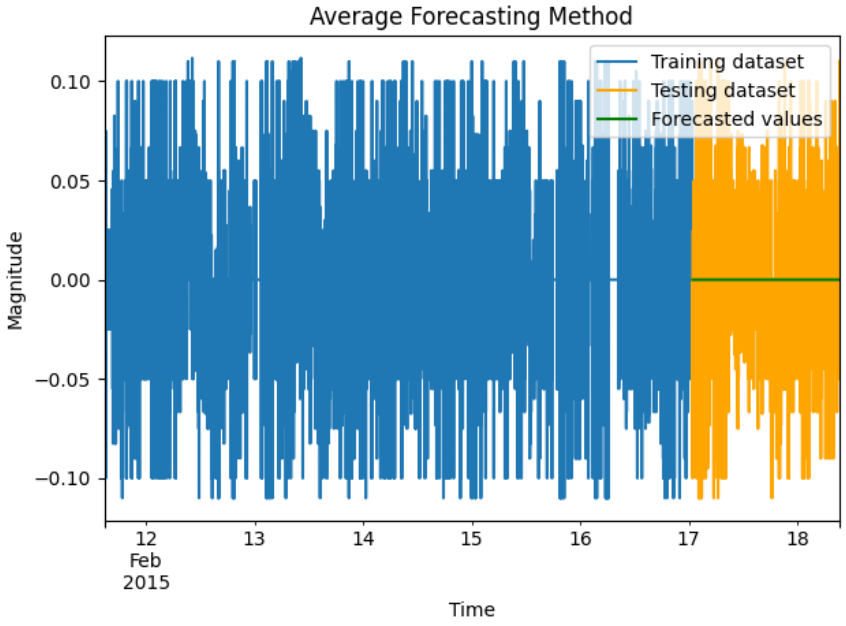

Figure 17: Train/Test/Forecast Plot from Average Method

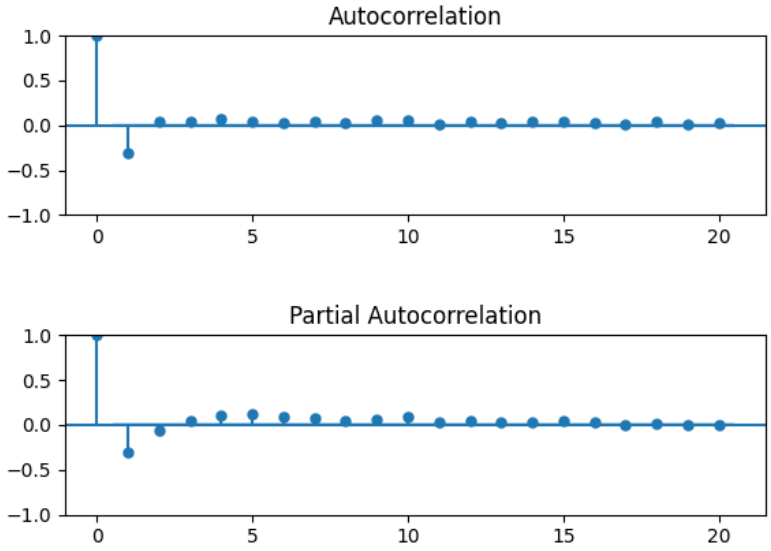

Figure 18: ACF/PCF Plot of Residuals of Average Method

6.2 Naive Method

The Naive Method is a basic forecasting approach that assumes the next value in a time series will be identical to the most recent observed value. This method is often employed as a benchmark to evaluate the performance of more complex forecasting models. Despite its simplicity, the Naive Method can be highly accurate for certain time series, particularly in short-term forecasting scenarios.

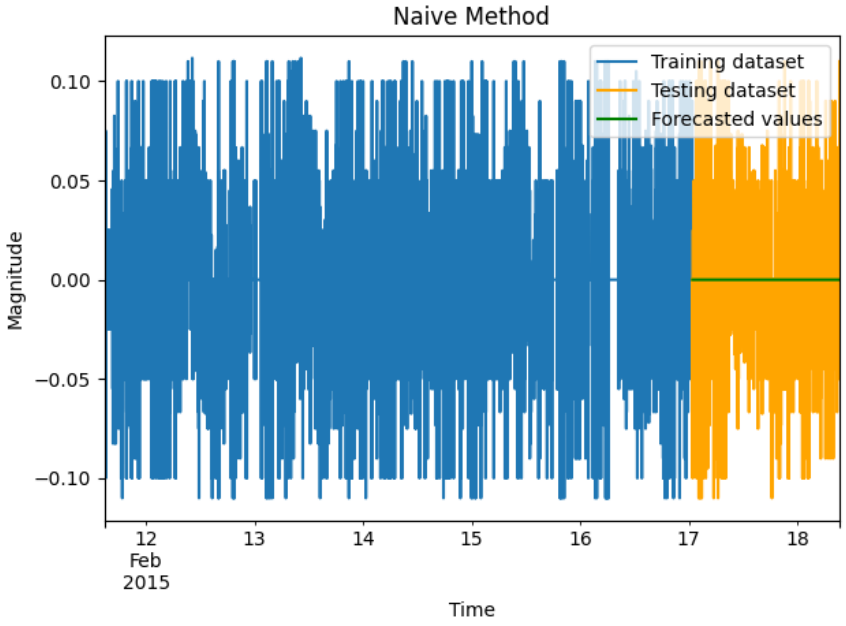

Figure 19: Train/Test/Forecast Plot from Naive Method

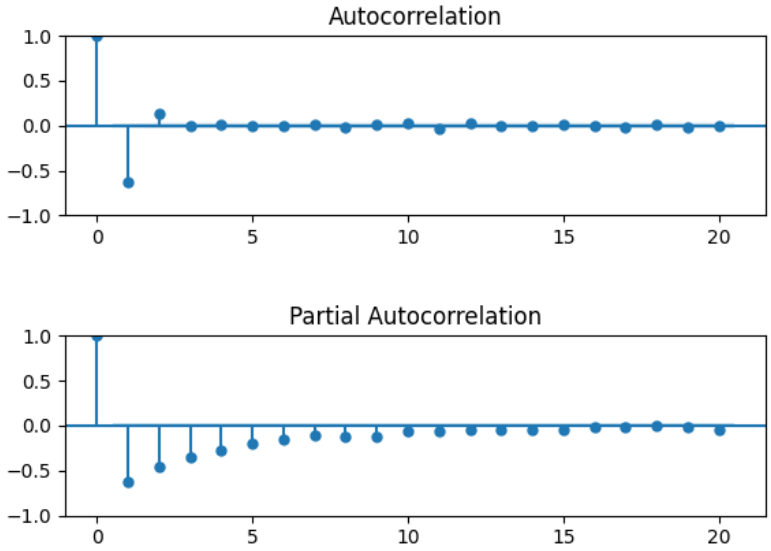

Figure 20: ACF/PCF Plot of Residuals of Naive Method

6.3 Drift Method

The drift method is a straightforward time series forecasting technique applied when there is a linear trend in the data. It involves calculating the average change per time period by dividing the difference between the first and final observations by the total number of observations. The forecast for the next time period is then generated by combining this drift estimate with the most recent value in the time series. This approach is useful when there is no apparent seasonal or cyclical pattern, assuming that the linear trend will persist. However, it may not be suitable for time series with intricate patterns, outliers, or abrupt changes in direction. The drift method is specifically designed for scenarios where a linear continuation of the trend is anticipated.

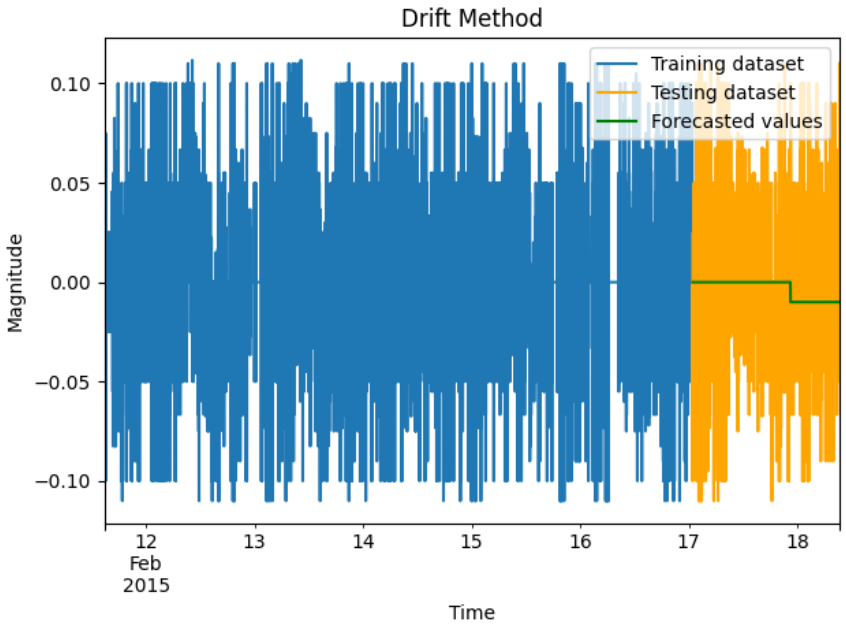

Figure 21: Train/Test/Forecast Plot from Drift Method

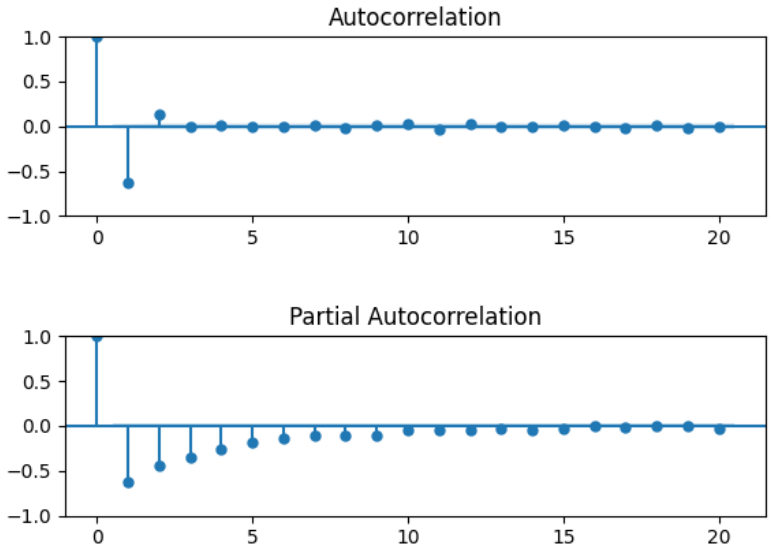

Figure 22: ACF/PCF Plot of Residuals of Drift Method

6.4 Simple and Exponential Smoothing

Simple Exponential Smoothing (SES) is a time series forecasting technique that utilizes an exponentially diminishing weighted average of previous data. It is commonly used for time series data that lacks a distinct pattern or seasonality. SES is a straightforward and effective method for predicting time series data that doesn't exhibit trend or seasonality. However, it may not be the most suitable approach for data with complex patterns, such as those

featuring a trend or seasonality.

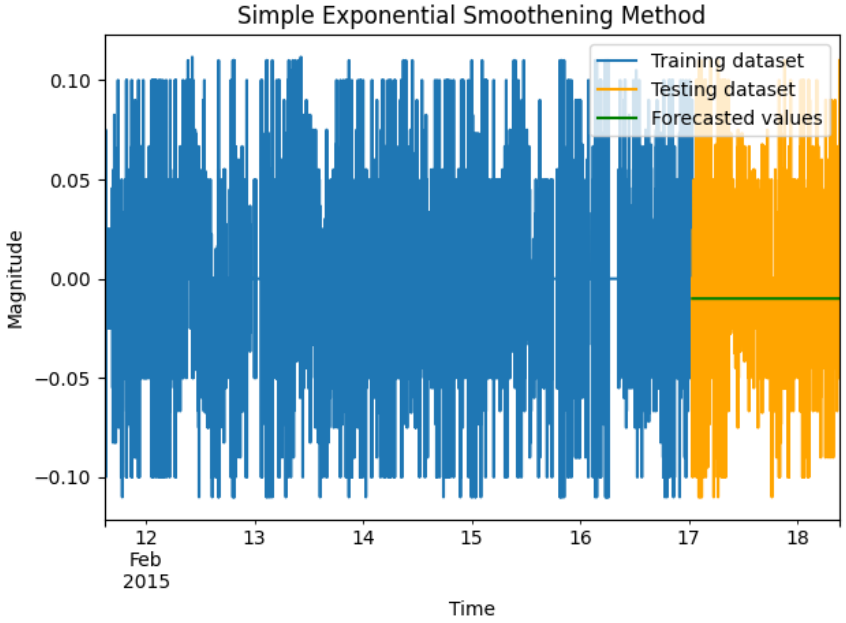

Figure 23: Train/Test/Forecast Plot from SES

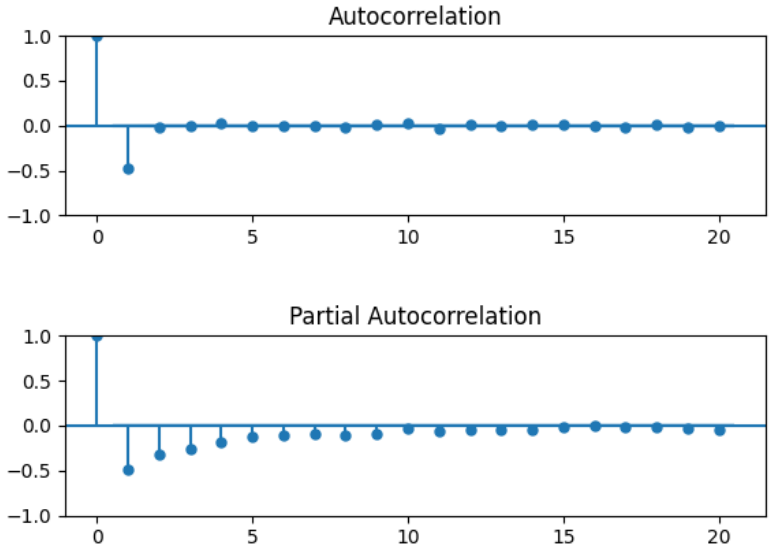

Figure 24: ACF/PCF Plot of Residuals for SES

6.5 Statistical Comparison of Basic Models

Model	Q-Value	Critical-Value	White-Residual	m_res	mse_res	var_res	m_pred	mse_pred	var_pred
Average	968.24	36.1909	No	-0.0002	0.0011	0.0011	0.0004	0.001	0.001
Naive	3282.9	36.1909	No	0	0.0029	0.0029	0.0004	0.001	0.001
Drift	3270.29	36.1909	No	0.0002	0.0029	0.0029	0.0038	0.001	0.001
SES	1854.69	36.1909	No	-0	0.0017	0.0017	0.0104	0.0011	0.001

Table 6: Statistical Comparison of Basic Models

Table Description

- M_RES = Mean of Residual Errors
- MSE_RES = Mean Squared of Residual Errors
- VAR_RES = Variance of Residual Errors
- M_PRED = Mean of Forecast Errors
- MSE_PRED = Mean Squared of Forecast Errors
- VAR_PRED = Variance of Forecast Errors

Observation: The mean and MSE of the residual error, along with the MSE of the forecast error, across all methods are close to 0. This suggests that the model effectively fits the training data with minimal residual errors. Both the variance of the residual error and the variance of the forecast error are also close to 0, indicating a small error variance and precise predictions. After analyzing various basic models, it is observed that the Average Method stands out as the best due to the lowest Q-value, despite the residuals not being entirely white.

7 Multiple Linear Regression

Multiple linear regression for time series forecasting involves extending the traditional linear regression framework to accommodate multiple predictor variables, allowing for a more nuanced understanding of the relationships within time-dependent data. In this approach, historical observations of the dependent variable are modeled as a linear combination of multiple independent variables. These variables can include time-related features or other relevant factors that contribute to the overall trend. The model aims to capture and quantify the influence of these variables on the time series, enabling more accurate predictions.

7.1 Singular value vs condition number

```
singular values of x are [72.21010138 37.17961348  9.91369823  7.05092052  3.15040651]
The condition number for x is 22.920883735819906
```

Figure 25: Singular value vs condition number

7.2 Least Square Estimator

	feature	Beta
0	const	20.6476
1	Humidity	-9.57546
2	Light	1.00327
3	CO2	-0.144398
4	HumidityRatio	8.17751
5	Occupancy	-0.278074

Table 7: LSE using Normal Equation

	feature	Beta
0	const	20.65
1	Humidity	-9.58
2	Light	1
3	CO2	-0.14
4	HumidityRatio	8.18
5	Occupancy	-0.28

Table 8: LSE using OLS model

7.3 Ordinary Least Regression Model

Ordinary Least Squares regression (OLS) is a common technique for estimating coefficients of linear regression equations which describe the relationship between one or more independent quantitative variables and a dependent variable (simple or multiple linear regression). [1]

7.3.1 All Features

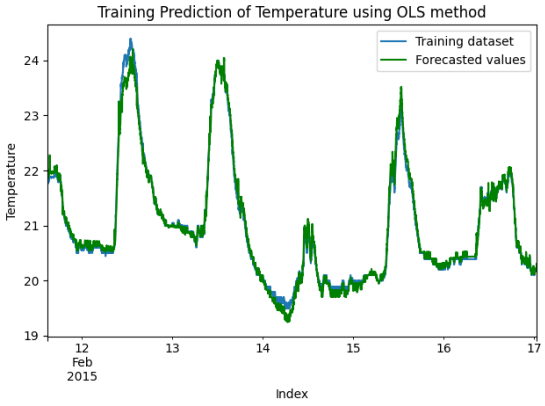

Figure 26: Plot of Train vs Prediction for all features

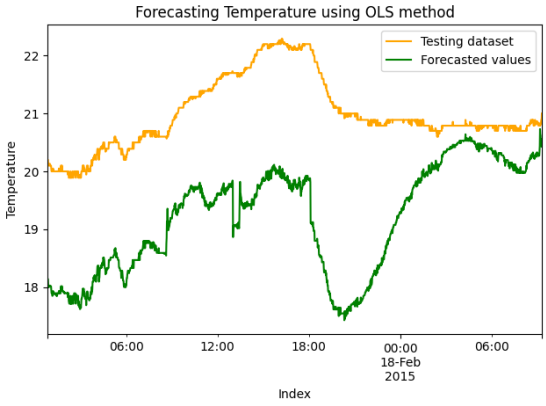

Figure 27: Plot of Test vs Forecast for all features

```
==============================================================================
Dep. Variable:            Temperature   R-squared:                       0.991
Model:                            OLS   Adj. R-squared:                  0.991
Method:                 Least Squares   F-statistic:                 1.734e+05
Date:                Wed, 06 Dec 2023   Prob (F-statistic):               0.00
Time:                        12:40:58   Log-Likelihood:                 6604.2
No. Observations:                7801   AIC:                        -1.320e+04
Df Residuals:                    7795   BIC:                        -1.315e+04
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         20.6476      0.004   5242.088      0.000      20.640      20.655
Humidity      -9.5755      0.017   -564.978      0.000      -9.609      -9.542
Light          1.0033      0.027     37.024      0.000       0.950       1.056
CO2           -0.1444      0.013    -11.240      0.000      -0.170      -0.119
HumidityRatio  8.1775      0.017    489.485      0.000       8.145       8.210
Occupancy     -0.2781      0.008    -36.117      0.000      -0.293      -0.263
==============================================================================
Omnibus:                     1792.857   Durbin-Watson:                   0.049
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             7413.288
Skew:                           1.079   Prob(JB):                         0.00
Kurtosis:                       7.261   Cond. No.                         35.8
==============================================================================
```

Figure 28: Model Summary for all features

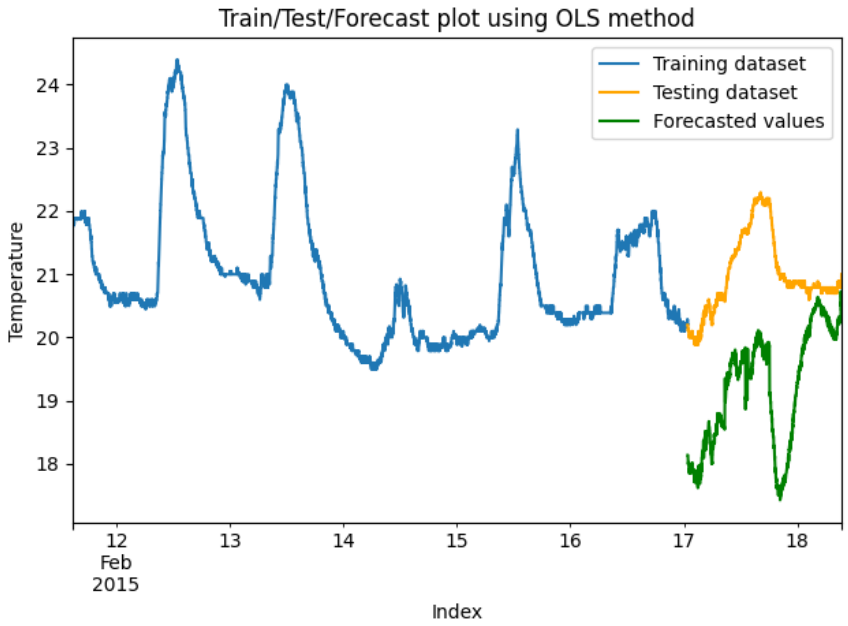

Figure 29: Plot of Train, Test, and Forecast for all features

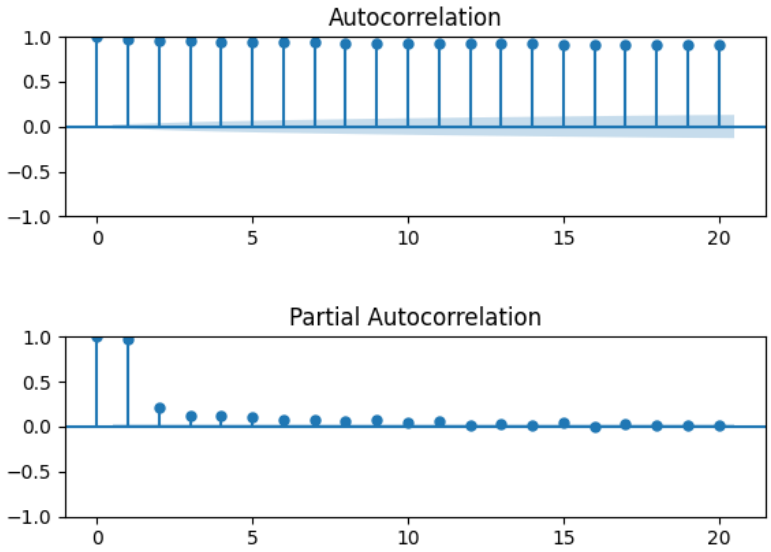

Figure 30: ACF/PACF residual plot for all features

7.3.2 Features from Backward Stepwise Regression

Dep. Variable:	Temperature	R-squared:	0.991
Model:	OLS	Adj. R-squared:	0.991
Method:	Least Squares	F-statistic:	1.734e+05
Date:	Wed, 06 Dec 2023	Prob (F-statistic):	0.00
Time:	12:40:59	Log-Likelihood:	6604.2
No. Observations:	7801	AIC:	-1.320e+04
Df Residuals:	7795	BIC:	-1.315e+04
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	20.6476	0.004	5242.088	0.000	20.640	20.655
Humidity	-9.5755	0.017	-564.978	0.000	-9.609	-9.542
Light	1.0033	0.027	37.024	0.000	0.950	1.056
CO2	-0.1444	0.013	-11.240	0.000	-0.170	-0.119
HumidityRatio	8.1775	0.017	489.485	0.000	8.145	8.210
Occupancy	-0.2781	0.008	-36.117	0.000	-0.293	-0.263

Omnibus:	1792.857	Durbin-Watson:	0.049
Prob(Omnibus):	0.000	Jarque-Bera (JB):	7413.288
Skew:	1.079	Prob(JB):	0.00
Kurtosis:	7.261	Cond. No.	35.8

Figure 31: Model Summary of model with BSR features

Observation: We can see that all the variables have pvalue less than 0.05. Hence, we

cannot remove any variable for feature selection using BSR.

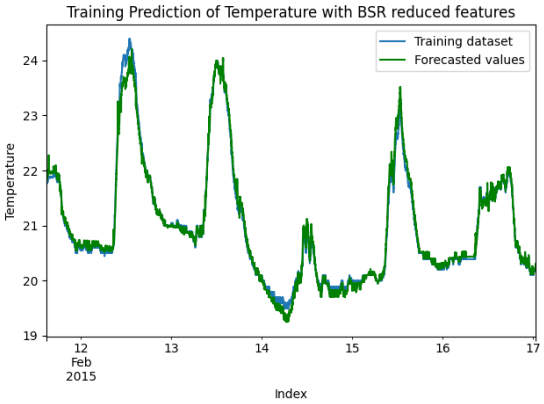

Figure 32: Plot of Train vs Prediction using features from BSR

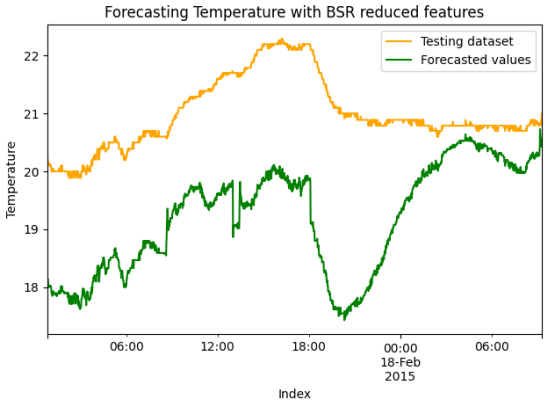

Figure 33: Plot of Test vs Forecast for features from BSR

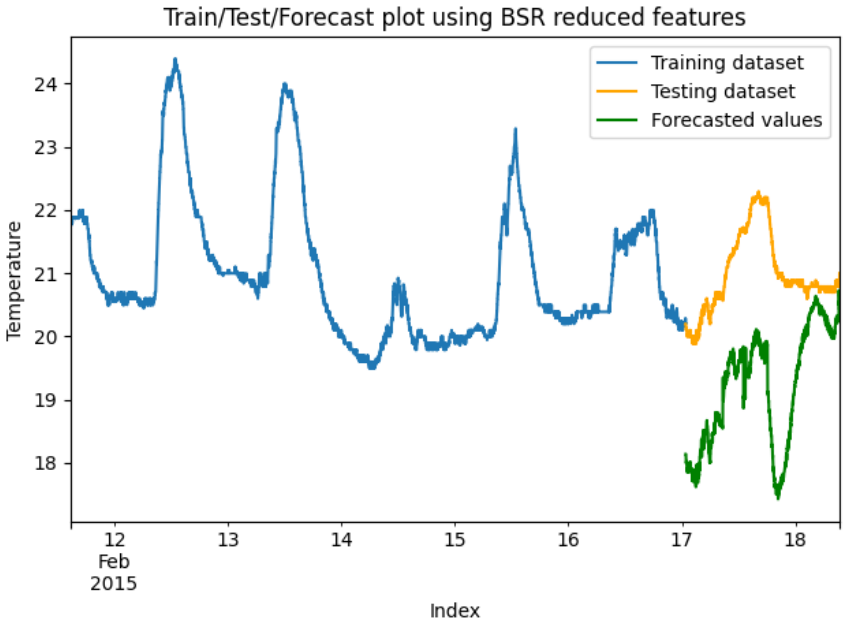

Figure 34: Plot of Train, Test, and Forecast from BSR features model

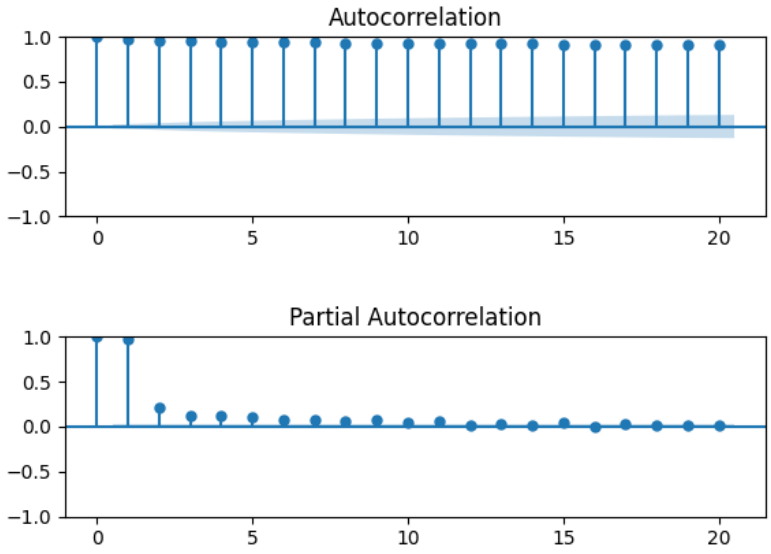

Figure 35: ACF/PACF residual plot for model with BSR features

7.3.3 Features from Variance Inflation Factor

Dep. Variable:	Temperature	R-squared:	0.991
Model:	OLS	Adj. R-squared:	0.991
Method:	Least Squares	F-statistic:	1.734e+05
Date:	Wed, 06 Dec 2023	Prob (F-statistic):	0.00
Time:	12:40:59	Log-Likelihood:	6604.2
No. Observations:	7801	AIC:	-1.320e+04
Df Residuals:	7795	BIC:	-1.315e+04
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	20.6476	0.004	5242.088	0.000	20.640	20.655
Humidity	-9.5755	0.017	-564.978	0.000	-9.609	-9.542
Light	1.0033	0.027	37.024	0.000	0.950	1.056
CO2	-0.1444	0.013	-11.240	0.000	-0.170	-0.119
HumidityRatio	8.1775	0.017	489.485	0.000	8.145	8.210
Occupancy	-0.2781	0.008	-36.117	0.000	-0.293	-0.263

Omnibus:	1792.857	Durbin-Watson:	0.049
Prob(Omnibus):	0.000	Jarque-Bera (JB):	7413.288
Skew:	1.079	Prob(JB):	0.00
Kurtosis:	7.261	Cond. No.	35.8

Figure 36: Model Summary of model before removing 'Humidity'

Feature	VIF Score
const	11.23
Humidity	11.67
Light	9.45
CO2	1.49
Humidity Ratio	10.42
Occupancy	6.67

Table 9: Dataset Variables

Observation: We can see that the variable 'Humidity' has highest VIF score, hence we remove the variable from the dataset and run the OLS model and compare the performance.

```
==============================================================================
Dep. Variable:            Temperature   R-squared:                       0.626
Model:                            OLS   Adj. R-squared:                  0.626
Method:                 Least Squares   F-statistic:                     3266.
Date:                Wed, 06 Dec 2023   Prob (F-statistic):               0.00
Time:                        12:51:50   Log-Likelihood:                -7969.9
No. Observations:                7801   AIC:                         1.595e+04
Df Residuals:                    7796   BIC:                         1.598e+04
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         20.9082      0.025    825.305      0.000      20.859      20.958
Light         11.1010      0.132     84.156      0.000      10.842      11.360
CO2            0.2257      0.083      2.716      0.007       0.063       0.389
HumidityRatio -0.7341      0.036    -20.592      0.000      -0.804      -0.664
Occupancy     -1.9702      0.046    -42.891      0.000      -2.060      -1.880
==============================================================================
Omnibus:                     1395.150   Durbin-Watson:                   0.082
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             9072.380
Skew:                           0.697   Prob(JB):                         0.00
Kurtosis:                       8.096   Cond. No.                         21.4
==============================================================================
```

Figure 37: Model Summary of model after removing 'Humidity'

Observation: We can see that after removing the variable 'Humidity' that has highest VIF score the performance of the model is degraded. Hence, we cannot remove any variable for feature selection using VIF.

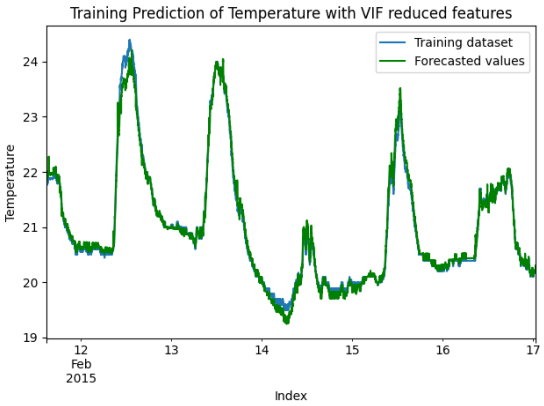

Figure 38: Plot of Train vs Prediction using features from VIF

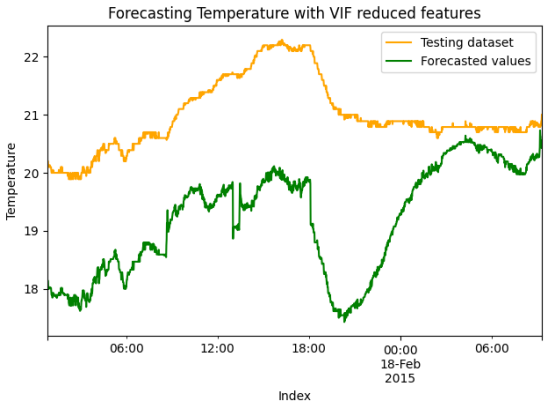

Figure 39: Plot of Test vs Forecast for features from VIF

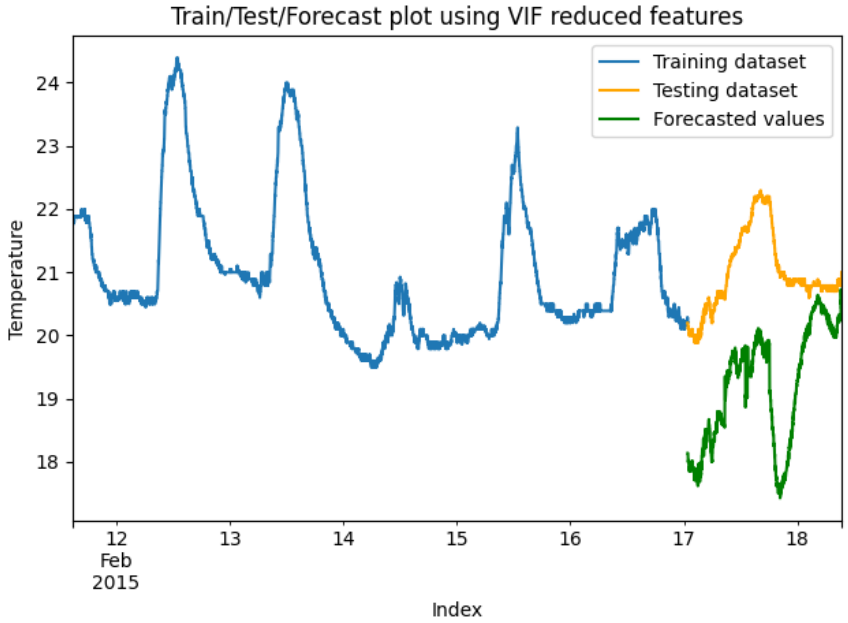

Figure 40: Plot of Train, Test, and Forecast from VIF features

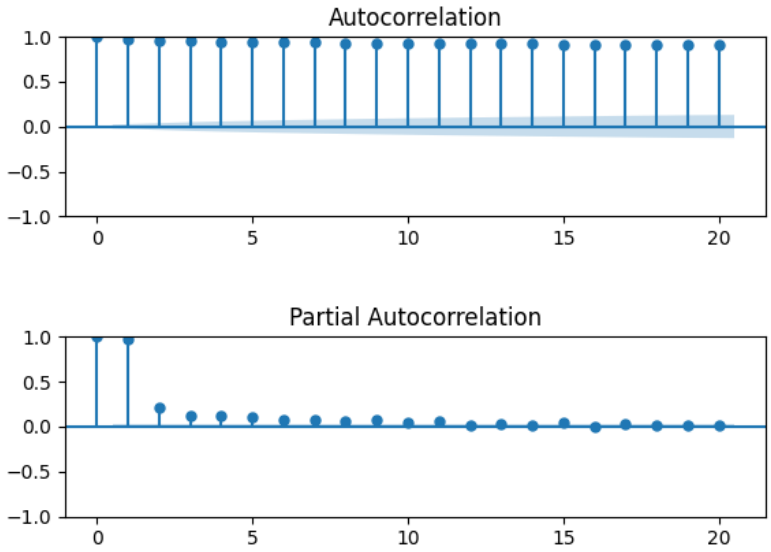

Figure 41: ACF/PACF residual plot for model with VIF features

7.3.4 Features from Principal Component Analysis

Dep. Variable:	Temperature	R-squared:	0.943
Model:	OLS	Adj. R-squared:	0.943
Method:	Least Squares	F-statistic:	3.217e+04
Date:	Wed, 06 Dec 2023	Prob (F-statistic):	0.00
Time:	12:51:55	Log-Likelihood:	-643.00
No. Observations:	7801	AIC:	1296.
Df Residuals:	7796	BIC:	1331.
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	21.0033	0.003	7057.694	0.000	20.997	21.009
x1	1.5381	0.007	218.330	0.000	1.524	1.552
x2	0.7862	0.009	83.411	0.000	0.768	0.805
x3	-2.2462	0.033	-68.595	0.000	-2.310	-2.182
x4	10.2600	0.039	263.379	0.000	10.184	10.336

Omnibus:	3873.111	Durbin-Watson:	0.223
Prob(Omnibus):	0.000	Jarque-Bera (JB):	137780.526
Skew:	-1.735	Prob(JB):	0.00
Kurtosis:	23.294	Cond. No.	13.1

Figure 42: Model Summary of model with PCA features

Observation: We can see that after using Principal Component Analysis on the given independent variables, the model reduced the features into 4 features. The performance drop from using the reduced components is negligible and can be used to create a linear equation for the dependent variable.

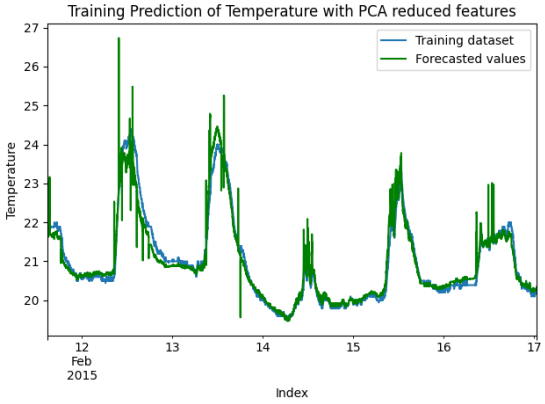

Figure 43: Plot of Train vs Prediction using features from PCA

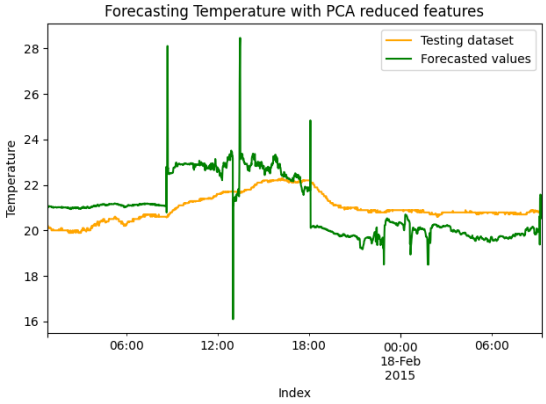

Figure 44: Plot of Test vs Forecast for features from PCA

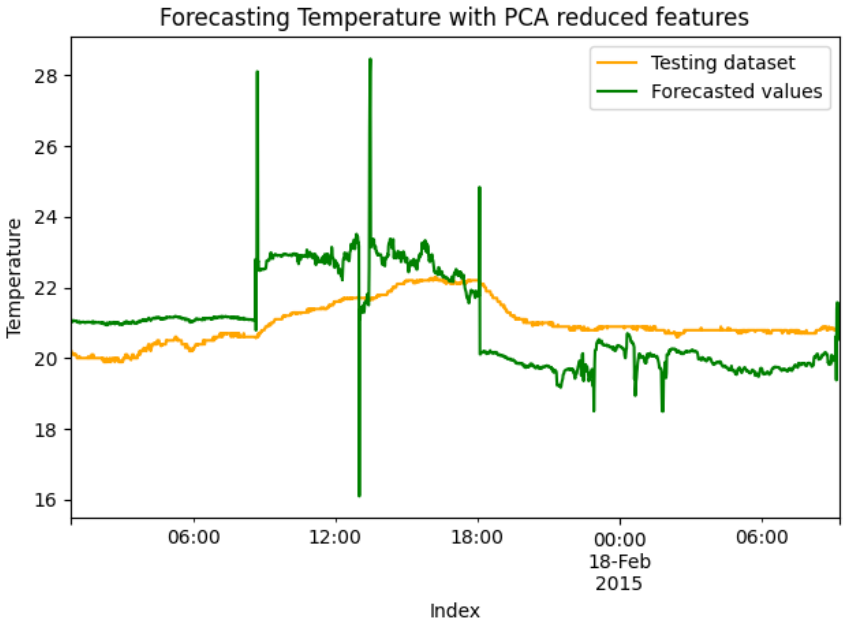

Figure 45: Plot of Train, Test, and Forecast from PCA features

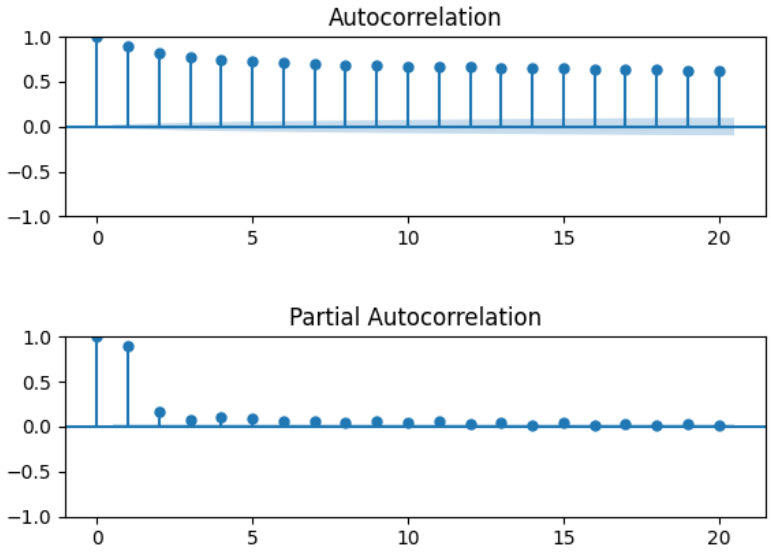

Figure 46: ACF/PACF residual plot for model with PCA features

7.3.5 Model Statistics

	Model	R2	Adjusted-R2	AIC	BIC	F-test	Mean Cross-val
0	All	0.9911	0.9911	-13196.3	-13154.5	0	0.7615
1	BSR	0.9911	0.9911	-13196.3	-13154.5	0	0.7615
2	VIF	0.9911	0.9911	-13196.3	-13154.5	0	0.7615
3	PCA	0.9429	0.9429	1296.01	1330.82	0	-5.2516

Table 10: Model Statistics comparison

7.3.6 Residual and Forecast Error Statistics

	Model	Q-Value	Critical-Value	White-Residual	mse_res	var_res	mse_pred	var_pred
0	All	134490	36.1909	No	0.0108	0.0108	4.0355	0.7771
1	BSR	134490	36.1909	No	0.0108	0.0108	4.0355	0.7771
2	VIF	134490	36.1909	No	0.0108	0.0108	4.0355	0.7771
3	PCA	74914.9	36.1909	No	0.069	0.069	1.2287	1.2287

Table 11: Residual and Forecast Error Statistics comparison

7.3.7 Conclusion

The provided table summarizes the evaluation metrics for different models. The R-squared (R2) and Adjusted R-squared values are consistently high across all models, indicating a strong fit of the models to the data. The AIC and BIC values are minimized, suggesting good model performance. The F-test, which assesses the overall significance of the models,

yields a value of 0 for all models, indicating a lack of statistical significance. The Mean Cross-validation scores are relatively high and consistent across models, further supporting their predictive capabilities. Overall, the model with all, BSR, and VIF reduced features perform similarly well, while the model with PCA reduced features, although still reasonably effective, shows a slightly lower R2 and Adjusted R2, indicating a marginally less robust fit to the data.

8 Order Determination using GPAC

GPAC (Generalized Partial Autocorrelation) is used to estimate the orders of autoregressive (AR) and moving average (MA) components in time series analysis, providing insights into the relationship between variables and aiding in model selection.

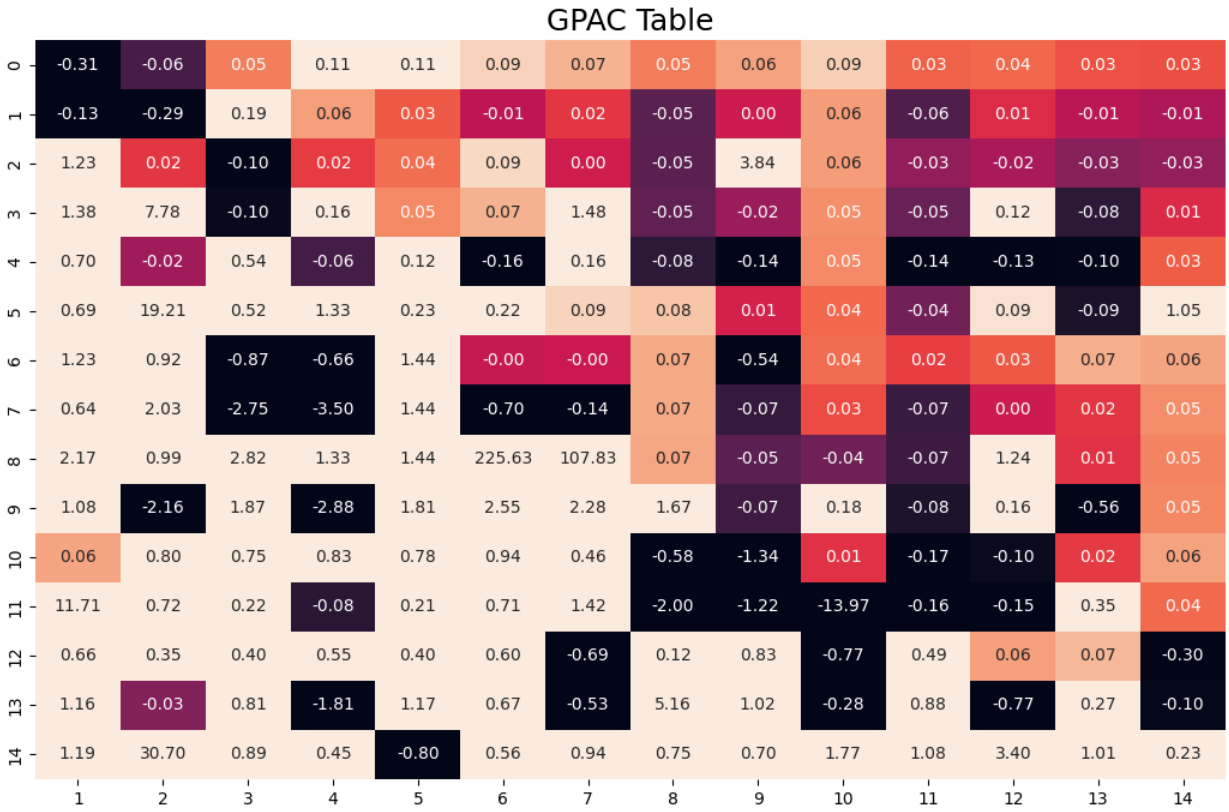

Figure 47: Generalized Partial Auto Correlation Function

Observation: We can observe that there are multiple order combination of AR and MA using the GPAC table. Amongst all observations, four combinations are considered as follows:

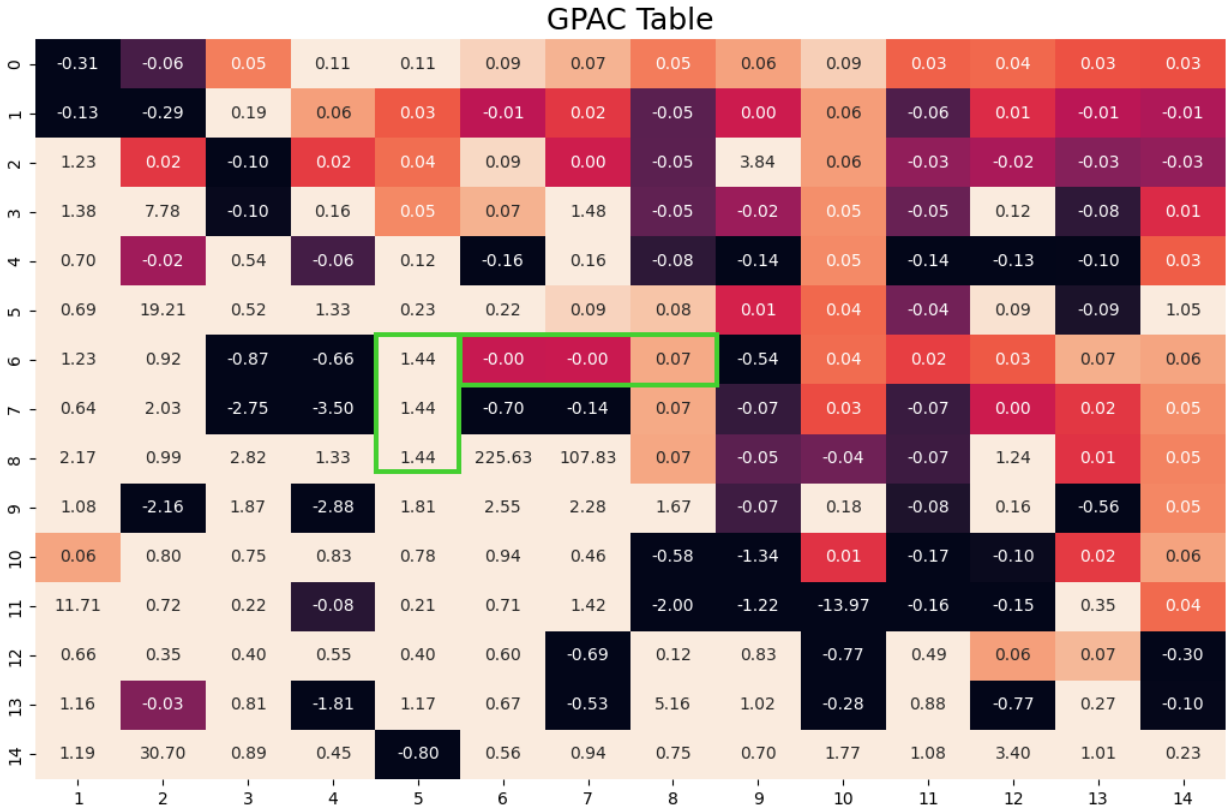

Figure 48: GPAC table with order of (5,6)

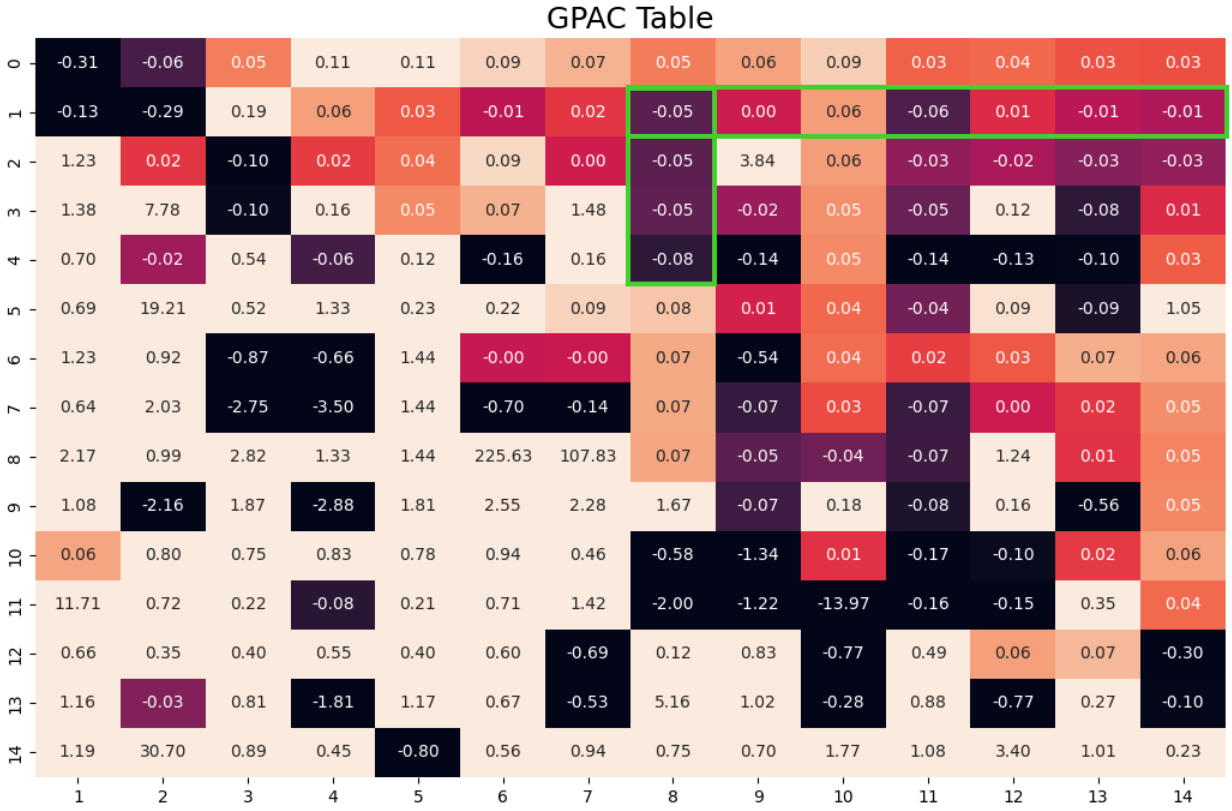

Figure 49: GPAC table with order of (8,1)

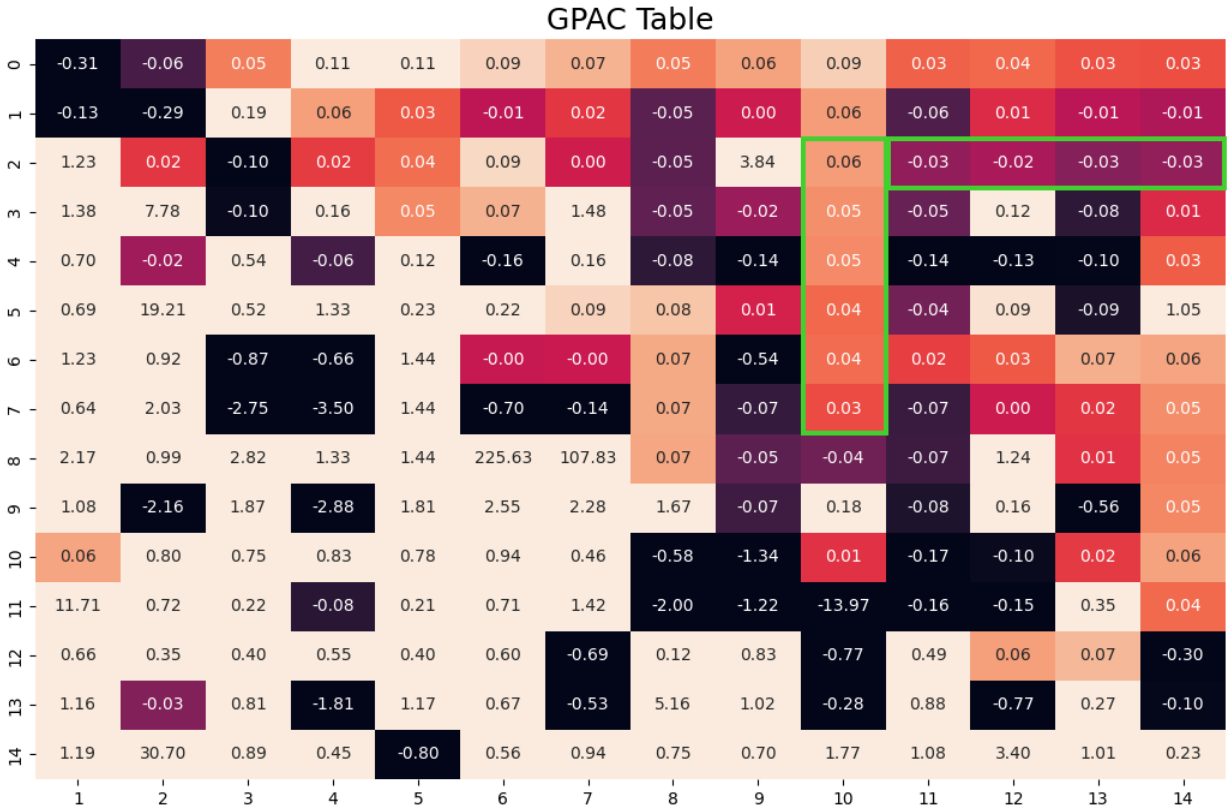

Figure 50: GPAC table with order of (10,2)

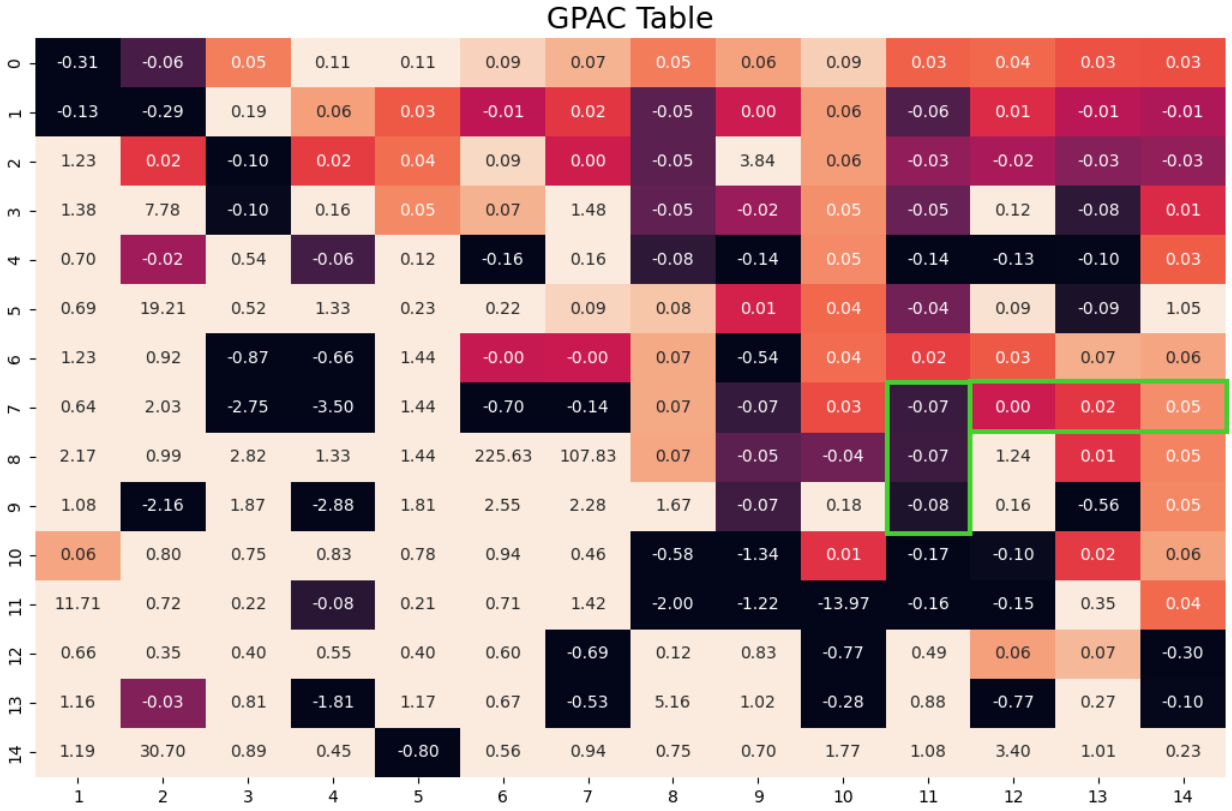

Figure 51: GPAC table with order of (11,7)

9 ARMA Model

9.1 ARMA(5,6)

SARIMAX Results

Dep. Variable:	Temperature	No. Observations:	7800
Model:	SARIMAX(5, 0, 6)	Log Likelihood	16138.754
Date:	Wed, 06 Dec 2023	AIC	-32253.508
Time:	17:35:09	BIC	-32169.965
Sample:	02-11-2015 - 02-17-2015	HQIC	-32224.876
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.0325	0.370	0.088	0.930	-0.694	0.759
ar.L2	0.0297	0.393	0.076	0.940	-0.741	0.801
ar.L3	0.0306	0.344	0.089	0.929	-0.644	0.706
ar.L4	0.1327	0.292	0.454	0.650	-0.440	0.705
ar.L5	0.6784	0.262	2.589	0.010	0.165	1.192
ma.L1	-0.4049	0.371	-1.091	0.275	-1.132	0.322
ma.L2	0.0111	0.516	0.021	0.983	-1.000	1.022
ma.L3	0.0206	0.486	0.043	0.966	-0.931	0.972
ma.L4	-0.0623	0.412	-0.151	0.880	-0.870	0.746
ma.L5	-0.5963	0.338	-1.765	0.078	-1.259	0.066
ma.L6	0.2564	0.107	2.406	0.016	0.048	0.465
sigma2	0.0009	1.05e-05	88.999	0.000	0.001	0.001

Ljung-Box (L1) (Q):	0.76	Jarque-Bera (JB):	1500.37
Prob(Q):	0.38	Prob(JB):	0.00
Heteroskedasticity (H):	1.08	Skew:	0.01
Prob(H) (two-sided):	0.05	Kurtosis:	5.15

Figure 52: Model Summary of ARMA with AR=5 and MA=6

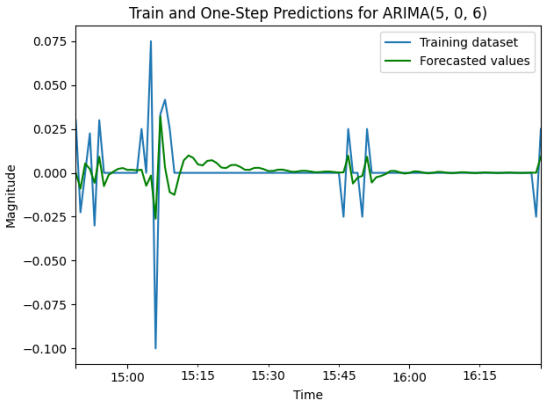

Figure 53: Plot of Train vs Prediction for ARMA(5,6)

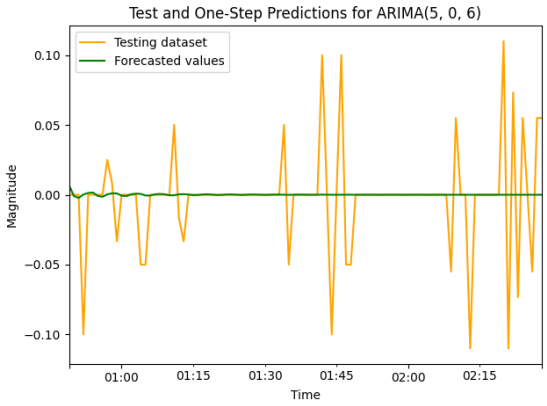

Figure 54: Plot of Test vs Forecast for ARMA(5,6)

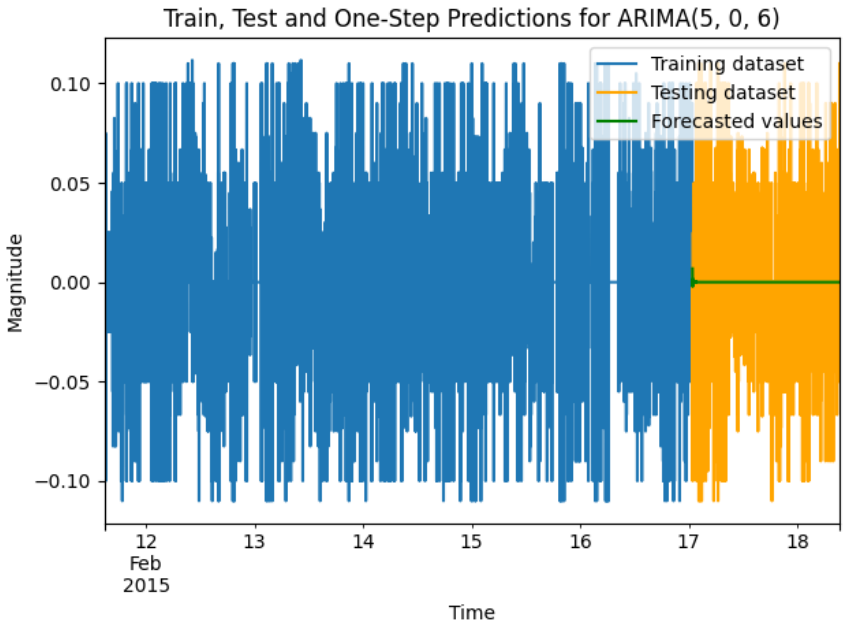

Figure 55: Plot of Train, Test, and Forecast for ARMA(5,6)

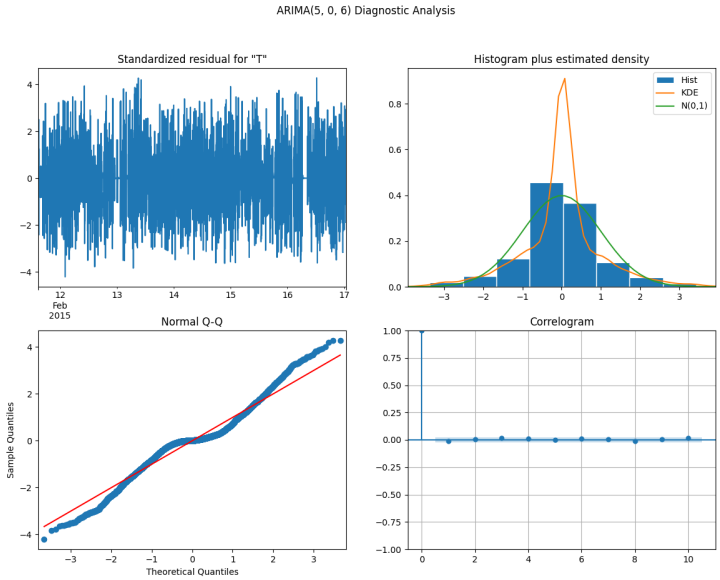

Figure 56: Diagnostic Analysis for for ARMA(5,6)

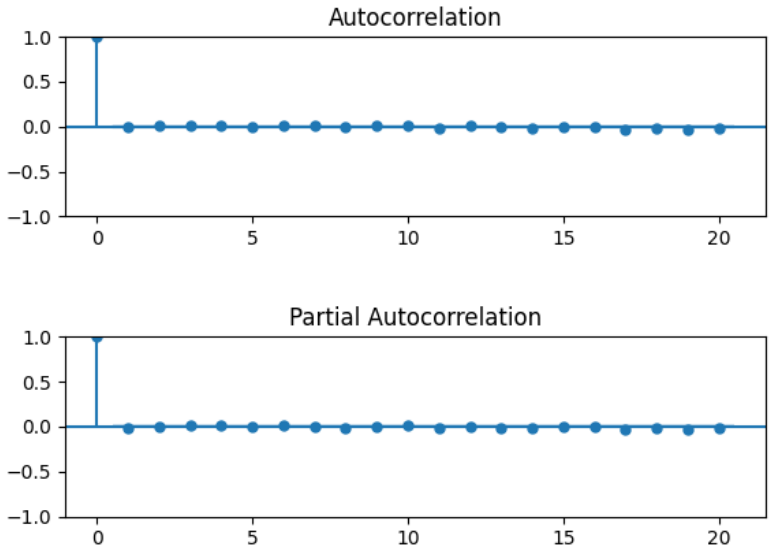

Figure 57: ACF/PCF Plot of ARMA(5,6)

Observation: We can observe from the ACF and PACF plot of residual errors that the residuals seems white as there are not significant values after lag 0. Therefore, we can assume that the residuals are white.

9.2 ARMA(8,1)

Dep. Variable:	Temperature	No. Observations:	7800
Model:	SARIMAX(8, 0, 1)	Log Likelihood	16119.744
Date:	Wed, 06 Dec 2023	AIC	-32219.489
Time:	17:35:37	BIC	-32149.870
Sample:	02-11-2015	HQIC	-32195.629
	- 02-17-2015		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.4221	0.025	16.563	0.000	0.372	0.472
ar.L2	0.2107	0.014	15.323	0.000	0.184	0.238
ar.L3	0.1038	0.011	9.239	0.000	0.082	0.126
ar.L4	0.0757	0.011	6.681	0.000	0.054	0.098
ar.L5	0.0221	0.012	1.867	0.062	-0.001	0.045
ar.L6	-0.0022	0.012	-0.178	0.858	-0.026	0.022
ar.L7	0.0118	0.012	0.949	0.343	-0.013	0.036
ar.L8	0.0391	0.012	3.219	0.001	0.015	0.063
ma.L1	-0.8197	0.024	-33.745	0.000	-0.867	-0.772
sigma2	0.0009	1.05e-05	89.423	0.000	0.001	0.001

Ljung-Box (L1) (Q):	2.49	Jarque-Bera (JB):	1543.46
Prob(Q):	0.11	Prob(JB):	0.00
Heteroskedasticity (H):	1.07	Skew:	0.02
Prob(H) (two-sided):	0.08	Kurtosis:	5.18

Figure 58: Model Summary of ARMA with AR=8 and MA=1

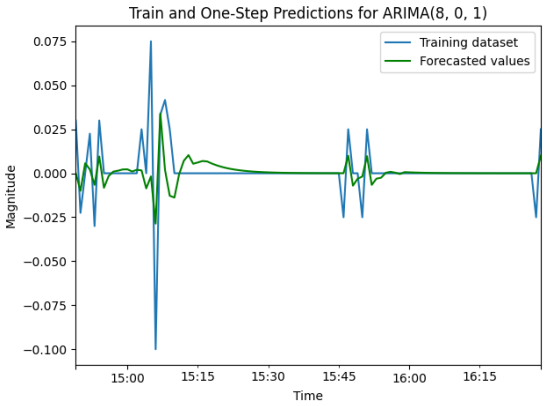

Figure 59: Plot of Train vs Prediction for ARMA(8,1)

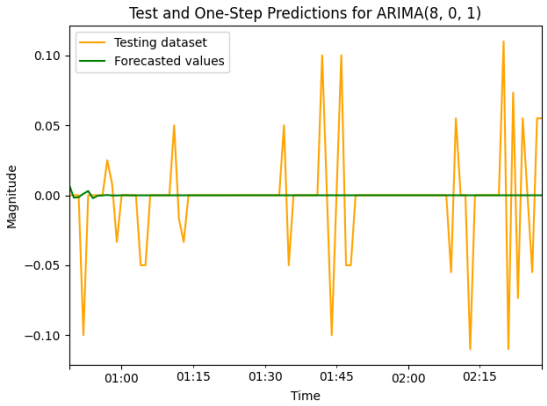

Figure 60: Plot of Test vs Forecast for ARMA(8,1)

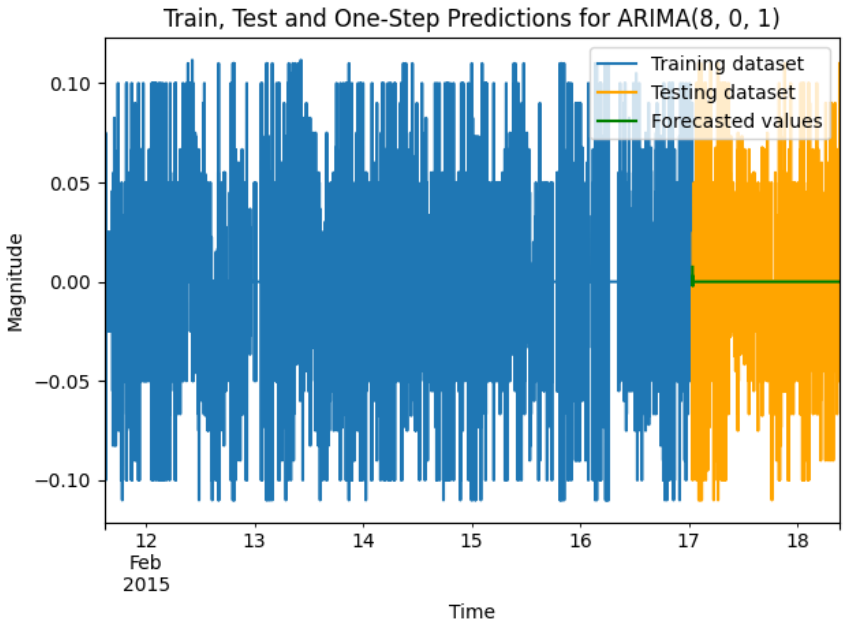

Figure 61: Plot of Train, Test, and Forecast for ARMA(8,1)

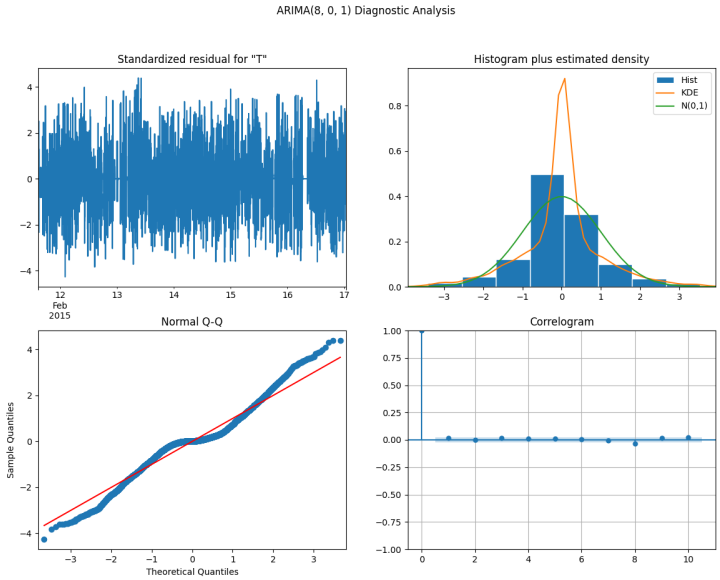

Figure 62: Diagnostic Analysis for for ARMA(8,1)

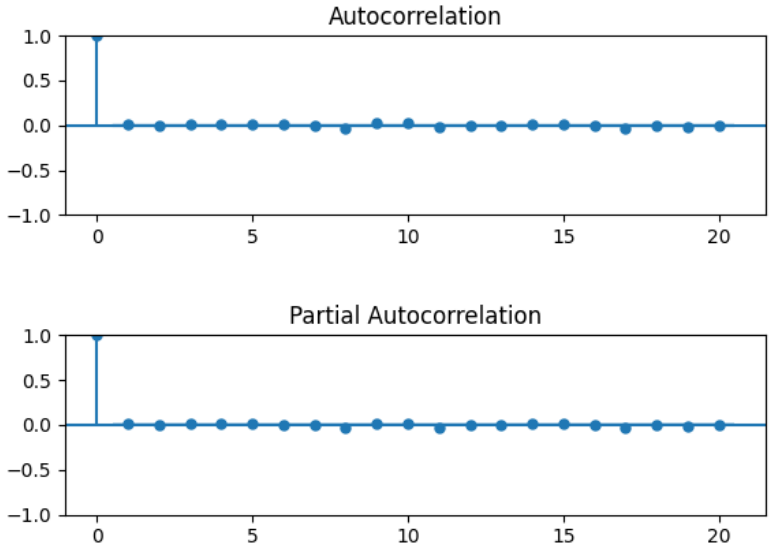

Figure 63: ACF/PCF Plot of ARMA(8,1)

Observation: We can observe from the ACF and PACF plot of residual errors that the residuals seems white as there are not significant values after lag 0. Therefore, we can assume that the residuals are white.

9.3 ARMA(10,2)

Dep. Variable:	Temperature	No. Observations:	7800
Model:	SARIMAX(10, 0, 2)	Log Likelihood	16130.184
Date:	Wed, 06 Dec 2023	AIC	-32234.369
Time:	17:36:11	BIC	-32143.864
Sample:	02-11-2015 - 02-17-2015	HQIC	-32203.351
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.2000	0.211	-0.947	0.344	-0.614	0.214
ar.L2	0.4143	0.103	4.007	0.000	0.212	0.617
ar.L3	0.2400	0.049	4.863	0.000	0.143	0.337
ar.L4	0.1480	0.031	4.844	0.000	0.088	0.208
ar.L5	0.0749	0.019	3.859	0.000	0.037	0.113
ar.L6	0.0074	0.015	0.500	0.617	-0.022	0.037
ar.L7	-0.0080	0.014	-0.566	0.572	-0.036	0.020
ar.L8	0.0041	0.013	0.306	0.760	-0.022	0.030
ar.L9	0.0429	0.013	3.269	0.001	0.017	0.069
ar.L10	0.0564	0.013	4.477	0.000	0.032	0.081
ma.L1	-0.1891	0.211	-0.895	0.371	-0.603	0.225
ma.L2	-0.4475	0.180	-2.486	0.013	-0.800	-0.095
sigma2	0.0009	1.05e-05	89.161	0.000	0.001	0.001

Ljung-Box (L1) (Q):	0.79	Jarque-Bera (JB):	1496.98
Prob(Q):	0.37	Prob(JB):	0.00
Heteroskedasticity (H):	1.07	Skew:	0.00
Prob(H) (two-sided):	0.08	Kurtosis:	5.15

Figure 64: Model Summary of ARMA with AR=10 and MA=2

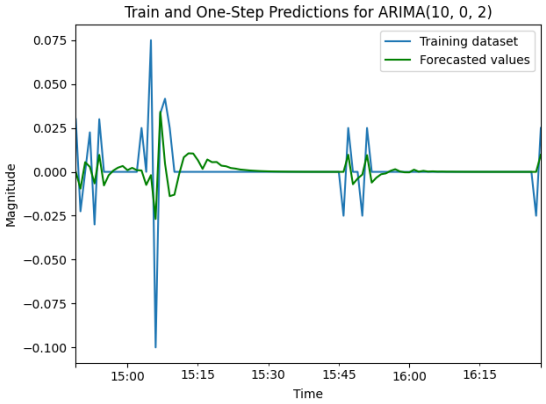

Figure 65: Plot of Train vs Prediction for ARMA(10,2)

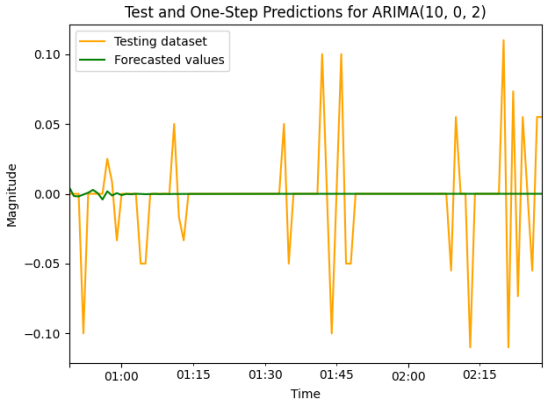

Figure 66: Plot of Test vs Forecast for ARMA(10,2)

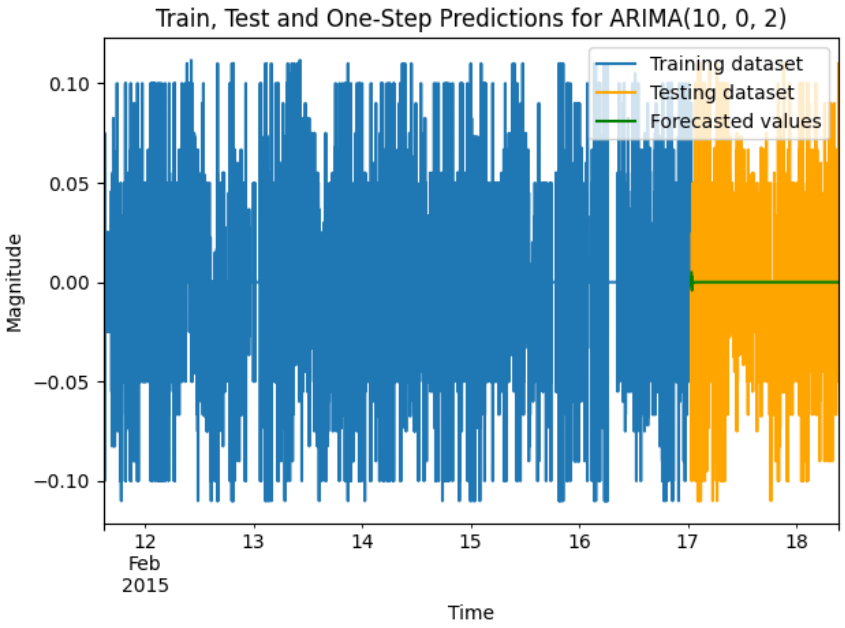

Figure 67: Plot of Train, Test, and Forecast for ARMA(10,2)

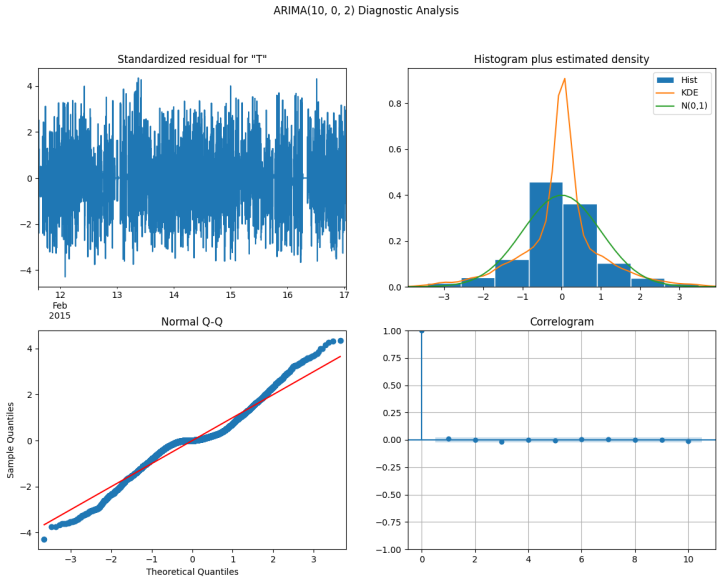

Figure 68: Diagnostic Analysis for for ARMA(10,2)

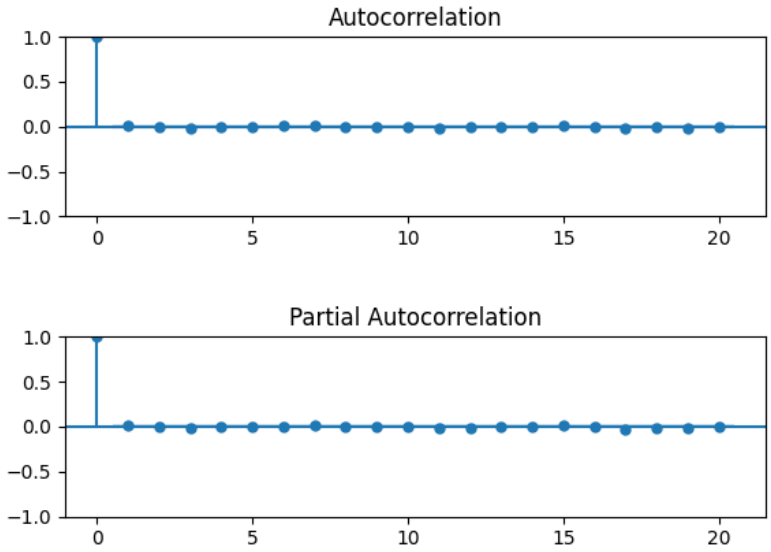

Figure 69: ACF/PCF Plot of ARMA(10,2)

Observation: We can observe from the ACF and PACF plot of residual errors that the residuals seems white as there are not significant values after lag 0. Therefore, we can assume that the residuals are white.

9.4 ARMA(11,7)

Dep. Variable:	Temperature	No. Observations:	7800
Model:	SARIMAX(11, 0, 7)	Log Likelihood	16148.665
Date:	Wed, 06 Dec 2023	AIC	-32259.330
Time:	17:37:04	BIC	-32127.054
Sample:	02-11-2015 - 02-17-2015	HQIC	-32213.997
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.1212	0.333	-0.365	0.715	-0.773	0.531
ar.L2	0.1171	0.343	0.342	0.733	-0.554	0.788
ar.L3	0.0049	0.279	0.018	0.986	-0.542	0.552
ar.L4	0.1445	0.222	0.651	0.515	-0.291	0.580
ar.L5	0.2007	0.233	0.860	0.390	-0.257	0.658
ar.L6	0.3582	0.266	1.345	0.178	-0.164	0.880
ar.L7	0.1744	0.284	0.613	0.540	-0.383	0.732
ar.L8	0.0554	0.119	0.465	0.642	-0.178	0.289
ar.L9	0.0301	0.045	0.673	0.501	-0.057	0.118
ar.L10	0.0034	0.017	0.198	0.843	-0.030	0.037
ar.L11	-0.0478	0.016	-2.961	0.003	-0.079	-0.016
ma.L1	-0.2612	0.333	-0.784	0.433	-0.914	0.391
ma.L2	-0.1273	0.293	-0.435	0.664	-0.702	0.447
ma.L3	0.0980	0.245	0.400	0.689	-0.382	0.578
ma.L4	-0.0783	0.216	-0.362	0.717	-0.502	0.346
ma.L5	-0.1143	0.219	-0.521	0.602	-0.544	0.316
ma.L6	-0.2775	0.226	-1.230	0.219	-0.720	0.165
ma.L7	-0.0279	0.231	-0.121	0.904	-0.481	0.425
sigma2	0.0009	1.06e-05	87.799	0.000	0.001	0.001

Ljung-Box (L1) (Q):	0.00	Jarque-Bera (JB):	1448.98
Prob(Q):	0.95	Prob(JB):	0.00
Heteroskedasticity (H):	1.09	Skew:	0.00
Prob(H) (two-sided):	0.04	Kurtosis:	5.11

Figure 70: Model Summary of ARMA with AR=12 and MA=7

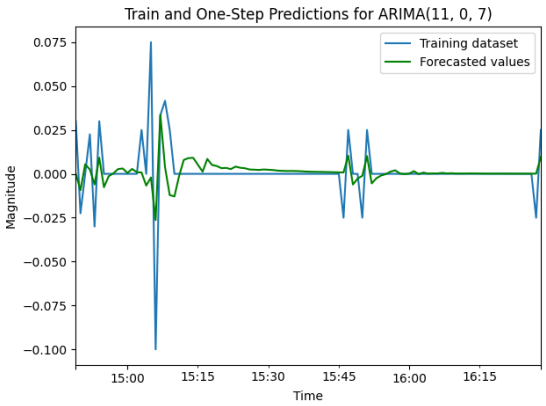

Figure 71: Plot of Train vs Prediction for ARMA(11,7)

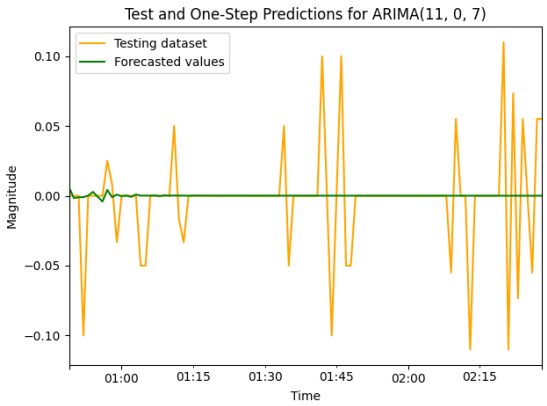

Figure 72: Plot of Test vs Forecast for ARMA(11,7)

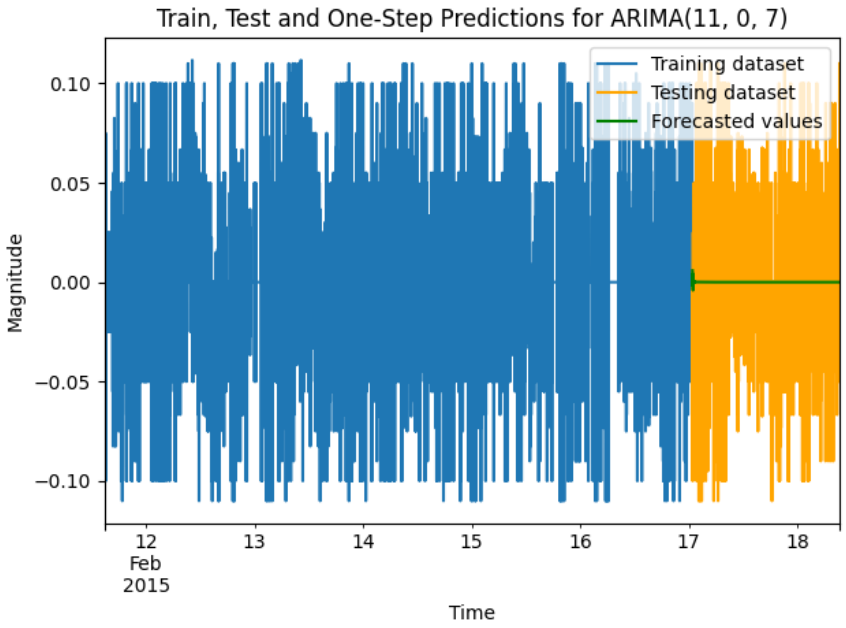

Figure 73: Plot of Train, Test, and Forecast for ARMA(11,7)

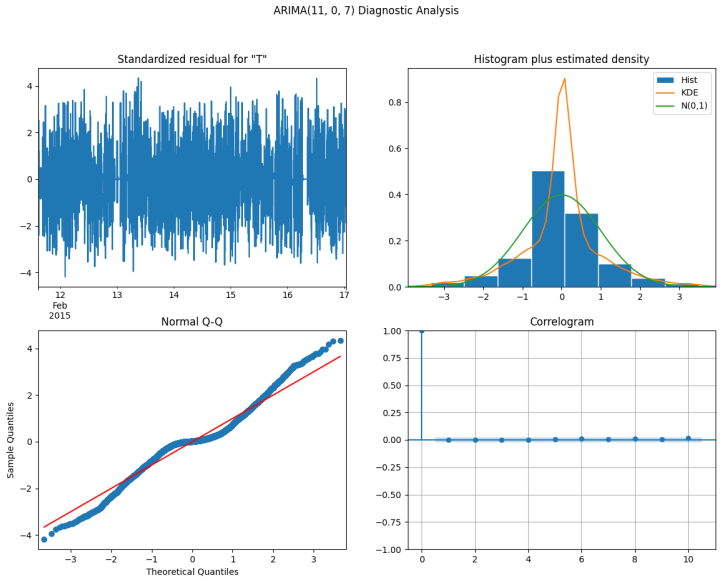

Figure 74: Diagnostic Analysis for for ARMA(11,7)

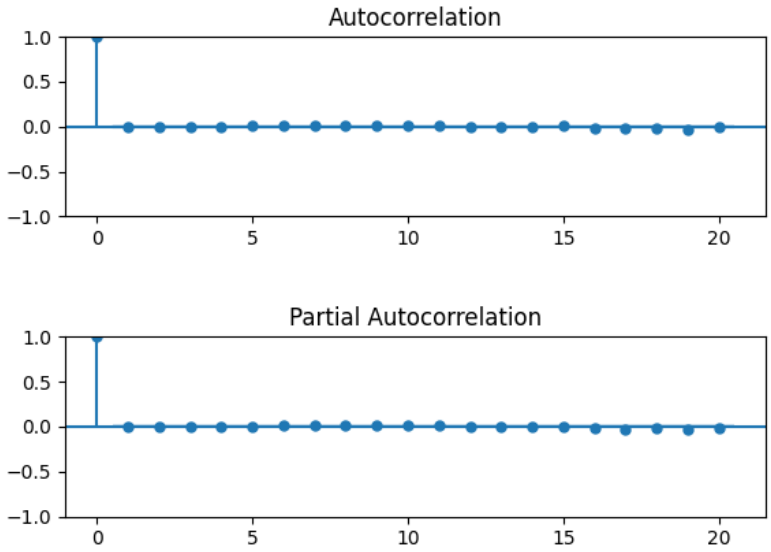

Figure 75: ACF/PCF Plot of ARMA(11,7)

Observation: We can observe from the ACF and PACF plot of residual errors that the residuals seems white as there are not significant values after lag 0. Therefore, we can assume that the residuals are white.

9.5 Model Comparison

Model	Q-Value	Critical-Value	White-Residual	m_res	mse_res	var_res	m_pred	mse_pred	var_pred
ARIMA(5, 0, 6)	35.3081	36.1909	Yes	-0.0001	0.0009	0.0009	0.0004	0.001	0.001
ARIMA(8, 0, 1)	40.5755	36.1909	No	-0.0001	0.0009	0.0009	0.0004	0.001	0.001
ARIMA(10, 0, 2)	17.8974	36.1909	Yes	-0.0001	0.0009	0.0009	0.0004	0.001	0.001
ARIMA(11, 0, 7)	21.3006	36.1909	Yes	-0.0001	0.0009	0.0009	0.0004	0.001	0.001

Table 12: Statistical Comparison of Basic Models

Table Description

- M_RES = Mean of Residual Errors
- MSE_RES = Mean Squared of Residual Errors
- VAR_RES = Variance of Residual Errors
- M_PRED = Mean of Forecast Errors
- MSE_PRED = Mean Squared of Forecast Errors
- VAR_PRED = Variance of Forecast Errors

9.6 Conclusion

ARMA(10, 2) exhibit the lowest Q value amongst all models and the second best model is ARMA(11, 7) both indicating a comparable goodness of fit, with the residuals following a white noise pattern. Whereas ARMA(5, 6) and ARMA(8, 2) have similar Q-Values. However, ARIMA(8, 1) does not meet the White-Residual criterion, suggesting a potential lack of randomness in the residuals.

The critical-value is consistent across all models, indicating a consistent threshold for assessing the statistical significance of the Q-Value. In terms of forecasting performance, all models show similar results for mean, mean squared error, and variance of both residuals and predictions.

In summary, all models except ARMA(8,1) perform well, satisfying the White-Residual criterion and demonstrating comparable forecasting accuracy, while ARMA(8,1) may need further investigation due to its failure to meet the White-Residual criterion.

10 Levenberg Marquardt algorithm

The Levenberg-Marquardt algorithm, devised in the 1960s, addresses nonlinear least squares problems, common in fitting mathematical models to data points. For linear models, solving least squares is straightforward, but when the model isn't linear, an iterative approach is needed. LM combines gradient descent and Gauss-Newton methods, behaving like gradient descent when parameters are far from optimal and like Gauss-Newton when close. This algorithm minimizes the sum of squared errors by iterative updating model parameters.

		Coefficient
0	AR coefficient 1	0.228
1	AR coefficient 2	-0.698
2	AR coefficient 3	-0.329
3	AR coefficient 4	-0.177
4	AR coefficient 5	-0.068
5	AR coefficient 6	0.015
6	AR coefficient 7	0.034
7	AR coefficient 8	0.033
8	AR coefficient 9	0.013
9	AR coefficient 10	-0.014
10	MA coefficient 11	-0.155
11	MA coefficient 12	-0.749

Table 13: Model parameters using LM Algorithm

		Lower Bound	Upper Bound
0	AR coefficient 1	-0.355	0.81
1	AR coefficient 2	-1.031	-0.365
2	AR coefficient 3	-0.48	-0.178
3	AR coefficient 4	-0.258	-0.096
4	AR coefficient 5	-0.121	-0.016
5	AR coefficient 6	-0.015	0.045
6	AR coefficient 7	0.002	0.065
7	AR coefficient 8	0.003	0.062
8	AR coefficient 9	-0.013	0.039
9	AR coefficient 10	-0.039	0.011
10	MA coefficient 11	-0.738	0.427
11	MA coefficient 12	-1.304	-0.194

Table 14: Confidence Interval for the estimated parameters

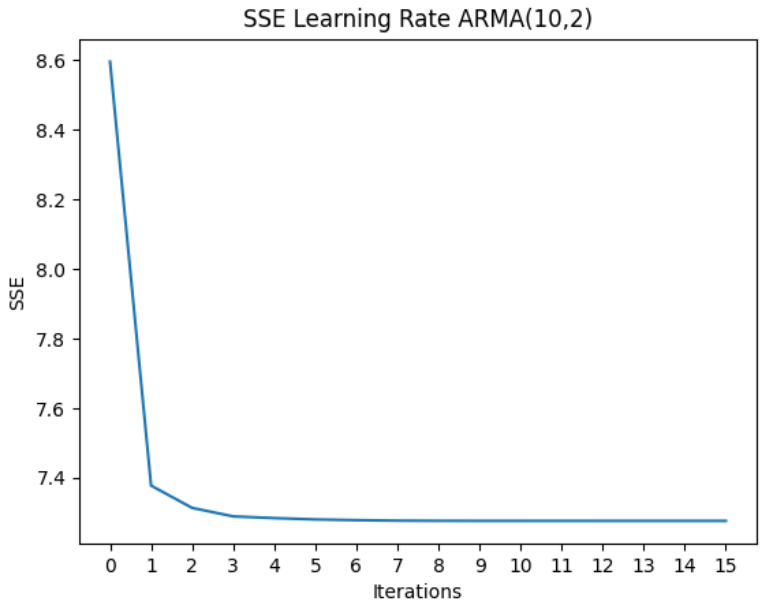

Figure 76: Plot of sum of squared errors converging at 15 iterations

Estimated variance of error: 0.001

Observation: We can observe that the estimated parameters are close to the one derived from the ARMA model in the previous section. Though we can see that some of the confidence intervals have zero included that states the insignificance of the the estimated coefficients. Therefore, we need to perform zero-pole cancellation to stabilize the time series.

11 Auto ARIMA

Dep. Variable:	y	No. Observations:	7800
Model:	SARIMAX(1, 0, 2)	Log Likelihood	16129.788
Date:	Wed, 06 Dec 2023	AIC	-32249.576
Time:	22:53:58	BIC	-32214.766
Sample:	02-11-2015	HQIC	-32237.646
	- 02-17-2015		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	-9.888e-06	2.03e-05	-0.487	0.626	-4.97e-05	2.99e-05
ar.L1	0.9697	0.005	206.581	0.000	0.960	0.979
ma.L1	-1.3494	0.009	-155.296	0.000	-1.366	-1.332
ma.L2	0.4074	0.008	51.203	0.000	0.392	0.423
sigma2	0.0009	1.05e-05	89.390	0.000	0.001	0.001

Ljung-Box (L1) (Q):	0.01	Jarque-Bera (JB):	1530.03
Prob(Q):	0.90	Prob(JB):	0.00
Heteroskedasticity (H):	1.08	Skew:	0.01
Prob(H) (two-sided):	0.05	Kurtosis:	5.17

Figure 77: Model summary from Auto AIMA Model

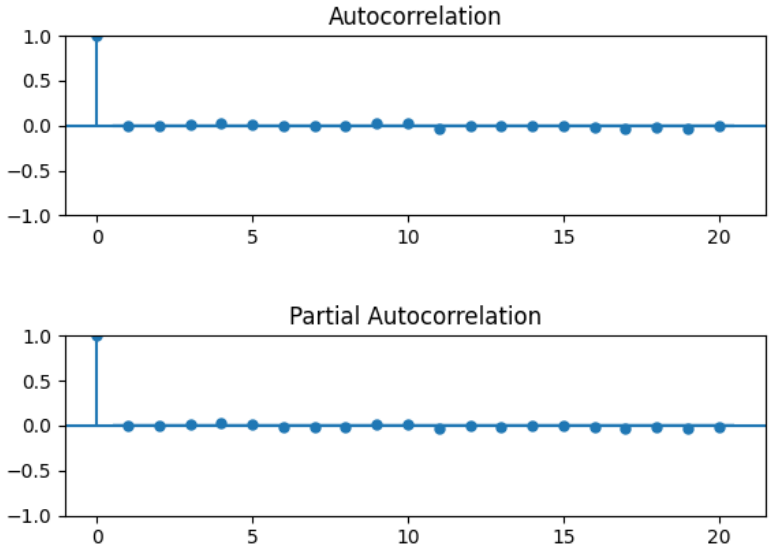

Figure 78: Plot of ACF/PACF of residuals from Auto ARIMA

	Model	Q-Value	Critical-Value	White-Residual	m_res	mse_res	var_res	m_pred	mse_pred	var_pred
0	ARIMA(1, 2)	49.0996	36.1909	No	-0.0001	0.0009	0.0009	0.0004	0.001	0.001

Table 15: Statistical values from Auto ARIMA Model

Observation: Auto ARIMA model was able to identify a low complexity model or AR = 1 and MA=2. Though the ACF and PACF plot seems stationary without any seasonal component. Upon performing residual analysis, we see that the Q-value is greater than critical value making the residuals non-white.

12 Forecast Function

Using the model parameter estimates from the ARMA(10, 2) model in section 9.3 the forecast function can be deduced as follows:

$$y(t)+0.41y(t-2)+0.24y(t-3)+0.15y(t-4)+0.75y(t-5)+0.04y(t-9)+0.06y(t-10) = \epsilon(t)-0.45\epsilon(t-2)$$

$$y(t) = \epsilon(t)-0.45\epsilon(t-2)0.41y(t-2)-0.24y(t-3)-0.15y(t-4)-0.75y(t-5)-0.04y(t-9)-0.06y(t-10)$$

Forecast function for h-step:

$$\hat{y}(1) = -0.45\epsilon(t-1)-0.41y(t-1)-0.24y(t-2)-0.15y(t-3)-0.75y(t-5)-0.04y(t-8)-0.06y(t-9)$$

$$\hat{y}(2) = -0.41y(t) - 0.24y(t-1) - 0.15y(t-2) - 0.75y(t-4) - 0.04y(t-7) - 0.06y(t-8)$$

$$\hat{y}(3) = -0.41\hat{y}(1) - 0.24y(t) - 0.15y(t-1) - 0.75y(t-3) - 0.04y(t-6) - 0.06y(t-7)$$

$$\hat{y}(4) = -0.41\hat{y}(2) - 0.24\hat{y}(1) - 0.15y(t) - 0.75y(t-2) - 0.04y(t-5) - 0.06y(t-6)$$

$$\hat{y}(5) = -0.41\hat{y}(3) - 0.24\hat{y}(2) - 0.15\hat{y}(1) - 0.75y(t-1) - 0.04y(t-4) - 0.06y(t-5)$$

$$\hat{y}(6) = -0.41\hat{y}(4) - 0.24\hat{y}(3) - 0.15\hat{y}(2) - 0.75y(t) - 0.04y(t-3) - 0.06y(t-4)$$

$$\hat{y}(7) = -0.41\hat{y}(5) - 0.24\hat{y}(4) - 0.15\hat{y}(3) - 0.75\hat{y}(1) - 0.04y(t-2) - 0.06y(t-3)$$

$$\hat{y}(8) = -0.41\hat{y}(6) - 0.24\hat{y}(5) - 0.15\hat{y}(4) - 0.75\hat{y}(2) - 0.04y(t-1) - 0.06y(t-2)$$

$$\hat{y}(9) = -0.41\hat{y}(7) - 0.24\hat{y}(6) - 0.15\hat{y}(5) - 0.75\hat{y}(3) - 0.04y(t) - 0.06y(t-1)$$

$$\hat{y}(10) = -0.41\hat{y}(8) - 0.24\hat{y}(7) - 0.15\hat{y}(6) - 0.75\hat{y}(4) - 0.04\hat{y}(1) - 0.06y(t)$$

$$\hat{y}(11) = -0.41\hat{y}(9) - 0.24\hat{y}(8) - 0.15\hat{y}(7) - 0.75\hat{y}(5) - 0.04\hat{y}(2) - 0.06\hat{y}(1)$$

.
.
.

For h>11

$$\hat{y}(h) = -0.41\hat{y}(h-2)-0.24\hat{y}(h-3)-0.15\hat{y}(h-4)-0.75\hat{y}(h-6)-0.04\hat{y}(h-9)-0.06\hat{y}(h-10)$$

Method	Q-Value	Improvement
Naive Method	3282.90	-
Drift Method	3270.29	0.38%
SES Method	1854.69	43.53%
Average Method	968.24	47.72%
Holt-Winter's Method	957.81	1.09%
ARMA (8,0)	40.58	95.76%
ARMA (5,6)	35.31	13.02%
ARMA (11,7)	21.30	39.85%
ARMA (10,2)	17.89	15.92%

Table 16: Performance comparison across models

13 Final Model Selection

Among all the models, it is evident that the ARMA(10, 2) model attains the lowest Q-value, reflecting a strong fit to the training data with a Q-value of 17.89, a statistical measure indicating goodness of fit. Additionally, the multi-linear regression model is noteworthy with an R-squared value of 0.99, signifying that the model accounts for 99% of the variance in the data, indicative of a robust fit.

14 h-step ahead Prediction

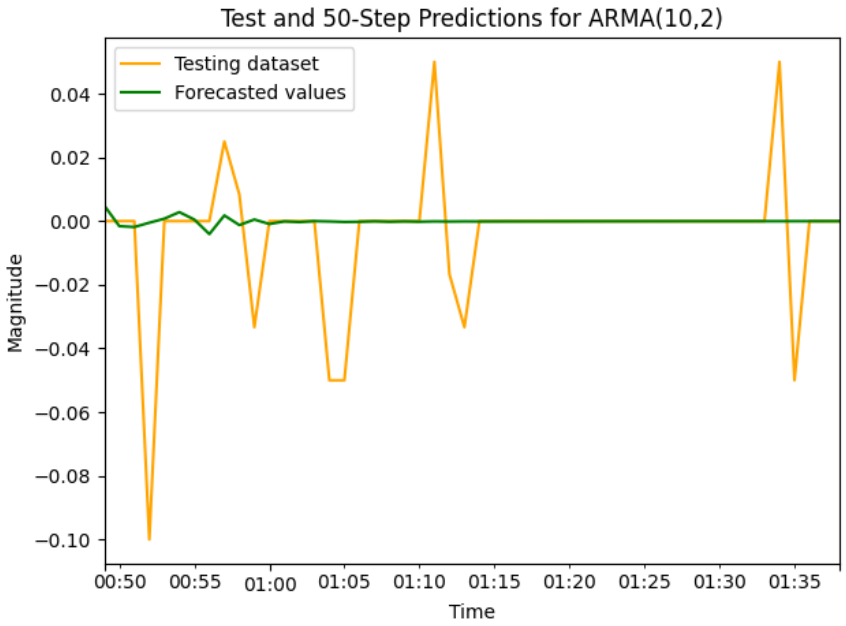

Figure 79: Plot of test and 50-step prediction

Observations: We can observe that the model is able to capture the variation at initial steps. But as the forecast horizon is increased the forecasted values are close to zero or we can see a flat prediction.

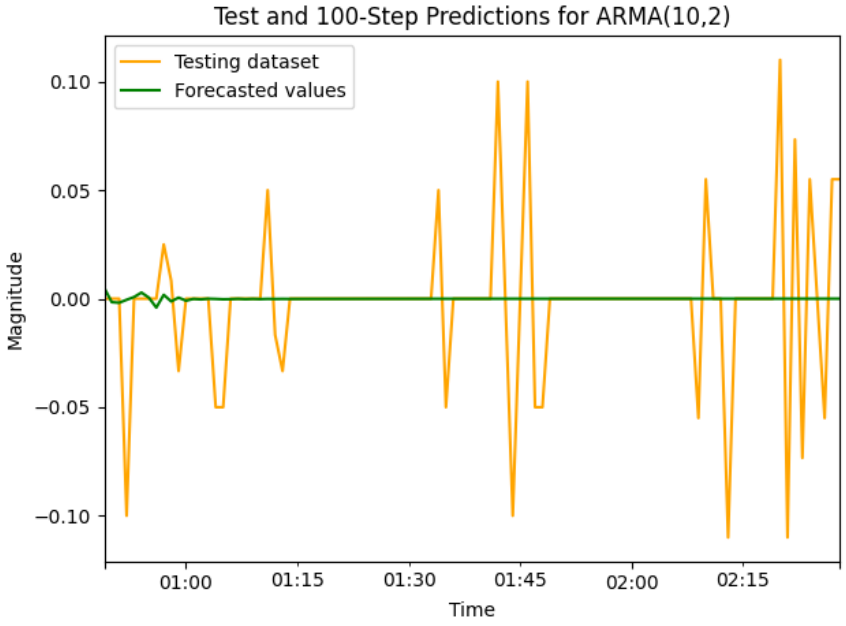

Figure 80: Plot of test and 100-step prediction

Observations: We can observe that the model is able to capture the variation at initial steps. But as the forecast horizon is increased the forecasted values are close to zero or we can see a flat prediction.

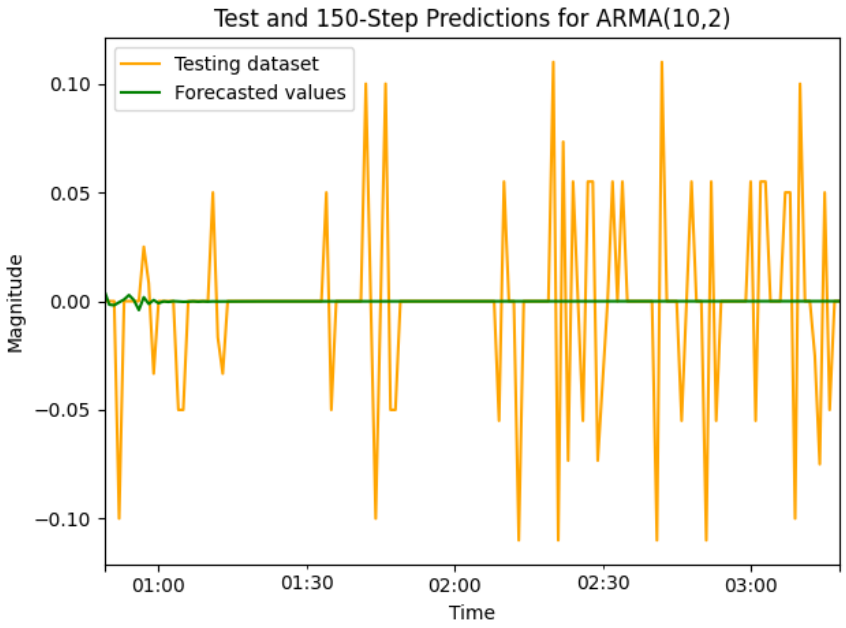

Figure 81: Plot of test and 150-step prediction

Observations: We can observe that the model is able to capture the variation at initial steps. But as the forecast horizon is increased the forecasted values are close to zero or we can see a flat prediction.

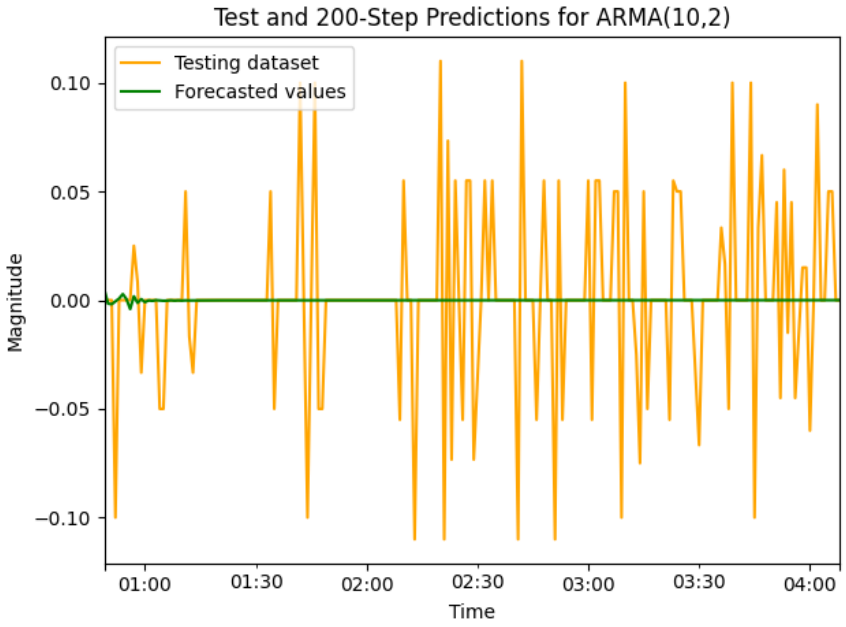

Figure 82: Plot of test and 200-step prediction

Observations: We can observe that the model is able to capture the variation at initial steps. But as the forecast horizon is increased the forecasted values are close to zero or we can see a flat prediction.

15 Conclusion

In conclusion, the analysis of the temperature dataset through basic techniques, OLS, ARMA, and ARIMA models has been experimented successfully. The data was transformed to be stationary, facilitating the application of basic techniques and the utilization of GPAC to assess the significance of AR and MA approaches. Despite SARIMA modestly improving the variance ratio of residuals over predictions, the leading ARMA (10, 2) model struggled in generalizing the data. The multiple linear regression model exhibited better predictive capabilities compared to univariate models.

For future work, considering Deep Learning models such as LSTM, DeepAR, and N-BEATS along with GridCV to determine the optimal model order may lead to improved model generalization. Overall, this project underscores the importance of selecting suitable models for time series analysis and emphasizes the ongoing exploration and enhancement of existing methods for better forecasting performance.

16 Appendix

16.1 Main

```
import importlib

data_preparation = importlib.import_module('01_data_preparation')
eda = importlib.import_module('02_eda')
stationarity = importlib.import_module('03_stationarity')
decomposition = importlib.import_module('04_decomposition')
data_preprocessing = importlib.import_module('05_data_preprocessing')
holt_winter = importlib.import_module('06_holt_winter')
base_models = importlib.import_module('07_base_models')
regression = importlib.import_module('08_regression')
arma = importlib.import_module('09_ARMA')
lm = importlib.import_module('10_LM')
auto_arima = importlib.import_module('11_auto_ARIMA')
hstep = importlib.import_module('12_Hstep')
```

16.2 Data Gathering

```
import pandas as pd
from tabulate import tabulate
import helperfunctions as hf  # custom function files

data_path = "../Data/"

temp_df = pd.read_csv(data_path+"raw_data.csv", index_col=0)

date = pd.date_range(
    start="2015-02-11 14:48",
    end='2015-02-18 09:19',
    freq='min'
)
temp_df['date'] = date
temp_df.index = temp_df['date']
temp_df.drop(columns=['date'], inplace=True)

print(f"\nTotal records: {temp_df.shape[0]}\n")
hf.print_tab(temp_df.head())
temp_df.to_csv("../Data/temp_data.csv")
```

16.3 EDA

```
import seaborn as sns
import matplotlib.pyplot as plt

import helperfunctions as hf  # custom function files

data_path = "../Data/"

target = 'Temperature'

temp_df = hf.make_df(data_path+"temp_data.csv", disp_samp=1)

min_date = temp_df.index.min()
max_date = temp_df.index.max()

print(f"\nEarliest Date: {min_date}")
print(f"Latest Date: {max_date}")

temp_df[target].plot()
plt.xlabel("Date")
plt.ylabel("USD($)")
plt.title("Target to predict: " + target)
plt.tight_layout()
plt.grid()
plt.show()

for i, col in enumerate(temp_df.columns):
    if col == 'Occupancy': continue
    temp_df[col].plot()
    plt.xlabel("Date")
    plt.ylabel("USD($)")
    plt.title(col)
    plt.tight_layout()
    plt.grid()
    plt.show()

print(temp_df.info())

hf.print_tab(temp_df.describe())

cor = temp_df.corr()
sns.heatmap(data = cor, annot = True, vmin=-1, vmax=1)
plt.tight_layout()
plt.show()

hf.plot_autocorr(temp_df[target].values, lags=20,
```

```
title=f"ACF Plot for {target}")
hf.plot_acf_pcf(temp_df[target].values, lags=20,
title=f"ACF/PACF Plot for {target}")
```

16.4 Stationarity

```
import numpy as np
import matplotlib.pyplot as plt

import helperfunctions as hf  # custom function files

data_path = "../Data/"
temp_data = hf.make_df(data_path+"temp_data.csv")
target = 'Temperature'

# Stationarity

print(f"\nChecking if dependable variable is stationary:")
hf.check_stationarity_init(temp_data[target], target)

print(f"\nChecking if dependable variable is stationary after
first differencing:")
temp_diff_1 = hf.non_seas_diff(temp_data[target])
hf.check_stationarity_init(temp_diff_1[1:], target)

print(); print(hf.adf_cal(temp_diff_1[1:]), end='\n\n')

print(hf.kpss_test(temp_diff_1[1:]))

temp_diff_1.plot.hist()
plt.xlabel("value")
plt.title("Histogram of 1st order differencing value")
plt.show()

hf.plot_autocorr(temp_diff_1[1:].values, lags=20,
title="ACF Plot for 1st order differencing value")
hf.plot_acf_pcf(temp_diff_1[1:].values, lags=20,
title="ACF Plot for 1st order differencing value")

temp_data[target] = temp_diff_1

temp_data[target].plot()
plt.xlabel("Time")
plt.ylabel("Value")
```

```
plt.title("Temperature over time")
plt.tight_layout()
plt.grid()
plt.show()

temp_data.dropna(inplace=True)
temp_data.to_csv(data_path+'temp_st.csv')
```

16.5 Decomposition

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tabulate import tabulate
from statsmodels.tsa.seasonal import STL

import helperfunctions as hf  # custom function files

data_path = "../Data/"

target = 'Temperature'

temp_df = hf.make_df(data_path+"temp_st.csv")

stl = STL(temp_df[target], period=1440)
res = stl.fit()

trend = np.array(res.trend)
season = np.array(res.seasonal)
residual = np.array(res.resid)

Ft = max(0, 1 - np.var(residual)/np.var(trend + residual))
print(f"\nThe strength of trend for this data set is
{round(Ft, 4)} or {round(Ft*100, 2)}%")

Fs = max(0, 1 - np.var(residual)/np.var(season + residual))
print(f"\nThe strength of seasonality for this data set is
{round(Fs, 4)} or {round(Fs*100, 2)}%")

res.plot()
plt.show()

plt.figure(figsize=(16, 8))
plt.plot(temp_df.index, trend, label='trend')
```

```
plt.plot(temp_df.index, season, label='Seasonal')
plt.plot(temp_df.index, residual, label='Residuals')
plt.xlabel('Year')
plt.ylabel(target)
plt.title('STL Decomposition')
plt.legend(loc='upper right')
plt.tight_layout()
plt.show()
```

16.6 Data Preprocessing

```
from tabulate import tabulate
from sklearn.model_selection import train_test_split

import helperfunctions as hf  # custom function files

data_path = "../Data/"
mode_data_path = "../Data/Model Data/"

target = 'Temperature'

temp_df = hf.make_df(data_path+"temp_data.csv")

temp_st_df = hf.make_df(data_path+"temp_st.csv")

X_train, X_test, y_train, y_test = train_test_split(
    temp_df.drop(columns=[target]),
    temp_df[target],
    shuffle=False,
    test_size=0.2
)

_, _, y_st_train, y_st_test = train_test_split(
    temp_st_df.drop(columns=[target]),
    temp_st_df[target],
    shuffle=False,
    test_size=0.2
)


def min_max_norm(series):
    sr_min = min(series)
    sr_max = max(series)
    normalized_series = (series - sr_min)/(sr_max - sr_min)
```

```
    return normalized_series


for col in X_train.columns:
    X_train[col] = min_max_norm(X_train[col])

for col in X_test.columns:
    X_test[col] = min_max_norm(X_test[col])

print(f'\nTraining set size: {X_train.shape[0]} rows and
{X_train.shape[1]+1} columns')
hf.print_tab(X_train.describe())

print(f'Testing set size: {X_test.shape[0]} rows and
{X_test.shape[1]+1} columns')
hf.print_tab(X_test.describe())

X_train.to_csv(mode_data_path+'X_train.csv')
X_test.to_csv(mode_data_path+'X_test.csv')
y_train.to_csv(mode_data_path+'y_train.csv')
y_test.to_csv(mode_data_path+'y_test.csv')
y_st_train.to_csv(mode_data_path+'y_st_train.csv')
y_st_test.to_csv(mode_data_path+'y_st_test.csv')
```

16.7 Holt-Winters

```
import helperfunctions as hf  # custom function files
from statsmodels.tsa.holtwinters import ExponentialSmoothing

mode_data_path = "../Data/Model Data/"

target = 'Temperature'

y_train = hf.make_df(mode_data_path+'y_st_train.csv', 0)[target]
y_test = hf.make_df(mode_data_path+'y_st_test.csv', 0)[target]

y_train.index.freq = 'T'
y_test.index.freq = 'T'

model = ExponentialSmoothing(y_train, seasonal='add',
seasonal_periods=365).fit()
y_pred = model.predict(start=y_test.index[0], end=y_test.index[-1])

hf.plot_forecast(y_train, y_test, y_pred, 'Holt-Winter Method',
```

```
x_label='Time', y_label='Magnitude')
```

16.8 Base Models

```
import helperfunctions as hf  # custom function files
from tabulate import tabulate
import pandas as pd

mode_data_path = "../Data/Model_Data/"

target = 'Temperature'

y_train = hf.make_df(mode_data_path+'y_st_train.csv', 0)[target]
y_test = hf.make_df(mode_data_path+'y_st_test.csv', 0)[target]

summary_res = pd.DataFrame(columns=range(10))

# ---------------------------------------------------------------------
# Average Method
# ---------------------------------------------------------------------
hf.print_h('Average_Forecasting_Method')

avg_train_pred, avg_forecast = hf.cal_average_forecast(y_train, y_test)
hf.plot_forecast(avg_train_pred.y, avg_forecast.y,
avg_forecast.y_pred, title='Average_Forecasting_Method')
res_e = (y_train - avg_train_pred.y_pred)[1:]
pred_e = (y_test - avg_forecast.y_pred)
error_stat_df, error_stat_avg =
hf.cal_error_stat(res_e, pred_e, nm='Average')
hf.print_tab(error_stat_df)
summary_res.loc[len(summary_res)] = error_stat_df

# ---------------------------------------------------------------------
# Naive Method
# ---------------------------------------------------------------------
hf.print_h('Naive_Method')

nv_train_pred, nv_forecast = hf.cal_naive_forecast(y_train, y_test)
hf.plot_forecast(nv_train_pred.y, nv_forecast.y,
nv_forecast.y_pred, title='Naive_Method')
res_e = (y_train - nv_train_pred.y_pred)[2:]
pred_e = (y_test - nv_forecast.y_pred)
error_stat_df, error_stat_avg =
hf.cal_error_stat(res_e, pred_e, nm='Naive')
```

```
hf.print_tab(error_stat_df)
summary_res.loc[len(summary_res)] = error_stat_df

# ------------------------------------------------------------------------
# Drift Method
# ------------------------------------------------------------------------
hf.print_h('Drift Method')

df_train_pred, df_forecast = hf.cal_drift_forecast(y_train, y_test)
hf.plot_forecast(df_train_pred.y, df_forecast.y,
df_forecast.y_pred, title='Drift Method')
res_e = (y_train - df_train_pred.y_pred)[2:]
pred_e = (y_test - df_forecast.y_pred)
error_stat_df, error_stat_avg =
hf.cal_error_stat(res_e, pred_e, nm='Drift')
hf.print_tab(error_stat_df)
summary_res.loc[len(summary_res)] = error_stat_df


# ------------------------------------------------------------------------
# SES Method
# ------------------------------------------------------------------------
hf.print_h('SES Method')

ses_train_pred, ses_forecast = hf.cal_ses_forecast(y_train, y_test, 0.5)
hf.plot_forecast(ses_train_pred.y, ses_forecast.y,
ses_forecast.y_pred, title='Simple Exponential Smoothening Method')
res_e = (y_train - ses_train_pred.y_pred)[2:]
pred_e = (y_test - ses_forecast.y_pred)
error_stat_df, error_stat_avg =
hf.cal_error_stat(res_e, pred_e, nm='SES')
hf.print_tab(error_stat_df)
summary_res.loc[len(summary_res)] = error_stat_df

hf.print_tab(summary_res)
```

16.9 Feature Selection and Regression

```
import helperfunctions as hf  # custom function files
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import statsmodels.api as sm
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.decomposition import PCA

mode_data_path = "../Data/Model_Data/"

target = 'Temperature'

X_train = hf.make_df(mode_data_path + 'X_train.csv', 0)
y_train = hf.make_df(mode_data_path + 'y_train.csv', 0)[target]
X_test = hf.make_df(mode_data_path + 'X_test.csv', 0)
y_test = hf.make_df(mode_data_path + 'y_test.csv', 0)[target]

summary_res = pd.DataFrame(columns=range(11))

# ---------------------------------------------------------------------
#  Singular values and Condition number
# ---------------------------------------------------------------------
hf.print_h("Singular values and Condition number")

s, d, v = np.linalg.svd(X_train, full_matrices=True)

print(f'singular values of x are {d}')
print(f'The condition number for x is {np.linalg.cond(X_train)}')

# ---------------------------------------------------------------------
# Coefficients using Normal equation
# ---------------------------------------------------------------------
hf.print_h("Coefficients using Normal equation")

beta_hat = hf.calc_lse(X_train, y_train)
lse_df = pd.DataFrame({
    'feature': ['const'] + X_train.columns.to_list(),
    'Beta': beta_hat
})
print(f"\nThe regression model unknown coefficients using the Normal
equation")
hf.print_tab(lse_df)

# ---------------------------------------------------------------------
# OLS Model with all features
# ---------------------------------------------------------------------
hf.print_h("OLS Model with all features")

X_train = sm.add_constant(X_train, prepend=True)
X_test = sm.add_constant(X_test, prepend=True)
```

```
ols_model_all_features = hf.calc_ols(X_train, y_train)
temp_df = np.round(ols_model_all_features.params, 2).reset_index()
lse_df = pd.DataFrame({
    'feature': temp_df['index'],
    'Beta': temp_df[0]
})
print("\nThe regression model unknown coefficients using OLS method")
hf.print_tab(lse_df)

r2 = ols_model_all_features.rsquared_adj
print(f"\n\nR-squared value model with all {X_train.shape[1]}
features: {r2}")
y_train_pred = ols_model_all_features.predict(X_train)
y_pred = ols_model_all_features.predict(X_test)
hf.plot_forecast(y_train, [], y_train_pred, 'Training Prediction of
Temperature using OLS method', y_label='Temperature',x_label="Index")
hf.plot_forecast([], y_test, y_pred, 'Forecasting Temperature using
OLS method', y_label='Temperature', x_label="Index")

res_e = (y_train - y_train_pred)
pred_e = (y_test - y_pred)
error_stat_df, error_stat = hf.cal_error_stat(res_e, pred_e, r2=r2,
nm='All Features')
hf.print_tab(error_stat_df)
summary_res.columns = error_stat_df.columns
summary_res.loc[len(summary_res)] = error_stat

# ------------------------------------------------------------------
# BSR Feature Selection
# ------------------------------------------------------------------
hf.print_h("BSR Feature Selection")

res_df, imp_features_bsr, features_to_drop_bsr =
hf.backward_stepwise_regression_fs(X_train, y_train, logs=1)

print(f"\n\nBSR | Important Features: {imp_features_bsr}")
print(f"BSR | Features to drop: {features_to_drop_bsr}")

X_train_BSR = X_train[imp_features_bsr]
X_test_BSR = X_test[imp_features_bsr]

ols_model_imp_features = sm.OLS(y_train, X_train_BSR).fit()
r2 = ols_model_imp_features.rsquared_adj
print(f"\n\nR-squared value model with all {X_train_BSR.shape[1]}
features: {ols_model_imp_features.rsquared_adj}")
```

```
y_train_pred = ols_model_imp_features.predict(X_train)
y_pred = ols_model_imp_features.predict(X_test_BSR)
hf.plot_forecast(y_train, [], y_train_pred, 'Training␣Prediction␣of
Temperature␣with␣BSR␣reduced␣features',y_label='Temperature',
x_label="Index")
hf.plot_forecast([], y_test, y_pred, 'Forecasting␣Temperature␣with
BSR␣reduced␣features', y_label='Temperature',x_label="Index")

res_e = (y_train - y_train_pred)
pred_e = (y_test - y_pred)
error_stat_df, error_stat = hf.cal_error_stat(res_e, pred_e, r2=r2,
nm='Features␣from␣BSR')
hf.print_tab(error_stat_df)
summary_res.loc[len(summary_res)] = error_stat

# ------------------------------------------------------------------
# VIF Feature Selection
# ------------------------------------------------------------------
hf.print_h("VIF␣Feature␣Selection")

imp_features_vif, features_to_drop_vif = hf.vif_fs(X_train, y_train
, logs=1)

print(f"\n\nVIF␣|␣Important␣Features:␣{imp_features_vif}")
print(f"VIF␣|␣Features␣to␣drop:␣{features_to_drop_vif}")

X_train_VIF = X_train[imp_features_vif]
X_test_VIF = X_test[imp_features_vif]

ols_model_imp_features = sm.OLS(y_train, X_train_VIF).fit()
r2 = ols_model_imp_features.rsquared_adj
print(f"\n\nR-squared␣value␣model␣with␣all␣{X_train_VIF.shape[1]}
features:␣{r2}")

y_train_pred = ols_model_imp_features.predict(X_train)
y_pred = ols_model_imp_features.predict(X_test_VIF)
hf.plot_forecast(y_train, [], y_train_pred, 'Training␣Prediction␣of
Temperature␣with␣VIF␣reduced␣features',
y_label='Temperature', x_label="Index")
hf.plot_forecast([], y_test, y_pred, 'Forecasting␣Temperature␣with
VIF␣reduced␣features', y_label='Temperature',x_label="Index")

res_e = (y_train - y_train_pred)
pred_e = (y_test - y_pred)
```

```
error_stat_df, error_stat = hf.cal_error_stat(res_e, pred_e, r2=r2,
nm='Features from VIF')
hf.print_tab(error_stat_df)
summary_res.loc[len(summary_res)] = error_stat

# ------------------------------------------------------------------------
# PCA
# ------------------------------------------------------------------------
hf.print_h("PCA")

sc = StandardScaler()
pca = PCA(n_components='mle')

X_train_PCA = X_train.drop(columns=['const'])
X_test_PCA = X_test.drop(columns=['const'])

X_train_PCA = pca.fit_transform(X_train_PCA)
X_test_PCA = pca.fit_transform(X_test_PCA)

X_train_PCA = sm.add_constant(X_train_PCA, prepend=True)
X_test_PCA = sm.add_constant(X_test_PCA, prepend=True)

ols_model_pca_features = sm.OLS(y_train, X_train_PCA).fit()
r2 = ols_model_pca_features.rsquared_adj
print(f"\n\nR-squared value model with all {X_train_PCA.shape[1]}
features: {r2}")

y_train_pred = ols_model_pca_features.predict(X_train_PCA)
y_train_pred = pd.Series(y_train_pred, index=y_train.index)
y_pred = ols_model_pca_features.predict(X_test_PCA)
y_pred = pd.Series(y_pred, index=y_test.index)
hf.plot_forecast(y_train, [], y_train_pred, 'Training Prediction of
Temperature with PCA reduced features',
y_label='Temperature', x_label="Index")
hf.plot_forecast([], y_test, y_pred, 'Forecasting Temperature with
PCA reduced features', y_label='Temperature',x_label="Index")

res_e = (y_train - y_train_pred)
pred_e = (y_test - y_pred)
error_stat_df, error_stat = hf.cal_error_stat(res_e, pred_e, r2=r2,
nm='Features from PCA')
hf.print_tab(error_stat_df)
summary_res.loc[len(summary_res)] = error_stat

hf.print_tab(summary_res)
```

16.10 ARMA

```
import pandas as pd
import matplotlib.pyplot as plt
import helperfunctions as hf  # custom function files
import statsmodels.api as sm
import numpy as np
import scipy.stats as stats

mode_data_path = "../Data/Model_Data/"
target = 'Temperature'
lags = 30

y_train = hf.make_df(mode_data_path + 'y_st_train.csv')[target]
y_test = hf.make_df(mode_data_path + 'y_st_test.csv')[target]

order_lst = [(5, 0, 6), (8, 0, 1), (10, 0, 2), (11, 0, 7)]
# order_lst = [(5, 0, 6), (8, 0, 1)]

res_df = pd.DataFrame(columns=range(9))

for order in order_lst:

    hf.print_h(f"ARIMA{order}")

    model = sm.tsa.SARIMAX(y_train, order=order)
    model_fit = model.fit()
    print(model_fit.summary())

    # simulate forecast function and compute errors
    y_train_hat = model_fit.predict()
    hf.plot_forecast(y_train[:100], [], y_train_hat[:100],
    f'Train and One-Step Predictions for ARIMA{order}')

    y_pred = model_fit.forecast(steps=len(y_test))
    hf.plot_forecast([], y_test[:100], y_pred[:100],
    f'Test and One-Step Predictions for ARIMA{order}')

    hf.plot_forecast(y_train, y_test, y_pred,
    f'Train, Test and One-Step Predictions for ARIMA{order}')

    model_fit.plot_diagnostics(figsize=(14, 10))
    plt.suptitle(f'ARIMA{order} Diagnostic Analysis')
    plt.grid()
    plt.show()
```

```
    # Residual errors
    res_e = y_train - y_train_hat
    pred_e = y_test - y_pred

    re = []
    for lag in range(0, lags + 1):
        re.append(hf.cal_autocorr(res_e, lag))

    hf.plot_autocorr(y=res_e, lags=lags,
    title=f"Autocorrelation of Residuals for ARIMA{order}")
    hf.plot_acf_pcf(res_e, lags=lags)

    Q = len(y_train) * np.sum(np.square(re[lags:]))

    DOF = lags - sum(order)
    alfa = 0.01
    chi_critical = stats.chi2.ppf(1 - alfa, DOF)

    error_stat = hf.cal_error_stat(res_e, pred_e)

    q_stats = {
        "Model": f"ARIMA{order}",
        "Q-Value": round(Q, 4),
        'Critical-Value': chi_critical,
        "White-Residual": 'Yes' if Q < chi_critical else 'No'
    }
    q_stats.update(error_stat)

    if len(res_df):
        res_df.loc[len(res_df)] = q_stats
    else:
        res_df.columns = list(q_stats.keys())
        res_df.loc[len(res_df)] = q_stats

hf.print_tab(res_df)
```

16.11 LM

```
import pandas as pd

import helperfunctions as hf  # custom function files
import numpy as np
```

```
round_off = 3
np.random.seed(6313)
na = 10
nb = 2
mode_data_path = "../Data/Model Data/"
target = 'Temperature'

y = hf.make_df(mode_data_path + 'y_st_train.csv')[target]

coeff_df = pd.DataFrame(columns=[' ', 'Coefficient'])

theta, sse, var_error, covariance_theta_hat, sse_list =
hf.lm_step3(y, na, nb)

theta2 = np.array(theta).reshape(-1)

for i in range(na + nb):
    if i < na:
        coeff_df.loc[len(coeff_df)] = {
            ' ': f"AR coefficient {i + 1}",
            'Coefficient': (np.round(theta2[i], round_off))
        }
    else:
        coeff_df.loc[len(coeff_df)] = {
            ' ': f"MA coefficient {i + 1}",
            'Coefficient': (np.round(theta2[i], round_off))
        }

hf.print_tab(coeff_df)

hf.lm_confidence_interval(theta, covariance_theta_hat, na, nb,
round_off=round_off)

print(f"\nEstimated Covariance Matrix of estimated parameters:
\n{np.round(covariance_theta_hat, decimals=round_off)}")

print(f"Estimated variance of error: {round(var_error, round_off)}")

hf.lm_find_roots(theta, na, round_off=round_off)

hf.plot_sse(sse_list, f"ARMA({na},{nb})")
```

16.12 Auto ARIMA

```
import helperfunctions as hf  # custom function files
from sktime.forecasting.arima import AutoARIMA
import pandas as pd
import statsmodels.api as sm

mode_data_path = "../Data/Model_Data/"
target = 'Temperature'

y_train = hf.make_df(mode_data_path + 'y_st_train.csv')[target]
y_test = hf.make_df(mode_data_path + 'y_st_test.csv')[target]

# p -> AR of non-seasonal
# P -> AR of seasonal
# q -> MA of non-seasonal
# Q -> MA of seasonal
# d -> non-seasonal differencing
# D -> seasonal differencing

forecast = AutoARIMA(
    start_p=0,
    max_p=10,
    start_q=0,
    max_q=10,
    start_Q=0,
    max_Q=5,
    max_d=5,
    max_D=5,
    stationary=True,
    n_fits=20,
    stepwise=False
)

auto_model = forecast.fit(y_train)
print(auto_model.summary())

model = sm.tsa.SARIMAX(y_train, order=(1,0,2))
model_fit = model.fit()
y_train_hat = model_fit.predict()
y_pred = model_fit.forecast(steps=len(y_test))

# Residual errors
res_e = y_train - y_train_hat
pred_e = y_test - y_pred

res_df = pd.DataFrame(columns=range(10))
```

```
_, error_stat = hf.cal_error_stat(res_e, pred_e, nm=f"ARMA{(1,2)}")

if len(res_df):
    res_df.loc[len(res_df)] = error_stat
else:
    res_df.columns = list(error_stat.keys())
    res_df.loc[len(res_df)] = error_stat

hf.print_tab(res_df)
```

16.13 H-Step Prediction

```
import pandas as pd
import matplotlib.pyplot as plt
import helperfunctions as hf  # custom function files
import statsmodels.api as sm

mode_data_path = "../Data/Model Data/"
target = 'Temperature'

y_train = hf.make_df(mode_data_path + 'y_st_train.csv')[target]
y_test = hf.make_df(mode_data_path + 'y_st_test.csv')[target]

steps = [50, 100, 150, 200]

model = sm.tsa.SARIMAX(y_train, order=(10, 0, 2))
model_fit = model.fit()

for step in steps:
    y_pred = model_fit.forecast(steps=step)
    hf.plot_forecast([], y_test[:step], y_pred,
    f'Test and {step}-Step Predictions for ARMA(10,2)')
```

16.14 Toolkit

```
import warnings
from statsmodels.tools.sm_exceptions import InterpolationWarning
from tabulate import tabulate
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller, kpss
```

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
import math
from scipy import signal
import seaborn as sns
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

warnings.simplefilter('ignore', InterpolationWarning)
warnings.filterwarnings("ignore")


def print_tab(df):
    print(tabulate(df, headers='keys', tablefmt='psql'))


def print_h(text):
    print("\n\n" + "-"*100)
    print("␣"*40 + text)
    print("-" * 100 + "\n\n")


def make_df(path, disp_samp=1):
    df = pd.read_csv(path)
    df['date'] = pd.to_datetime(df['date'])
    df.index = df['date']
    df.drop(columns=['date'], inplace=True)

    if disp_samp:
        print_tab(df.head())

    return df


def get_data_repo_url():
    return "https://raw.githubusercontent.com/rjafari979/"+
    "Information-Visualization-Data-Analytics-Dataset-/main/"


def cal_rolling_mean_var(data):
    roll_mean = []
    roll_var = []
    for i in range(0, len(data)):
        roll_mean.append(np.mean(data[:i + 1]))
        roll_var.append(np.var(data[:i + 1]))
    return roll_mean, roll_var
```

```
def plot_rol_mean_var(data, column_name, main_title=''):

    if main_title == '':
        main_title = 'Plot of Rolling Mean and Variance of {0}'
        .format(column_name)

    roll_mean, roll_var = cal_rolling_mean_var(data)

    f, axes = plt.subplots(2)

    axes[0].plot(np.arange(1, len(roll_mean) + 1), roll_mean,
    label='Varying mean')
    axes[0].set_xlabel('Samples')
    axes[0].set_ylabel('Magnitude')
    axes[0].set_title('Rolling Mean - {}'.format(column_name))
    axes[0].legend()

    axes[1].plot(np.arange(1, len(roll_var) + 1), roll_var,
    label='Varying variance')
    axes[1].set_xlabel('Samples')
    axes[1].set_ylabel('Magnitude')
    axes[1].set_title('Rolling Variance - {}'.format(column_name))
    axes[1].legend()
    plt.tight_layout()
    plt.show()


def adf_cal(ts):
    result = adfuller(ts)

    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: %f' % (key, value))


def kpss_test(ts):
    print('Results of KPSS Test:')

    kpsstest = kpss(ts, 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)


def check_stationarity_init(ts, column_name):
    plot_rol_mean_var(ts, column_name)

    adf_p_val = adfuller(ts)[1]
    kpss_p_val = kpss(ts, regression='c', nlags="auto")[1]

    if adf_p_val < 0.05:
        print("According to ADF test, the time series is stationary")
    else:
        print("According to ADF test, the time series is not stationary")

    if kpss_p_val > 0.05:
        print("According to KPSS test, the time series is stationary")
    else:
        print("According to KPSS test, the time series is not stationary")


def non_seas_diff(ts, order=1):
    counter = 0
    while counter != order:
        ts = ts - ts.shift(periods=1)
        counter += 1
    return ts


def correlation_coefficient_cal(x, y):
    if len(x) == len(y):
        x_bar = np.mean(x)
        y_bar = np.mean(y)
        num = sum((x - x_bar) * (y - y_bar))
        deno = np.sqrt(sum(np.square(x - x_bar))) *
        np.sqrt(sum(np.square(y - y_bar)))
        return round(num / deno, 2)
    else:
        return 0


# Generate random numbers from a normal distribution
```

```
def generate_normal_distribution(mean=0, variance=1,
num_observations=1000):
    np.random.seed(6313)
    random_numbers = np.random.normal(mean, np.sqrt(variance),
    num_observations)
    return random_numbers


# White noise data
def generate_white_noise(mean=0, std=1, sample_size=100):
    np.random.seed(6313)
    return np.random.normal(mean, std, sample_size)


# Auto Correlation Function
def cal_autocorr(y, lag):
    mean_y = np.mean(y)
    numerator = 0
    denominator = 0

    for t in range(0, len(y)):
        denominator += (y[t] - mean_y) ** 2

    for t in range(lag, len(y)):
        numerator += (y[t] - mean_y) * (y[t - lag] - mean_y)

    return numerator / denominator


# Auto Correlation Function graph
def plot_autocorr(y, lags, title='', show_plot=1):
    ryy = []
    ryy_final = []
    lags_final = []

    for lag in range(0, lags + 1):
        ryy.append(cal_autocorr(y, lag))

    ryy_final.extend(ryy[:0:-1])
    ryy_final.extend(ryy)
    lags = list(range(0, lags + 1, 1))
    lags_final.extend(lags[:0:-1])
    lags_final = [value * (-1) for value in lags_final]
    lags_final.extend(lags)
```

```
    markers, stem_lines, baseline = plt.stem(lags_final, ryy_final)
    plt.setp(markers, color='red', marker='o')
    plt.setp(stem_lines, color='#74aad0')
    plt.setp(baseline, color='grey', linewidth=2, linestyle='-')
    plt.axhspan((-1.96 / np.sqrt(len(y))), (1.96 / np.sqrt(len(y))),
    alpha=0.1, color='blue')
    plt.xlabel('Lags')
    plt.ylabel('Magnitude')
    plt.title(title)
    plt.tight_layout()

    if show_plot:
        plt.show()


def cal_average_forecast(train_data, test_data, step=1):
    train_pred_lst = [
        np.nan if i - step < 0
        else np.mean(train_data[:i - step + 1])
        for i in range(len(train_data))
    ]
    test_pred_lst = [np.mean(train_data)] * len(test_data)

    train_pred = pd.DataFrame({
        "y": train_data,
        "y_pred": np.round(train_pred_lst, 2)
    })

    test_pred = pd.DataFrame({
        "y": test_data,
        "y_pred": np.round(test_pred_lst, 2)
    })

    return train_pred, test_pred


def cal_naive_forecast(train_data, test_data, step=1):
    train_pred_lst = [np.nan if i - step < 0 else train_data[i - step]
    for i in range(len(train_data))]
    test_pred_lst = [train_data[len(train_data) - 1]] * len(test_data)

    train_pred = pd.DataFrame({
        "y": train_data,
        "y_pred": np.round(train_pred_lst, 2)
    })
```

```
    test_pred = pd.DataFrame({
        "y": test_data,
        "y_pred": np.round(test_pred_lst, 2)
    })

    return train_pred, test_pred


def cal_drift_forecast(train_data, test_data, step=1):
    len_train_data = len(train_data)
    len_test_data = len(test_data)

    train_pred_lst = [
        np.nan if i - step <= 0
        else train_data[i - step] + ((train_data[i - step] - train_data[0])
        / (i - 1))
        for i in range(len_train_data)
    ]
    test_pred_lst = [
        train_data[len_train_data - 1] + (i + 1) *
        (train_data[len_train_data - 1]
        - train_data[0]) / (len_train_data - 1)
        for i in range(len_test_data)
    ]

    train_pred = pd.DataFrame({
        "y": train_data,
        "y_pred": np.round(train_pred_lst, 2)
    })

    test_pred = pd.DataFrame({
        "y": test_data,
        "y_pred": np.round(test_pred_lst, 2)
    })

    return train_pred, test_pred


def cal_ses_forecast(train_data, test_data, alpha=0):
    len_train_data = len(train_data)
    len_test_data = len(test_data)
    train_pred_lst = []
    test_pred_lst = []
```

```
    for idx in range(len_train_data):
        if idx == 0:
            train_pred_lst.append(train_data[0])
        else:
            train_pred_lst.append((alpha * train_data[idx - 1])
            + ((1 - alpha) * train_pred_lst[idx - 1]))

    for idx in range(len_test_data):
        test_pred_lst.append((alpha * train_data[len_train_data-1])
        + ((1 - alpha) * train_pred_lst[-1]))

    train_pred = pd.DataFrame({
        "y": train_data,
        "y_pred": np.round(train_pred_lst, 2)
    })

    test_pred = pd.DataFrame({
        "y": test_data,
        "y_pred": np.round(test_pred_lst, 2)
    })

    return train_pred, test_pred


def plot_forecast(y_train, y_test, y_pred, title, x_label='Time',
y_label='Magnitude', show_plot=1):
    train_n = len(y_train)
    test_n = len(y_test)
    pred_n = len(y_pred)
    dim = train_n

    if train_n:
        plt.plot([i + 1 for i in range(dim)], y_train,
        label='Training dataset')

    if test_n:
        dim += test_n
        plt.plot([i + 1 for i in range(train_n, dim)],
        y_test, label='Testing dataset', color='orange')

    if pred_n:
        if test_n:
            plt.plot([i + 1 for i in range(train_n, dim)],
            y_pred, label='Forecasted values', color='green')
        else:
```

```
            plt.plot([i + 1 for i in range(dim)], y_pred,
            label='Forecasted values', color='green')

    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.title(title)
    plt.legend()
    plt.tight_layout()
    if show_plot:
        plt.show()


def calc_stats(y, y_pred, skip=0):
    temp_data = pd.DataFrame({
        "y": y.to_list(),
        "y_pred": y_pred.to_list()
    })
    error_lst = y - y_pred

    temp_data['e'] = temp_data.y - temp_data.y_pred
    temp_data['e2'] = np.square(temp_data.e)

    error_w_skip = error_lst.iloc[skip:]
    mse = np.nansum(np.square(error_w_skip)) / len(error_w_skip)
    var = np.var(error_w_skip)
    mean_res_err = np.nanmean(error_w_skip)

    return temp_data, mse, var, mean_res_err


def cal_q_value(data, lags):
    auto_corr_val = [cal_autocorr(data, lag)**2 for
    lag in range(1, lags+1)]
    q_value = len(data) * np.sum(auto_corr_val)
    return np.round(q_value, 2)


def calc_lse(features, target):
    features = sm.add_constant(features, prepend=True)
    beta_hat = np.dot(np.linalg.inv(np.dot(features.T,
    features)), np.dot(features.T, target))
    return beta_hat


def calc_ols(features, target):
```

```
    features = sm.add_constant(features, prepend=True)
    model_ols = sm.OLS(target, features)
    result = model_ols.fit()
    return result


def predict_ols(model, features):
    features = sm.add_constant(features, prepend=True)
    y_pred = model.predict(features)
    return y_pred


def backward_stepwise_regression_fs(inp_features,
inp_target, logs=0):
    features = inp_features.copy(deep=True)
    target = inp_target.copy(deep=True)

    model = sm.OLS(target, features).fit()
    aic, bic, adj_r2 = model.aic.round(2), model.bic.round(2),
    model.rsquared_adj.round(2)

    res_df = pd.DataFrame({
        "dropped_features": ["none"],
        "aic": [aic],
        "bic": [bic],
        "adj_r2": [adj_r2]
    })

    rank_df = pd.DataFrame(model.pvalues[1:],
    index=features.columns, columns=['p-value'])
    rank_df['imp_rank'] = rank_df['p-value'].rank(ascending=False)

    if logs:
        print(f'========Step {1}, include all '+
        '{len(features.columns)-1} ========features\n')
        print(model.summary())
        print('\nAIC = ', aic.round(2))
        print('BIC = ', bic.round(2))
        print('Adj_R2 = ', adj_r2.round(2), '\n')
        print(rank_df)

    features_to_drop = []

    i = 1
    while len(features.columns) > 1:
```

```
        i += 1
        if (rank_df[rank_df.imp_rank == 1]
        ['p-value'] < 0.05).values[0]:
            if logs:
                print("\nAll insignificant features" +
                "are removed hence we stop here.\n")
                print(model.summary())
            break
        else:
            least_imp_feature = rank_df[rank_df.imp_rank == 1]
            .index[0]
            features_to_drop.append(least_imp_feature)
            features.drop(columns=[least_imp_feature], axis=1,
            inplace=True)

            model = sm.OLS(target, features).fit()
            aic, bic, adj_r2 = model.aic.round(2), model.bic.round(2),
            model.rsquared_adj.round(4)

            rank_df = pd.DataFrame(model.pvalues[1:],
            index=features.columns, columns=['p-value'])
            rank_df['imp_rank'] = rank_df['p-value'].
            rank(ascending=False)

            res_df.loc[len(res_df)] = {
                "dropped_features": least_imp_feature,
                "aic": aic,
                "bic": bic,
                "adj_r2": adj_r2
            }
            if logs:
                print(f"\n\n========Step {i}, removing '"+
                "{least_imp_feature}' and include "+
                "{len(features.columns)-1} ========features\n")
                print(model.summary())
                print('\nAIC = ', aic)
                print('BIC = ', bic)
                print('Adj_R2 = ', adj_r2, '\n')
                print(rank_df)

    imp_features = features.columns.to_list()

    if logs:
        print(res_df)
```

```
    return res_df, imp_features, features_to_drop


def cal_vif(data):
    vif_data = pd.DataFrame()
    vif_data['feature'] = data.columns
    vif_data['vif'] = [round(variance_inflation_factor
    (data.values, i), 2) for i in range(len(data.columns))]
    vif_data['imp_rank'] = vif_data['vif'].rank(ascending=False)
    return vif_data


def vif_fs(inp_features, inp_target, logs=0):
    features = inp_features.copy(deep=True)
    target = inp_target.copy(deep=True)

    model = sm.OLS(target, features).fit()
    best_aic, best_bic, best_adj_r2 = model.aic, model.bic,
    model.rsquared_adj

    vif_df = cal_vif(features)

    if logs:
        print(f'\n\n========Step {1}, include all '+
        '{len(features.columns)-1} ========features\n')
        print(model.summary())
        print('\nAIC = ', best_aic.round(2))
        print('BIC = ', best_bic.round(2))
        print('Adj_R2 = ', best_adj_r2.round(4), '\n')
        print(vif_df)

    features_to_drop = []

    i = 2
    while len(features.columns) > 1:

        least_imp_feature = vif_df[vif_df.imp_rank == 1].feature.values[0]
        features_to_drop.append(least_imp_feature)
        temp_features =
        features.drop(columns=[least_imp_feature], axis=1)

        model = sm.OLS(target, temp_features).fit()
        aic, bic, adj_r2 = model.aic, model.bic, model.rsquared_adj

        vif_df = cal_vif(temp_features)
```

```
        if logs:
            print(f"\n\n========Step {i}, removing "+
            "'{least_imp_feature}' and include "+
            "{len(features.columns)-1} "+
            "========features\n")
            print(model.summary())
            print('\nAIC = ', aic.round(2))
            print('BIC = ', bic.round(2))
            print('Adj_R2 = ', adj_r2.round(4), '\n')
            print(vif_df)

        if aic <= best_aic or bic <= best_bic or best_adj_r2 <= adj_r2:
            best_aic = aic
            best_bic = bic
            best_adj_r2 = adj_r2
            i += 1
            features.drop(columns=[least_imp_feature], axis=1,
            inplace=True)
        else:
            if logs: print("\nThere is no improvement in the"+
            "accuracy metrics hence we stop here.")
            features_to_drop.pop()
            break

    imp_features = features.columns.to_list()

    return imp_features, features_to_drop


def calc_moving_average(lst, ma_order):
    lst_len = len(lst)
    end_point = math.floor(ma_order / 2)
    mid_point = math.ceil(ma_order/2)
    ma_res = [np.nan]*lst_len

    if ma_order % 2 != 0:
        for i in range(mid_point, lst_len-end_point+1):
            values = lst[i-end_point-1:i+end_point]
            ma_res[i-1] = np.sum(values)/len(values)
    elif ma_order > 2:
        for i in range(mid_point, lst_len - end_point + 1):
            values = lst[i - end_point:i + end_point]
            ma_res[i-1] = np.sum(values)/len(values)
    elif ma_order == 2:
```

```
        for i in range(1, lst_len):
            values = lst[i - 1:i+1]
            ma_res[i] = np.sum(values)/len(values)

    return np.round(ma_res, 2)


def dlsim_method(n, num, den, mean=0, std=1):
    e = np.random.normal(mean, std, n)
    system = (num, den, 1)
    t, y_dlsim = signal.dlsim(system, e)
    return e.round(4), y_dlsim.reshape(-1).round(4)


def plot_acf_pcf(y, lags, title='ACF/PACF of the raw data'):
    fig = plt.figure()
    plt.subplot(211)
    plt.title(title)
    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()


def get_arma_input():
    n = int(input('Number of observations: '))
    mean_e = int(input('Enter the mean of white noise: '))
    var_e = int(input('Enter the variance of white noise: '))
    lags = int(input('Enter number of lags: '))
    na = int(input('Enter AR order: '))
    nb = int(input('Enter MA order: '))
    den = []
    for i in range(1, na + 1):
        den.append(float(input(f'Enter the coefficient {i} of AR'+
        'process: ')))
    num = []
    for i in range(1, nb + 1):
        num.append(float(input(f'Enter the coefficient {i} of MA'+
        'process: ')))
    max_order = max(na, nb)
    ar_coef = [0] * max_order
    ma_coef = [0] * max_order
    for i in range(na):
        ar_coef[i] = den[i]
```

```
    for i in range(nb):
        ma_coef[i] = num[i]
    ar_params = np.array(ar_coef)
    ma_params = np.array(ma_coef)
    ar = np.r_[1, ar_params]
    ma = np.r_[1, ma_params]
    return n, na, nb, ar, ma, mean_e, var_e, lags


def gpac(ry, show_heatmap=1, j_max=7, k_max=7, round_off=3,
seed=6313):
    np.random.seed(seed)
    gpac_table = np.zeros((j_max, k_max-1))

    for j in range(0, j_max):
        for k in range(1, k_max):
            phi_num = np.zeros((k, k))
            phi_den = np.zeros((k, k))
            for x in range(0, k):
                for z in range(0, k):
                    phi_num[x][z] = ry[abs(j + 1 - z + x)]
                    phi_den[x][z] = ry[abs(j - z + x)]
            phi_num = np.roll(phi_num, -1, 1)
            det_num = np.linalg.det(phi_num)
            det_den = np.linalg.det(phi_den)
            if det_den != 0 and not np.isnan(det_den):
                phi_j_k = det_num / det_den
            else:
                phi_j_k = np.nan
            gpac_table[j][k - 1] = phi_j_k

    if show_heatmap:
        plt.figure(figsize=(16, 8))
        x_axis_labels = list(range(1, k_max))
        sns.heatmap(gpac_table, annot=True, xticklabels=x_axis_labels,
        fmt=f'.{round_off}f', vmin=-0.1, vmax=0.1)
        plt.title(f'GPAC Table', fontsize=18)
        plt.show()

    return gpac_table


def arma_gpac_pacf(j_max=7, k_max=7, precision=2):
    n, na, nb, ar, ma, mean_e, var_e, lags = get_arma_input()
    print('\nAR coefficients: ', ar)
```

```
    print('MA coefficients:', ma)

    arma_process = sm.tsa.ArmaProcess(ar, ma)
    mean_y = mean_e*(1 + np.sum(ar))/(1 + np.sum(ma))
    y = arma_process.generate_sample(n, scale=np.sqrt(var_e)) + mean_y
    print('\nARMA Process:', list(np.around(np.array(y[:15]), precision)))

    ry = arma_process.acf(lags=lags)
    print('\nACF:', list(np.around(np.array(ry[:15]), precision)))

    gpac(ry, j_max=j_max, k_max=k_max, round_off=precision)
    plot_acf_pcf(y, lags=20)


def lm_cal_e(y, na, theta, seed=6313):
    np.random.seed(seed)
    den = theta[:na]
    num = theta[na:]
    if len(den) > len(num):  # matching len of num and den
        for x in range(len(den) - len(num)):
            num = np.append(num, 0)
    elif len(num) > len(den):
        for x in range(len(num) - len(den)):
            den = np.append(den, 0)
    den = np.insert(den, 0, 1)
    num = np.insert(num, 0, 1)
    sys = (den, num, 1)
    _, e = signal.dlsim(sys, y)
    return e


def lm_step1(y, na, nb, delta, theta):
    n = na + nb
    e = lm_cal_e(y, na, theta)
    sse_old = np.dot(np.transpose(e), e)
    X = np.empty(shape=(len(y), n))
    for i in range(0, n):
        theta[i] = theta[i] + delta
        e_i = lm_cal_e(y, na, theta)
        x_i = (e - e_i) / delta
        X[:, i] = x_i[:, 0]
        theta[i] = theta[i] - delta
    A = np.dot(np.transpose(X), X)
    g = np.dot(np.transpose(X), e)
    return A, g, X, sse_old
```

```
def lm_step2(y, na, A, theta, mu, g):
    delta_theta = np.matmul(np.linalg.inv
    (A + (mu * np.identity(A.shape[0]))), g)
    theta_new = theta + delta_theta
    e_new = lm_cal_e(y, na, theta_new)
    sse_new = np.dot(np.transpose(e_new), e_new)
    if np.isnan(sse_new):
        sse_new = 10 ** 10
    return sse_new, delta_theta, theta_new


def lm_step3(y, na, nb):
    N = len(y)
    n = na+nb
    mu = 0.01
    mu_max = 10 ** 20
    max_iterations = 500
    delta = 10 ** -6
    var_error = 0
    covariance_theta_hat = 0
    sse_list = []
    theta = np.zeros(shape=(n, 1))

    for iterations in range(max_iterations):
        A, g, X, sse_old = lm_step1(y, na, nb, delta, theta)
        sse_new, delta_theta, theta_new = lm_step2(y, na, A,
        theta, mu, g)
        sse_list.append(sse_old[0][0])
        if iterations < max_iterations:
            if sse_new < sse_old:
                if np.linalg.norm(np.array(delta_theta), 2) < 10 ** -3:
                    theta_hat = theta_new
                    var_error = sse_new / (N - n)
                    covariance_theta_hat = var_error * np.linalg.inv(A)
                    print(f"\nConvergence Occured in {iterations}"+
                    "iterations")
                    break
                else:
                    theta = theta_new
                    mu = mu / 10
            while sse_new >= sse_old:
                mu = mu * 10
                if mu > mu_max:
```

```
                print('\nNo Convergence')
                break
            sse_new, delta_theta, theta_new = lm_step2(y, na, A,
            theta, mu, g)
        if iterations > max_iterations:
            print('\nMax Iterations Reached')
            break
        theta = theta_new
    return theta_new, sse_new, var_error[0][0],
    covariance_theta_hat, sse_list


def lm_confidence_interval(theta, cov, na, nb, round_off=4):
    print("\nConfidence Interval for the estimated parameter(s)")
    lower_bound = []
    upper_bound = []
    for i in range(len(theta)):
        lower_bound.append(theta[i] - 2 * np.sqrt(cov[i, i]))
        upper_bound.append(theta[i] + 2 * np.sqrt(cov[i, i]))
    lower_bound = np.round(lower_bound, decimals=round_off)
    upper_bound = np.round(upper_bound, decimals=round_off)

    coeff_df =
    pd.DataFrame(columns=[' ', 'Lower Bound', 'Upper Bound'])

    for i in range(na + nb):
        if i < na:
            coeff_df.loc[len(coeff_df)] = {
                ' ': f"AR coefficient {i + 1}",
                'Lower Bound': lower_bound[i][0],
                'Upper Bound': upper_bound[i][0]
            }
        else:
            coeff_df.loc[len(coeff_df)] = {
                ' ': f"MA coefficient {i + 1}",
                'Lower Bound': lower_bound[i][0],
                'Upper Bound': upper_bound[i][0]
            }
    print_tab(coeff_df)


def lm_find_roots(theta, na, round_off=4):
    den = theta[:na]
    num = theta[na:]
    if len(den) > len(num):
```

```
        for x in range(len(den) - len(num)):
            num = np.append(num, 0)
    elif len(num) > len(den):
        for x in range(len(num) - len(den)):
            den = np.append(den, 0)
    else:
        pass
    den = np.insert(den, 0, 1)
    num = np.insert(num, 0, 1)
    print("\nRoots of numerator:", np.round(np.roots(num),
    decimals=round_off))
    print("Roots of denominator:", np.round(np.roots(den),
    decimals=round_off))


def plot_sse(sse_list, model_name):
    plt.plot(sse_list)
    plt.xlabel('Iterations')
    plt.ylabel('SSE')
    plt.title(f'SSE Learning Rate {model_name}')
    plt.xticks(np.arange(0, len(sse_list), step=1))
    plt.show()


def run_lm():
    round_off = 3
    np.random.seed(6313)
    N, na, nb, den, num, mean_e, var_e, lags =
    get_arma_input()
    e, y = dlsim_method(N, num, den, mean=mean_e,
    std=np.sqrt(var_e))
    theta, sse, var_error, covariance_theta_hat, sse_list =
    lm_step3(y, na, nb)
    # Q1
    theta2 = np.array(theta).reshape(-1)
    for i in range(na+nb):
        if i < na:
            print('The AR coefficient {} is: {:.3f}'
            .format(i + 1, np.round(theta2[i], 3)))
        else:
            print('The MA coefficient {} is: {:.3f}'
            .format(i + 1 - na, np.round(theta2[i], 3)))
    # Q2
    lm_confidence_interval(theta, covariance_theta_hat,
    na, nb, round_off=round_off)
```

```
    # Q3
    print(f"\nEstimated Covariance Matrix of estimated "+
    "parameters: \n{np.round(covariance_theta_hat,"+
    "decimals=round_off)}")
    # Q4
    print(f"Estimated variance of error: {round(var_error,"+
    "round_off)}")
    # Q5
    lm_find_roots(theta, na, round_off=round_off)
    # Q6
    plot_sse(sse_list, f"ARMA({na},{nb})")
    print('\nUsing stats model package:')
    arma_process = sm.tsa.ArmaProcess(den, num)
    mean_y = mean_e * (1 + np.sum(den)) / (1 + np.sum(num))
    y = arma_process.generate_sample(N, scale=np.sqrt(var_e)) + mean_y
    model = sm.tsa.ARIMA(y, order=(na, 0, nb))
    results = model.fit()
    for i in range(na):
        print('The AR coefficient {} is: {:.3f}'.format(i+1,
        round(-results.arparams[i], 3)))
    for i in range(nb):
        print('The MA coefficient {} is: {:.3f}'.format(i+1,
        round(results.maparams[i], 3)))
    print(results.summary())


def check_stationarity(y):
    stationary = adf_cal(y)
    rm, rv = cal_rolling_mean_var(y)
    plot_rol_mean_var(rm, rv)
    return stationary


def run_intial_sarima():
    np.random.seed(6313)
    n, na, nb, ar, ma, mean_e, var_e, lags = get_arma_input()
    e, ts = dlsim_method(n, ma, ar, mean=mean_e, std=np.sqrt(var_e))
    plot_acf_pcf(ts, lags=20)
    stationary_1 = check_stationarity(ts)
    arma_process = sm.tsa.ArmaProcess(ar, ma)
    stationary_2 = arma_process.isstationary
    stationary = stationary_1 and stationary_2
    return ts, ar, ma, lags, stationary
```

```
def run_gpac(ar, ma, lags, size=10):
    arma_process = sm.tsa.ArmaProcess(ar, ma)
    ry = arma_process.acf(lags=lags)
    gpac(ry, j_max=size, k_max=size, round_off=2)


def plot_ts(data, diff=0, seas=0):
    plt.plot(data[:500], label='Original')
    if diff:
        plt.plot(diff[seas:500], label='Differenced')
    plt.xlabel('Samples')
    plt.ylabel('ARIMA Process')
    plt.title('Time Series Plot')
    plt.legend()
    plt.tight_layout()
    plt.show()


def cal_error_stat(residual_error, forecast_error, order=1,
nm='Model', rnd=4):
    re = []
    lags = 20

    for lag in range(1, lags + 1):
        re.append(cal_autocorr(residual_error, lag))

    plot_acf_pcf(residual_error, lags=lags)

    Q = sm.stats.acorr_ljungbox(residual_error, lags=[20],
    boxpierce=True, return_df=True)['bp_stat'].values[0]

    DOF = lags - order
    alfa = 0.01
    chi_critical = stats.chi2.ppf(1 - alfa, DOF)

    error_stat = {
        "Model": nm,
        "Q-Value": round(Q, rnd),
        'Critical-Value': chi_critical,
        "White-Residual": 'Yes' if Q < chi_critical else 'No',
        'm_res': np.round(np.mean(residual_error), rnd),
        'mse_res': np.round(np.mean(residual_error ** 2), rnd),
        'var_res': np.round(np.var(residual_error), rnd),
        'm_pred': np.round(np.mean(forecast_error), rnd),
        'mse_pred': np.round(np.mean(forecast_error ** 2), rnd),
```

```
        'var_pred': np.round(np.var(forecast_error), rnd)
    }

    temp = pd.DataFrame(columns=range(10))
    temp.columns = list(error_stat.keys())
    temp.loc[len(temp)] = error_stat

    return temp, error_stat


def cal_reg_stat(model, x, y_tr, y_tr_pred, y_tst, y_tst_pred,
nm='Model', rnd=4):

    print(model.summary())

    residual_error = y_tr - y_tr_pred
    forecast_error = y_tst - y_tst_pred

    re = []
    lags = 20

    for lag in range(1, lags + 1):
        re.append(cal_autocorr(residual_error, lag))

    plot_acf_pcf(residual_error, lags=lags)

    Q = sm.stats.acorr_ljungbox(residual_error, lags=[20],
    boxpierce=True, return_df=True)['bp_stat'].values[0]

    DOF = 20 - 1
    alfa = 0.01
    chi_critical = stats.chi2.ppf(1 - alfa, DOF)

    error_stat = {
        "Model": nm,
        "R2": round(model.rsquared, rnd),
        "Adjusted-R2": round(model.rsquared_adj, rnd),
        "AIC": round(model.aic, rnd),
        "BIC": round(model.bic, rnd),
        "F-test": round(model.f_pvalue, rnd),
        "Mean_Cross-val": round(np.mean(
            cross_val_score(linear_model.LinearRegression(), x,
            y_tst, cv=5)
        ),4),
        "Q-Value": round(Q, rnd),
```

```
    'Critical-Value': chi_critical,
    "White-Residual": 'Yes' if Q < chi_critical else 'No',
    'mse_res': np.round(np.mean(residual_error ** 2), rnd),
    'var_res': np.round(np.var(residual_error), rnd),
    'mse_pred': np.round(np.mean(forecast_error ** 2), rnd),
    'var_pred': np.round(np.var(forecast_error), rnd)
}

temp = pd.DataFrame(columns=range(14))
temp.columns = list(error_stat.keys())
temp.loc[len(temp)] = error_stat

return temp, error_stat
```

References

[1] S. Mauriz, "(simple) linear regression and ols: Introduction to the theory," *Towards Data Science*, 2020.