Time series Analysis and Modelling DATS 6313

(MS In Data Science)

Project Report

Occupancy Detection

Instructor: Reza Jafari

Authors:

Sudhanshu Deshpande

Date: 05/10/2023

TABLE OF CONTENTS

TABLE OF FIGURES AND TABLES

Abstract

The Occupancy Detection dataset is a collection of environmental sensor readings from an office building, used to predict occupancy levels in the rooms. The dataset includes temperature, humidity, light, CO2, and humidity ratio measurements taken every minute over seven days. The goal of the project is to develop a model that can accurately predict occupancy levels based on the sensor data. The project explores various machine learning algorithms, including logistic regression and Time series methods to determine which algorithm provides the best results. The results show that the ARIMA model works best with data. This dataset and project have applications in building energy management, occupancy detection, and indoor environmental quality control.

Introduction

Time series analysis is a branch of statistics and data analysis that deals with data points ordered in time. In time series data, observations are made over a period of time, and the order of these observations is important. Time series data is commonly used in a wide range of fields, including finance, economics, engineering, and environmental science.

In this report, we will perform time series analysis on the Occupancy Detection dataset obtained from the UCI Machine Learning Repository. The dataset contains data collected from an office room, where sensors were used to monitor the room's occupancy, temperature, humidity, and CO2 levels. The objective of this analysis is to predict occupancy levels in the office room based on the other variables.

The report will begin with an exploration of the dataset, including visualizations and summary statistics. We will then perform time series modelling using various techniques, including Naïve Method, Simple Exponential Smoothing, Holt-Winter Model, ARIMA, and drift method. We will evaluate the performance of these models using various metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and AIC and BIC values.

Finally, we will present our conclusions and discuss the strengths and limitations of each model. The report aims to provide an overview of the time series analysis and modelling process, highlighting the importance of time series data in real-world applications.

1) <u>Data Description</u>

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 whether the room is occupied or not based on the sensor readings.

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	in Celsius
Relative Humidity	%
Light	in Lux
CO2	in ppm
Humidity Ratio	Derived quantity from temperature and relative humidity, in kgwater-vapor/kg-air
Occupancy	0 or 1, 0 for not occupied, 1 for occupied status

Table 1: Dataset Variables

a) Pre-processing dataset:

The data dose not have any missing value, any NAN or outliers. So there is no need to perform the pre-processing on dataset.

```
date             0
Temperature      0
Humidity         0
Light            0
CO2              0
HumidityRatio    0
Occupancy        0
dtype: int64
```

Fig 1: Variables with zero missing data

b) Dependent variable versus time

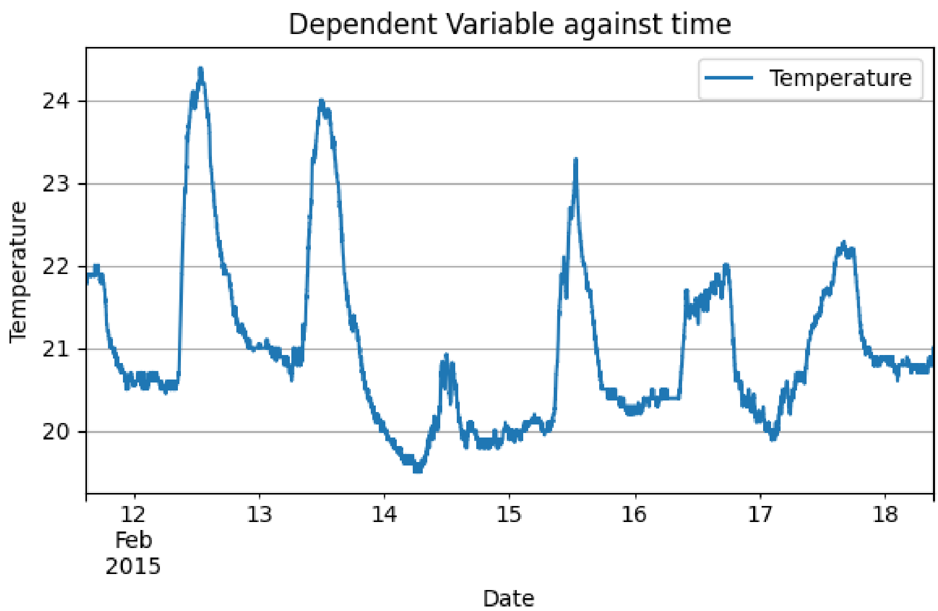


Fig 2: Dependent variable against time

From the plot we can see that the dependent/target variable is not at all stationary. Also, we can see the seasonality in the variable that is we can see there is low temperature at start of the day and it gradually increases over the day and again becomes low at start of the next day.

c) ACF/PACF plot of dependent variable

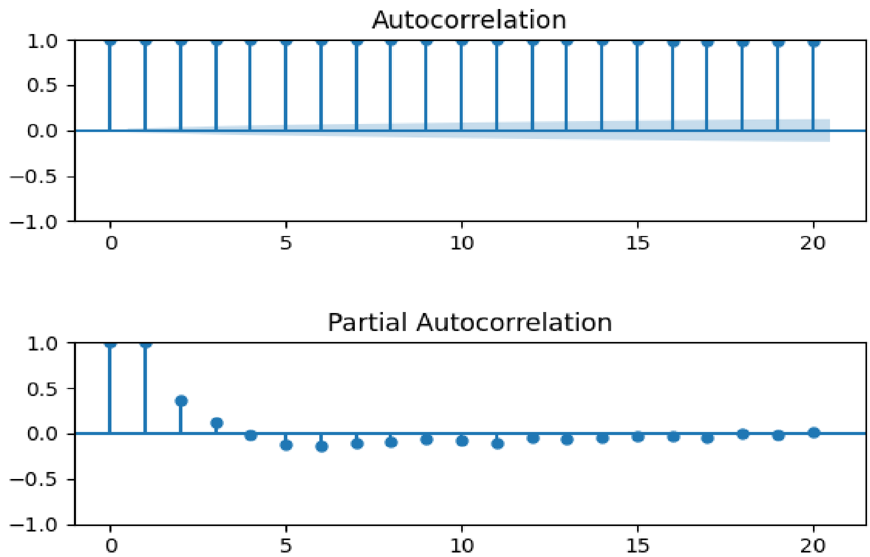


Fig 3: ACF/PACF plot for dependent variable

From the ACF/PACF plot also we can see that the dependent variable is not stationary.

d) Correlation Matrix

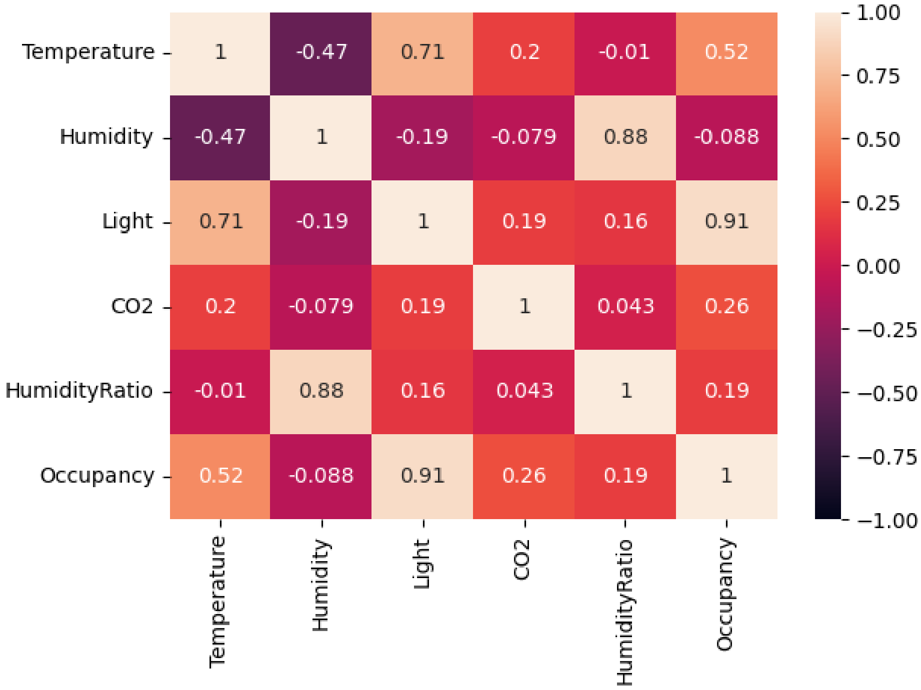


Fig 4: Correlation Matrix

From the correlation matrix we can see that variable light is positively correlated to the temperature, that is when the intensity of the light increases the temperature will also increase. Whereas the variable humidity is negatively correlated to the temperature, that is when the humidity increases the temperature will decrease.

e) Splitting the dataset

```
x = df.drop(['Temperature','date'], axis=1)
y = df['Temperature']
x_train1, x_test1,y_train1, y_test1 = train_test_split(x,y, shuffle=False, test_size=0.20)
```

Fig 5: Splitting dataset

2) Stationarity

- **Checking Stationarity of raw data**

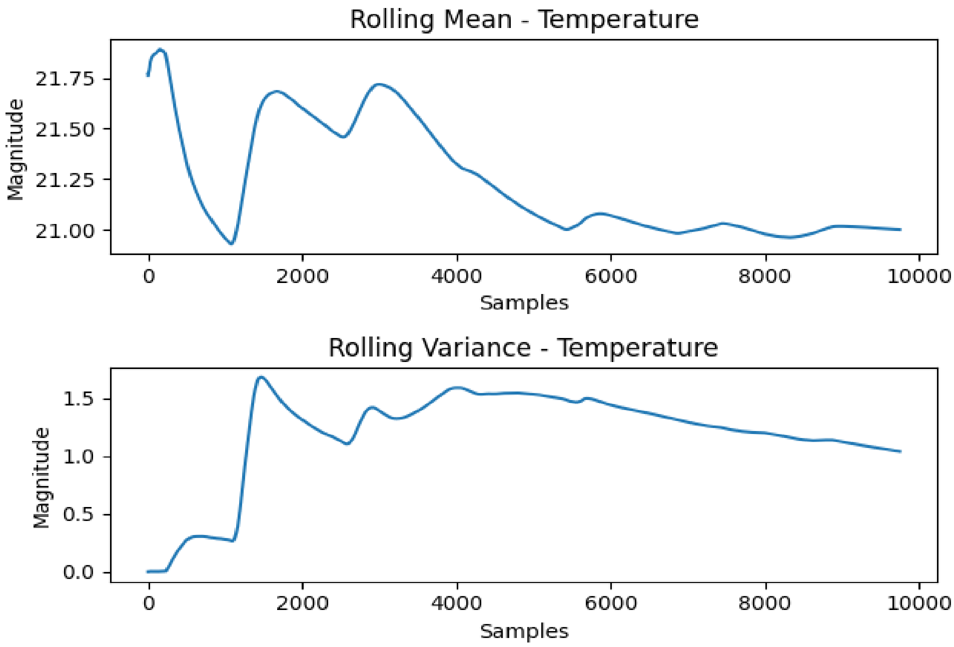


Fig 6: Rolling Mean and Variance of raw data

From the plot we can see the data is not stationary as the mean and variance is not getting stable.

- **Checking Stationarity after differencing (1st Order Non-seasonal)**

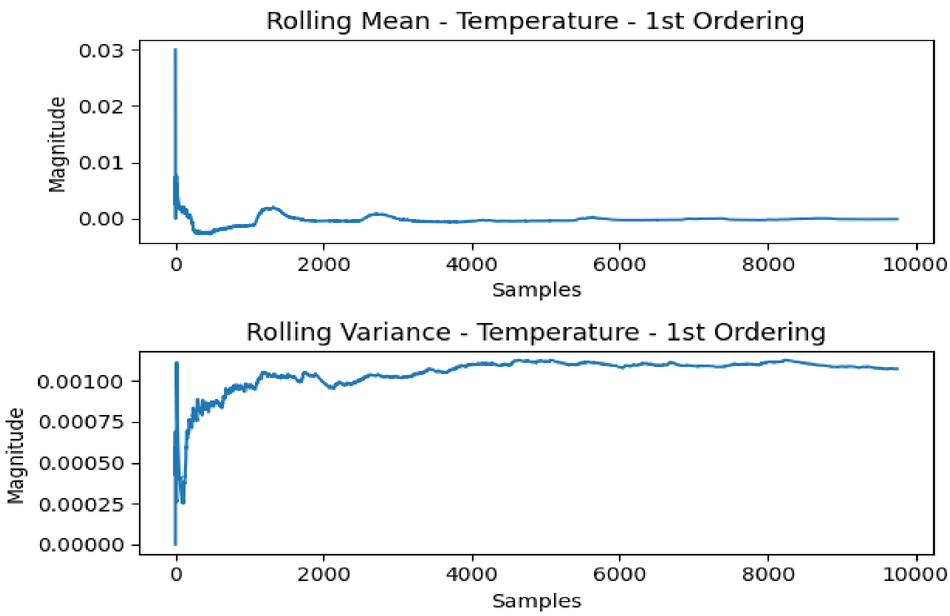


Fig 7: Rolling Mean and Variance of differenced data

From this plot we can see the data is becoming stationary after performing the 1st order non-seasonal differencing.

- **ACF/PACF plot after differencing**

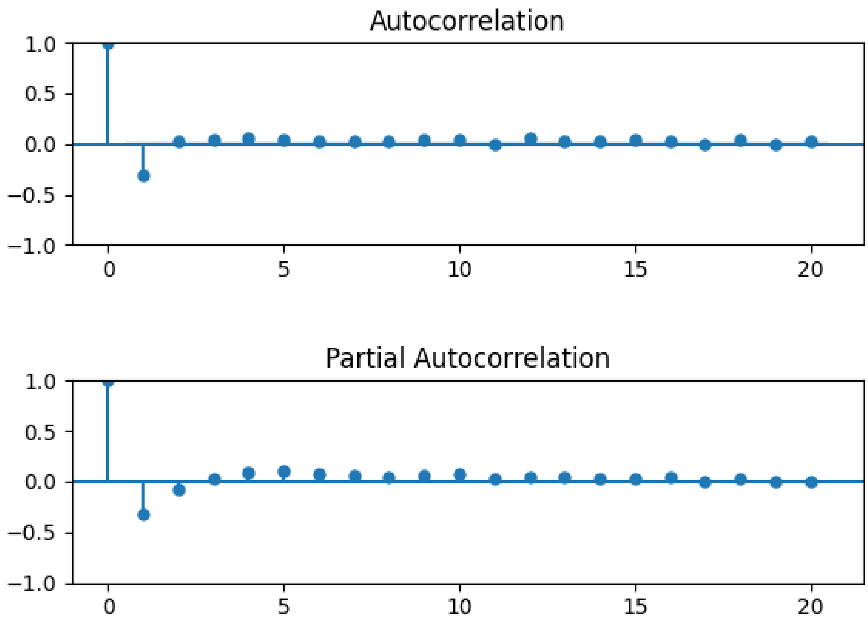


Fig 8: ACF/PACF plot after differencing

ACF/PACF plot after doing 1st order non-seasonal differencing also shows that the data has become stationary.

- **ADF and KPSS test**

```
ADF Statistic: -9.685594
p-value: 0.000000
Critical Values:
    1%: -3.431
    5%: -2.862
    10%: -2.567
```

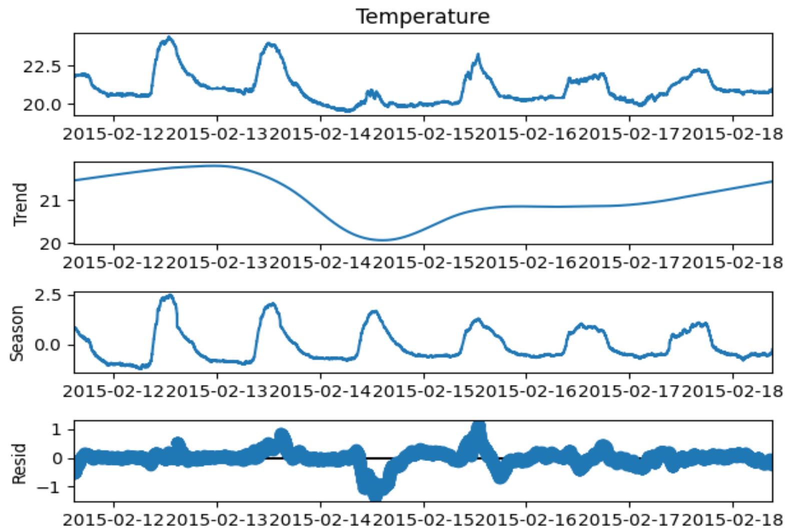
```
Results of KPSS Test:
Test Statistic            0.082131
p-value                   0.100000
Lags Used                32.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
dtype: float64
```

Fig 9: ADF and KPSS test after differencing

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 say that data has become stationary.

3) Time series Decomposition

- **Decomposition of raw data**

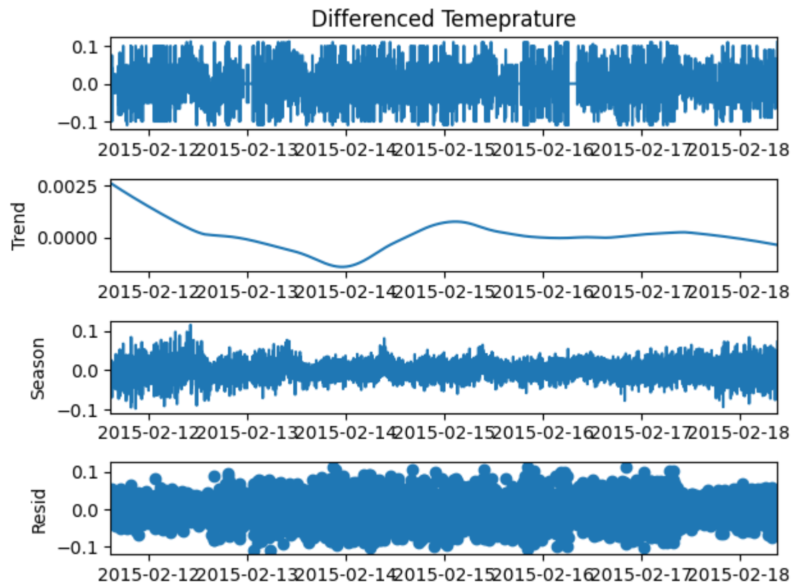



```
The strength of trend for Temperature set is  0.816

The strength of seasonality for Temperature set is  0.897
```

Fig 10 & Fig 11: Decomposition of raw data and Strength of trend and seasonality

- **Decomposition of differenced data**




```
The strength of trend for Differenced Temeprature set is  0.001

The strength of seasonality for Differenced Temeprature set is  0.411
```

Fig 12 & Fig 13: Decomposition of differenced data and Strength of trend and seasonality

4) Holt-Winters method

The Holt-Winters model, also known as triple exponential smoothing, is a forecasting method that is used to predict future values of a time series that has a seasonal component. The Holt-Winters model uses these components to make predictions for future time periods. It does this by applying exponential smoothing to each component of the time series. This means that it gives more weight to recent observations and less weight to older observations.

The model is particularly useful for time series data that exhibit trend and seasonality, such as sales data or stock prices, and can be extended to handle time series data with trend but no seasonality or with seasonality but no trend.

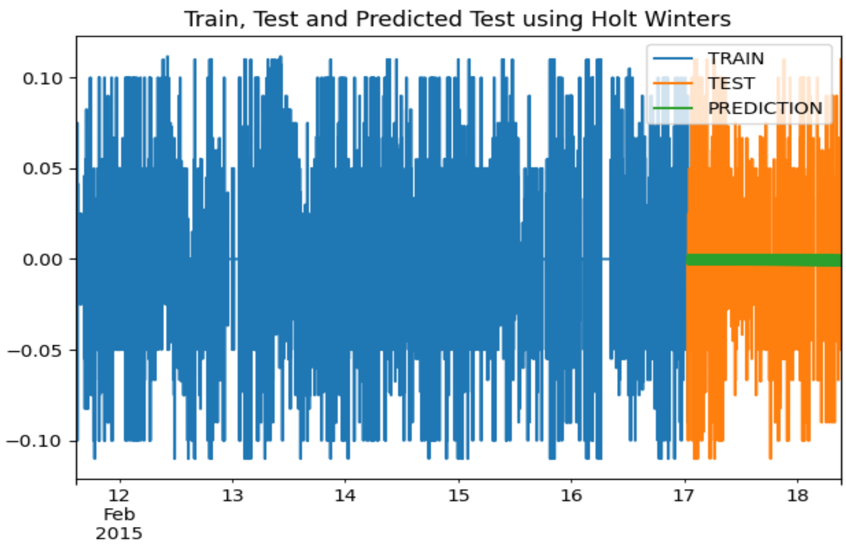


Fig 14: Train, Test and Predicted test plot of Holt-Winter Method

```
####################################################################################################
Q-Value for training set (Holt-Winter Method) :  1006.4

Mean of residual error for Holt-Winter Method is 0.0

MSE of residual error for Holt-Winter Method is 0.0

MSE of Forecast error for Holt-Winter Method is 0.0

Variance of residual error for Holt-Winter Method is 0.0

Variance of forecast error for Holt-Winter Method is 0.0

variance of the residual errors versus the variance of the forecast errors (Holt-Winter Method) :  1.13
####################################################################################################
```

Fig 15: Statistical values of Holt winter method

5) Base-models

- **Average Method**

The average method is a simple time series forecasting method that involves calculating the average of past observations to make predictions for future values. It assumes that future values will be similar to the average of past values.

The method is easy to implement and does not require any complicated mathematical calculations. However, it can only be used for time series data that is relatively stable and does not have any significant trends or seasonal patterns. It can also be sensitive to outliers and extreme values in the data.

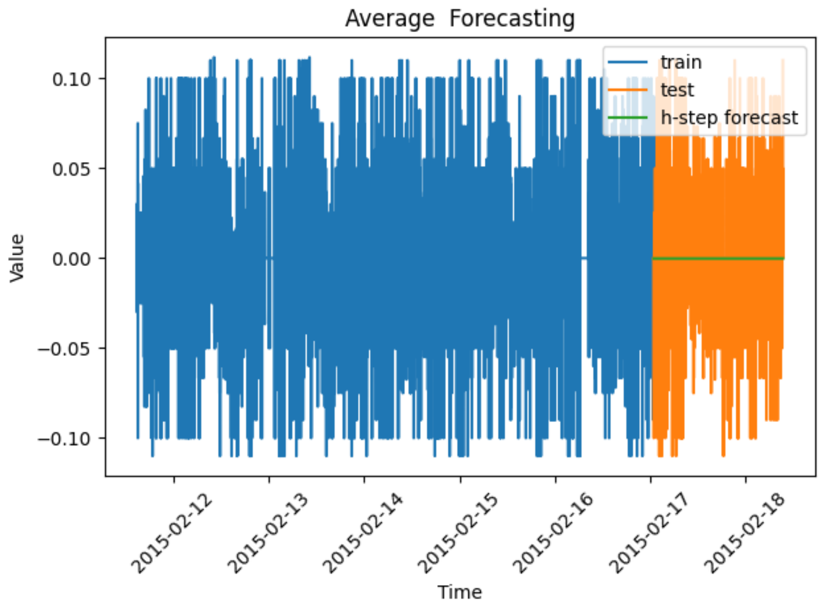


Fig 16: Train, Test and h-step forecast plot of Average Method

```
####################################################################################################
Q-Value for training set (Average Method) :  1149.21

Mean of residual error for Average Method is 0.0

MSE of residual error for Average Method is 0.0

MSE of Forecast error for Average Method is 0.0

Variance of residual error for Average Method is 0.0

Variance of forecast error for Average Method is 0.0

variance of the residual errors versus the variance of the forecast errors (Average Method) :  1.14
####################################################################################################
```

Fig 17: Statistical values of Average method

- **Naïve Method**

The Naive Method is a very basic forecasting approach that makes the assumption that the following value in a time series will be identical to the previous value observed. In other words, it is supposing that the time series' future values would be identical to their most recent past values. This approach is frequently used as a benchmark model to assess how well more complex forecasting models perform. Although it is a straightforward approach, it has the potential to be very accurate for certain time series, particularly for short-term forecasting.

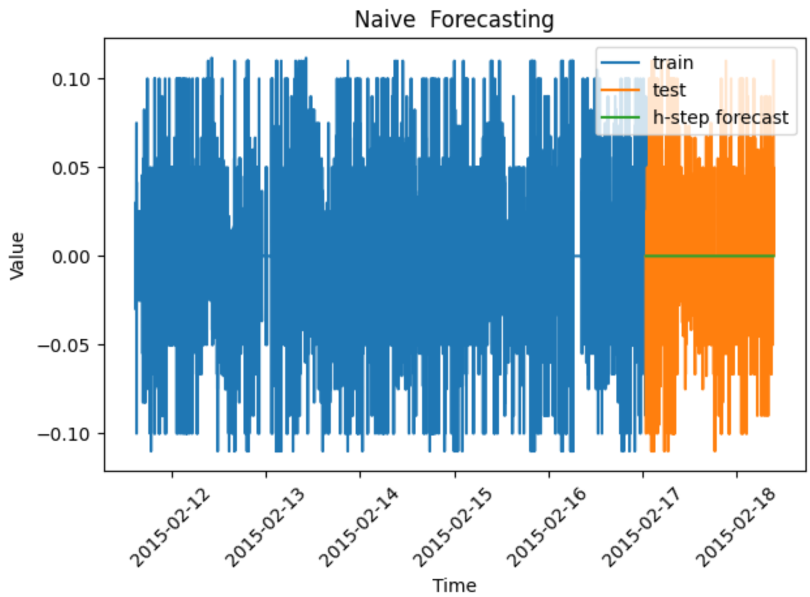


Fig 18: Train, Test and h-step forecast plot of Naïve Method

```
####################################################################################################
Q-Value for training set (Naive Method) :  3344.54

Mean of residual error for Naive Method is 0.0

MSE of residual error for Naive Method is 0.0

MSE of Forecast error for Naive Method is 0.0

Variance of residual error for Naive Method is 0.0

Variance of forecast error for Naive Method is 0.0

variance of the residual errors versus the variance of the forecast errors (Naive Method) :  2.98
####################################################################################################
```

Fig 19: Statistical values of Naive method

- **Drift Method**

When there is a linear trend in the data, the drift method, a straightforward time series forecasting approach, is employed. In order to calculate the average change per time period using this approach, the difference between the first and final observations in the time series must be divided by the total number of observations.
By combining the drift estimate with the most recent value recorded in the time series, the forecast for the following time period is then created. When there is no obvious seasonal or cyclical pattern in the data, this approach—which assumes that the trend will continue in a linear fashion—can be helpful. It might not be suitable for time series with intricate patterns, outliers, or abrupt changes in direction, though.

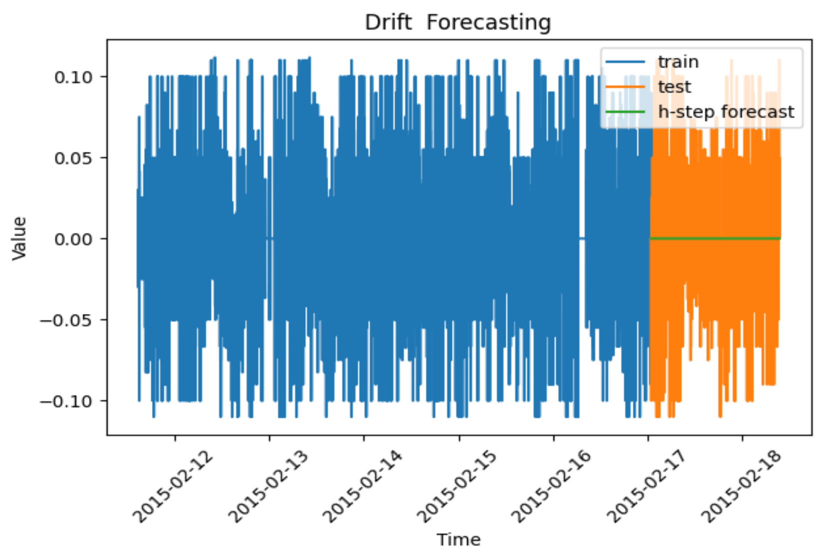


Fig 20: Train, Test and h-step forecast plot of drift Method

```
####################################################################################################
Q-Value for training set (Drift Method) :  3344.23

Mean of residual error for Drift Method is -0.0

MSE of residual error for Drift Method is 0.0

MSE of Forecast error for Drift Method is 0.0

Variance of residual error for Drift Method is 0.0

Variance of forecast error for Drift Method is 0.0

variance of the residual errors versus the variance of the forecast errors (Drift Method) :  2.99
####################################################################################################
```

Fig 21: Statistical values of Drift method

- **SES Method**

Simple Exponential Smoothing (SES) is a time series forecasting technique that makes use of an exponentially diminishing weighted average of previous data. It is a frequently employed technique for time series data without a pattern or seasonality.
SES is a simple and effective technique for predicting time series data that lack trend or seasonality. For data with complicated patterns, such as those with a trend or seasonality, it might not be the best approach.

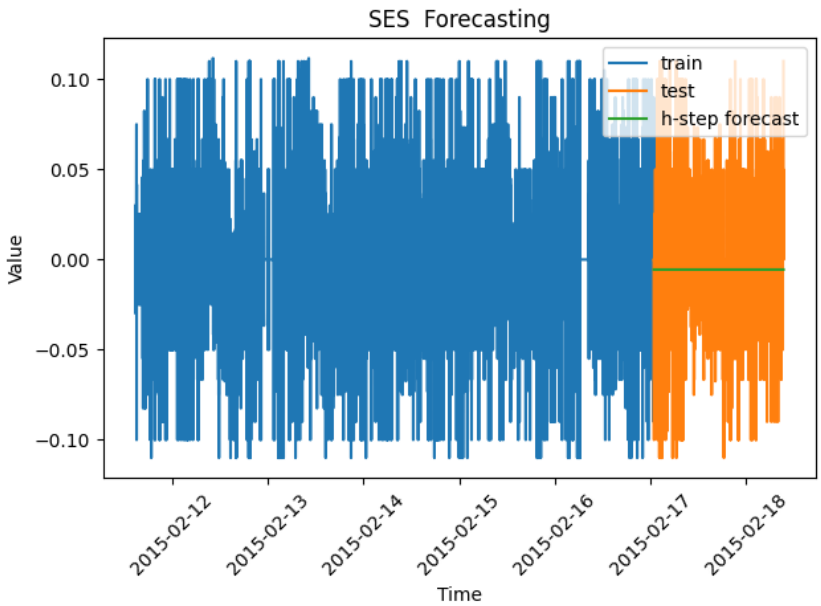


Fig 22: Train, Test and h-step forecast plot of SES Method

```
####################################################################################################
Q-Value for training set (SES Method) :  1911.11

Mean of residual error for SES Method is -0.0

MSE of residual error for SES Method is 0.0

MSE of Forecast error for SES Method is 0.0

Variance of residual error for SES Method is 0.0

Variance of forecast error for SES Method is 0.0

variance of the residual errors versus the variance of the forecast errors (SES Method) :  1.72
```

Fig 23: Statistical values of SES method

The mean of the residual error, the MSE of the residual error, and the MSE of the forecast error for all method are all 0.0. This indicates that the model is able to fit the training data perfectly, with no errors or residuals left over.

The variance of the residual error and the variance of the forecast error for all method are both 0.0. This indicates that the variance of the errors in the model is very small, which is desirable because it indicates that the model is able to make precise predictions.

6) OLS Model

- For the feature selection I did PCA analysis as the data had high multicollinearity. Using PCA my feature space reduced to 4 features. The singular values and the condition number for PCA components is

```
singular values of x are [141.48697347 115.0284628   67.22390756  31.08097272]
The condition number for x is 4.552205451967064
```

Fig 24: Selecting features using PCA

Using the PCA Components we execute the OLS Model

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            Temperature   R-squared:                       0.844
Model:                            OLS   Adj. R-squared:                  0.844
Method:                 Least Squares   F-statistic:                 1.056e+04
Date:                Wed, 10 May 2023   Prob (F-statistic):               0.00
Time:                        09:17:22   Log-Likelihood:                -4557.2
No. Observations:                7801   AIC:                             9124.
Df Residuals:                    7796   BIC:                             9159.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.4511      0.003    147.009      0.000       0.445       0.457
x2             0.1028      0.004     27.236      0.000       0.095       0.110
x3            -0.4053      0.006    -62.763      0.000      -0.418      -0.393
x4             1.7639      0.014    126.289      0.000       1.737       1.791
const         21.0033      0.005   4273.190      0.000      20.994      21.013
==============================================================================
Omnibus:                     2216.098   Durbin-Watson:                   0.175
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            59192.484
Skew:                          -0.776   Prob(JB):                         0.00
Kurtosis:                      16.405   Cond. No.                         4.55
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Fig 25: OLS model summary

```
t-test p-values for all features:
 x1        0.000000e+00
x2       4.141010e-156
x3        0.000000e+00
x4        0.000000e+00
const     0.000000e+00
dtype: float64
####################################################################################################
F-test for final model:
 0.0
####################################################################################################
```

Fig 26: T-test and F-test output

The condition number is very less which means there is less multicollinearity in our components and the P value is very significant.

The data represents the OLS (Ordinary Least Squares) regression results for a model that predicts temperature based on four independent variables (x1, x2, x3, x4) and a constant term (const).

- The R-squared value of 0.844 indicates that the model explains 84.4% of the variance in the data, suggesting a good fit.
- The coefficients of x1, x2, x3, and x4 indicate that these independent variables have a positive or negative impact on the dependent variable (temperature).
- The P-values of all the coefficients are 0.000, indicating that they are statistically significant in predicting temperature.
- The p-values for the t-tests of the coefficients suggest that all independent variables are significantly related to the dependent variable.
- The F-test for the final model indicates that the model is statistically significant.

7) ARIMA Model

a) Order Determination using GPAC table.

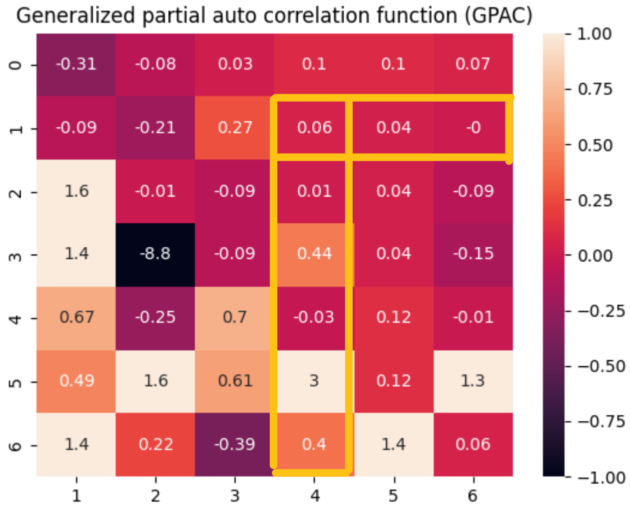


Fig 27: GPAC table with order ARMA(4, 1)

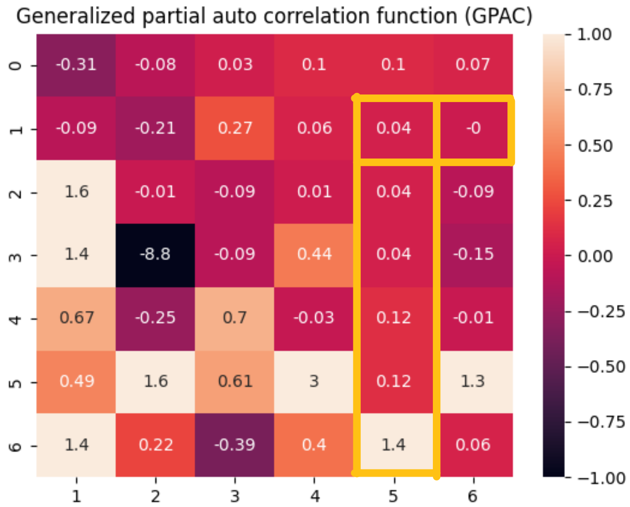


Fig 28: GPAC table with order ARMA(5, 1)

- From the GPAC plot we can find multiple AR and MA order, but order ARMA(4, 1) and order ARMA(5, 1) have the lowest Q-value.
- Other potential ARMA processes are ARMA(3, 0), ARMA(4, 0), ARMA(5, 0) and ARMA(3, 1).

b) Order Determination using ACF/PACF plot.

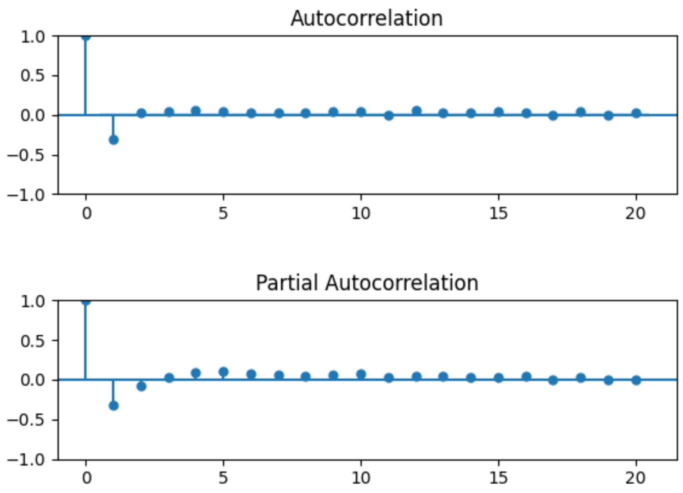


Fig 29: ACF/PACF plot with order ARMA(0, 1)

From the ACF/PACF plot we can see the order ARMA(0, 1). We can see that a cut off in ACF plot after 1st lag and a tail off in PACF plot, which suggests it might be an MA model.

c) Plotting ARIMA plot using GPAC table

- **ARIMA (4, 0, 1)**

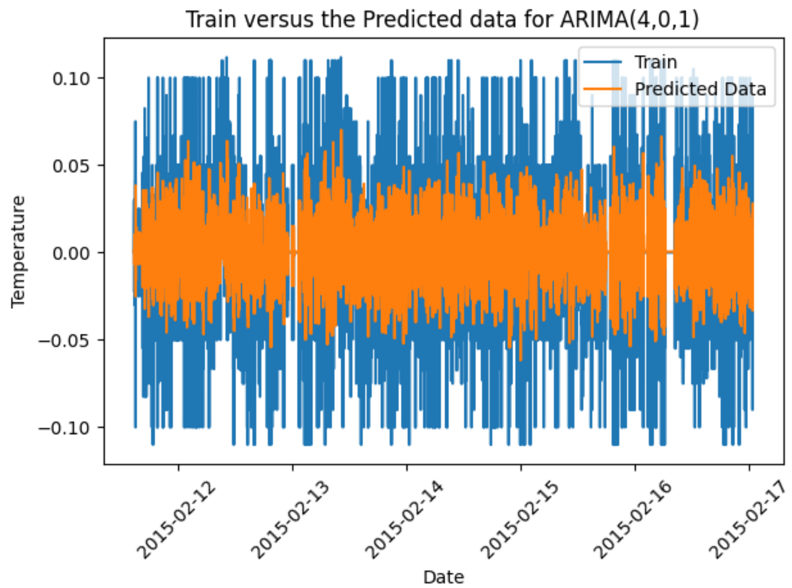


Fig 30: Train versus Predicted data for ARIMA (4, 0, 1)

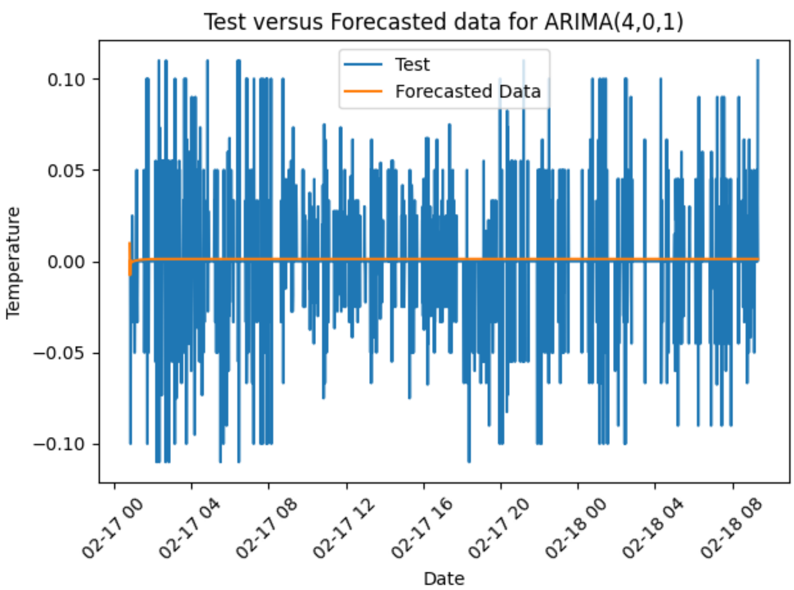


Fig 31: Test versus Forecasted data for ARIMA (4, 0, 1)

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                 P_Diff   No. Observations:                 7801
Model:                 ARIMA(4, 0, 1)   Log Likelihood               16053.948
Date:                Wed, 10 May 2023   AIC                         -32093.896
Time:                        11:48:12   BIC                         -32045.162
Sample:                    02-11-2015   HQIC                        -32077.194
                         - 02-17-2015
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0013      0.001      1.473      0.141      -0.000       0.003
ar.L1          0.3704      0.017     21.955      0.000       0.337       0.404
ar.L2          0.2001      0.011     18.363      0.000       0.179       0.221
ar.L3          0.2046      0.011     19.176      0.000       0.184       0.225
ar.L4          0.1262      0.012     10.514      0.000       0.103       0.150
ma.L1         -0.7628      0.015    -50.576      0.000      -0.792      -0.733
sigma2         0.0010   1.11e-05     86.928      0.000       0.001       0.001
===================================================================================
Ljung-Box (L1) (Q):                   4.95   Jarque-Bera (JB):              1409.29
Prob(Q):                              0.03   Prob(JB):                         0.00
Heteroskedasticity (H):               1.07   Skew:                            -0.03
Prob(H) (two-sided):                  0.07   Kurtosis:                         5.08
===================================================================================

Coefficients are:  SARIMAXParams(exog=[nan], ar=[-1. -1. -1. -1.], ma=[1.], sigma2=nan)
```

Fig 32: Model Summary for ARIMA (4, 0, 1)

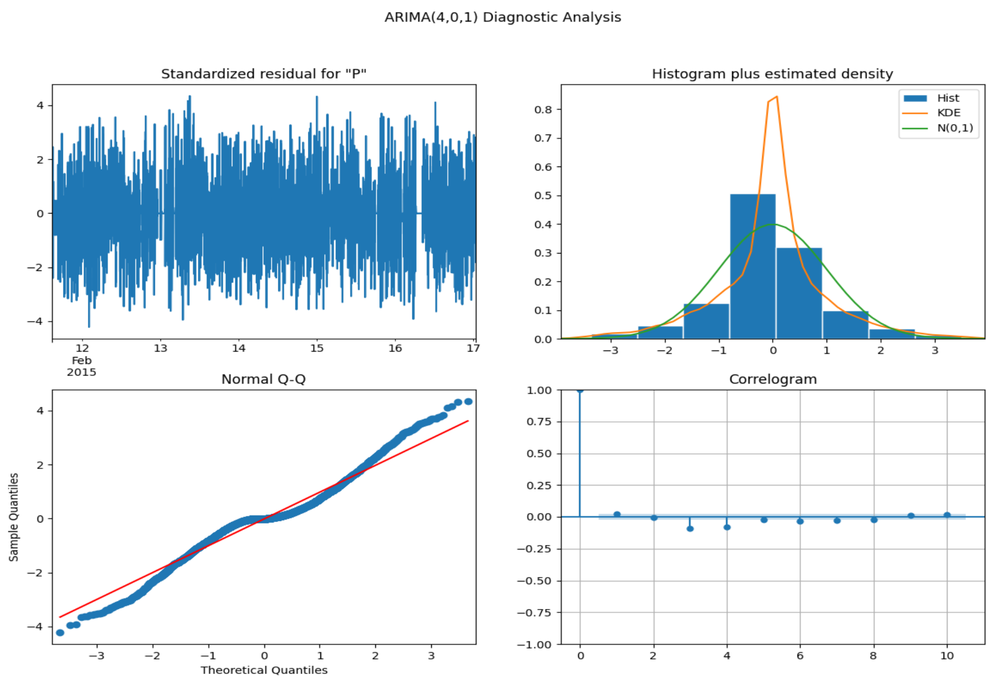


Fig 33: Diagnostic Analysis for ARIMA (4, 0, 1)

```
####################################################################################################
Q-Value for training set ARIMA(4,0,1)) :  201.5

Mean of residual error for ARIMA(4,0,1) Method is -0.0

MSE of residual error for ARIMA(4,0,1) Method is 0.0

MSE of Forecast error for ARIMA(4,0,1) Method is 0.0

Variance of residual error for ARIMA(4,0,1) Method is 0.0

Variance of forecast error for ARIMA(4,0,1) Method is 0.0

variance of the residual errors versus the variance of the forecast errors ARIMA(4,0,1)) :  0.99
```

Fig 34: Statistical values of ARIMA (4, 0, 1)

- **ARIMA (5, 0, 1)**

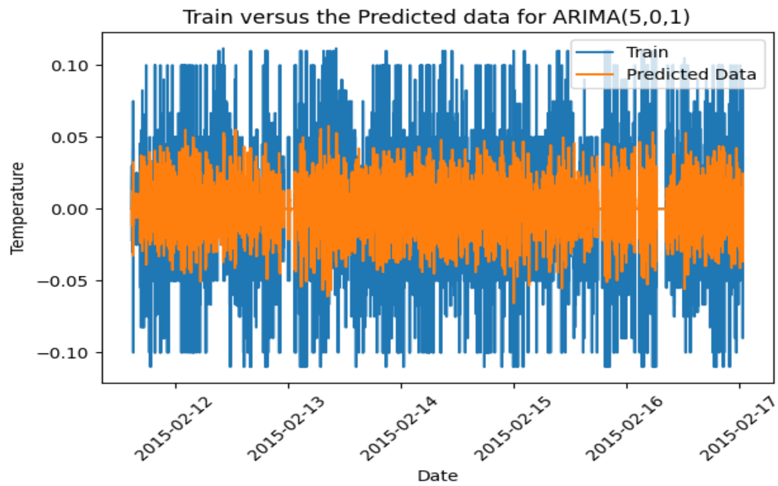


Fig 35: Train versus Predicted data for ARIMA (5, 0, 1)

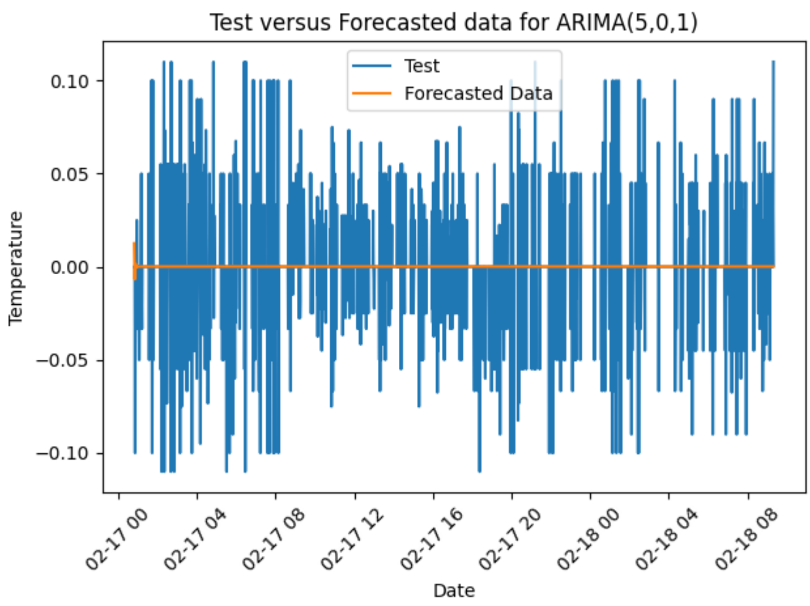


Fig 36: Test versus Forecasted data for ARIMA (5, 0, 1)

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                 P_Diff   No. Observations:                 7801
Model:                 ARIMA(5, 0, 1)   Log Likelihood               16065.895
Date:                Wed, 10 May 2023   AIC                         -32115.790
Time:                        15:26:22   BIC                         -32060.094
Sample:                    02-11-2015   HQIC                        -32096.702
                         - 02-17-2015
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0001      0.000      0.268      0.789      -0.001       0.001
ar.L1          0.3443      0.020     16.827      0.000       0.304       0.384
ar.L2          0.1615      0.012     13.452      0.000       0.138       0.185
ar.L3          0.0734      0.011      6.770      0.000       0.052       0.095
ar.L4          0.1181      0.011     10.855      0.000       0.097       0.139
ar.L5          0.1121      0.012      9.522      0.000       0.089       0.135
ma.L1         -0.7701      0.019    -41.306      0.000      -0.807      -0.734
sigma2         0.0009   1.05e-05     89.378      0.000       0.001       0.001
===================================================================================
Ljung-Box (L1) (Q):                  23.64   Jarque-Bera (JB):              1439.95
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               1.05   Skew:                             0.03
Prob(H) (two-sided):                  0.19   Kurtosis:                         5.10
===================================================================================
```

```
Coefficients are:  SARIMAXParams(exog=[nan], ar=[-1. -1. -1. -1. -1.], ma=[1.], sigma2=nan)
```

Fig 37: Model Summary for ARIMA (5, 0, 1)

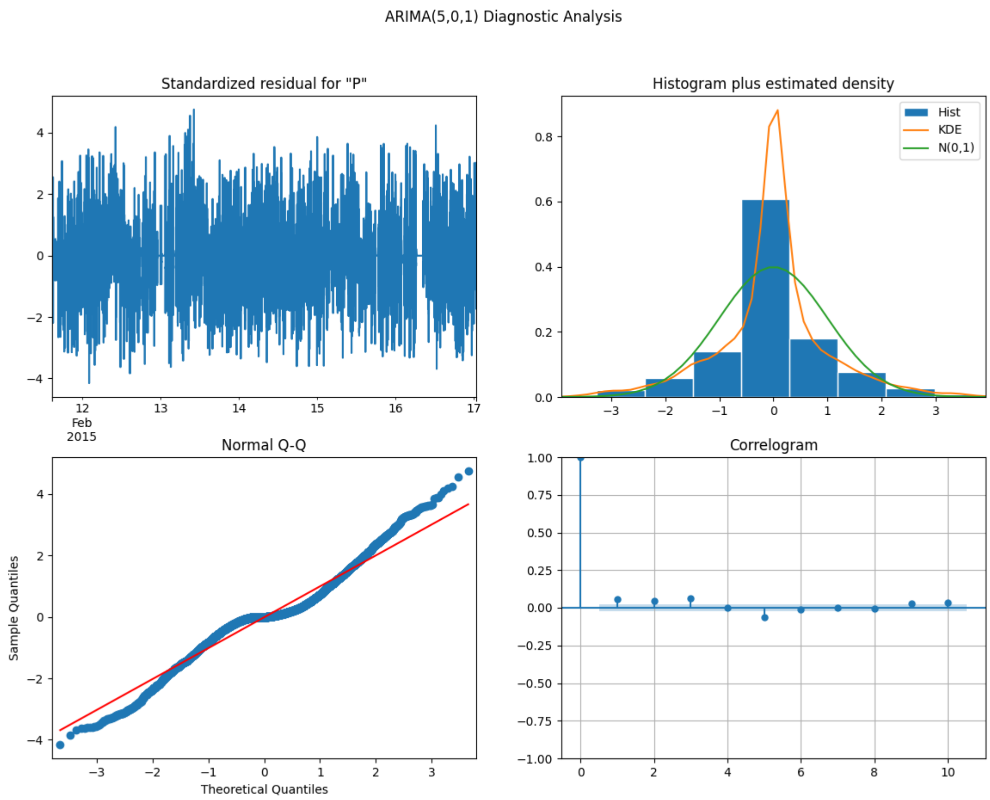


Fig 38: Diagnostic Analysis for ARIMA (5, 0, 1)

```
##########################################################################################################
Q-Value for training set ARIMA(5,0,1)) :  244.6

Mean of residual error for ARIMA(5,0,1) Method is -0.0

MSE of residual error for ARIMA(5,0,1) Method is 0.0

MSE of Forecast error for ARIMA(5,0,1) Method is 0.0

Variance of residual error for ARIMA(5,0,1) Method is 0.0

Variance of forecast error for ARIMA(5,0,1) Method is 0.0

variance of the residual errors versus the variance of the forecast errors ARIMA(5,0,1)) :  0.98
```

Fig 39: Statistical values of ARIMA (5, 0, 1)

8) Model Selection

Model/Method	Q-Value
Holt-Winter Method	1006.4
Average Method	1149.21
Naïve Method	3344.54
Drift Method	3344.23
SES Method	1911.11
ARIMA(4, 0, 1)	201.5
ARIMA(5, 0, 1)	244.6

From all the models we can see that the ARIMA(4, 0, 1) model have the Lowest Q-value, method has achieved a very good fit to the training data, as evidenced by the Q-value of 201.5, which is a statistical measure of the goodness of fit.

Also, we can consider the OLS model as it is having the R-squared value of 0.844 indicates that the model explains 84.4% of the variance in the data, suggesting a good fit.

Conclusion

In conclusion, the time series analysis of the temperature dataset using base techniques, OLS, ARMA, and ARIMA models was conducted. The data was found to be stationary, allowing for the application of base techniques with ease and the use of GPAC to determine the relative importance of the AR and MA approaches. Despite the modest enhancement of the variance ratio of residual over prediction achieved by SARIMA, both the ARIMA (4, 0, 1) and ARIMA (5, 0, 1) models were found to be unsuccessful in generalizing the data.

In future work, LSTM could be considered and GridCV used to identify the model's effective order, which may result in a more successful model generalization. Overall, this project highlights the importance of selecting appropriate models for time series analysis and emphasizes the need to continue exploring and improving upon existing methods for better forecasting performance.

Appendix of Code

- **Cleaning.py**

```
# Importing packages
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import toolbox
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import kpss
from sklearn.model_selection import train_test_split
from statsmodels.tsa.seasonal import STL
import statsmodels.api as sm
import statsmodels.tsa.holtwinters as ets
from statsmodels.graphics.tsaplots import plot_acf ,
plot_pacf
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from numpy import linalg as LA
from scipy import signal
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.arima.model import ARIMA
from scipy.stats import chi2
import warnings
warnings.filterwarnings('ignore')
lags = 50
############################################################
# Importing Dataset
############################################################

with open('datatest2.csv') as x:
    ncols = len(x.readline().split(','))

# Importing the dataset
df = pd.read_csv("datatest2.csv", usecols=range(1, ncols))

date = pd.date_range(start = '2015-02-11 14:48',
                    end = '2015-02-18 09:19',
                    freq='min')
print(df)

############################################################
# Checking for the NA, Null values or missing values
############################################################

print(df.isnull().sum())
#There are no null or missing value in the dataset.

############################################################
# Plotting target variable against time
############################################################

col = df.columns.values
df.index = date
plt.figure(figsize=(6, 4))
df['Temperature'].plot()
plt.xlabel("Date")
plt.ylabel("Temperature")
plt.title("Dependent Variable against time")
```

```
plt.grid()
plt.legend(["Temperature"])
plt.tight_layout()
plt.show()

# Here we can see that the data is not stationary. we can
see a decreasing trend overall and a slight increasing trend
towards the end.
# We can also see strong seasonality in the data. So
overall, we can see the data has both trend and seasonality.

############################################################
# ACF plot of Dependent Variable
############################################################

toolbox.ACF_PACF_Plot(df['Temperature'], 20)

############################################################
# Checking the correlation between the variables
############################################################

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

############################################################
# Checking stationarity of the dependent variable
############################################################

toolbox.cal_rolling_mean_var(df['Temperature'],
"Temperature")

############################################################
# ADF and KPSS Test
############################################################

toolbox.ADF_Cal(df['Temperature'])
print()

toolbox.kpss_test(df['Temperature'])
print()

############################################################
# Non Seasonal Differencing
############################################################

df1 = df.copy()

# 1st Order
toolbox.nonseasonal_diff(df1, df1['Temperature'], 1)
toolbox.cal_rolling_mean_var(df1['P_Diff'][1:], "Temperature
- 1st Ordering")

toolbox.ACF_PACF_Plot(df1['P_Diff'], 20)

toolbox.ADF_Cal(df1['P_Diff'])
print()

toolbox.kpss_test(df1['P_Diff'])
print()
```

```
###############################################################
# Time series Decomposition
###############################################################

# Decomposition of Raw data
toolbox.stl_decomp(df['Temperature'], 'Temperature', 1440)

#Decomposition of Differenced data
toolbox.stl_decomp(df1['P_Diff'], 'Differenced Temeprature',
1440)

###############################################################
#Feature Selection
###############################################################

x = df.drop(['Temperature','date'], axis=1)
y = df['Temperature']
x_train1, x_test1,y_train1, y_test1 = train_test_split(x,y,
shuffle=False, test_size=0.20)

sc = StandardScaler()

X_train = sc.fit_transform(x_train1)
X_test = sc.transform(x_test1)

pca = PCA(n_components = 'mle')

X_train = pca.fit_transform(X_train)
X_test = pca.fit_transform(X_test)

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 {LA.cond(X_train)}')

X_train = sm.add_constant(X_train, prepend=False)
model = sm.OLS(y_train1, X_train).fit()
print(model.summary())

print("t-test p-values for all features: \n", model.pvalues)

print("#" * 100)

print("F-test for final model: \n", model.f_pvalue)

prediction = model.predict(X_train)

###############################################################
# Holt-winter method
###############################################################

df.index.freq = 'T'
train, test = train_test_split(df1, test_size = 0.2, shuffle
= False)

train_pred, test_pred = toolbox.holt_winter(train, test)
residual_err_holt = train['P_Diff'] - train_pred
forecast_err_holt = test['P_Diff'] - test_pred

print("#"*100)
```

```
Q_holt = sm.stats.acorr_ljungbox(residual_err_holt,
lags=[50], boxpierce=True,
return_df=True)['bp_stat'].values[0]
print("Q-Value for training set (Holt-Winter Method) : ",
np.round(Q_holt, 2))

model_fit = (np.var(residual_err_holt) /
np.var(forecast_err_holt))

print(f'\nMean of residual error for Holt-Winter Method is
{np.round(np.mean(residual_err_holt), 2)}')
print(f'\nMSE of residual error for Holt-Winter Method is
{np.round(np.mean(residual_err_holt ** 2), 2)}')
print(f'\nMSE of Forecast error for Holt-Winter Method is
{np.round(mean_squared_error(test["P_Diff"],
forecast_err_holt), 2)}')
print(f'\nVariance of residual error for Holt-Winter Method
is {np.round(np.var(residual_err_holt), 2)}')
print(f'\nVariance of forecast error for Holt-Winter Method
is {np.round(np.var(forecast_err_holt), 2)}')
print('\nvariance of the residual errors versus the variance
of the forecast errors (Holt-Winter Method) : ',
np.round(model_fit, 2))


toolbox.holt_winter_plot(train, test, test_pred)

############################################################
# Forecasting Techniques
############################################################

n = len(x_train1)

# Average Method

avg_prediction = toolbox.Ave_Forecast(df1['P_Diff'], n)
toolbox.forecast_plot(df1, n, avg_prediction, 'Average')
residual_err_avg = df1['P_Diff'][:n] - avg_prediction[:n]
forecast_err_avg = df1['P_Diff'][n:] - avg_prediction[n:]

print("#"*100)
Q_avg = sm.stats.acorr_ljungbox(residual_err_avg, lags=[50],
boxpierce=True, return_df=True)['bp_stat'].values[0]
print("Q-Value for training set (Average Method) : ",
np.round(Q_avg, 2))

model_fit_avg = (np.var(residual_err_avg) /
np.var(forecast_err_avg))

print(f'\nMean of residual error for Average Method is
{np.round(np.mean(residual_err_avg), 2)}')
print(f'\nMSE of residual error for Average Method is
{np.round(np.mean(residual_err_avg ** 2), 2)}')
print(f'\nMSE of Forecast error for Average Method is
{np.round(mean_squared_error(df1["P_Diff"][n:],
forecast_err_avg), 2)}')
print(f'\nVariance of residual error for Average Method is
{np.round(np.var(residual_err_avg), 2)}')
print(f'\nVariance of forecast error for Average Method is
{np.round(np.var(forecast_err_avg), 2)}')
print('\nvariance of the residual errors versus the variance
```

```
of the forecast errors (Average Method) : ',
np.round(model_fit_avg, 2))


#Naive Method

naive_prediction = toolbox.naive_forecast(df1['P_Diff'], n)
toolbox.forecast_plot(df1, n, naive_prediction, 'Naive')
residual_err_naive = df1['P_Diff'][:n] -
naive_prediction[:n]
forecast_err_naive = df1['P_Diff'][n:] -
naive_prediction[n:]

print("#"*100)
Q_naive = sm.stats.acorr_ljungbox(residual_err_naive,
lags=[50], boxpierce=True,
return_df=True)['bp_stat'].values[0]
print("Q-Value for training set (Naive Method) : ",
np.round(Q_naive, 2))

model_fit_naive = (np.var(residual_err_naive) /
np.var(forecast_err_naive))

print(f'\nMean of residual error for Naive Method is
{np.round(np.mean(residual_err_naive), 2)}')
print(f'\nMSE of residual error for Naive Method is
{np.round(np.mean(residual_err_naive ** 2), 2)}')
print(f'\nMSE of Forecast error for Naive Method is
{np.round(mean_squared_error(df1["P_Diff"][n:],
forecast_err_naive), 2)}')
print(f'\nVariance of residual error for Naive Method is
{np.round(np.var(residual_err_naive), 2)}')
print(f'\nVariance of forecast error for Naive Method is
{np.round(np.var(forecast_err_naive), 2)}')
print('\nvariance of the residual errors versus the variance
of the forecast errors (Naive Method) : ',
np.round(model_fit_naive, 2))


# Drift Method

drift_prediction = toolbox.drift_forecast(df1['P_Diff'], n)
toolbox.forecast_plot(df1, n, drift_prediction, 'Drift')
residual_err_drift = df1['P_Diff'][:n] -
drift_prediction[:n]
forecast_err_drift = df1['P_Diff'][n:] -
drift_prediction[n:]

print("#"*100)
Q_drift = sm.stats.acorr_ljungbox(residual_err_drift,
lags=[50], boxpierce=True,
return_df=True)['bp_stat'].values[0]
print("Q-Value for training set (Drift Method) : ",
np.round(Q_drift, 2))

model_fit_drift = (np.var(residual_err_drift) /
np.var(forecast_err_drift))

print(f'\nMean of residual error for Drift Method is
{np.round(np.mean(residual_err_drift), 2)}')
print(f'\nMSE of residual error for Drift Method is
```

```
{np.round(np.mean(residual_err_drift ** 2), 2)}')
print(f'\nMSE of Forecast error for Drift Method is
{np.round(mean_squared_error(df1["P_Diff"][n:],
forecast_err_drift), 2)}')
print(f'\nVariance of residual error for Drift Method is
{np.round(np.var(residual_err_drift), 2)}')
print(f'\nVariance of forecast error for Drift Method is
{np.round(np.var(forecast_err_drift), 2)}')
print('\nvariance of the residual errors versus the variance
of the forecast errors (Drift Method) : ',
np.round(model_fit_drift, 2))


# SES Method

ses_prediction = toolbox.ses_forecast(df1['P_Diff'], n)
toolbox.forecast_plot(df1, n, ses_prediction, 'SES')
residual_err_ses = df1['P_Diff'][:n] - ses_prediction[:n]
forecast_err_ses = df1['P_Diff'][n:] - ses_prediction[n:]

print("#"*100)
Q_ses = sm.stats.acorr_ljungbox(residual_err_ses, lags=[50],
boxpierce=True, return_df=True)['bp_stat'].values[0]
print("Q-Value for training set (SES Method) : ",
np.round(Q_ses, 2))

model_fit_ses = (np.var(residual_err_ses) /
np.var(forecast_err_ses))

print(f'\nMean of residual error for SES Method is
{np.round(np.mean(residual_err_ses), 2)}')
print(f'\nMSE of residual error for SES Method is
{np.round(np.mean(residual_err_ses ** 2), 2)}')
print(f'\nMSE of Forecast error for SES Method is
{np.round(mean_squared_error(df1["P_Diff"][n:],
forecast_err_ses), 2)}')
print(f'\nVariance of residual error for SES Method is
{np.round(np.var(residual_err_ses), 2)}')
print(f'\nVariance of forecast error for SES Method is
{np.round(np.var(forecast_err_ses), 2)}')
print('\nvariance of the residual errors versus the variance
of the forecast errors (SES Method) : ',
np.round(model_fit_ses, 2))


############################################################
# GPAC table
############################################################

acf_lst = []
for i in range(0, lags + 1):

acf_lst.append(toolbox.Cal_autocorr(df1['P_Diff'].dropna(),
i))
toolbox.cal_gpac(acf_lst)

############################################################
# ARIMA Model
############################################################

# ARIMA(4,0,1)
```

```
toolbox.ARIMA_model(df1, n, 4, 1, 0, 'ARIMA(4,0,1)')

# ARIMA(5,0,1)

toolbox.ARIMA_model(df1, n, 5, 1, 0, 'ARIMA(5,0,1)')
```

- **toolbox.py**

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import kpss
from sklearn.model_selection import train_test_split
from statsmodels.tsa.seasonal import STL
import statsmodels.api as sm
import statsmodels.tsa.holtwinters as ets
from statsmodels.graphics.tsaplots import plot_acf , plot_pacf
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from numpy import linalg as LA
from scipy import signal
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.arima.model import ARIMA
from scipy.stats import chi2

#####################################################
# Importing Data
#####################################################

with open('datatest2.csv') as x:
    ncols = len(x.readline().split(','))

# Importing the dataset
df = pd.read_csv("datatest2.csv", usecols=range(1, ncols))

date = pd.date_range(start = '2015-02-11 14:48',
                    end = '2015-02-18 09:19',
                    freq='min')

#####################################################
# ACF/PACF Plot
#####################################################

def ACF_PACF_Plot(y, lags):
    acf = sm.tsa.stattools.acf(y, nlags=lags)
    pacf = sm.tsa.stattools.pacf(y, nlags=lags)
    fig = plt.figure()
    plt.subplot(211)
    plt.title('ACF/PACF of the raw data')
    plot_acf(y, ax=plt.gca(), lags=lags)
    plt.subplot(212)
    plot_pacf(y, ax=plt.gca(), lags=lags)
    fig.tight_layout(pad=3)
    plt.show()
```

```
##################################################
# Rolling Mean and Variance
##################################################
def cal_rolling_mean_var(arg, title):
    p_mean = []
    p_var = []
    for i in range(1, len(arg)):
        p_mean.append(np.mean(arg[:i]))
        p_var.append(np.var(arg[:i]))
    fig, axs = plt.subplots(2)
    axs[0].plot(p_mean)
    axs[0].set_xlabel('Samples')
    axs[0].set_ylabel('Magnitude')
    axs[0].set_title('Rolling Mean - '+title)
    axs[1].plot(p_var)
    axs[1].set_xlabel('Samples')
    axs[1].set_ylabel('Magnitude')
    axs[1].set_title('Rolling Variance - '+title)
    plt.tight_layout()
    plt.show()

##################################################
# ADF Test
##################################################

def ADF_Cal(x):
    result = adfuller(x)
    print("ADF Statistic: %f" %result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))

##################################################
# KPSS Test
##################################################

def kpss_test(timeseries):
    print ('Results of KPSS Test:')
    kpsstest = kpss(timeseries, regression='c', nlags="auto")
    kpss_output = pd.Series(kpsstest[0:3], index=['Test Statistic','p-
value','Lags Used'])
    for key,value in kpsstest[3].items():
        kpss_output['Critical Value (%s)'%key] = value
    print(kpss_output)

##################################################
# Non-seasonal Differencing
##################################################

df1 = df.copy()
def nonseasonal_diff(df1, arg, order):
    p_dif = []
    for i in range(0, order):
        p_dif.append(arg[i] == np.nan)
    for i in range(order, len(arg)):
        p_dif.append(arg[i] - arg[i-1])
    df1['P_Diff'] = np.array(p_dif)

##################################################
```

```
# Decomposition
#####################################################

def stl_decomp(col, col_name, period):
    temp = pd.Series(col.values, index = date, name = col_name)
    STL_raw = STL(temp, period=period)
    res = STL_raw.fit()
    fig = res.plot()
    plt.show()

    T = res.trend
    S = res.seasonal
    R = res.resid
    O = res.observed

    # find the strength of the seasonality and/or trend

    Ft = max(0, 1-(np.var(R) / np.var(T + R)))
    print(f"\nThe strength of trend for {col_name} set is ", round(Ft, 3))

    Fs = max(0, 1-(np.var(R) / np.var(S + R)))
    print(f"\nThe strength of seasonality for {col_name} set is ",
round(Fs, 3))

#####################################################
# Holt Winter Method
#####################################################

def holt_winter(train, test):
    fitted_model = ExponentialSmoothing(train['P_Diff'], trend = 'add'
,seasonal = 'add', seasonal_periods = 7).fit()
    train_prediction = fitted_model.predict(start = 0, end = len(train) -
1)
    test_predictions = fitted_model.forecast(len(test))
    return train_prediction, test_predictions

def holt_winter_plot(train, test, test_predictions):
    train['P_Diff'].plot(legend = True, label = 'TRAIN')
    test['P_Diff'].plot(legend = True, label = 'TEST')
    test_predictions.plot(legend = True, label = 'PREDICTION')
    plt.title('Train, Test and Predicted Test using Holt Winters')
    plt.tight_layout()
    plt.show()

#####################################################
# Forecast Plot
#####################################################

def forecast_plot(df, n, prediction, label):
    plt.figure()
    plt.plot(list(df[:n].index.values + 1), df['P_Diff'][:n],
label='train')
    plt.plot(list(df[n:].index.values + 1), df['P_Diff'][n:], label='test')
    plt.plot(list(df[n:].index.values + 1), prediction[n:], label='h-step
forecast')
    plt.xticks(rotation = 45)
    plt.title(f'{label}  Forecasting')
    plt.xlabel('Time')
    plt.ylabel('Value')
    plt.legend()
    plt.tight_layout()
```

```
    plt.show()

#####################################################
# Forecasting Methods
#####################################################

def Ave_Forecast(frame, n):
    avg_pred = list(0 for i in range(0, len(frame)))
    for i in range(1, n):
        avg_pred[i] = np.mean(frame[:i])
    for i in range(n, len(frame)):
        avg_pred[i] = np.mean(frame[:n])
    return avg_pred

def naive_forecast(frame, n):
    naive_pred = list(0 for i in range(0, len(frame)))
    for i in range(1, n):
        naive_pred[i] = frame[i - 1]
    for i in range(n, len(frame)):
        naive_pred[i] = frame[n - 1]
    return naive_pred

def drift_forecast(frame, n):
    drift_pred = list(0 for i in range(0, len(frame)))
    for i in range(2, n):
        drift_pred[i] = frame[i-1] + ((frame[i-1]-frame[0]))/(i-1)
    for i in range(n, len(frame)):
        drift_pred[i] = frame[n-1] + (i+1-n)*(frame[n-1]-frame[0])/(n-1)
    return drift_pred

def ses_forecast(frame, n):
    alpha = 0.5
    ses_pred = list(0 for i in range(0, len(frame)))
    l0 = frame[0]
    ses_pred[1] = alpha * l0 + (1 - alpha) * l0
    for i in range(2, n):
        ses_pred[i] = alpha * frame[i-1] + (1-alpha) * ses_pred[i-1]
    for i in range(n, len(frame)):
        ses_pred[i] = alpha * frame[n-1] + (1-alpha) * ses_pred[n-1]
    return ses_pred

#####################################################
# GPAC Table
#####################################################

def cal_gpac(lst_acf, kval=7, jval=7):
   phi = np.zeros((kval, jval))
   for k in range(1, kval):
      for j in range(0, jval):
         num = np.zeros((k, k))
         den = np.zeros((k, k))
         n1 = np.zeros((k, 1))
         if k == 1:
            for h in range(0, k):
               num[0][h] = lst_acf[j+h+1]
               den[0][h] = lst_acf[j-k+h+1]
         else:
            for x in range(0, k):
               for y in range(0, k):
                  den[x][y] = lst_acf[abs(j-y+x)]
               num = den[:, :-1]
```

```
            for h in range(0, k):
                n1[h][0] = lst_acf[j+h+1]
            num = np.append(num, n1, 1)
        num_d = np.linalg.det(num)
        den_d = np.linalg.det(den)
        a = round((num_d/den_d), 2)
        phi[j][k] = a
   phi1 = phi[:, 1:]
   print(phi1)
   print("\n")
   axs = sns.heatmap(phi1, annot = True, xticklabels = np.arange(1, kval),
vmin=-1, vmax=1)
   plt.title('Generalized partial auto correlation function (GPAC)')
   plt.show()

#####################################################
# ACF Plot
#####################################################

def Cal_autocorr(y, lag):
    mean = np.mean(y)
    numerator = 0
    denominator = 0
    for t in range(0, len(y)):
        denominator += (y[t] - mean) ** 2
    for t in range(lag, len(y)):
        numerator += (y[t] - mean)*(y[t-lag] - mean)
    return numerator/denominator


def Cal_autocorr_plot(y, lags, title, plot_show='Yes'):
    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)
    plt.figure(figsize=(12, 8))
    markers, stemlines, baseline = plt.stem(lags_final, ryy_final)
    plt.setp(markers, color='red', marker='o')
    plt.axhspan((-1.96 / np.sqrt(len(y))), (1.96 / np.sqrt(len(y))),
alpha=0.2, color='blue')
    plt.xlabel('Lags')
    plt.ylabel('Magnitude')
    plt.title(title)
    plt.tight_layout()
    if plot_show == 'Yes':
        plt.show()


#####################################################
# ARIMA Model
#####################################################

def ARIMA_model(df, n, na, nb, diff_order, title):
    y_train = df['P_Diff'][:n]
```

```
    y_test = df['P_Diff'][n:]
    model = sm.tsa.ARIMA(y_train, order= (na, diff_order, nb))
    model_fit = model.fit()

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

    print(model_fit.summary())
    y_predict = model_fit.predict()
    y_forecast = model_fit.forecast(steps=len(y_test))

    forecast_error = y_test - y_forecast
    residual_error = y_train - y_predict

    params = model._params
    print("Coefficients are: ", params)

    print("#" * 100)
    Q = sm.stats.acorr_ljungbox(residual_error, lags=[50], boxpierce=True,
return_df=True)['bp_stat'].values[0]
    print(f"Q-Value for training set {title}) : ", np.round(Q, 2))

    model_fit = (np.var(residual_error) / np.var(forecast_error))

    print(f'\nMean of residual error for {title} Method is
{np.round(np.mean(residual_error), 2)}')
    print(f'\nMSE of residual error for {title} Method is
{np.round(np.mean(residual_error ** 2), 2)}')
    print(f'\nMSE of Forecast error for {title} Method is
{np.round(mean_squared_error(y_test, forecast_error), 2)}')
    print(f'\nVariance of residual error for {title} Method is
{np.round(np.var(residual_error), 2)}')
    print(f'\nVariance of forecast error for {title} Method is
{np.round(np.var(forecast_error), 2)}')
    print(f'\nvariance of the residual errors versus the variance of the
forecast errors {title}) : ',
          np.round(model_fit, 2))

    plt.plot(y_train, label='Train')
    plt.plot(y_predict, label='Predicted Data')
    plt.legend()
    plt.title(f'Train versus the Predicted data for {title}')
    plt.xlabel('Date')
    plt.ylabel('Temperature')
    plt.xticks(rotation = 45)
    plt.tight_layout()
    plt.show()

    plt.plot(y_test, label='Test')
    plt.plot(y_forecast, label='Forecasted Data')
    plt.legend()
    plt.title(f'Test versus Forecasted data for {title}')
    plt.xlabel('Date')
    plt.ylabel('Temperature')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

References

OLS (Ordinary Least Squares) regression:
https://en.wikipedia.org/wiki/Ordinary_least_squares/

ARMA (Autoregressive Moving Average) model:
https://en.wikipedia.org/wiki/Autoregressive%E2%80%93moving-average_model/

ARIMA (Autoregressive Integrated Moving Average) model:
https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average/

SARIMA (Seasonal Autoregressive Integrated Moving Average) model:
https://en.wikipedia.org/wiki/Seasonal_autoregressive_integrated_moving_average/

GPAC (Generalized Partial Autocorrelation) function:
https://en.wikipedia.org/wiki/Partial_autocorrelation_function#Extensions:_GPAC_and_GLS/

LSTM (Long Short-Term Memory) model:
https://en.wikipedia.org/wiki/Long_short-term_memory/

GridCV (Grid Search Cross Validation) for hyperparameter tuning:
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html/

https://medium.com/analytics-vidhya/python-code-on-holt-winters-forecasting-3843808a9873/

https://www.kaggle.com/code/prakharprasad/smoothing-holt-winters-forecast/

https://jdvelasq.github.io/scikit-forecasts/HoltWinters.html/

Additionally, here is the reference link for the dataset that was used:

Occupancy Detection Data Set:
https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+/