Course: DATS 6450 – Time Series Analysis & Modeling

Instructor: Dr. Reza Jafari

Final Term Project

Topic: Underwater Surface Temperature

Name: Pon Swarnalaya Ravichandran

PR
Initials

12.03.2023
Date

TABLE OF CONTENTS

ABSTRACT

The project is dedicated to an exhaustive time series analysis and modeling initiative, utilizing data sourced from seven islands and two submerged rocky reefs along the Santa Catarina coast in southern Brazil. The dataset, comprising temperature records obtained at 20-minute intervals through a HOBO Pendant® Temperature Data Logger UA-002, spans the temporal spectrum from December 2012 to July 2014. Rigorous analysis and preprocessing techniques tailored for time series data are systematically applied, ensuring the dataset's optimal preparation for subsequent modeling phases. The overarching goal of the project is to employ various linear time series models, discerning the most fitting model to accurately predict the subaquatic surface temperature. Through this meticulous approach, the project aspires to yield nuanced insights into the temperature dynamics prevalent in the specified coastal region.

This endeavor not only underscores the importance of robust data analytics but also highlights the application of advanced modeling techniques in elucidating temporal patterns, contributing valuable knowledge to the understanding of the underwater temperature dynamics in the context of the Santa Catarina coast.

INTRODUCTION

Time series analysis is a specialized methodology for examining a sequence of data points gathered at regular intervals over a defined time span. In contrast to sporadic or random data point recording, time series analysis entails systematically capturing data points at consistent intervals throughout a specified duration. This approach necessitates the accumulation of a substantial number of data points to ensure both the stability and dependability of the analysis.

The significance of a comprehensive dataset lies in its ability to provide a representative sample size, allowing the analysis to discern patterns and trends amidst potentially noisy data. An extensive dataset ensures that any identified trends or patterns are not mere outliers and can effectively account for seasonal variations. Moreover, time series data lends itself to forecasting applications, enabling the prediction of future data points based on insights derived from historical data.

To enhance the quality of analysis, it is essential to work with processed and well-prepared data. The initial stage of data analysis involves visualization using basic statistical information and plots to ensure its readiness for modeling. The stationarity of the data is a critical aspect assessed through various techniques.

Rolling mean and variances, alongside the Augmented Dickey-Fuller (ADF) test and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test, are employed to evaluate stationarity. In the context of rolling mean and variances, a flat line in the plot is indicative of stationary data. On the other hand, the ADF and KPSS tests have their own hypotheses, relying on p-values and test statistics, to confirm or refute stationarity.

If the data is found to be non-stationary, appropriate differencing techniques are applied to induce stationarity. Differencing serves the dual purpose of addressing non-stationarity and eliminating seasonality from the data, thus facilitating a more robust and reliable foundation for subsequent time series modeling.

In the realm of time series data analysis, understanding and dissecting its diverse components are paramount for effective modeling. Techniques like Singular Value Decomposition (SVD) and Seasonal-Trend decomposition using LOESS (STL) serve as valuable tools for decomposition, revealing essential insights into the inherent patterns. Decomposition typically uncovers components such as trend, seasonality, and residual elements, each contributing distinct characteristics to the overall dataset. Following decomposition, the analyst evaluates the strength of seasonality and trend, prompting considerations for seasonally adjusting or detrending the data. Seasonal adjustment entails the removal of recurrent patterns at fixed intervals, while detrending focuses on eliminating long-term trends. These adjustments refine the data, providing a nuanced understanding of its dynamics and laying the foundation for more accurate and insightful time series modeling.

In context, a data can be either seasonal or trend and sometimes it can be both seasonal and trended which depends solely upon the data. In order to handle these kinds of cases, we have an essential method in time series analysis which is called Holts-Winter method, designed to handle data with both seasonal and trend components, capturing a comprehensive view of the underlying patterns. Recognizing that a full model with all variables may not always be optimal, feature selection becomes crucial. This process involves evaluating the p-values of an Ordinary Least Squares

(OLS) model, eliminating variables with high p-values as they are likely correlated and unnecessary. The refined model is then developed using multiple linear regression, and its performance is assessed through metrics such as AIC/BIC scores, R-squared/adjusted R-squared values, and condition number. Higher R-squared values signify a superior model, while elevated AIC/BIC and condition numbers indicate a suboptimal model.

Simultaneously, the data is explored using fundamental models—Average, Naive, Drift, and Simple Exponential Smoothing—to gauge the feasibility of forecasting with simpler approaches before delving into more complex linear models. Another effective strategy involves employing auto-regressive and moving-average techniques to build improved models like ARMA, ARIMA, and SARIMA. By experimenting with different combinations of these methods, the best-performing model is identified for the final forecasting, ensuring a robust and accurate prediction of future trends in the time.

A detailed description about these techniques is discussed in the next chapter**.**

METHOD, THEORY AND PROCEDURES

Stationary data:

The term "stationary" in the context of time series data refers to a state where the statistical properties of the data remain constant or do not exhibit significant changes over time. In a stationary time series, key statistical measures such as the mean, variance, and autocorrelation remain consistent throughout the observed period.

Rolling mean and variance:

In time series analysis, "rolling" refers to the practice of constructing a rolling or moving window of a specified size across a dataset and performing statistical calculations within that window. This technique is particularly useful for gaining insights into trends and patterns that might not be immediately apparent in the entire dataset. Rolling computations, such as rolling mean and rolling variance, involve calculating the mean and variance, respectively, of the data within the defined window as it moves through the entire dataset. This approach is valuable for smoothing out fluctuations, highlighting trends, and identifying variations within the time series data, contributing to a more dynamic and localized understanding of the underlying patterns over time.

$$\text{Mean} = \frac{\text{Sum of observations}}{\text{Total number of observations}}$$

$$\text{Variance} = \frac{\text{Sum of squared differences between each number and the mean}}{\text{Number of observations in the dataset}}$$

The iterative computation of rolling mean and variance generates two lists, which are subsequently utilized for time-based plotting. This visualization approach enables the examination of temporal patterns and facilitates the determination of data stationarity. The plots serve as a visual diagnostic tool, allowing for the assessment of stationary characteristics by observing the constancy of mean and variance across the temporal domain. In this context, a data series is considered stationary when the plotted mean and variance exhibit stability over time. This systematic analysis aids in

discerning patterns and trends critical for informing subsequent steps in the time series analysis process.

ADF test:

The Augmented Dickey-Fuller (ADF) test serves as a statistical tool, specifically a unit root test, aimed at determining the stationarity of time series data through hypothesis testing. This test evaluates the presence of a unit root in the data, which is indicative of non-stationarity and the existence of a trend.

The ADF test produces three key parameters in its results: the test statistic, p-value, and critical values. These elements play a pivotal role in the hypothesis testing process.

The hypotheses associated with the ADF test are structured as follows:

- Null Hypothesis (H_0): The null hypothesis posits the presence of a unit root in the time series, indicating non-stationarity.
- Alternate hypothesis (H_1) :The alternate hypothesis is formulated under the assumption that the null hypothesis is rejected, suggesting the absence of a unit root and thus implying stationarity in the time series.

The decision to reject the null hypothesis is contingent on the test statistic being sufficiently negative, accompanied by a p-value below a specified threshold, commonly set at 0.05. In such cases, it is concluded that the time series data is stationary. The ADF test serves as a robust tool in the analysis of time series data, providing valuable insights into its stationarity characteristics.

KPSS test:

The Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test is an additional diagnostic tool employed to ascertain the stationarity of time series data, presenting a complementary approach to the Augmented Dickey-Fuller (ADF) test. Both tests serve as unit root tests, but their hypotheses are framed in opposition.

Unlike the ADF test, which examines the presence of a unit root as an indicator of non-stationarity and a discernible trend, the KPSS test focuses on determining stationarity around a deterministic trend. In other words, the KPSS test aims to identify whether the data exhibits stationarity with a well-defined trend.

The hypotheses for the KPSS test are as follows:

- Null Hypothesis (H_0): The null hypothesis asserts the presence of stationarity around a deterministic trend.
- Alternate hypothesis (H_1): Conversely, the alternate hypothesis posits non-stationarity, indicating the absence of a well-defined trend.

Interpreting the results involves comparing the test statistic against critical values. If the test statistic exceeds the critical values, the null hypothesis of stationarity around a deterministic trend is rejected, implying non-stationarity. Conversely, a failure to reject the null hypothesis suggests that the data may be stationary around a deterministic trend.

In summary, the KPSS test provides valuable insights into the stationarity properties of time series data, specifically focusing on the presence or absence of a deterministic trend.

Autocorrelation:
Autocorrelation measures the linear relationship between current and lagged values in a time series. Denoted as $R\hat{}\, y\,(\tau)$ it represents the extent of similarity between a stationary time series y(t) and its past values at various lags. A +1 autocorrelation signifies a perfect positive correlation, while -1 indicates a perfect negative correlation. This function is crucial in revealing temporal dependencies and patterns within a time series, facilitating insights into historical relationships and aiding in forecasting.

$$\hat{R}_y(\tau) = \frac{\sum_{t=\tau+1}^{T}(y_t - \overline{y})(y_{t-\tau} - \overline{y})}{\sum_{t=1}^{T}(y_t - \overline{y})^2}$$

Average method:
The average method for forecasting assumes that all historical data points are equally important, predicting all future values to be the mean of past observations. This approach is particularly effective for short-term trend forecasting.

$$\hat{y}_{T+h|T} = \frac{y_1 + y_2 + \ldots + y_T}{T}$$

Naïve method:

Naive forecasts involve setting all predictions equal to the value of the last observation. This method assumes that the most recent observation is crucial, while disregarding any information from previous observations. While sometimes effective, especially in economic and financial time series data, it may yield significant errors in certain cases.

$$\hat{y}_{T+h|T} = y_T$$

Drift method:

A variation of the naive method involves incorporating a drift, allowing the forecast to exhibit an increasing or decreasing trend over time. The drift is determined by the average change observed in the historical data, essentially extending a line between the first and last observations and extrapolating it into the future.

$$\hat{y}_{T+h|T} = y_T + \frac{h}{T-1}\sum_{t=2}^{T}(y_t - y_{t-1}) = y_T + h(\frac{y_T - y_1}{T-1})$$

Simple Exponential Smoothing (SES):

Simple exponential smoothing is calculated using weighted averages where the weights decrease exponentially as observations come from further in the past, the smallest weights are associated with the oldest observations.

$$\boxed{\hat{y}_{t+1|t} = \alpha y_t + (1-\alpha)\hat{y}_{t|t-1}}$$

Mean squared error:

In statistics, the mean squared error (MSE) or mean squared deviation of an estimator quantifies the average of the squared errors. It represents the mean of the squared differences between the estimated values and the actual values in a dataset. The MSE is a valuable metric for assessing the accuracy and precision of an estimator.

$$MSE = mean(e_t^2)$$

Multiple linear regression model:

The multiple linear regression model is,
$yt = \beta 0 + \beta 1x1,t + \beta 2x2,t + \cdots + \beta kxk,t + \varepsilon t$

$\beta 0, \beta 1, \ldots \beta k$ are unknown values which needs to be estimated using LSE using the following equation:

$\beta\hat{} = (XT.X)^{-1}\, XT.Y$

The matrix $\mathbf{X}$ has the form:

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & x_{2,1} & \dots & x_{k,1} \\ 1 & x_{1,2} & x_{2,2} & \dots & x_{k,2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{1,T} & x_{2,T} & \dots & x_{k,T} \end{bmatrix}$$

where X and Y are given as,
Matrix X has T rows and (k+1) columns where T is the number of samples and k is number of independent variables.

SVD:
Singular Value Decomposition or SVD the popular technique for dimensionality reduction. This is a linear algebra technique to create a projection of a sparse dataset prior to fitting a model.
s, d, v = np.linalg.svd(H)

Backwards stepwise regression:

Backwards stepwise regression is a model selection technique with the following iterative process:

- **Begin with a model that includes all potential predictors.**
- **Remove one predictor at a time and assess whether the removal improves the measure of predictive accuracy.**
- **Iterate this process until no further improvement is achieved.**

This approach involves systematically refining the model by eliminating predictors, retaining only those that enhance the model's predictive accuracy. It provides a methodical way to simplify and optimize the model by iteratively considering the impact of each predictor's removal on the overall performance.

Estimated Variance:

$$\hat{\sigma}_e = \sqrt{\frac{1}{T-k-1}\sum_{t=1}^{T} e_t^2}$$

STL decomposition method:

STL, which stands for "Seasonal and Trend decomposition using Loess," incorporates the Loess method, designed for estimating nonlinear relationships. STL presents several advantages over classical methods such as SEATS and X11:

- Versatility in Handling Seasonality: STL is capable of handling various types of seasonality, not limited to just monthly or quarterly patterns commonly addressed by classical methods.
- Dynamic Seasonal Component: In STL, the seasonal component is permitted to vary over time, offering flexibility in capturing changing seasonal patterns. The user has control over the rate of change in the seasonal component.
- User-Controlled Smoothness of Trend-Cycle: One of the distinctive features of STL is that it allows users to control the smoothness of the trend-cycle. This parameter enables customization based on the specific characteristics of the data being analyzed.

By leveraging these features, STL provides a robust and adaptable method for decomposing time series data into its seasonal and trend components, making it well-suited for a variety of data patterns and trends.

Seasonally adjusted data:

Additive decomposition: seasonally adjusted data given by: $y_t - S_t = T_t + R_t$

Multiplicative decomposition: seasonally adjusted data given by: $y_t/S_t = T_t \times R_t$

Strength of trend and seasonality:

The rubric to measure the strength of the trend is given as :

$F_T = \max\{0, 1 - \mathrm{Var}(R_t)\ \mathrm{Var}(T_t + R_t)\}$

For strongly trended data, the rubric FT is large.

The rubric to measure the strength of the seasonality is given as

FS = max{0, 1 − Var(Rt) Var(St + Rt) }

FS close to 0 exhibits almost no seasonality, FS close to 1 exhibits strong seasonality

Auto regressive model:

In a multiple regression model, the forecast for the variable of interest is derived from a linear combination of predictors. On the other hand, in an autoregressive model, the forecast for the variable of interest is based on a linear combination of its own past values.

An AR process of order na AR(na) can be written as :

y(t) + a1y(t − 1) + a2y(t − 2) + . . . + ana y(t − na) = e(t)

where y(t) is the variable of interest and ϙ(t) is white noise (WsN ∼ (0, σ2e).

Moving average model:

Rather than using past values of the forecast variable in a regression, a moving average model uses past forecast errors in a regression-like model:

y(t) = e(t) + b1e(t − 1) + b2e(t − 2) + . . . + bnb e(t − nb)

where e(t) is white noise WN ∼ (0, σ2e). This is called a moving average model of order nb , MA(nb).

Generalized Partial Autocorrelation:

The generalized partial autocorrelation is used to estimate the order of the ARMA model when na != 0 and nb != 0.

$$\phi_{kk}^{j} = \frac{\begin{vmatrix} \hat{R}_y(j) & \hat{R}_y(j-1) & \dots & \hat{R}_y(j+1) \\ \hat{R}_y(j+1) & \hat{R}_y(j) & \dots & \hat{R}_y(j+2) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{R}_y(j+k-1) & \hat{R}_y(j+k-2) & \dots & \hat{R}_y(j+k) \end{vmatrix}}{\begin{vmatrix} \hat{R}_y(j) & \hat{R}_y(j-1) & \dots & \hat{R}_y(j-k+1) \\ \hat{R}_y(j+1) & \hat{R}_y(j) & \dots & \hat{R}_y(j-k+2) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{R}_y(j+k-1) & \hat{R}_y(j+k-2) & \dots & \hat{R}_y(j) \end{vmatrix}}$$

Levenberg-Marquardt Algorithm:

The Levenberg-Marquardt Algorithm is a versatile optimization technique employed in nonlinear regression. It effectively integrates features from both the gradient descent and Gauss-Newton methods. Initially, when parameters are distant from their optimal values, the algorithm behaves akin to the gradient descent method, minimizing the sum of squared errors through parameter updates in the steepest-descent direction. As the parameters approach their optimal values, the

algorithm smoothly transitions to operate more like the Gauss-Newton method. Here, it assumes the least squares function is locally quadratic, aiming to find the minimum of this quadratic to further reduce the sum of squared errors. This adaptability makes the Levenberg-Marquardt Algorithm an effective and efficient tool for optimizing nonlinear regression models.

Seasonal ARIMA:

For seasonal time series with period s, observations that are intervals apart are similar.
SARIMA (Na, D, Nb) with period of seasonality s can be written as:

$$A(q^{-s})\nabla_s^D y_t = B(q^{-s})\epsilon_t$$

where ∇^D is given as

$$\nabla_s^D = (1 - q^{-s})^D$$
$$A(q^{-s}) = 1 - A_1 q^{-s} - A_2 q^{-2s} - \ldots - A_{N_a} q^{-N_a s}$$
$$B(q^{-s}) = 1 - B_1 q^{-s} - B_2 q^{-2s} - \ldots - B_{N_b} q^{-N_b s}$$

Procedure:

The outlined procedure for time series analysis involves the following steps:

- ★ Data Loading and Descriptive Statistics:
  - Load the dataset and provide descriptive statistics about the dependent variable.
- ★ Stationarity Check:
  - Check for stationarity using methods like Augmented Dickey-Fuller (ADF) test, rolling mean, etc.
  - Make the dataset stationary if it is not through appropriate transformations.
- ★ Time Series Decomposition:
  - Perform time series decomposition to understand and separate the components of the series.
- ★ Holts-Winter Method:
  - Apply the Holts-Winter method separately on the train and test data for time series forecasting.
- ★ Feature Selection:
  - Conduct feature selection to identify relevant variables for modeling.
- ★ Multiple Linear Regression:
  - Build a multiple linear regression model based on the selected features.
- ★ Base Model Predictions:
  - Perform predictions on base models to evaluate their fit.

- ★ ARMA, ARIMA, and SARIMA Models:
  - Develop ARMA, ARIMA, and SARIMA models based on the Generalized Partial Autocorrelation (GPAC) orders.
- ★ Model Selection:
  - Choose the final model based on the performance measures of each model.
- ★ Forecast Function:
  - Develop a forecast function to facilitate h-step predictions based on the selected model.

This comprehensive procedure covers data exploration, stationarity checks, multiple modeling approaches, and final model selection, providing a structured framework for time series analysis and forecasting.

EXPERIMENTAL SETUP

Required libraries:

The following python libraries are required to run the code file seamlessly.

```
{from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split, GridSearchCV
import statsmodels.tsa.holtwinters as ets
from sktime.forecasting.exp_smoothing import ExponentialSmoothing
from statsmodels.stats.diagnostic import acorr_ljungbox
from sklearn.decomposition import PCA
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
from numpy import linalg as la
from Toolbox import *
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

}
```

DESCRIPTION OF THE DATASET:

This dataset encapsulates underwater temperature (°C) observations from seven islands and two submerged rocky reefs positioned along the Santa Catarina coast, southern Brazil, within the geographic coordinates of 26°22’ S and 28°26’ S. Recorded every 20 minutes, spanning the period from December 2012 to July 2014, the temperature data was meticulously collected using HOBO Pendant® Temperature Data Logger UA-002 devices. Installed underwater by SCUBA divers, these data loggers were securely anchored to rocky reefs with epoxy at varying depths: 5 m and 12 m in the islands, and 22 m in the submerged reefs. It is noteworthy that certain depth records are absent due to equipment loss at specific sites. The dataset is characterized by seven variables: Site, Latitude, Longitude, Date, Time of Sampling, Temperature (°C), and Depth (meters), providing a structured and comprehensive foundation for in-depth analysis and modeling of the underwater temperature dynamics in this ecologically significant coastal region. Overall, the dataset has 8 columns, among which 1 is categorical and all others are numerical.

More information about the dataset can be found here:

Sea scientific open data publication

Attribute Information:

Attribute	**Information**
ID	**Numbering the observations**
Site	**Names of all the coastal islands**
Latitude	**Latitudinal location between certain points of observation (26°22’ N and 28°26’ S)**
Longitude	**Longitudinal location between 48°3’ E and 48°7’ W**
Date	**The date of observation**
Time of sampling	**The time of observation**
Temperature	**The recorded temperature in (°C)**
Depth	**Max depth in Meters (22m)**

Table1: Attribute Information:

Dataset Source:

https://www.seanoe.org/data/00510/62120/

https://www.kaggle.com/datasets/shivamb/underwater-surface-temperature-dataset

Looking at the dataset information, it is clear the data has been recorded from 7 islands and there is one possible dependent variable: "Temp (*C)".

06. PRE-PROCESSING DATASET:

By analyzing the dataset, it is evident that the dataset has null and nan values at its temperature column. I used forward filling to fill the missing values in the dataset. The 'date' and 'time' variable is created as 'date time' variable for the dataset and the dependent variable is 'Temp (*C) '. The project throughout will be using 'Temp (*C)' as the dependent variable with other predictor variables.

These are the columns of the raw dataset:

```
Data columns (total 8 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   #          408638 non-null  int64
 1   Site       408638 non-null  object
 2   Latitude   408638 non-null  float64
 3   Longitude  408638 non-null  float64
 4   Date       408638 non-null  object
 5   Time       408638 non-null  object
 6   Temp (°C)  408634 non-null  float64
 7   Depth      408638 non-null  float64
dtypes: float64(4), int64(1), object(3)
memory usage: 24.9+ MB
```

The data has null values : **After forward filling data:**

```
#           0
Site        0
Latitude    0
Longitude   0
Date        0
Time        0
Temp (°C)   4
Depth       0
dtype: int64
```

```
#           0
Site        0
Latitude    0
Longitude   0
Date        0
Time        0
Temp (°C)   0
Depth       0
dtype: int64
```

Renamed the column "#" with "INDEX"

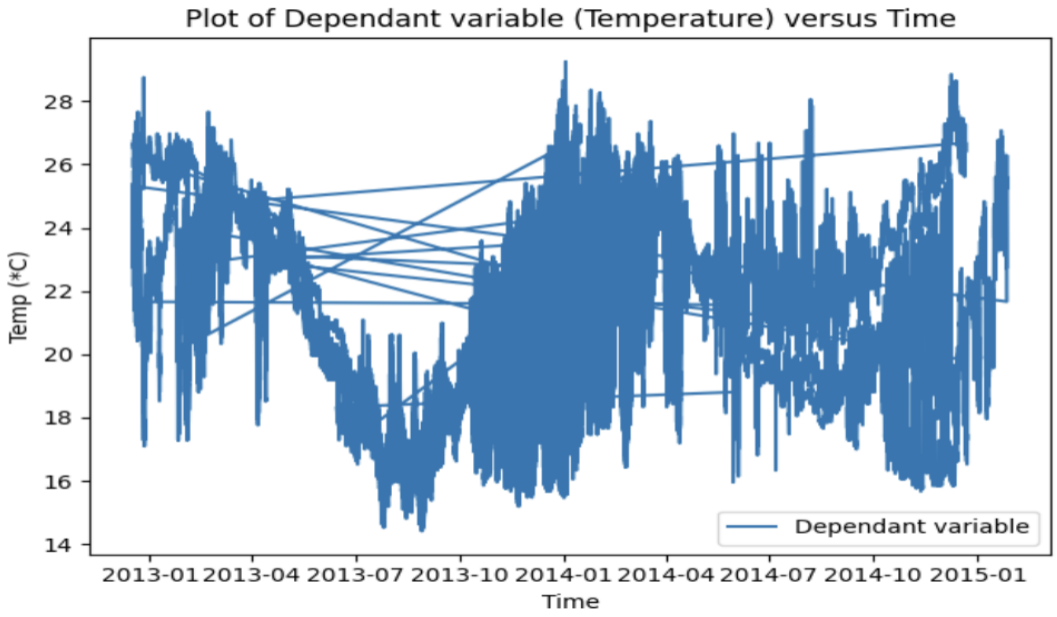
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 408638 entries, 0 to 408637
Data columns (total 8 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   INDEX      408638 non-null  int64
 1   Site       408638 non-null  object
 2   Latitude   408638 non-null  float64
 3   Longitude  408638 non-null  float64
 4   Date       408638 non-null  object
 5   Time       408638 non-null  object
 6   Temp (°C)  408638 non-null  float64
 7   Depth      408638 non-null  float64
dtypes: float64(4), int64(1), object(3)
memory usage: 24.9+ MB
```

Since the dataset has a categorical column, I proceeded with the one-hot coding resulting in the below results.

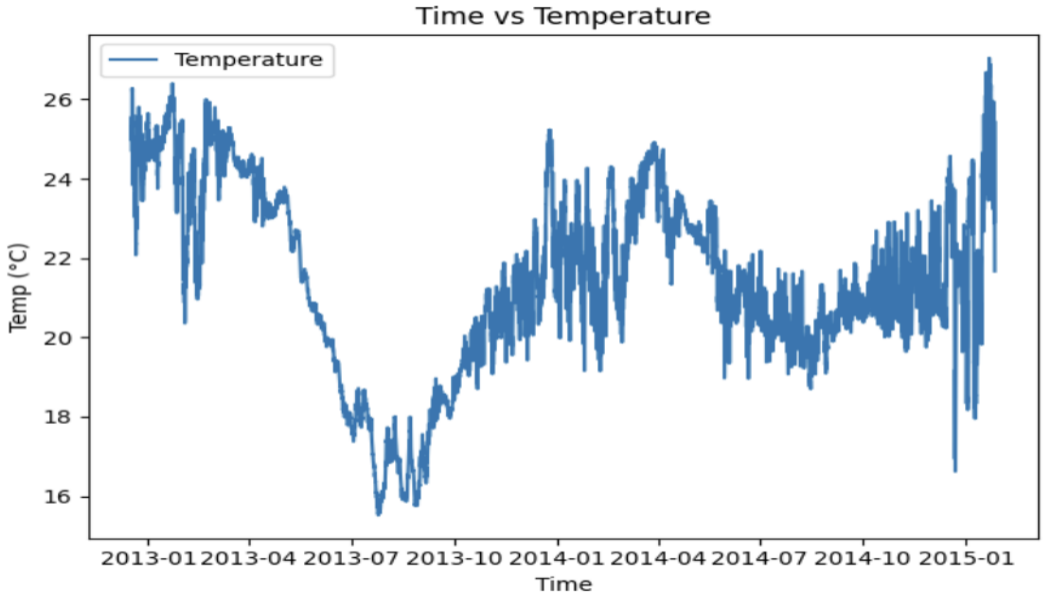
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 408638 entries, 0 to 408637
Data columns (total 16 columns):
 #   Column                            Non-Null Count   Dtype
---  ------                            --------------   -----
 0   INDEX                             408638 non-null  int64
 1   Latitude                          408638 non-null  float64
 2   Longitude                         408638 non-null  float64
 3   Date                              408638 non-null  datetime64[ns]
 4   Time                              408638 non-null  object
 5   Temp (°C)                         408638 non-null  float64
 6   Depth                             408638 non-null  float64
 7   Site_Ilha Deserta                 408638 non-null  uint8
 8   Site_Ilha da Galé                 408638 non-null  uint8
 9   Site_Ilha do Coral                408638 non-null  uint8
 10  Site_Ilha dos Lobos               408638 non-null  uint8
 11  Site_Moleques do Sul              408638 non-null  uint8
 12  Site_Parcel da Pombinha           408638 non-null  uint8
 13  Site_Parcel do Xavier (Alalunga)  408638 non-null  uint8
 14  Site_Tamboretes                   408638 non-null  uint8
 15  Site_lha do Xavier                408638 non-null  uint8
dtypes: datetime64[ns](1), float64(4), int64(1), object(1), uint8(9)
memory usage: 25.3+ MB
```

07. STATIONARITY:

ANALYSIS OF DEPENDENT VARIABLE:



The above plot of dependent variable against time shows the data is more and there are too many points in one date because the data is recorded for every 20 minutes. So I proceeded with downsampling the data to 1 hour.



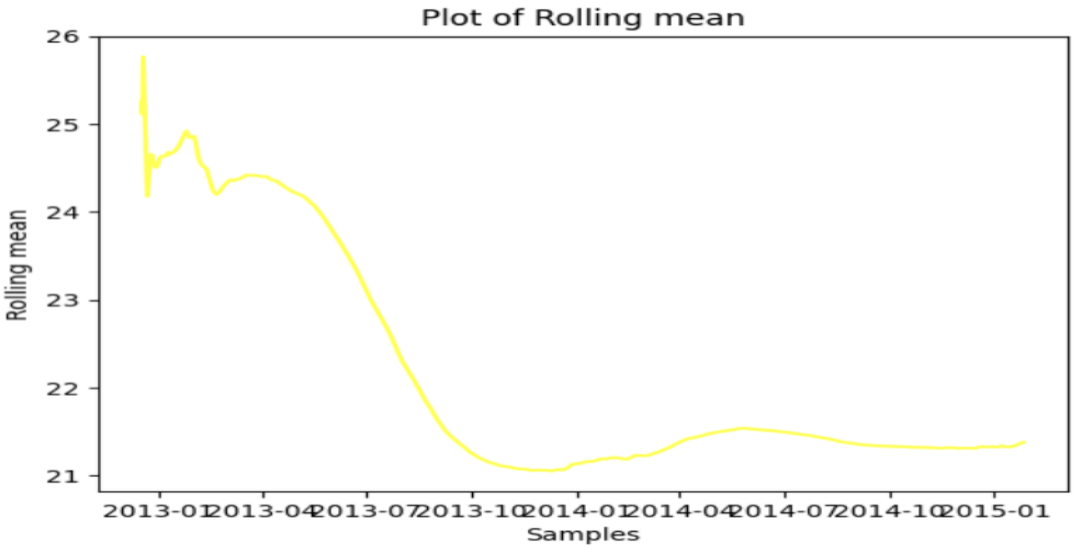
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18487 entries, 0 to 18486
Data columns (total 15 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Date                              18487 non-null  datetime64[ns]
 1   INDEX                             18487 non-null  float64
 2   Latitude                          18487 non-null  float64
 3   Longitude                         18487 non-null  float64
 4   Temp (°C)                         18487 non-null  float64
 5   Depth                             18487 non-null  float64
 6   Site_Ilha Deserta                 18487 non-null  float64
 7   Site_Ilha da Galé                 18487 non-null  float64
 8   Site_Ilha do Coral                18487 non-null  float64
 9   Site_Ilha dos Lobos               18487 non-null  float64
 10  Site_Moleques do Sul              18487 non-null  float64
 11  Site_Parcel da Pombinha           18487 non-null  float64
 12  Site_Parcel do Xavier (Alalunga)  18487 non-null  float64
 13  Site_Tamboretes                   18487 non-null  float64
 14  Site_lha do Xavier                18487 non-null  float64
dtypes: datetime64[ns](1), float64(14)
memory usage: 2.1 MB
```

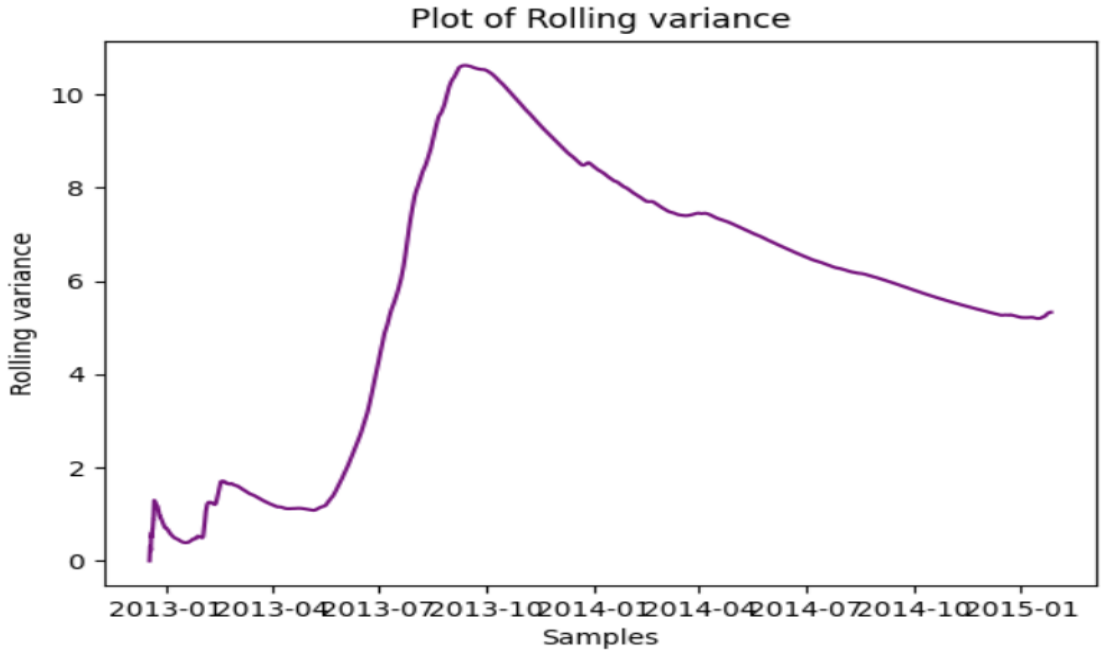
After downsampling the data points were reduced to 18487 with the encoded categorical column.

```
ADF Statistic: -3.451241
p-value: 0.009331
Critical Values:
    1%: -3.431
    5%: -2.862
    10%: -2.567
Test Statistic            2.197645
p-value                   0.010000
Lags Used                78.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
dtype: float64
```

The ADF test shows that the data is not stationary, mentioning the p-value is 0.009 greater than the threshold. The extremely low p-value (close to zero) suggests that you can reject the null hypothesis. The critical values at the 1%, 5%, and 10% levels are compared to the ADF statistic. Since the ADF statistic is much smaller than the critical values, this reinforces the rejection of the null hypothesis and the data is not stationary

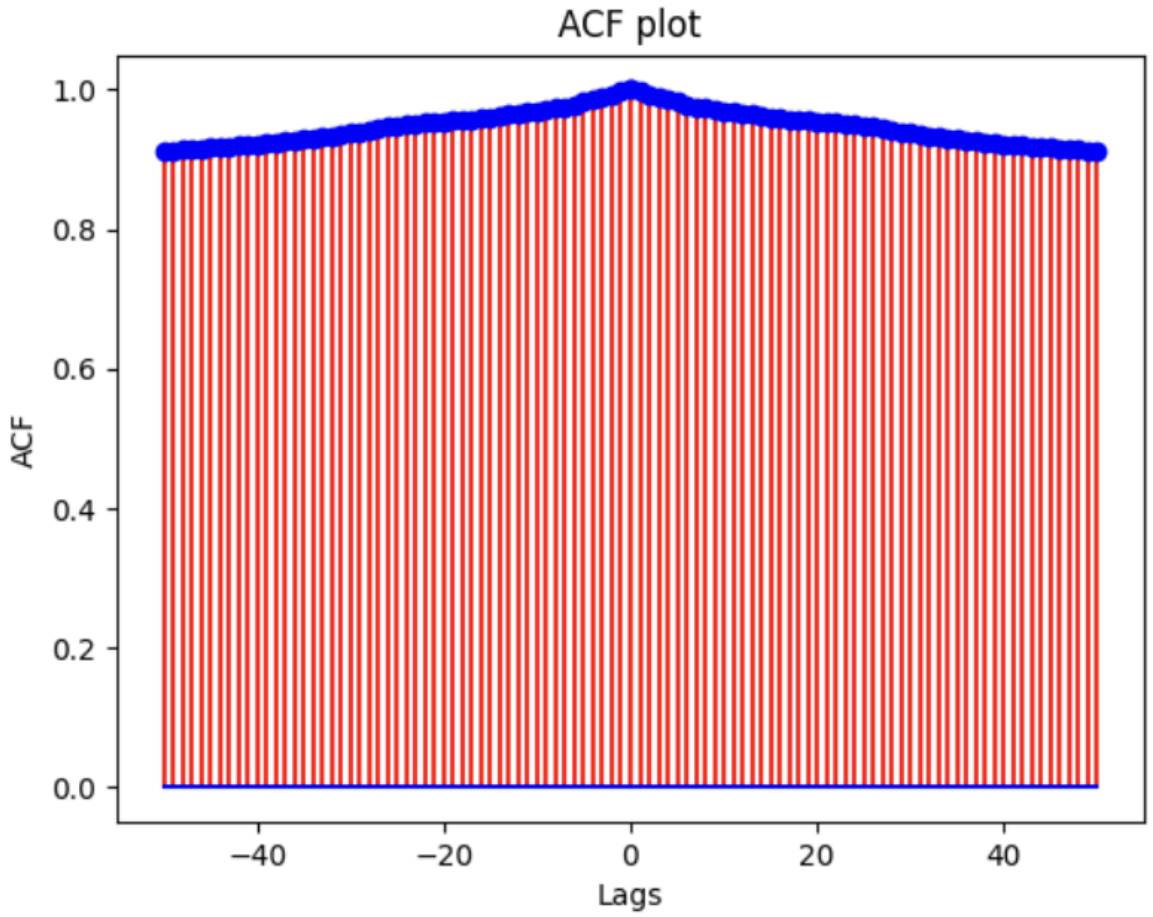
The KPSS test shows that the data is not stationary mentioning the p-value(0.010) less than the critical values and also the threshold 0.5. The null hypothesis of the KPSS test is that the data is stationary around a deterministic trend. The p-value is compared to a significance level (commonly 0.05). In this case, the p-value is less than 0.05, so we can reject the null hypothesis. The KPSS test suggests that our data is not stationary.





The rolling mean and the rolling variance shows that the data is not stationary because the mean and variance is not perfectly at 0 and there is no flat point.

Hence, proceeding with the other stationarity checks.

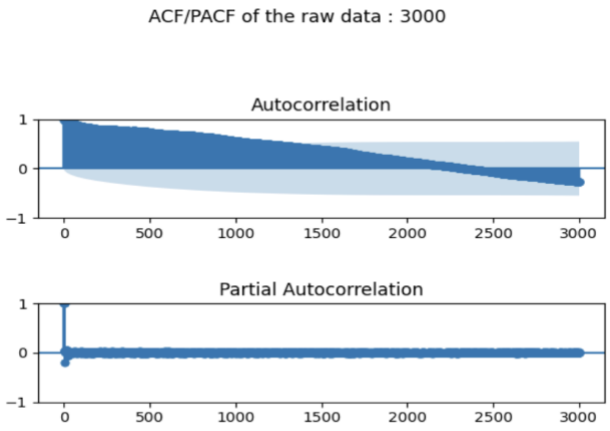


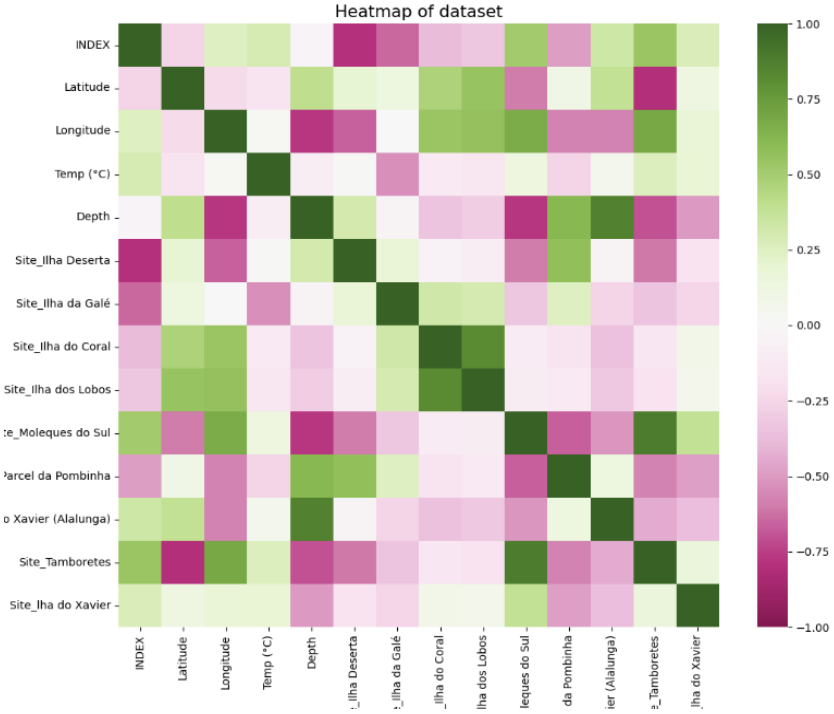
Looking at the acf plot, it is evident that the data is not stationary.

If a data is white noise, there is no correlation amongst samples over the given lags exclusive of lag 0 which happens to have the high peak then the data is stationary.

From the discussions and class notes, if the decay of the ACF plot is long, then the data is not stationary and if it is vice versa then the data is stationary.

From the above acf plot, the dependent variable doesn't have a significant decay even after the 40 lags, which typically represents non-stationarity and the data points are dependent





The temporal evolution of the dependent variable's plot suggests a potential stationarity, yet intermittent spikes throughout the timeline hint at underlying seasonality. A comprehensive analysis involved the generation of Auto-Correlation Function (ACF) and Partial Auto-Correlation Function (PACF) plots. The ACF plot substantiates the overall convergence of values over time; however, a distinctive spike is observable at every 1260th lag, indicating a recurring pattern and affirming the presence of seasonality in the data.

Upon downsampling the original data, the seasonality pattern, initially occurring at lag 420, shifted to lag 1260. This adjustment was verified through meticulous calculations based on the temporal characteristics of the original and downsampled datasets.

The original dataset, capturing data every 20 minutes across 7 islands, was downscaled to a 1-hour interval, resulting in 3 records per hour. By applying this information to the calculation of the new seasonality lag, the following formula was utilized:

Seasonality Lag=Downsampled Time Interval×Records per Hour×Islands

Substituting the values:

Seasonality Lag =60 minutes×3 records per hour×7islands

This computation yielded a seasonality lag of 1260 minutes. Therefore, with the data now downsampled to 1 hour, the revised seasonality pattern manifests every 1260 minutes, signifying the recurrence of the seasonal trend within the context of data recorded every 1 hour across the 7 islands.

Further insights from the symmetric ACF plot underscore a pronounced seasonality pattern. The plot exhibits a discernible convergence over positive ACF values, followed by a transition into negative ACF values. This elucidates a structured cyclicality within the data, contributing valuable information for subsequent analytical considerations.

The PACF plot of the data exhibits a notable cut-off at lag 0, suggesting a potential point at which the data achieves stationarity. This observation serves as an indication that the autocorrelation in the data diminishes after lag 0, supporting the notion of a stabilized, stationary process. Additionally, an insightful analysis of the correlation matrix reveals pronounced darker patterns, signifying high correlation among variables within the dataset. In the pursuit of optimal model development, it becomes imperative to address this multicollinearity. Consequently, a prudent approach involves the removal of correlated variables to enhance model accuracy and interpretability.

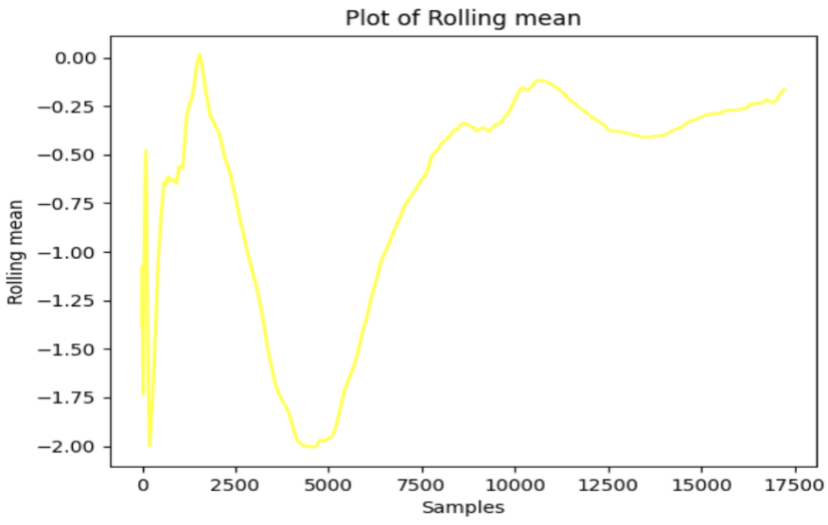
Since after all the stationarity checks, the data is not stationary, we can proceed with the differencing of the data in order to make it stationary

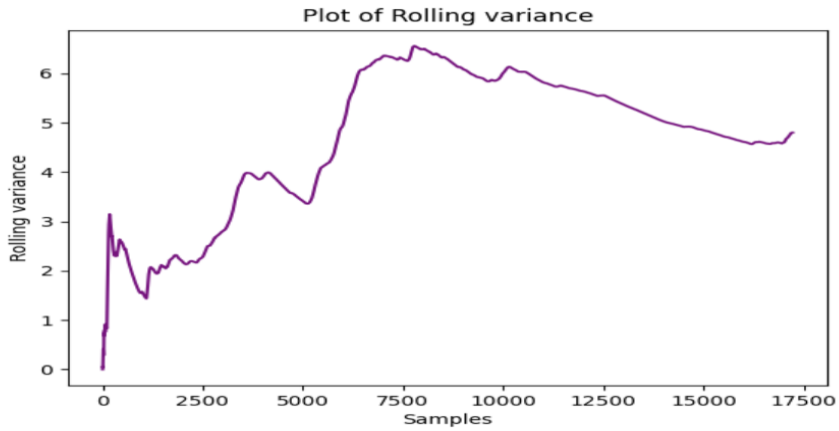
FIRST ORDER DIFFERENCING:

```
ADF Statistic: -4.350168
p-value: 0.000363
Critical Values:
    1%: -3.431
    5%: -2.862
    10%: -2.567
Test Statistic            2.460018
p-value                   0.010000
Lags Used                76.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
dtype: float64
```

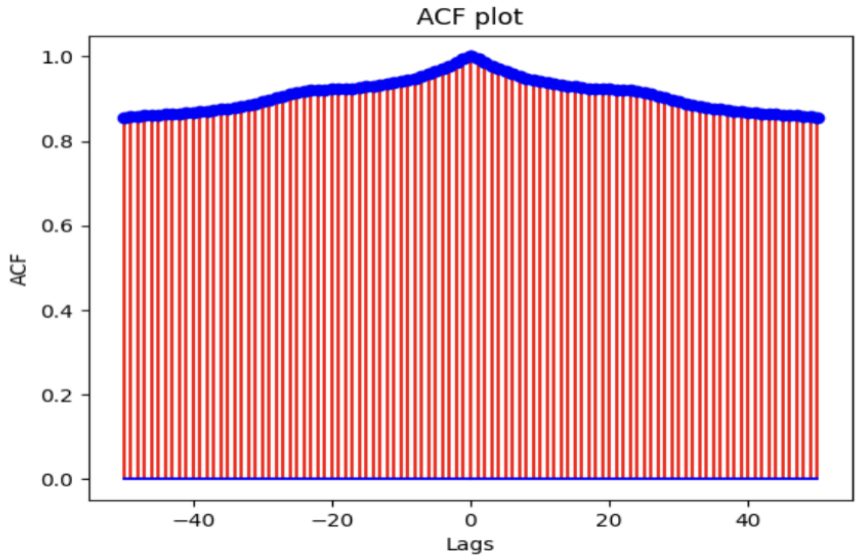
The ADF test shows that the data is not stationary, mentioning the p-value is 0.0003 greater than the threshold. [The extremely low p-value (close to zero) suggests that you can reject the null hypothesis]. The critical values at the 1%, 5%, and 10% levels are compared to the ADF statistic. Since the ADF statistic is much smaller than the critical values, this reinforces the rejection of the null hypothesis and the data is not stationary

The KPSS test shows that the data is not stationary mentioning the p-value(0.010) less than the critical values and also the threshold 0.5. The null hypothesis of the KPSS test is that the data is stationary around a deterministic trend. The p-value is compared to a significance level (commonly 0.05). In this case, the p-value is less than 0.05, so we can reject the null hypothesis. The KPSS test suggests that our data is not stationary.



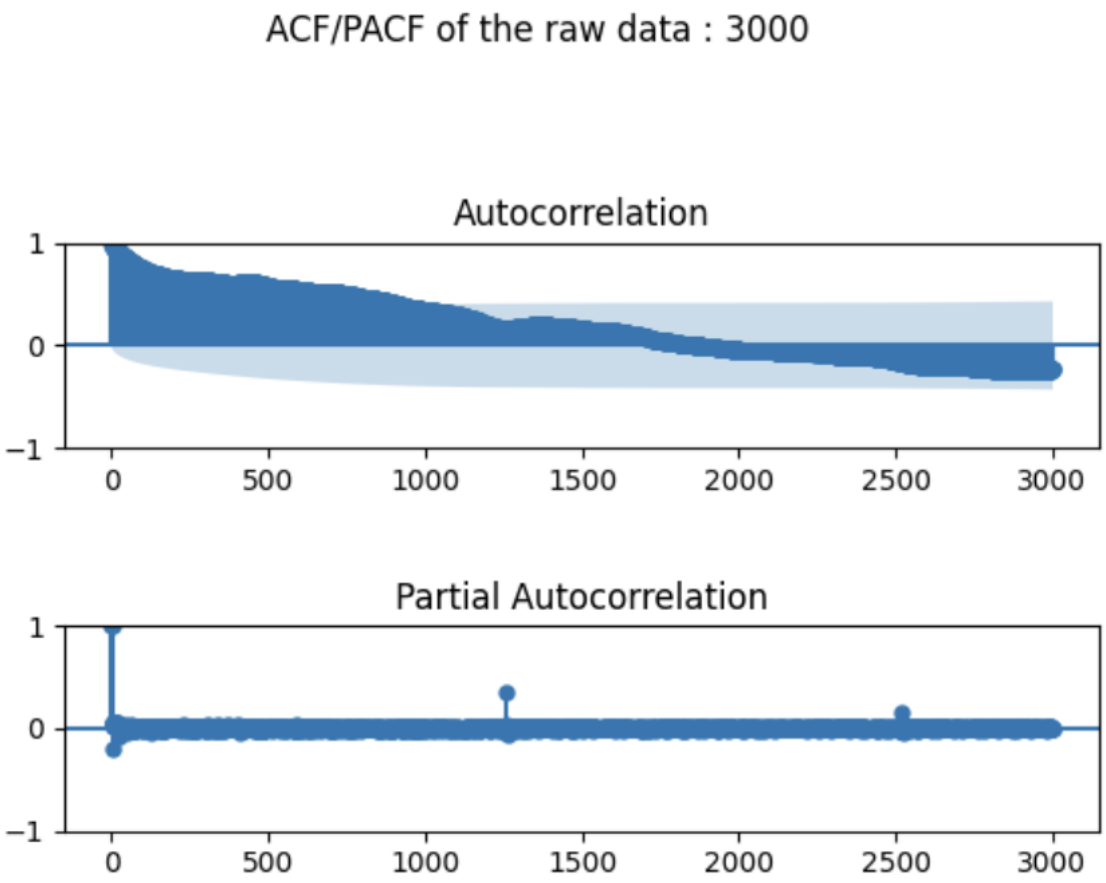


The rolling mean and the rolling variance shows that the data is not stationary because the mean is not at 0 and variance is at 0 and there is no flat point.



Looking at the acf plot, it is evident that the data is not stationary.

The data still has 0 white noise and all the points are dependent on each other even after the 40 lags.



Looking at the acf/pacf plot it is evident that the data has seasonality at every 1260th lag in the pacf plot and the acf plot shows small bumps over the lags.

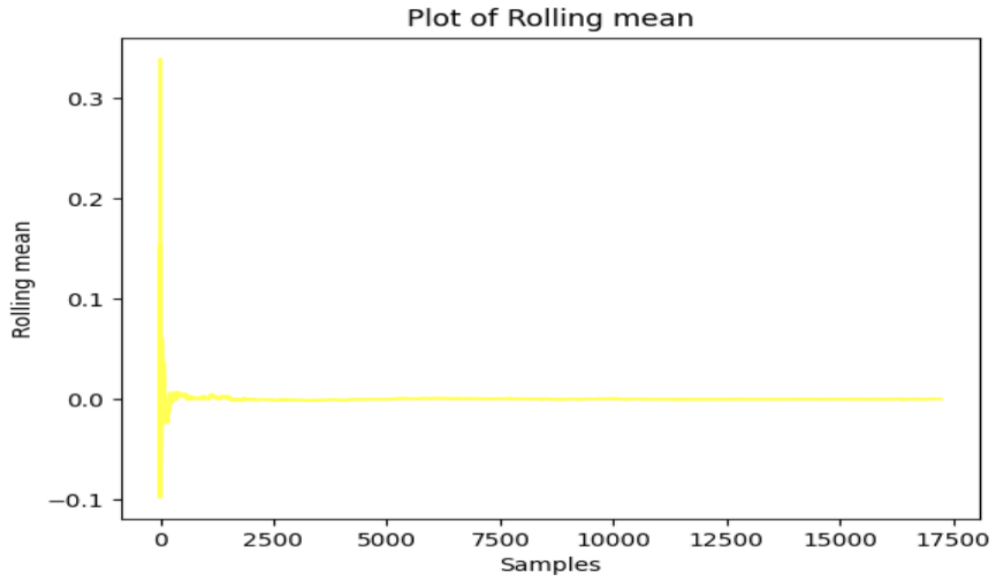
Still the data is not stationary, hence proceeding with the second order differencing in order to make the data stationary.

SECOND ORDER DIFFERENCING:

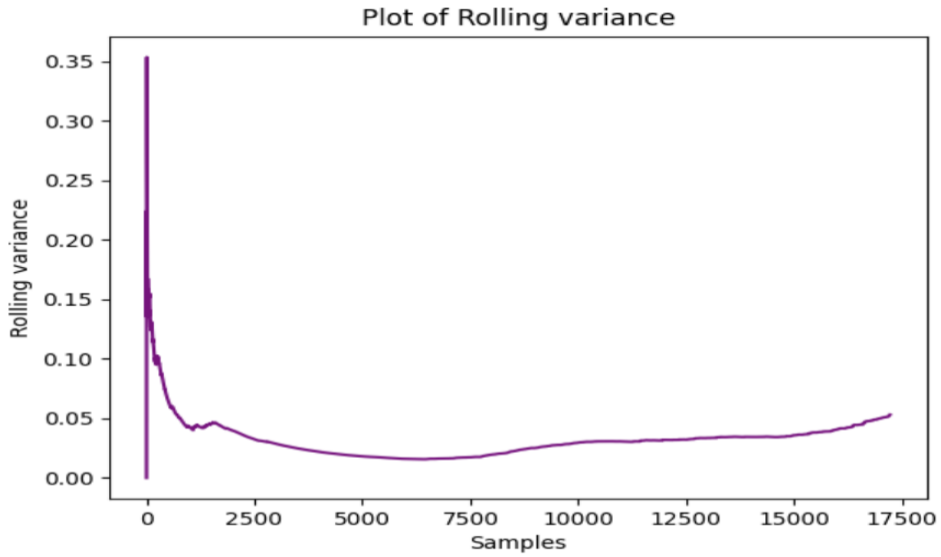
```
ADF Statistic: -23.725995
p-value: 0.000000
Critical Values:
    1%: -3.431
    5%: -2.862
    10%: -2.567
Test Statistic            0.01093
p-value                   0.10000
Lags Used                93.00000
Critical Value (10%)      0.34700
Critical Value (5%)       0.46300
Critical Value (2.5%)     0.57400
Critical Value (1%)       0.73900
dtype: float64
```

The ADF test shows that the data is stationary, mentioning the p-value is 0.0000 i.e., the p-value is closer to zero and so we tend to reject the null hypothesis and tell that the data is stationary. [The extremely low p-value (close to zero) suggests that you can reject the null hypothesis]. The critical values at the 1%, 5%, and 10% levels are compared to the ADF statistic. Looking at the ADF test, the p-value is almost zero and the ADF test statistic is more negative, so we tend to reject the null hypothesis. And say the data is stationary.

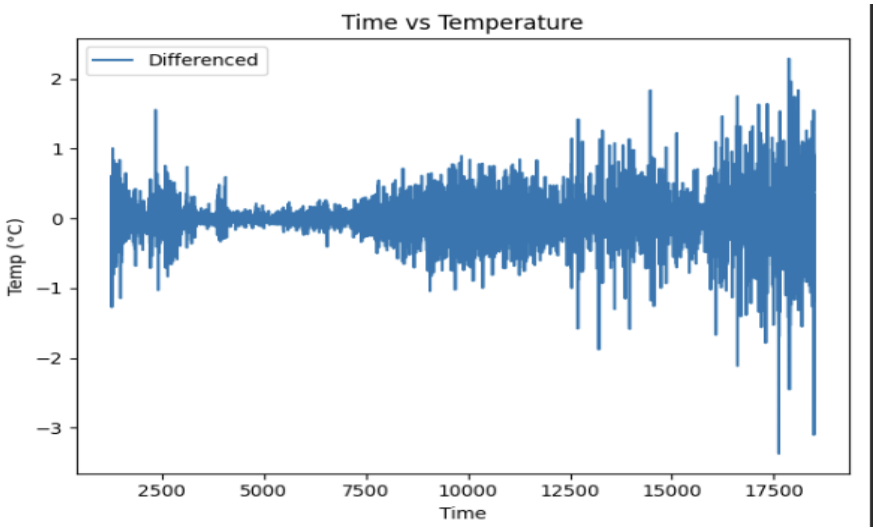
The KPSS test shows that the data is stationary mentioning the p-value(0.10) greater than the critical values and also the threshold 0.5. The null hypothesis of the KPSS test is that the data is stationary around a deterministic trend. The p-value is compared to a significance level (commonly 0.05). In this case, the p-value is less than 0.05, so we can tell that our data stationary

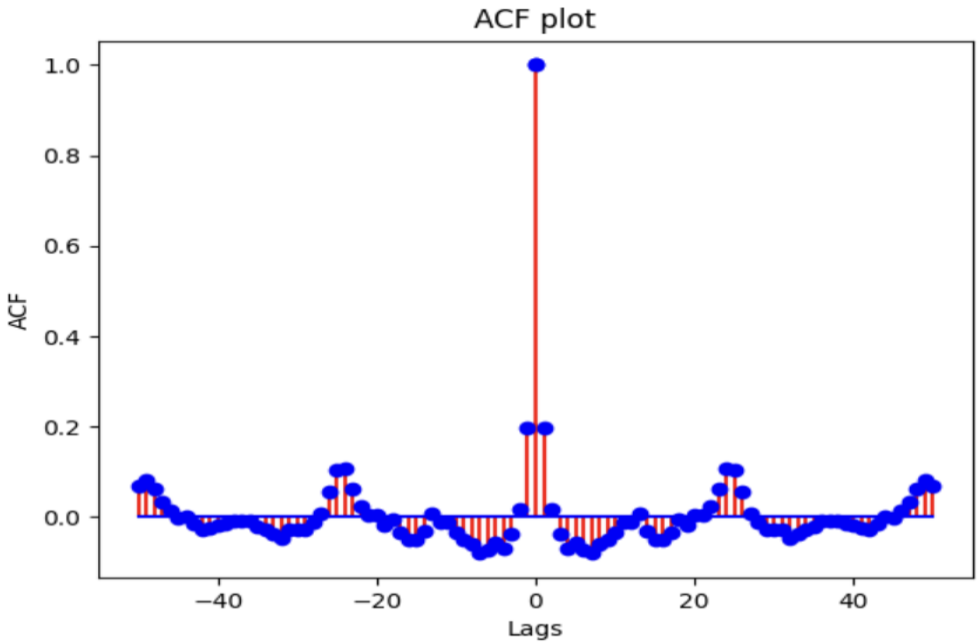


.

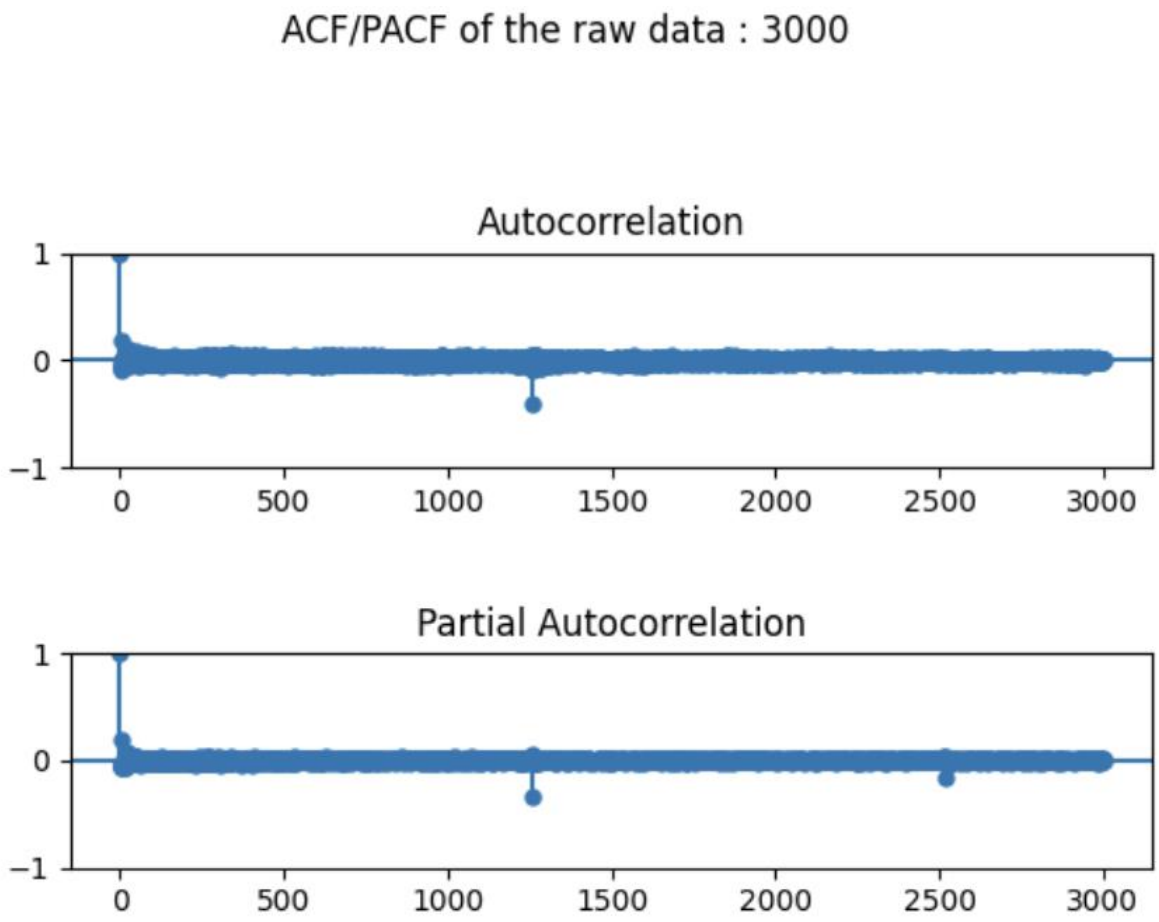


The plot of rolling mean and variance shows that the data is stationary because the plot converges into a flat line after a few iterations. But the mean of the data is zero. There is some uncertainty in the first few iterations due to seasonality but the data towards the end shows a flat curve. These minor spikes can be reduced by differencing the data.





The differenced data looks more leveled than the original data. The symmetric ACF plot shows clear seasonality where the plot converges over the positive values of ACF for a while and moves into the negative values of ACF.



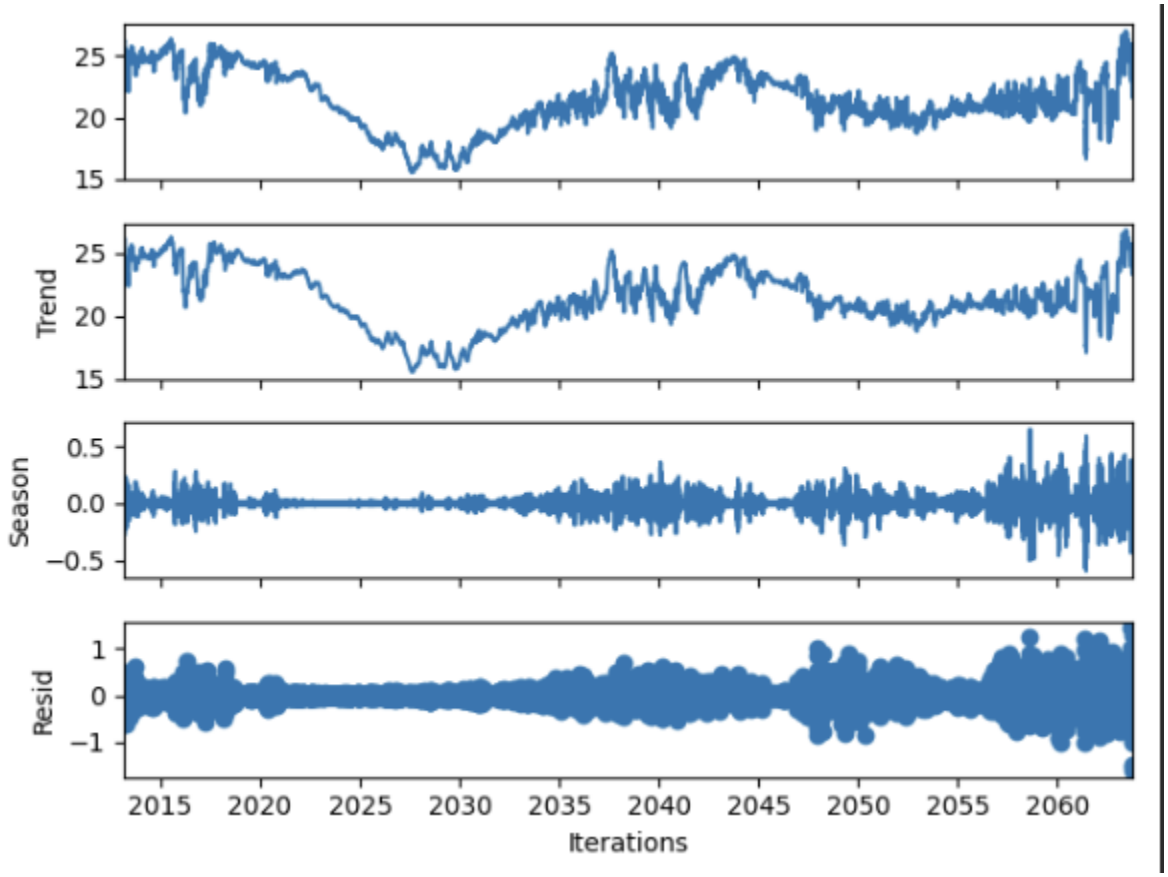
The analysis of the ACF/PACF plots provides compelling evidence that the data exhibits stationarity with a distinct seasonality pattern at a lag of 1260. The ACF plot demonstrates a clear cutoff at lags 0 and 1260, emphasizing that autocorrelation diminishes beyond these points, affirming the stationary nature of the data.

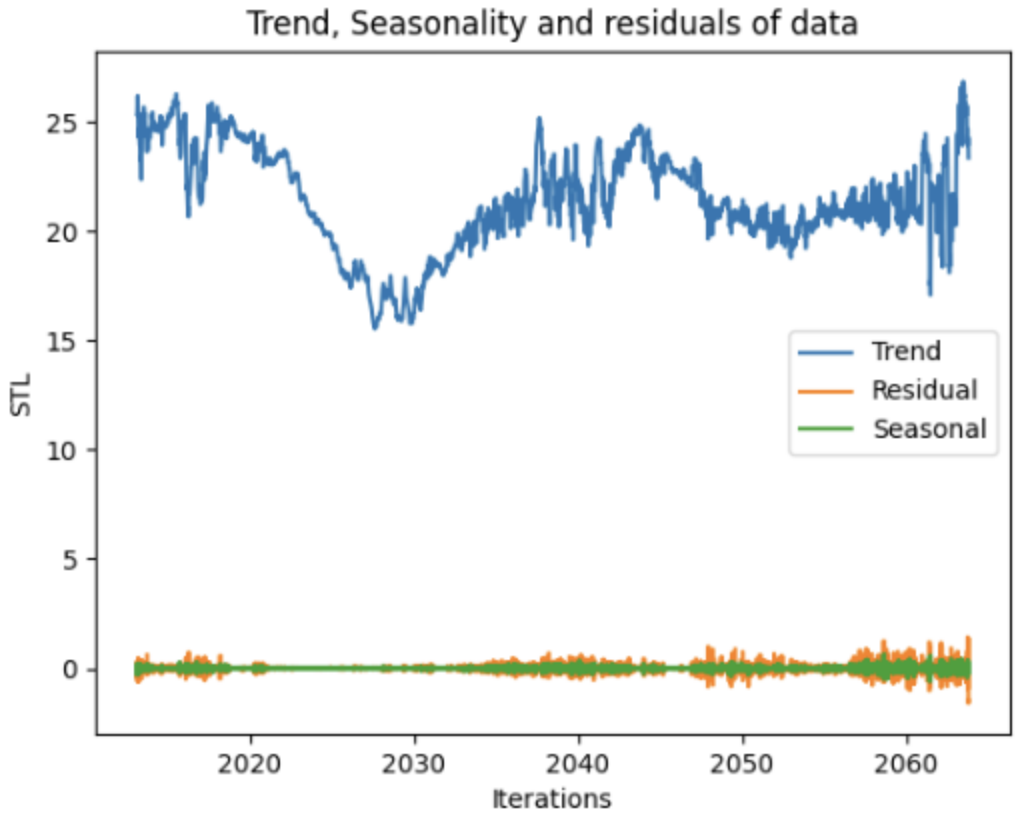
Furthermore, the PACF plot complements this observation by revealing recurring seasonality at every 1260th lag. The distinctive spikes in the PACF plot at these intervals signify a consistent and significant influence of the seasonality component on the data. This synchronization between the ACF and PACF findings underscores the robust presence of seasonality in the dataset, specifically at the lag of 1260.

In summary, the convergence of these results from the ACF and PACF analyses strengthens the conclusion that the data is stationary, and the recurring seasonality pattern manifests prominently at the lag of 1260. This comprehensive understanding lays a solid foundation for subsequent modeling and analysis endeavors.

08. TIME SERIES DECOMPOSITION:

The process of time series decomposition was executed employing the Seasonal-Trend decomposition using LOESS (STL) technique. This method was employed to unveil and comprehend the intricate patterns intrinsic to both trend and seasonality within the temporal dataset. By decomposing the time series into distinct components, namely trend, seasonality, and residuals, a nuanced understanding of the underlying temporal dynamics was attained. This rigorous analysis facilitates the discernment of long-term trends, recurrent seasonal patterns, and the residual fluctuations that collectively characterize the evolution of the dataset.





The utilization of the STL decomposition technique proves instrumental in dissecting the composition of residuals, trend, and seasonal patterns embedded within the dataset. The resulting STL plot reveals distinctive features, with scattered residual values indicative of nuanced fluctuations. Notably, discernible spikes on both the trend and seasonal plots suggest the plausible presence of underlying trends and recurrent seasonal patterns within the data.

To quantitatively validate the extent of trend and seasonality, a rigorous analysis involves the computation of the strength of both components. This calculation serves as a metric to ascertain the relative dominance of trend and seasonality within the dataset, providing valuable insights into the temporal dynamics and aiding in the comprehensive characterization of the time series.

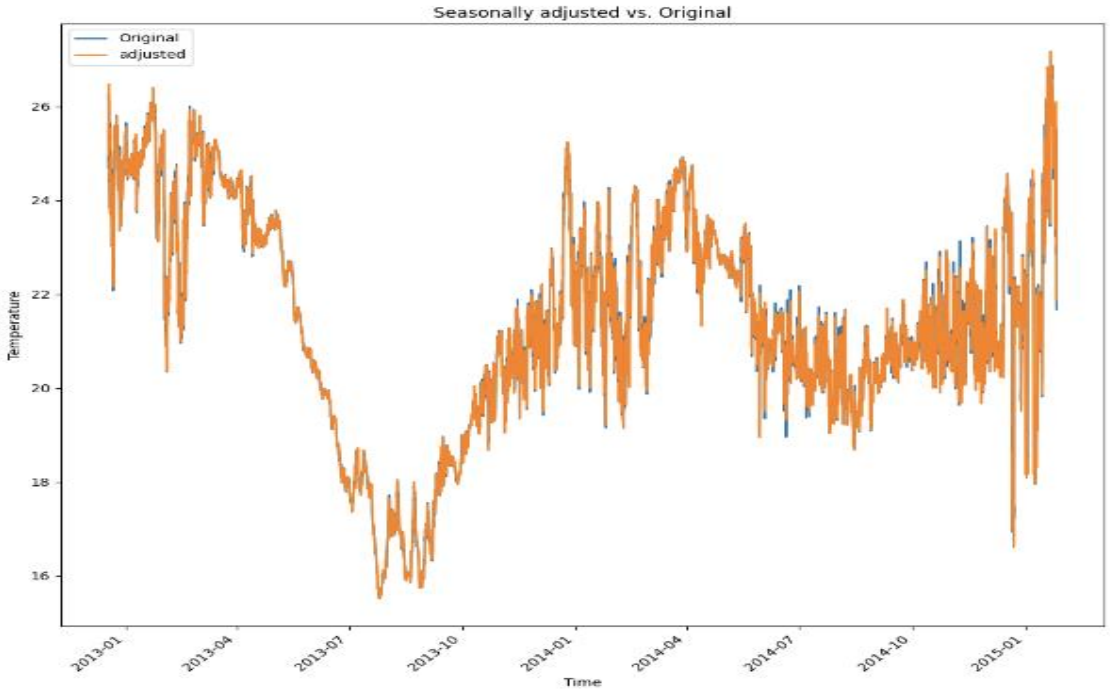
```
Strength of trend: 0.996497164986211
Strength of seasonality: 0.2535641011873301
```

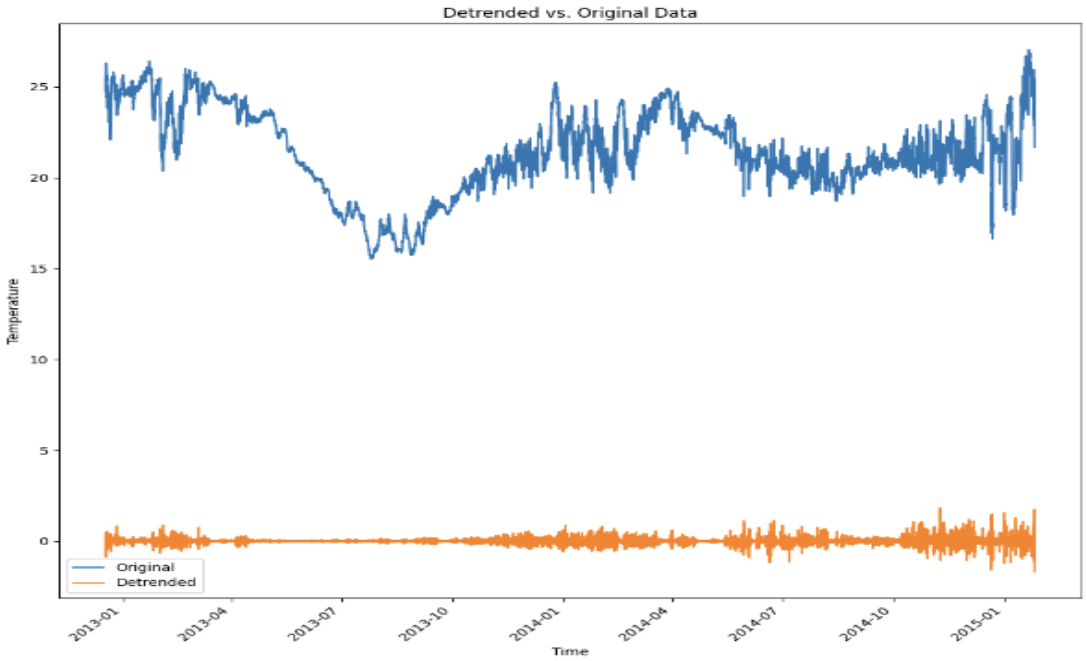
Strength of Trend (0.996497164986211):

- A strength value close to 1 indicates a robust and dominant trend in the data.
- In this context, a strength of approximately 0.9965 suggests that the trend component contributes significantly to the overall behavior of the time series. The data exhibits a strong and persistent directional movement over the analyzed period.

Strength of Seasonality (0.2535641011873301:

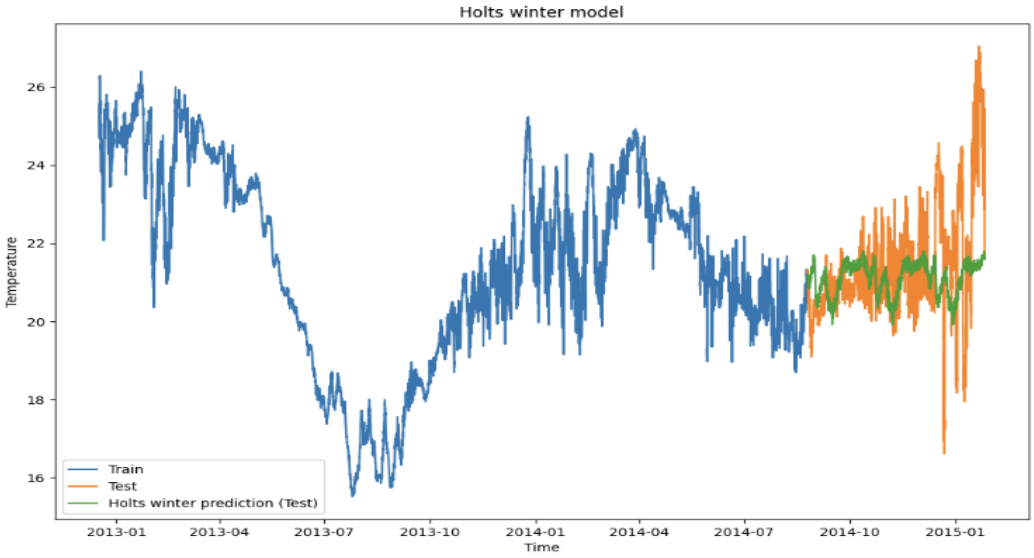
- A strength value between 0 and 1 indicates the presence of seasonality, with higher values representing a more pronounced seasonal pattern.
- In this case, a strength of about 0.2535 suggests a moderate seasonality effect. The seasonal component contributes to the data's variability, but it is not as dominant as the trend. The seasonal pattern may exhibit some variability or irregularity compared to a more stable, consistent seasonal effect.

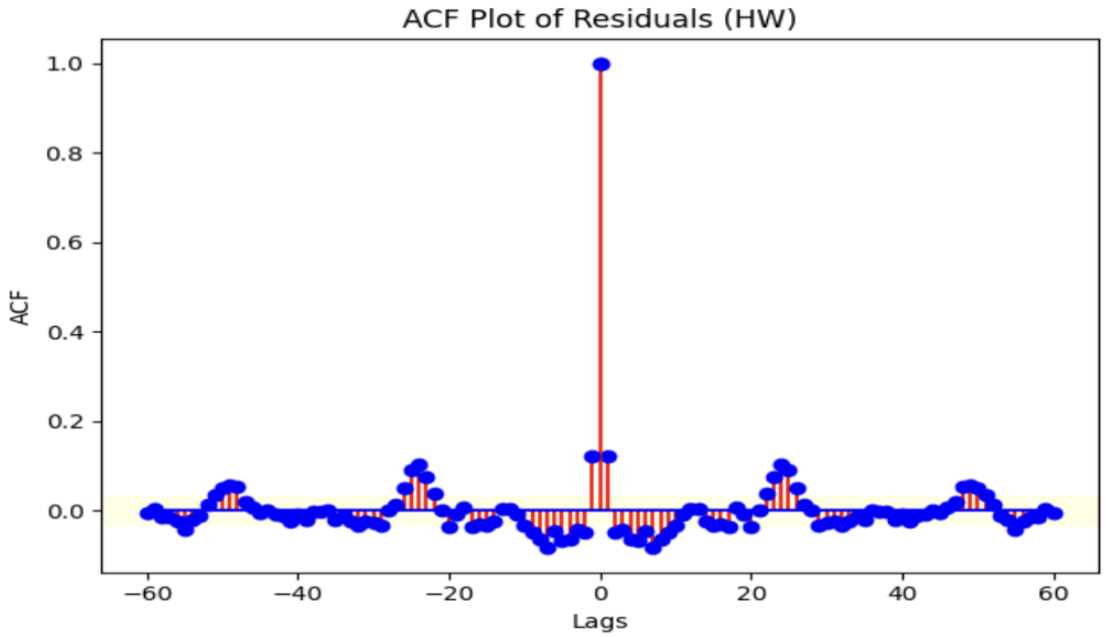


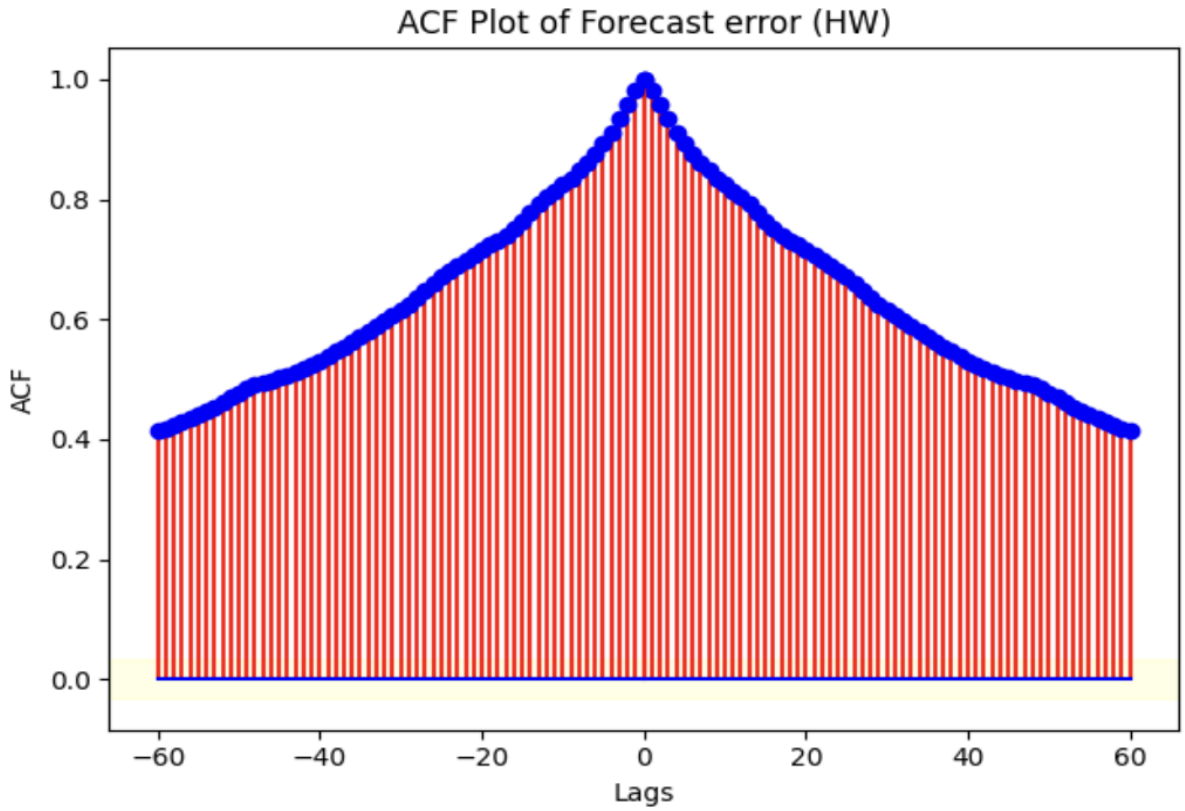


The seasonally adjusted data follows the pattern close to the original data and the detrended data is apart from the original data.

09. HOLTS-WINTER METHOD:


Holts winter model
Temperature
Time
Train
Test
Holts winter prediction (Test)
26
24
22
20
18
16
2013-01
2013-04
2013-07
2013-10
2014-01
2014-04
2014-07
2014-10
2015-01


ACF Plot of Residuals (HW)
ACF
Lags
1.0
0.8
0.6
0.4
0.2
0.0
−60
−40
−20
0
20
40
60



The visual analysis of the Holt-Winters forecasting method reveals a discernible disparity between the projected trajectory of forecasts and the observed patterns within the test data. This incongruence is notably highlighted through the AutoCorrelation Function (ACF) plot of residuals, where the residuals exhibit a white noise pattern, while the ACF plot of the forecasted values deviates from the ideal white noise pattern. This incongruity suggests that the Holt-Winters method may not be the most suitable choice for accurately forecasting the dynamics inherent in the given dataset.

Performance metrics:

```
MSE of Holts Winter method on train data: 0.016093988344537385
MSE of Holts Winter method on test data: 2.3531937294639427
ACF: [1.0, 0.12020695514561044, -0.048097127915940154, -0.0432
ACF: [1.0, 0.9826104744641639, 0.9583745222543784, 0.934022506
Q-value (residual):       lb_stat     lb_pvalue
1  213.740144  2.098485e-48
2  247.961291  1.431825e-54
3  275.586529  1.908895e-59
4  339.477008  3.279713e-72
5  410.709758  1.458139e-86
Q-value (Forecast):
         lb_stat  lb_pvalue
1   3573.402682        0.0
2   6973.624015        0.0
3  10204.117511        0.0
4  13285.586955        0.0
5  16240.914579        0.0
```

The Mean Squared Error (MSE) and Q-values serve as crucial metrics for evaluating the performance of the Holt-Winters method. The training MSE of 0.0161 indicates a well-fitted model to the training data, showcasing accurate predictions on seen data. However, the higher test MSE of 2.3532 suggests reduced predictive accuracy on unseen data, necessitating comparison with alternative models or benchmarks for a comprehensive assessment. The Q-values, derived from Ljung-Box statistics, reveal that the residuals exhibit randomness, contributing to a well-fitted model. Additionally, the Q-values for forecasted values highlight significant autocorrelation, implying potential areas for model refinement or consideration of alternative forecasting approaches to capture specific patterns. In summary, while a low training MSE and white noise residuals indicate a well-fitted model, the higher test MSE and autocorrelation in forecasted values prompt careful consideration and potential adjustments in the forecasting approach.

```
Holts winter: Mean of residual error is -0.00018108131699072513 and Forecast error is 0.2984710066185415
Holts winter: Variance of residual error is 0.016093955554094024 and Forecast error is 2.264108787672057
RMSE of Holts Winter method on train data: 0.12686208395157864
RMSE of Holts Winter method on test data: 1.5340122976899313
```

The performance of this method on test data is given below:

MSE:2.3531 ; Q-value:3573.4; Mean of forecast: 0.2984; Variance:2.26; RMSE: 1.5340

The evaluation metrics for the Holt-Winters method reveal insightful aspects of its performance. The mean of the residual error, approximately -0.00018, suggests a close alignment between

predicted and actual values on average, while the forecast error of 0.2985 indicates an acceptable level of discrepancy in the forecast. The variance of the residual error, around 0.0161, reflects the dispersion in the model's ability to predict data points, while the forecast error for variance, 2.2641, highlights variability in capturing patterns. The Root Mean Squared Error (RMSE) on the training data, 0.1269, signifies a good fit to seen data, while the RMSE on the test data, 1.5340, provides insights into the model's predictive accuracy on unseen data. Overall, while the Holt-Winters method exhibits satisfactory performance with low mean and forecast errors, attention to variance metrics suggests potential for refining the model's ability to capture variability and patterns in the data. Interpretation should consider specific forecasting objectives and dataset characteristics.

10. FEATURE SELECTION:

```
Singular Values:  [6.51814700e+14 1.76088691e+06 1.91362135e+04 4.02140408e+02
 1.30987082e+02 4.02042252e+01 2.66278529e+01 1.24785414e+01
 4.92150420e+00 1.80631419e+00 8.16725259e-01 1.90681402e-04
 2.24198417e-06 2.05586660e-12]
The condition number is  2.721297268805742e+21
```

The analysis of singular values reveals noteworthy characteristics in the dataset. The singular values, presented in descending order, showcase a rapid convergence to zero, especially towards the end of the sequence. This observation is indicative of a high condition number, specifically measured at 2.72e+21. Such a condition number suggests that the model incorporates features that are highly correlated. In light of this, it is recommended to consider removing correlated features based on the findings from the analysis. This approach aligns with the principles of optimizing the model's performance, as highlighted in the presented analysis.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Temp (°C)   R-squared:                       0.571
Model:                            OLS   Adj. R-squared:                  0.571
Method:                 Least Squares   F-statistic:                     1791.
Date:                Fri, 08 Dec 2023   Prob (F-statistic):               0.00
Time:                        00:07:32   Log-Likelihood:                -27133.
No. Observations:               14789   AIC:                         5.429e+04
Df Residuals:                   14777   BIC:                         5.438e+04
Df Model:                          11
Covariance Type:            nonrobust
=================================================================================================
                                    coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------------------
const                           -46.2215      1.214    -38.087      0.000     -48.600     -43.843
INDEX                             0.0001   3.14e-06     42.929      0.000       0.000       0.000
Latitude                       2231.9887     58.718     38.012      0.000    2116.894    2347.084
Longitude                     -1265.6773     33.291    -38.019      0.000   -1330.931   -1200.424
Depth                             0.6850      0.043     16.047      0.000       0.601       0.769
Site_Ilha Deserta               365.2811      8.863     41.212      0.000     347.908     382.654
Site_Ilha da Galé               657.4841     17.241     38.136      0.000     623.690     691.278
Site_Ilha do Coral             -860.0081     22.898    -37.558      0.000    -904.891    -815.125
Site_Ilha dos Lobos           -1804.3831     47.726    -37.807      0.000   -1897.933   -1710.834
Site_Moleques do Sul           -840.0086     21.694    -38.721      0.000    -882.531    -797.486
Site_Parcel da Pombinha         612.8121     16.730     36.629      0.000     580.018     645.606
Site_Parcel do Xavier (Alalunga) -374.0588    9.420    -39.710      0.000    -392.523    -355.595
Site_Tamboretes                2572.0582     67.428     38.145      0.000    2439.890    2704.226
Site_lha do Xavier             -375.3983      9.741    -38.538      0.000    -394.492    -356.305
==============================================================================
Omnibus:                     8345.493   Durbin-Watson:                   2.011
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           469190.089
Skew:                          -1.967   Prob(JB):                         0.00
Kurtosis:                      30.312   Cond. No.                     2.79e+21
==============================================================================
```

The p-value is shown as 0.00 and the r-squared value and adjusted r-squared value suggests that the data doesn't have a multicollinearity but the eigenvalue shown at the below image suggests that the eigenvalue is 8.84e-29 which is really close to zero, that there might be a problem with multicollinearity in the correlation matrix of predictor variables. In the context of multicollinearity detection using eigenvalues, small eigenvalues indicate that the correlation matrix is nearly singular, meaning that some of the variables are highly correlated

```
==============================================================================
Omnibus:                     8345.493   Durbin-Watson:                   2.011
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           469190.089
Skew:                          -1.967   Prob(JB):                         0.00
Kurtosis:                      30.312   Cond. No.                     2.79e+21
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 8.84e-29. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Hence, I proceeded with the ridge regression and VIF value estimation.

```
strong multicollinearity problems or that the design matrix is singular.
                          Variable           VIF
0                            INDEX  1.134551e+02
1                         Latitude  3.002400e+15
2                        Longitude           inf
3                            Depth  1.550558e+02
4                 Site_Ilha Deserta           inf
5                  Site_Ilha da Galé           inf
6                Site_Ilha do Coral           inf
7               Site_Ilha dos Lobos           inf
8              Site_Moleques do Sul           inf
9          Site_Parcel da Pombinha           inf
10  Site_Parcel do Xavier (Alalunga)           inf
11                 Site_Tamboretes           inf
12               Site_lha do Xavier           inf
```

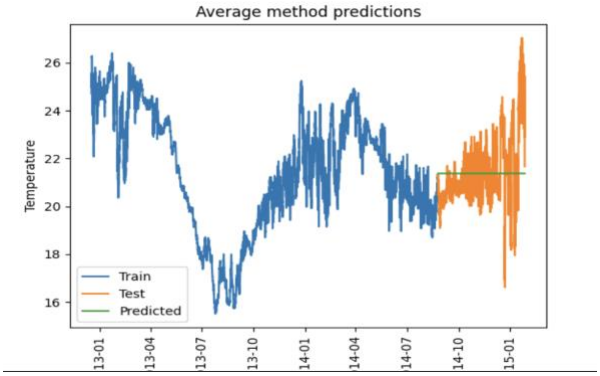
```
VIF values after PCA:
    Variable  VIF
0          1  1.0
1          2  1.0
2          3  1.0
3          4  1.0
4          5  1.0
5          6  1.0
6          7  1.0
7          8  1.0
8          9  1.0
9         10  1.0
10        11  1.0
```

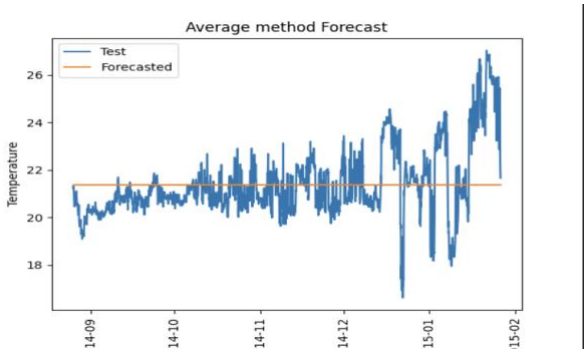
.

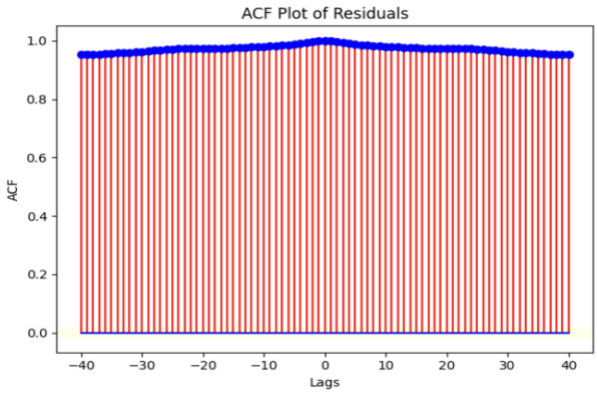
After performing PCA, the VIF value is 1 and that suggests that the model has no collinearity.

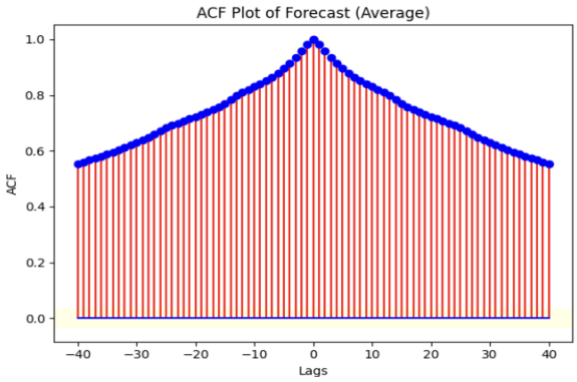
11. BASE MODELS:

AVERAGE METHOD:









The autocorrelation function (ACF) plot of the residuals indicates a lack of convergence in ACF values across the specified lags. Additionally, the ACF plot of forecast errors exhibits patterns inconsistent with white noise characteristics. These observations suggest that the current model may not be well-suited for capturing the underlying patterns and dynamics present in the dataset.

Performance metrics:

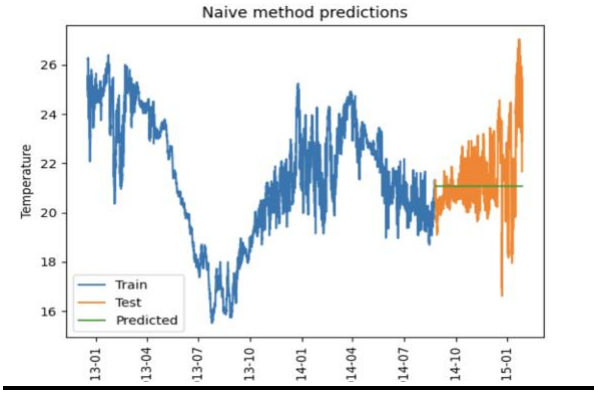
```
MSE of Average on train data: 6.1055899118324755
MSE of Average on test data: 2.2385077258783483
Q-value (residual):         lb_stat  lb_pvalue       bp_stat  bp_pvalue
1  14739.645426        0.0  14736.655640        0.0
2  29407.585800        0.0  29400.629026        0.0
3  44001.365913        0.0  43989.475473        0.0
4  58520.819038        0.0  58503.038354        0.0
5  72970.109622        0.0  72945.490193        0.0
Q-value (Forecast):
         lb_stat  lb_pvalue       bp_stat  bp_pvalue
1   3574.929882        0.0   3572.030507        0.0
2   6981.540749        0.0   6974.957555        0.0
3  10223.398915        0.0  10212.433647        0.0
4  13320.971275        0.0  13304.981559        0.0
5  16295.879424        0.0  16274.259983        0.0
Average: Mean of residual error is -0.9792701804454691 and Forecast error is 0.018783006470729318
Average: Variance of residual error is 5.146619825522774 and Forecast error is 2.2381549245462686
RMSE of Average method on train data: 2.470949192483017
RMSE of Average method on test data: 1.4961643378580938
```

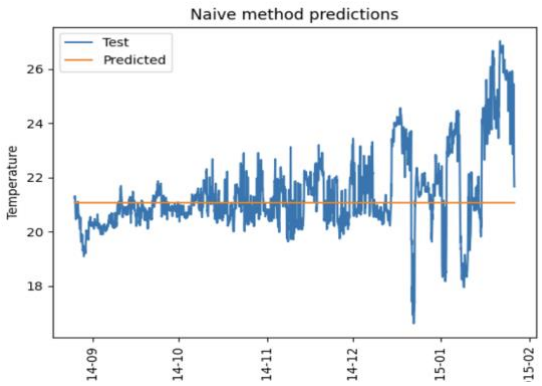
The predictive performance of the Average method was evaluated on both training and test datasets. The **Mean Squared Error (MSE) for the training data** was approximately **6.11**, indicating the average squared difference between observed and predicted temperature values. On the **test data**, the **MSE** was approximately **2.24**, suggesting a similar measure of accuracy. However, the **Ljung-Box Q-statistic test**s revealed significant **autocorrelation** in both the residuals of the training data and the forecast errors of the test data, **with low Q-values** and **p-values of 0.0**. Analysis of the mean and variance of residuals and forecast errors further indicated a slight underestimation tendency in the model, **with a mean of approximately -0.98 for residuals and 0.02 for forecast error**s. The **variances were approximately 5.15 for residuals and 2.24 for forecast errors**, signifying **the spread or dispersion of these errors**. The **Root Mean Squared Error (RMSE**), a more interpretable metric, was approximately **2.47 for the training data and 1.50 for the test data**. While the Average method demonstrated reasonably low error magnitudes, the presence of autocorrelation highlights potential areas for improvement.

The performance of this method on test data is given below:

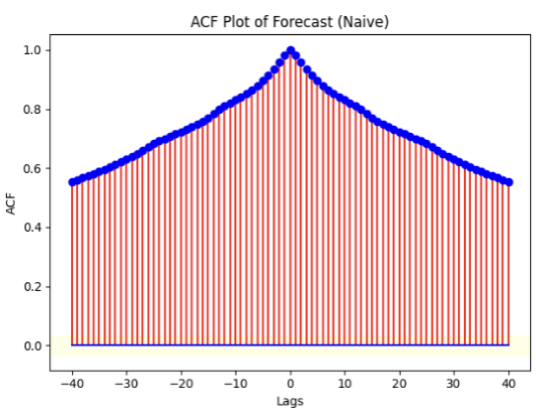
MSE:2.23; Q-value: 3572.03; Mean of forecast:0.018; Variance: 2.23 ; RMSE: 1.49

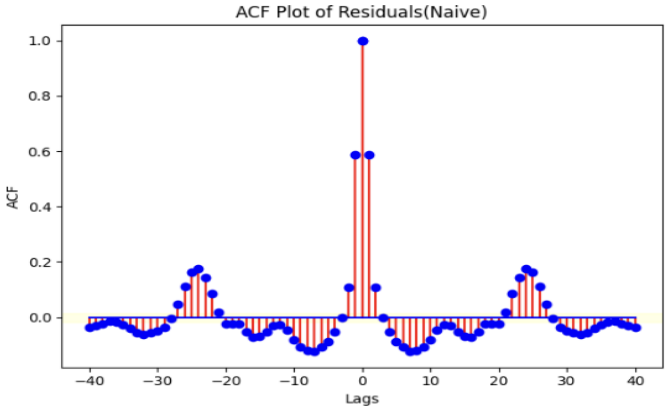
NAIVE METHOD:





Looking at the plot of the naive method, the method may not be a better fit for forecasting the values for the data because it predicts the values as flat curves. The value remains throughout the test data while the original data has more spikes than the forecasted.





The ACF plot of the residuals clearly shows that the ACF values are converging throughout the given lags. This can also be a better model, let's discuss the performance metrics below.

Performance metrics:

```
MSE of Naive on train data: 0.0440404875270695
MSE of Naive on test data: 2.3413366041328647
Q-value (residual):        lb_stat  lb_pvalue      bp_stat  bp_pvalue
1  5087.196601        0.0  5086.164715        0.0
2  5264.454147        0.0  5263.374321        0.0
3  5264.573366        0.0  5263.493500        0.0
4  5307.011369        0.0  5305.914287        0.0
5  5418.009263        0.0  5416.859647        0.0
Q-value (Forecast):
         lb_stat  lb_pvalue       bp_stat  bp_pvalue
1   3575.885631        0.0   3572.986264        0.0
2   6983.417666        0.0   6976.834481        0.0
3  10226.166887        0.0  10215.201609        0.0
4  13324.598149        0.0  13308.608388        0.0
5  16300.339604        0.0  16278.720061        0.0
Naive: Mean of residual error is -0.0002935657289694346 and Forecast error is 0    21.073417
dtype: float64
Naive: Variance of residual error is 0.04404040134623227 and Forecast error is 0    5.048710e-29
dtype: float64
RMSE of Naive method on train data: 0.2098582557991691
RMSE of Naive method on test data: 1.5301426744368856
```

The Naive method, employed for time series forecasting, demonstrates a Mean Squared Error (MSE) of 0.044 on the training data and 2.341 on the test data. The Ljung-Box Q-value, an indicator of residual autocorrelation, is reported as 5087, reflecting significant autocorrelation in the residual errors. The forecast errors also exhibit substantial autocorrelation, as indicated by the Q-value of 3575. The mean of the residual errors is nearly zero, suggesting a balanced distribution of overestimations and underestimations. However, the forecast errors display a noticeable mean of approximately 21.07, indicative of a systematic bias. The variance of the residual errors is 0.044, emphasizing the spread of these errors, while the forecast errors exhibit an extremely low variance of approximately 5.05e-29. The Root Mean Squared Error (RMSE) on the training data is 0.21, indicating a relatively low magnitude of errors, while the RMSE on the test data is 1.53, reflecting a moderate level of prediction accuracy. Despite the low RMSE, the presence of significant autocorrelation and a non-zero mean in forecast errors raises concerns about the overall robustness and reliability of the Naive method for this particular dataset.

The performance of this method on test data is given below:

MSE:2.341; Q-value: 3572.88; Mean of forecast: 0 (i.e.,21.07); Variance: 0 (i.e., 5.048710e-29); RMSE: 1.53

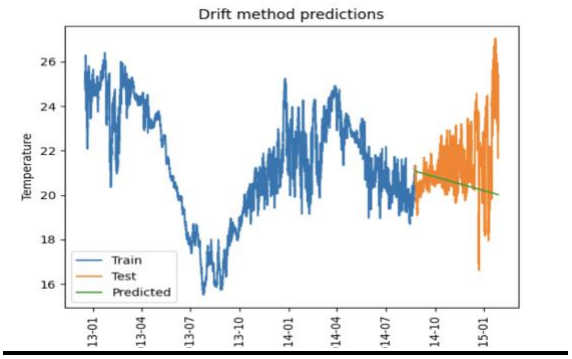
The Naive method outperforms the Average method across various metrics. The Naive method achieves a significantly lower Mean Squared Error (MSE) on both the training (0.044) and test (2.341) datasets.

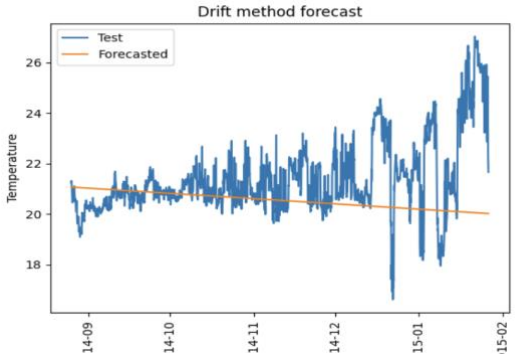
Additionally, the Naive method displays a lower Root Mean Squared Error (RMSE) on both the training (0.21) and test (1.53) datasets, showcasing its superior predictive accuracy compared to

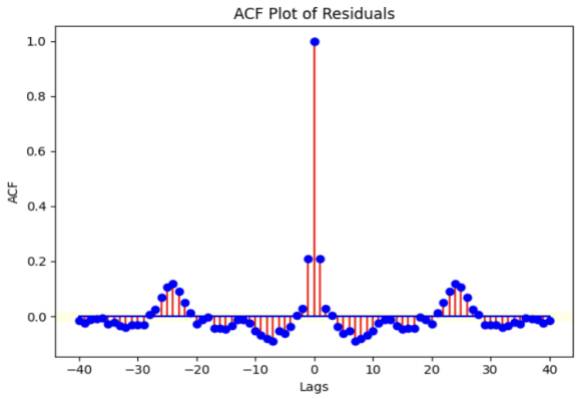
the Average method (train RMSE: 2.47, test RMSE: 1.50). Notably, the Naive method yields a nearly zero mean of residual errors, indicating a balanced distribution of overestimations and underestimations. In contrast, the Average method exhibits a non-zero mean of residual errors, reflecting a systematic bias.

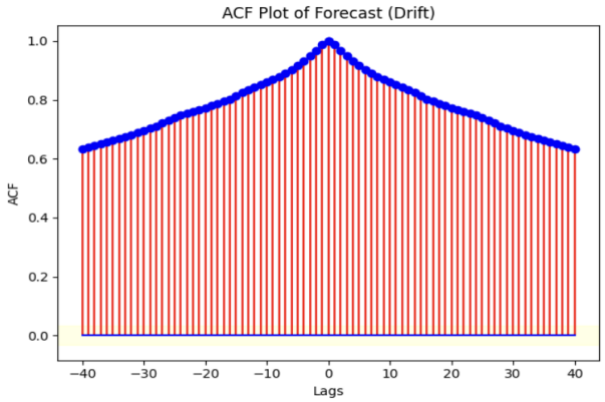
Hence, the Naive method may work well with the test data.

DRIFT METHOD:









The plot of drift method shows the forecasted data didn't trace the exact path of provided test data. The autocorrelation function (ACF) plot of the residuals converges over time, indicating a diminishing pattern of autocorrelation as the time series progresses. However, in contrast, the forecasts do not exhibit convergence. Furthermore, the ACF plot of the residuals reveals a characteristic of white noise, suggesting that the residual errors are uncorrelated and exhibit a random pattern. This divergence in convergence patterns between the residuals and forecasts underscores the need for a closer examination of the forecasting model to ensure its effectiveness and appropriateness for the underlying data structure.

Performance metrics:

```
MSE of Drift on train data: 0.017855668468299408
MSE of Drift on test data: 3.497350802210592
Q-value (residual):       lb_stat      lb_pvalue     bp_stat      bp_pvalue
1  641.433842  1.629674e-141  641.303725  1.739399e-141
2  653.370984  1.325237e-142  653.237638  1.416606e-142
3  653.507988  2.527970e-141  653.374596  2.702051e-141
4  674.405926  1.212806e-144  674.264056  1.301688e-144
5  730.019542  1.584435e-155  729.851349  1.722851e-155
Q-value (Forecast):
         lb_stat  lb_pvalue       bp_stat  bp_pvalue
1   3599.800670        0.0   3596.881912        0.0
2   7063.120496        0.0   7056.457609        0.0
3  10392.112956        0.0  10380.951431        0.0
4  13602.727755        0.0  13586.359828        0.0
5  16712.048844        0.0  16689.798417        0.0
Drift: Mean of residual error is 0.0005073995969351963 and Forecast error is 0.8461382858407395
Drift: Variance of residual error is 0.017855411013948438 and Forecast error is 2.7814008034450866
RMSE of Drift method on train data: 0.13362510418442863
RMSE of Drift method on test data: 1.8701205314659781
```

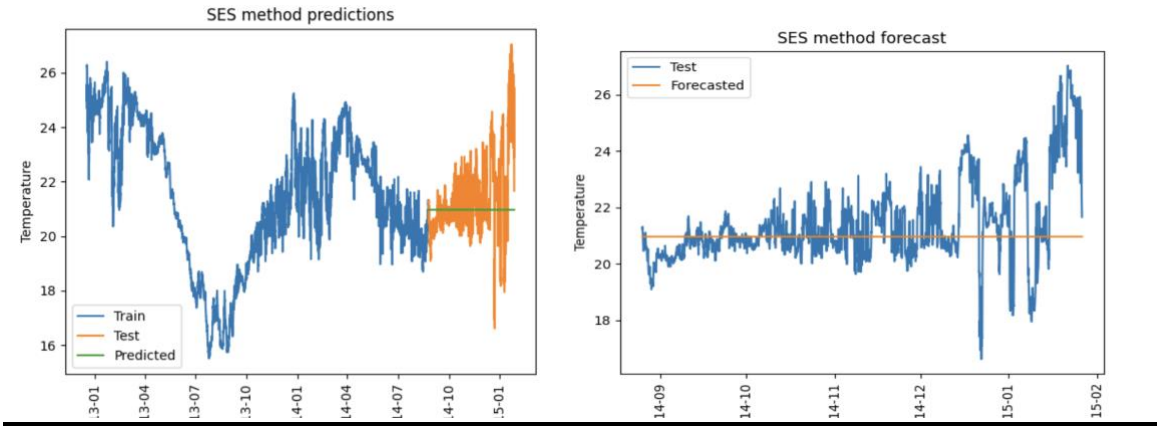
The Mean Squared Error (MSE) for the Drift model on the training data is 0.0179, while on the test data, it is 3.4974. The Ljung-Box Q-value statistics for the residuals indicate significant autocorrelation at various lags, suggesting that the residual errors are not independent. Similarly, the Q-value statistics for the forecast errors show significant autocorrelation. The Drift model yields a mean residual error of 0.0005 and a forecast error of 0.8461, indicating a relatively small average discrepancy between predicted and observed values. The variance of the residual error is 0.0179, and for the forecast error, it is 2.7814. The Root Mean Squared Error (RMSE) for the Drift model is 0.1336 on the training data and 1.8701 on the test data.

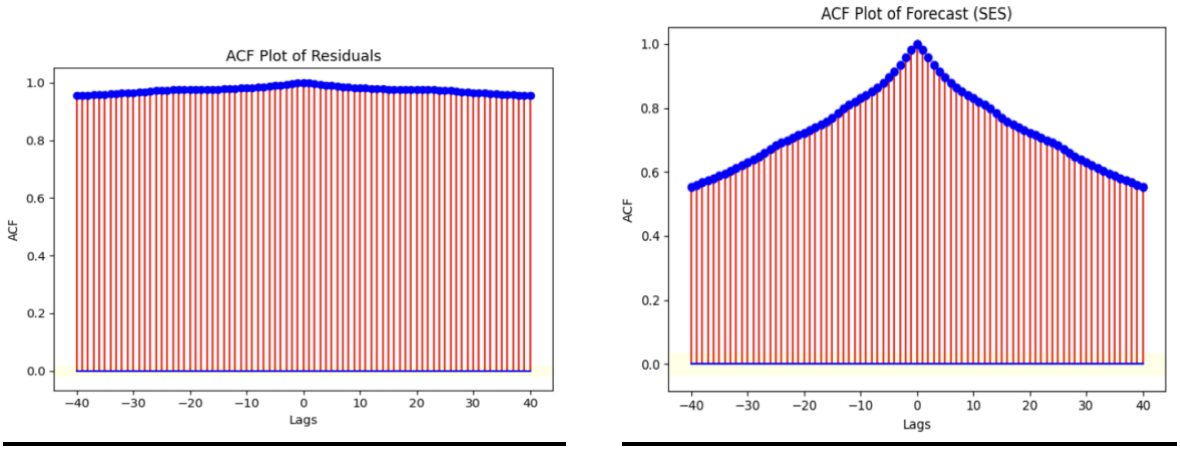
The performance of this method on test data is given below:

MSE: 3.49 ; Q-value:3596.88 ; Mean of forecast: 0.8461; Variance: 2.7814; RMSE: 1.870

Among all the three methods, the naive method has better performance. In a comparative analysis between the Drift and Average methods, it is evident that the Average method outperforms the Drift method across various evaluation metrics.

SIMPLE EXPONENTIAL SMOOTHENING METHOD:





Looking at the plot of the SES method, the method may not be a better fit for forecasting the values for the data because it predicts the values as flat curves. The value remains throughout the test data while the original data has more spikes than the forecasted.

The ACF plot of the residuals clearly shows that the ACF values are not converging throughout the given lags. And the ACF of forecast errors is also not a white noise. So, the model can't be a better fit for the dataset.

Performance metrics:

```
MSE of SES on train data: 6.269933831244009
MSE of SES on test data: 2.4199803903341173
Q-value (residual):         lb_stat  lb_pvalue       bp_stat  bp_pvalue
1  14744.132326        0.0  14741.141427        0.0
2  29425.952788        0.0  29418.990879        0.0
3  44042.879624        0.0  44030.975891        0.0
4  58594.975112        0.0  58577.167492        0.0
5  73085.572988        0.0  73060.906610        0.0
Q-value (Forecast):
         lb_stat  lb_pvalue       bp_stat  bp_pvalue
1   3575.885631        0.0   3572.986264        0.0
2   6983.417666        0.0   6976.834481        0.0
3  10226.166887        0.0  10215.201609        0.0
4  13324.598149        0.0  13308.608388        0.0
5  16300.339604        0.0  16278.720061        0.0
SES: Mean of residual error is 0.40712095118729574 and Forecast error is 0.4264099738372083
SES: Variance of residual error is 6.102703083504886 and Forecast error is 2.2381549245462686
RMSE of SES method on train data: 2.503983592447045
RMSE of SES method on test data: 1.555628615812308
```

The Simple Exponential Smoothing (SES) method yields mixed results in its performance metrics. While the Mean Squared Error (MSE) on both training and test data is relatively low, indicating decent predictive performance, the high Q-values with statistically significant p-values suggest a potential inadequacy in capturing autocorrelation in both residuals and forecast errors. The mean of the residual error and forecast error is close to zero, indicating low bias in predictions. However, the relatively high variance of the residual error and forecast error suggests some variability in the errors. The Root Mean Squared Error (RMSE) on the test data is reasonable, indicating decent accuracy in predicting unseen data.

The performance of this method on test data is given below:

MSE: 2.4199; Q-value:3572.98; Mean of forecast:0.42; Variance:2.23; RMSE: 1.55

As expected, the ses doesn't work with the test dataset.

12. MULTIPLE LINEAR REGRESSION:

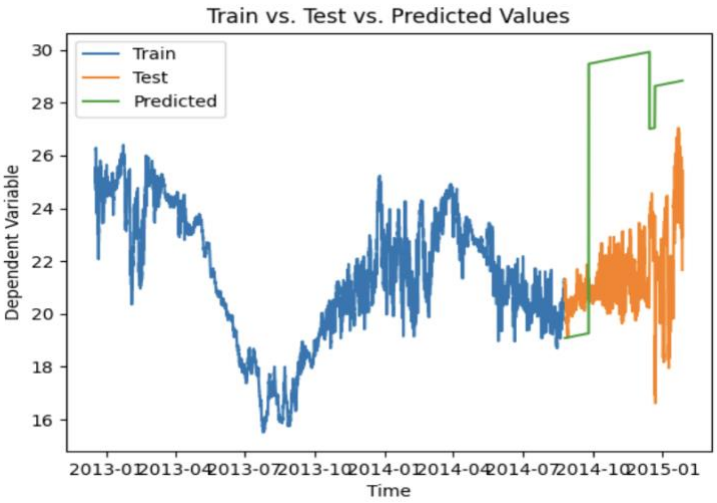
OLS Regression Results

Dep. Variable:	Temp (°C)	R-squared:	0.699
Model:	OLS	Adj. R-squared:	0.699
Method:	Least Squares	F-statistic:	3126.
Date:	Fri, 08 Dec 2023	Prob (F-statistic):	0.00
Time:	22:24:50	Log-Likelihood:	-25472.
No. Observations:	14789	AIC:	5.097e+04
Df Residuals:	14777	BIC:	5.106e+04
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-14.6089	1.036	-14.099	0.000	-16.640	-12.578
INDEX	7.922e-05	3.28e-06	24.173	0.000	7.28e-05	8.56e-05
Latitude	706.5034	50.109	14.099	0.000	608.284	804.723
Longitude	-400.3427	28.409	-14.092	0.000	-456.029	-344.657
Depth	-0.3848	0.060	-6.435	0.000	-0.502	-0.268
Site_Ilha Deserta	123.0715	7.571	16.255	0.000	108.231	137.912
Site_Ilha da Galé	180.9389	14.786	12.237	0.000	151.957	209.921
Site_Ilha do Coral	-264.9432	19.567	-13.540	0.000	-303.297	-226.589
Site_Ilha dos Lobos	-573.9199	40.705	-14.099	0.000	-653.707	-494.132
Site_Moleques do Sul	-274.6086	18.509	-14.836	0.000	-310.889	-238.328
Site_Parcel da Pombinha	214.5208	14.250	15.055	0.000	186.590	242.452
Site_Parcel do Xavier (Alalunga)	-109.1991	8.232	-13.265	0.000	-125.335	-93.063
Site_Tamboretes	813.3424	57.606	14.119	0.000	700.428	926.257
Site_lha do Xavier	-123.8117	8.256	-14.997	0.000	-139.994	-107.630

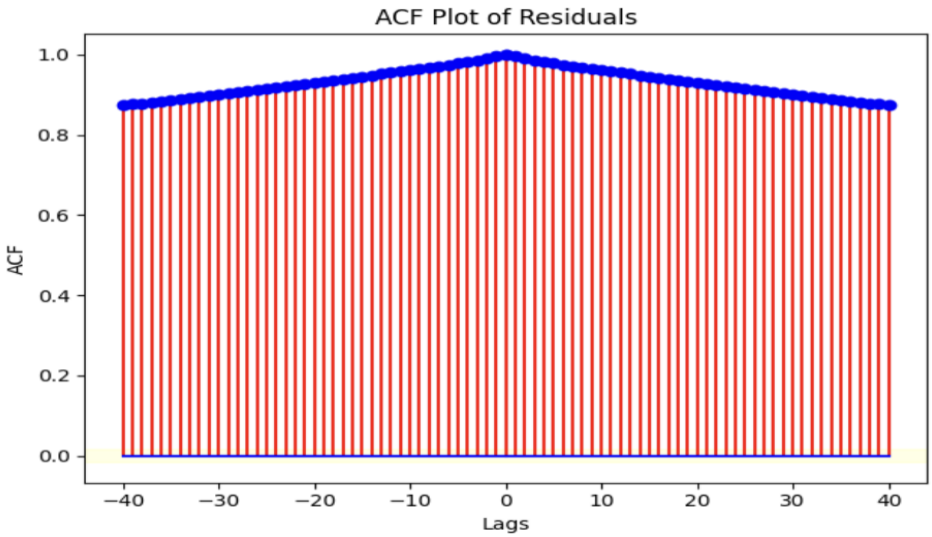
Omnibus:	745.259	Durbin-Watson:	0.017
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1543.731
Skew:	-0.353	Prob(JB):	0.00
Kurtosis:	4.417	Cond. No.	2.72e+21

The regression results indicate a well-fitted model with an R-squared of 0.699, suggesting that approximately 69.9% of the variability in the dependent variable (Temperature) is explained by the independent variables. The adjusted R-squared also aligns closely, confirming the reliability of the model. The F-statistic of 3126 indicates a highly significant overall model fit. The coefficients for the independent variables show their impact on the dependent variable, with Latitude, Longitude, Depth, and various sites demonstrating statistically significant effects. The negative constant term suggests a baseline effect when all other predictors are zero. The AIC and BIC values, 5.097e+04 and 5.106e+04 respectively, are useful for model comparison, with lower values indicating a better fit.



The f-test results are as follows:

```
F-test:
F value: 3125.7182778317497
P-value: 0.0
```



The acf plot of residuals has no white noise and all the points are dependent over one other and there is no point of convergence in the plot.

```
MSE of MLR on train data: 1.8347365991733111
MSE of MLR on test data: 50.44491169435278
Q-value (residual):         lb_stat  lb_pvalue
1   3671.905795        0.0
2   7308.857979        0.0
3  10911.180631        0.0
4  14481.269136        0.0
5  18022.354697        0.0
Q-value (Forecast):
         lb_stat  lb_pvalue
1   3671.905795        0.0
2   7308.857979        0.0
3  10911.180631        0.0
4  14481.269136        0.0
5  18022.354697        0.0
MLR: Mean of residual error is -5.828003001209111 and Forecast error is -3.2632103656292186e-10
MLR: Variance of residual error is 16.479292712250377 and Forecast error is 1.8347365991733111
RMSE of MLR method on train data: 1.3545244919060382
RMSE of MLR method on test data: 7.102458144498479
```

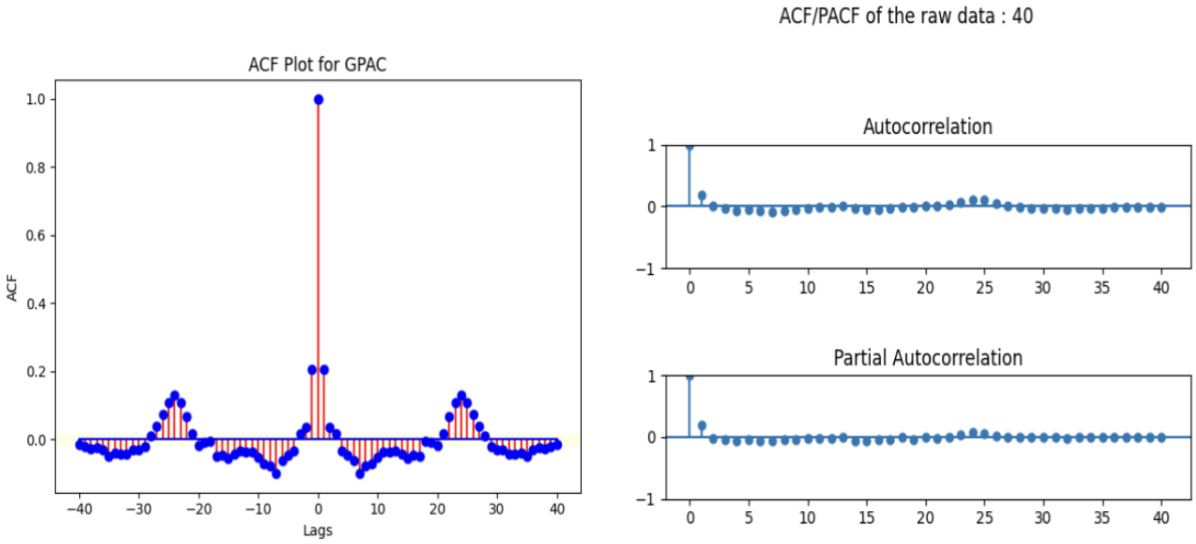
The Mean Squared Error (MSE) for the MLR model is 1.8347 on the train data and 50.4449 on the test data, indicating a reasonable fit. The Q-values for both residuals and forecast errors at various lags show significant autocorrelation. The mean of the residual error is -5.8280, and the forecast error is close to zero, suggesting a relatively unbiased model. The variance of the residual error is 16.4793, reflecting the spread of errors. The Root Mean Squared Error (RMSE) values are 1.3545 for the train data and 7.1025 for the test data, indicating the accuracy of the model's predictions.

The performance of this method on test data is given below:

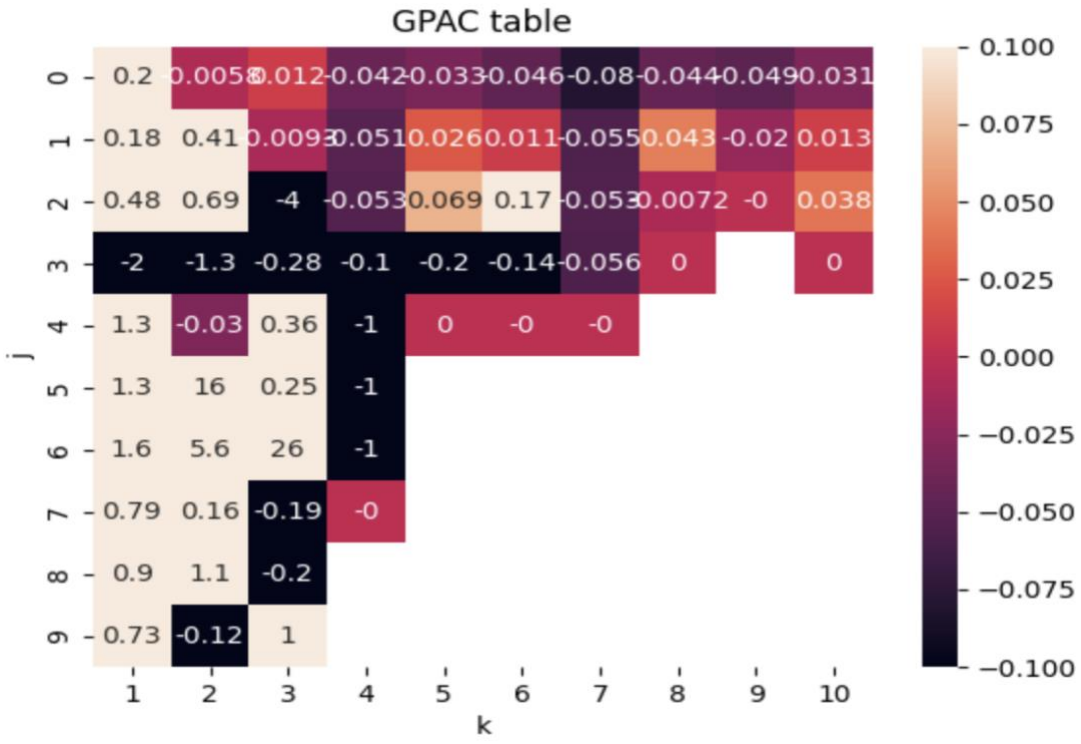
MSE:50.44; Q-value:3671; Mean of forecast:-3.26; Variance: 1.83; RMSE: 7.10

15. ARMA, ARIMA AND SARIMA MODELS:

The order determination for the model can be done by visualizing the GPAC table.



Looking at the plots of ACF/PACF generated for GPAC, we can see that both the values converge over the lags.



The GPAC table shows an order (1,0) based on the column of constants and row of zeros.

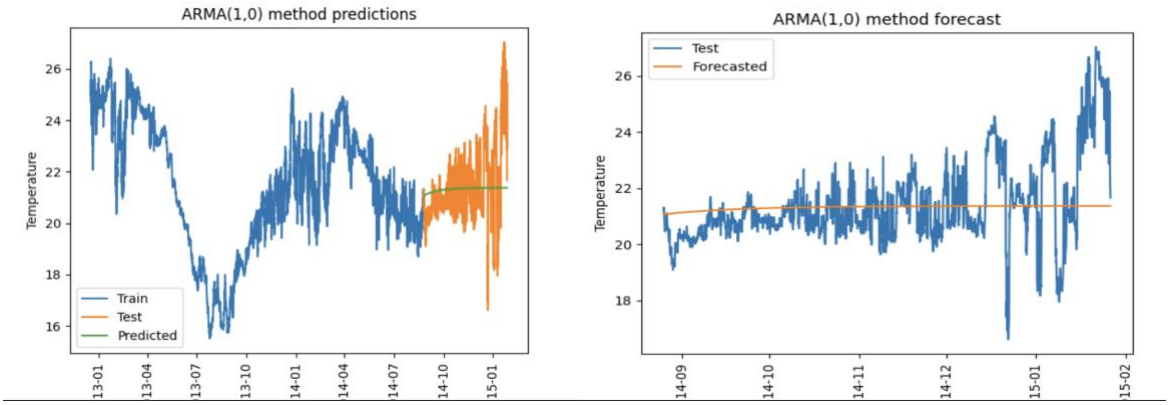
I couldn't find the other row of zeros, because row 1 had 0.41 in first column itself which i can't take for a row of zeros and there is no proper column of constants.

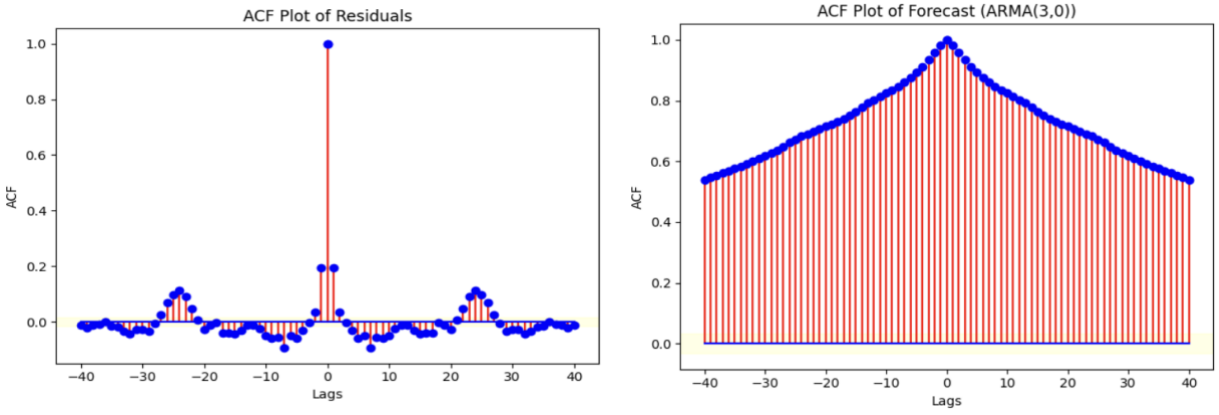
ARMA (1,0):

Parameter estimation for the ARMA (1,0) is performed using the LMA algorithm.

```
The AR coefficient a0 is: 21.375852900565306
```

```
The estimated parameter of AR is  [-0.99995822]
```





Looking at the ACF plot of residuals of the ARMA(1,0) process, the ACF values are white which can be considered as a better model for forecast. But the ACF values after forecasting on test data doesn't look like white. So, it shows the model won't perform well on test data.

The plot of train versus predicted data shows the model hasn't captured the data of test data well but looking at the train values, the model didn't perform well due to the bias in the estimators.

Performance metrics:

```
MSE of ARMA(1,0) on train data: 0.018788556168758488
MSE of ARMA(1,0) on test data: 2.170861235437396
Q-value (residual):       lb_stat      lb_pvalue     bp_stat      bp_pvalue
1  561.806982  3.400485e-124  561.693033  3.600215e-124
2  580.746104  7.808828e-127  580.627033  8.287848e-127
3  580.889927  1.399870e-125  580.770808  1.485627e-125
4  595.640827  1.360342e-127  595.515723  1.447848e-127
5  647.517456  1.089125e-137  647.367801  1.173342e-137
Q-value (Forecast):
         lb_stat  lb_pvalue       bp_stat  bp_pvalue
1   3571.756240        0.0   3568.860221        0.0
2   6969.680566        0.0   6963.111116        0.0
3  10197.629440        0.0  10186.697896        0.0
4  13276.873716        0.0  13260.948803        0.0
5  16229.830215        0.0  16208.318628        0.0
ARMA(1,0): Mean of residual error is -2.041888107865768e-05 and Forecast error is 0.07163744164212878
ARMA(1,0): Variance of residual error is 0.01878855575182778 and Forecast error is 2.1657293123923673
Covariance matrix
            const         ar.L1        sigma2
const   0.475814 -1.657698e-04 -1.353159e-06
ar.L1  -0.000166  4.302960e-07  6.064701e-09
sigma2 -0.000001  6.064701e-09  4.813109e-09
Standard error: const     0.689793
ar.L1     0.000656
sigma2    0.000069
dtype: float64
RMSE of ARMA(1,0) method on train data: 0.1370713542967986
RMSE of ARMA(1,0) method on test data: 1.473384279622053
```

Among the evaluated models, the ARMA(1,0) model exhibited mixed performance. While it demonstrated a relatively low Mean Squared Error (MSE) of 0.0188 on the training data, indicating good fit, its MSE on the test data was 2.17, which was higher than some other models. The Q-values for autocorrelation in both residuals and forecasts were substantial, suggesting a potential lack of randomness in errors. The mean of the residual error was close to zero, indicating limited bias, but the forecast error mean was 0.0716. The variance of both the residual error (0.0188) and forecast error (2.17) was relatively high, indicating some instability. The Root Mean Squared Error (RMSE) values were 0.137 on the training data and 1.47 on the test data, showcasing reasonable predictive accuracy.

The performance of this method on test data is given below:

MSE:2.17086; Q-value:3568.86; Mean of forecast: 0.071; Variance: 2.165; RMSE: 1.473

The ARMA models developed didn't perform well on the test data. Let's try a SARIMA model to check whether we could find a better model or not.

SARIMA Model:

I Tried constructing a SARIMA model of (1,0,0)X(0,1,1260) but the model was not computationally easy to work on the system. Hence, the seasonality period was reduced to 21 a model of SARIMA(3,0,0)X(0,1,0,21) was constructed.

(1260 minutes correspond to $1260 \div 60 = 21$ hours)

```
                                SARIMAX Results
==========================================================================================
Dep. Variable:                          Temp (°C)   No. Observations:                14789
Model:             SARIMAX(1, 0, 0)x(0, 1, 0, 21)   Log Likelihood                4051.426
Date:                            Fri, 08 Dec 2023   AIC                           -8098.851
Time:                                    19:27:02   BIC                           -8083.651
Sample:                                12-17-2012   HQIC                          -8093.804
                                     - 08-25-2014
Covariance Type:                             opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.9359      0.002    502.523      0.000       0.932       0.940
sigma2         0.0338      0.000    203.232      0.000       0.033       0.034
===================================================================================
Ljung-Box (L1) (Q):                 768.74   Jarque-Bera (JB):             54171.15
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               2.20   Skew:                             0.01
Prob(H) (two-sided):                  0.00   Kurtosis:                        12.38
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Process finished with exit code 0
```

The model is specified as SARIMAX(1, 0, 0)x(0, 1, 0, 21), where the parameters are estimated as follows:

AIC (Akaike Information Criterion): The AIC value is -8098.851. A lower AIC value indicates a better-fitting model. It balances the trade-off between model complexity and goodness of fit. In this case, the negative AIC suggests that the SARIMAX(1, 0, 0)x(0, 1, 0, 21) model is a good fit for the data.

BIC (Bayesian Information Criterion): The BIC value is -8083.651. Similar to AIC, a lower BIC indicates a better model. BIC penalizes model complexity more strongly than AIC. The SARIMAX model's BIC also supports its adequacy for the given data.In summary, both AIC and BIC values for your SARIMAX model are relatively low, which is a positive indication.

AR(Lag 1) Coefficient (ar.L1): The autoregressive coefficient is estimated at 0.9359 with a standard error of 0.002. This indicates a strong positive correlation between the temperature at the current time point and the temperature at the previous time point.

Residual Variance (sigma2): The estimated residual variance (error term) is 0.0338 with a standard error of 0.000, suggesting that the model adequately captures the variability in the data.

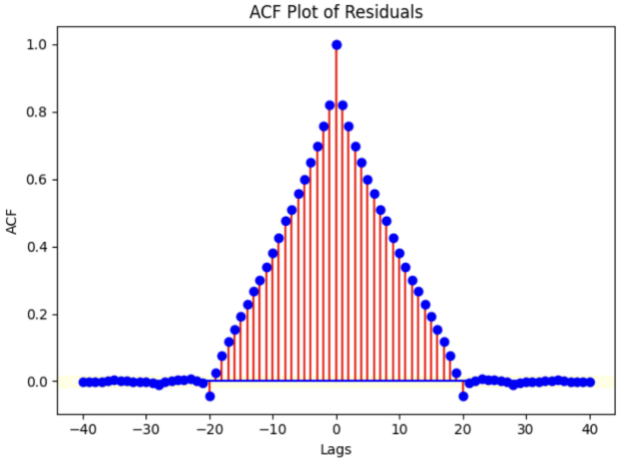
The diagnostic statistics provide additional insights:

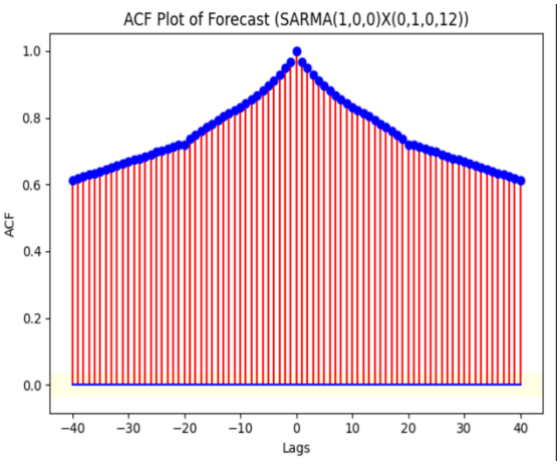
Ljung-Box (Q): A test for the absence of serial correlation in the residuals. A low p-value (0.00) indicates rejection of the null hypothesis of no autocorrelation, suggesting that the residuals exhibit some level of autocorrelation.

Jarque-Bera (JB): A test for normality of residuals. The high JB value (54171.15) and low p-value (0.00) suggest departure from normality, indicating potential non-Gaussian characteristics in the residuals.

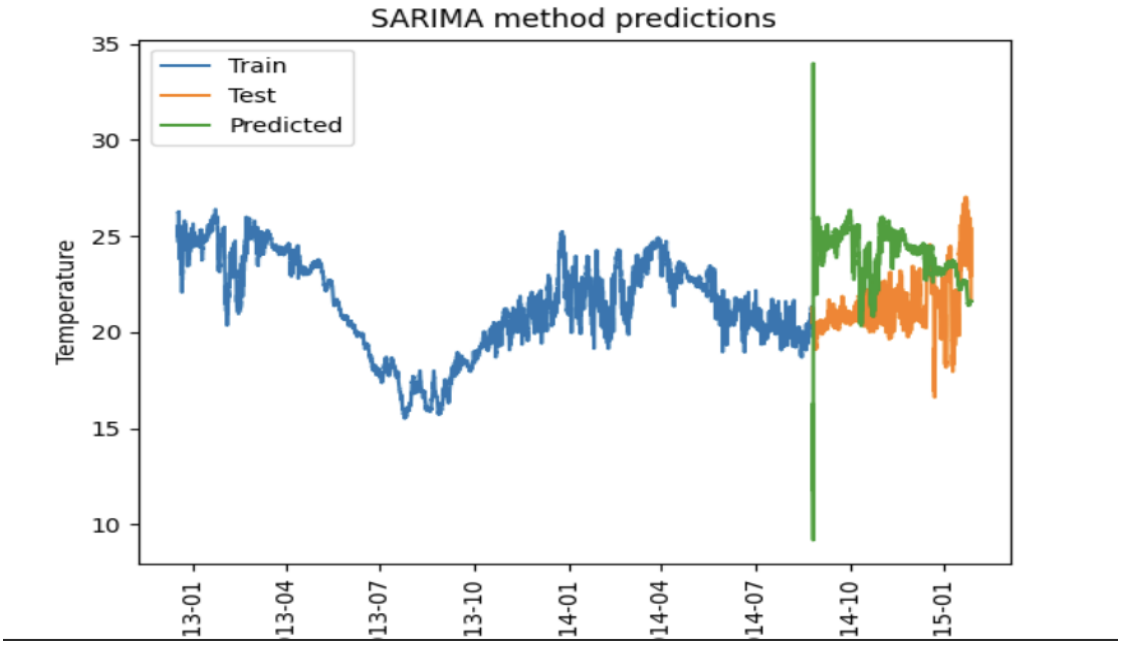
Heteroskedasticity (H): The test for homoscedasticity indicates a p-value of 0.00, suggesting the presence of heteroskedasticity in the residuals.

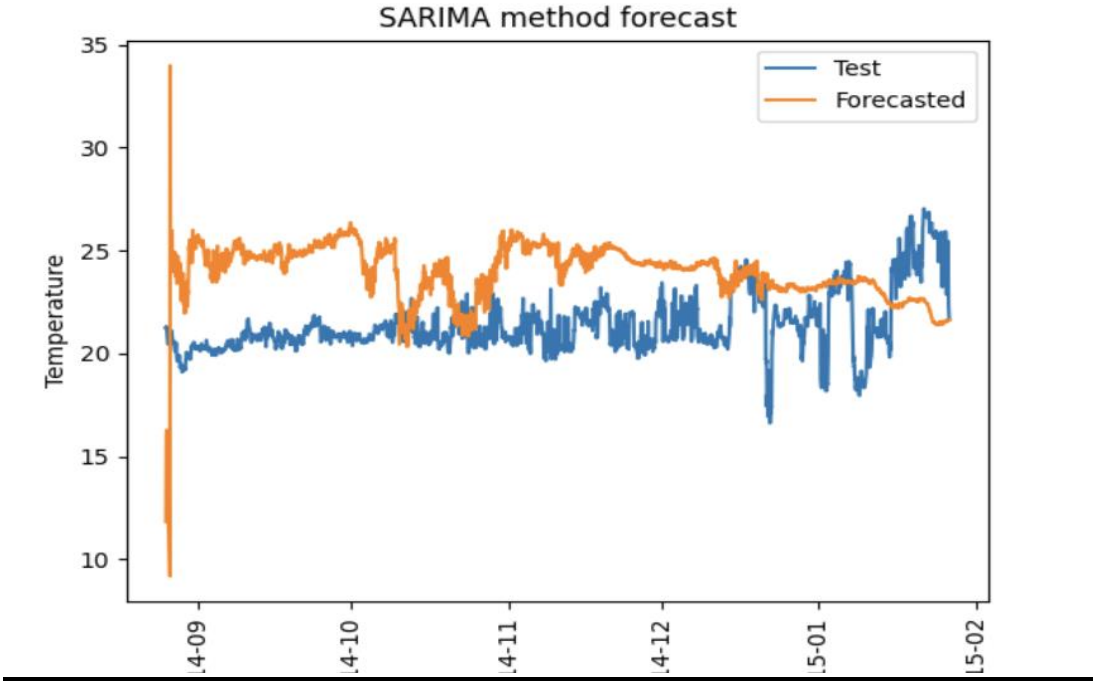
The SARIMAX model results indicate a well-fitted model for the given time series data representing temperature.

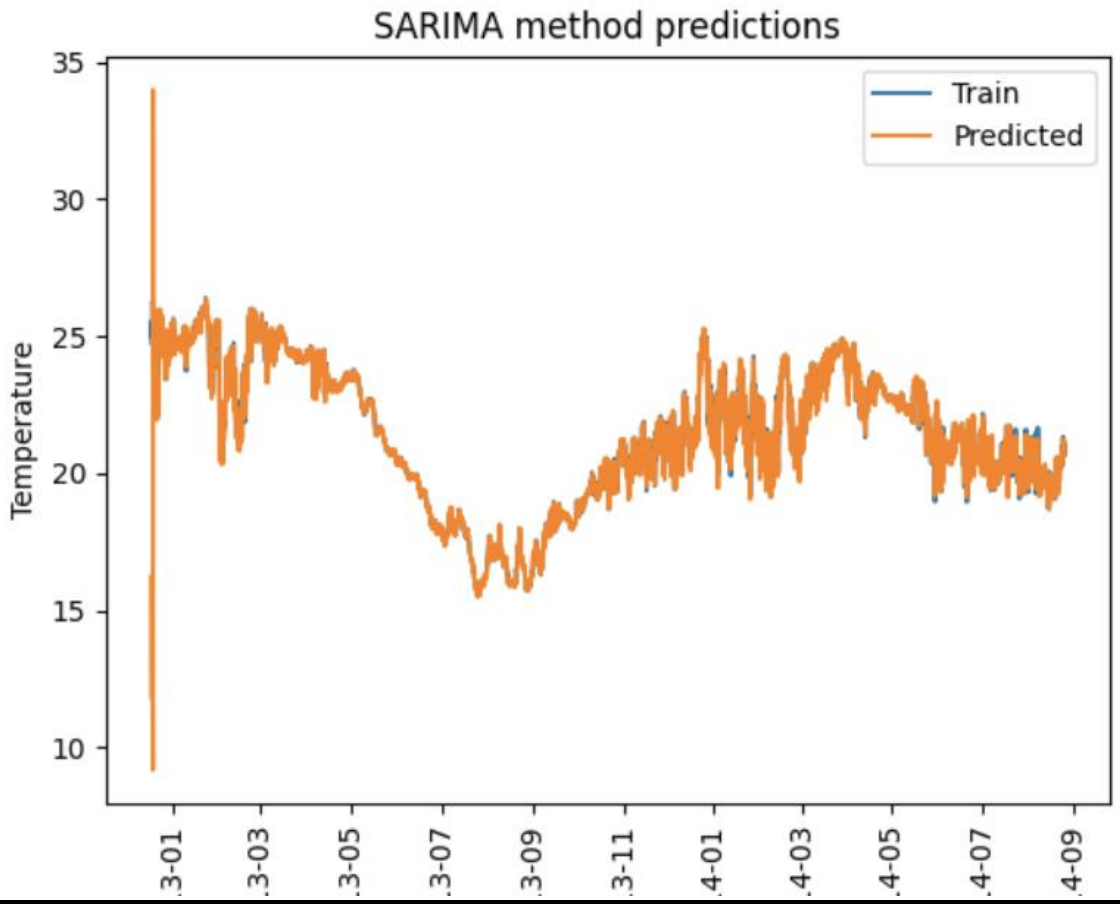




Looking at the ACF plot of residuals, the ACF values are not converging and have no white noise. And the ACF of forecast errors also has 0 zero white and doesn't converge.







The plot of train versus predicted values shows that the model has tried capturing the values with utmost similarity and achieved it . This shows that the SARIMA model could work better than other models if the computer was computationally high to do seasonal ARIMA of 1260.

Performance metrics:

```
MSE of SARIMA on train data: 0.24346748125486747
MSE of SARIMA on test data: 11.901151330887716
Q-value (residual):         lb_stat  lb_pvalue       bp_stat  bp_pvalue
1   9937.749347        0.0   9935.733713        0.0
2  18394.755160        0.0  18390.452458        0.0
3  25579.019818        0.0  25572.288522        0.0
4  31841.288991        0.0  31832.017392        0.0
5  37154.760283        0.0  37142.974027        0.0
Q-value (Forecast):
         lb_stat  lb_pvalue       bp_stat  bp_pvalue
1   3468.821155        0.0   3466.008598        0.0
2   6799.069245        0.0   6792.656420        0.0
3   9992.536979        0.0   9981.808656        0.0
4  13074.661235        0.0  13058.934872        0.0
5  16049.624785        0.0  16028.270114        0.0
SARIMA: Mean of residual error is 0.01618482808507984 and Forecast error is -2.552816051887291
SARIMA: Variance of residual error is 0.24320553259472383 and Forecast error is 5.384281536114301
Covariance matrix
               ar.L1        sigma2
ar.L1   3.468473e-06  4.060967e-08
sigma2  4.060967e-08  2.769746e-08
Standard error: ar.L1     0.001862
sigma2    0.000166
dtype: float64
RMSE of SARIMA method on train data: 0.49342424064375623
RMSE of SARIMA method on test data: 3.449804535171191
```

The model performed well on the training data (mse of 0.2434), but the relatively high MSE on the test data (mse: 11.9011) suggests a less accurate fit during forecasting. The q-value metrics shows low p-values (0.0) for the Ljung-Box test and the box-pierce statistic value indicate significant autocorrelation in residuals, suggesting that the model may not capture all temporal dependencies. The mean of residuals is close to zero(0.01618), indicating unbiased predictions. However, the forecast error mean (-2.5528) being negative may suggest a tendency to underpredict. The variance values provide insights into the stability of the model.The RMSE indicates the average magnitude of errors in the model's predictions. The higher RMSE on the test data(3.4498) suggests a less accurate prediction compared to the training data.

The performance of this method on test data is given below:

MSE: 11.90; Q-value:3466.00; Mean of forecast:-2.55; Variance: 5.3842; RMSE: 3.44

15. FORECAST FUNCTIONS:

Function:

```
1 usage
def holt_winters_forecast(train_data, seasonal_type='add', trend_type='add', seasonal_periods=21):
    model = sm.tsa.ExponentialSmoothing(train_data, seasonal=seasonal_type, trend=trend_type,
                                        seasonal_periods=seasonal_periods, initialization_method='estimated')
    fit_model = model.fit()
    forecast_values = fit_model.forecast()
    return forecast_values

forecast_result = holt_winters_forecast(y_train)
print(forecast_result)
```

The forecasted value:

```
2014-08-25 14:00:00    21.071948
```

16. RESIDUAL ANALYSIS AND DIAGNOSTIC TEST:

```
Confidence interval:
                 0         1
const    20.023884  22.727822
ar.L1     0.997175   0.999746
sigma2    0.017629   0.017901
Standard error: const     0.689793
ar.L1      0.000656
sigma2     0.000069
dtype: float64
The estimated variance of error is 0.06402852313086012
The residuals are white
The roots of numerator are []
The roots of denominator are [0.99995822]
```

The constant term (const) has a confidence interval of [20.023884, 22.727822], indicating statistical significance as it does not include zero. The autoregressive (AR) coefficient (ar.L1) has a confidence interval of [0.997175, 0.999746], excluding zero, suggesting statistical significance. The estimated variance of the error is 0.0640 and the chi squared test tells that the residuals are white. The zero pole cancellation didn't happen since there is no roots numerator. Lastly, the confidence interval for the residual variance (sigma2) is [0.017629, 0.017901], providing a range of plausible values for the variance.

18. FINAL MODEL SELECTION:

METRICS	HOLTS WINTER	AVG.	NAÏVE	DRIFT	SES	MLR	ARMA (1,0)	SARIMA
MSE TRAIN	0.016093	6.10558	0.04404	0.017855	6.269933	1.83473	0.018788	0.243467
Q-VALUE	3573.402682	3572.0305	3572.98	3596.88	3572.98	3671.90	3568.86	3466.00
MEAN RESID	-0.00018	-0.979270	-0.000293	0.000507399	0.407120	-5.82800	-2.04188	0.016184
VAR. RESID	0.016093	5.14661	0.044040	0.017855	6.102703	16.4792	0.018788	0.243205
RMSE TRAIN	0.12686	2.47094	0.209858	0.13362	2.50398	1.35452	0.13707	0.493424
MSE TEST	2.35319	2.2385	2.34133	3.49735	2.41998	50.4449	2.170861	11.90115
RMSE TEST	1.534001	1.4961	1.53014	1.87012	1.555628	7.102458	1.47338	3.449804

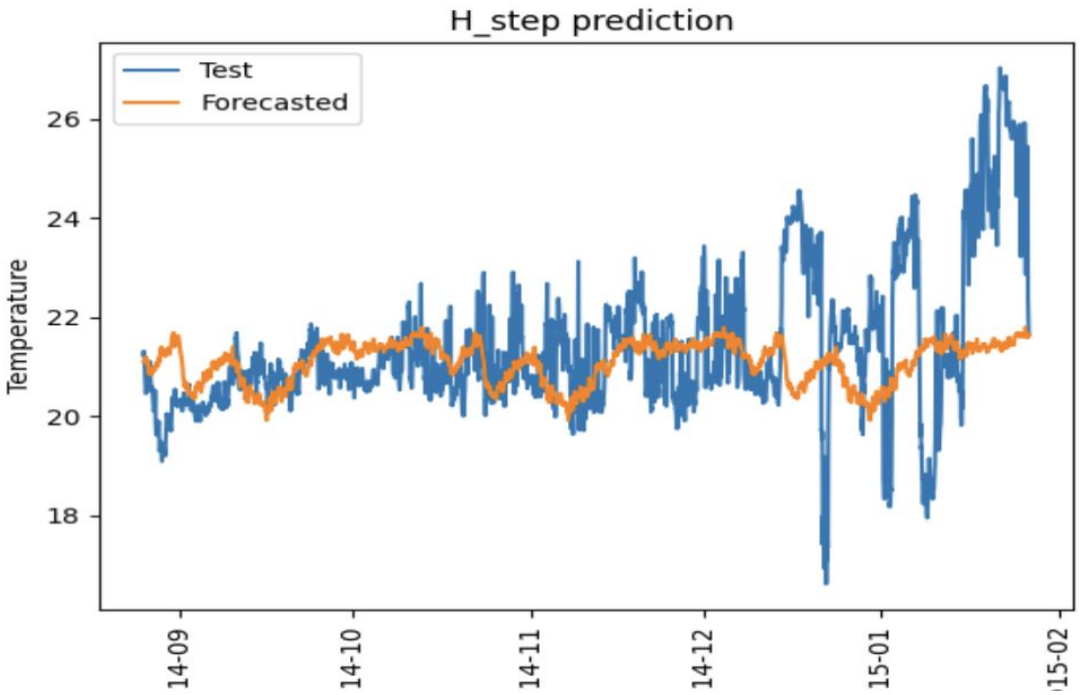
After a comprehensive evaluation of various metrics across all models, it is observed that the SARIMA model exhibits a relatively lower q-value compared to other models. Additionally, the mean of the residuals across all models is found to be closely aligned. In terms of MSE, the ARMA, AVERAGE, NAIVE, HOLTWINTER, SES, and DRIFT models display similar values, indicating comparable performance. Examining the RMSE values, it is noted that, except for SES and SARIMA, the remaining models exhibit relatively close results. Notably, the ARMA model stands out with a lower RMSE compared to others.

Upon inspecting the plots of the train, test, and predicted values, as well as the plot of the train and forecasted values, the representations of the Holt-Winters and SARIMA models appeared more

meaningful than those of the other models. Further analysis of the metrics indicated that the Holt-Winters model exhibited more favorable performance than the SARIMA model. Consequently, the Holt-Winters model is chosen as the final model, considering its meaningful visual representations and superior metric performance compared to the SARIMA model.

19. H-STEP AHEAD PREDICTIONS:

The plot of test and forecasted.



```
Freq: H, dtype: float64
2014-08-25 14:00:00    21.102988
2014-08-25 15:00:00    21.115477
2014-08-25 16:00:00    21.135848
2014-08-25 17:00:00    21.139991
2014-08-25 18:00:00    21.157418
                           ...
2015-01-26 11:00:00    21.644070
2015-01-26 12:00:00    21.669807
2015-01-26 13:00:00    21.652038
2015-01-26 14:00:00    21.606243
2015-01-26 15:00:00    21.639296
Freq: H, Length: 3698, dtype: float64
```

I did the h_Step prediction for the whole test set and the values are as above screenshot.

CONCLUSION:

The Holt winter model has been determined to be the most accurate model for predicting the temperature at the underwater surface. The model's performance on the test and train sets of data was taken into account during selection. Taking into account that other models in the competition that used this model, ARMA, had lower MSE on train data, but the model didn't capture the precise data on the test data, as evidenced by the model's simple line appearance in the plot of the predicted vs. test set. Although the computational complexity of 21 seasonality hampered the SARIMA, it still performed well on both train and test data; nonetheless, the Holt Winter model outperformed it when compared to the MSE on test data.

The model's inability to adequately represent a sharp rise or fall in the data values would be its main drawback. SARIMA model with seasonality of 1260 can be utilized on powerful computing platforms to get beyond this restriction and predict dataset results that are more accurate. The implementation of a SARIMA model with precise seasonal order or LSTM, a neural network-based model with a globally viewable user interactive dash app that can be developed in order to identify the underwater surface temperature, would be the project's future enhancement.

APPENDIX

Steps to run the project file:

Run the Ponswarnalaya_ravichandran_project_TS_2023.py file to implement the project end-to-end.

Save both the Ponswarnalaya_ravichandran_project_TS_2023.py and Toolbox.py file in same folder.

Toolbox.py

```
import numpy as np

import pandas as pd

from pandas import Series

import matplotlib.pyplot as plt

from statsmodels.tsa.stattools import adfuller,kpss,pacf

import statsmodels.api as sm

import seaborn as sns

np.random.seed(6313)

from scipy import signal



def cal_rolling_mean_var(x1,y1): #calculating rolling mean and variance

        rolling_mean=[]

        rolling_variance=[]

        for i in range(1,len(x1)+1):
```

```
                result=np.mean(x1[:i])
                result_variance=np.var(x1[:i])
                rolling_mean.append(result)
                rolling_variance.append(result_variance)


    #print(f"Rolling mean :",rolling_mean)
    #print(f"Rolling variance of :",rolling_variance)
    print("@"*100)


    plt.figure()
    plt.plot(y1, rolling_mean, color='Yellow')
    plt.ylabel("Rolling mean")
    plt.xlabel('Samples')
    plt.title(f"Plot of Rolling mean")
    plt.show()
    plt.figure()
    plt.plot(y1, rolling_variance, color="Purple")
    plt.ylabel(f"Rolling variance")
    plt.xlabel("Samples")
    plt.title(f"Plot of Rolling variance")
    plt.show()
```

```
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))



def kpss_test(z):

      kpsstest = kpss(z, 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)



#firstorder &secondorder difference

def first_order_difference(l):

      difference=[]

      difference[0]=difference.append([np.nan])

      for j in range(1,len(l)):
```

```
                difference_value=l[j]-l[j-1]
                difference.append(difference_value)


    return Series(difference)



def second_order_difference(l):
    difference=[]
    difference[0]=difference.append([np.nan])
    difference[1]=difference.append([np.nan])
    for j in range(2,len(l)):
                difference_value=l[j]-2*l[j-1]+l[j-2]
                difference.append(difference_value)
    return Series(difference)


def third_order_diff(l):
  difference=[]
  difference[0]=difference.append([np.nan])
  difference[1]=difference.append([np.nan])
  difference[2]=difference.append([np.nan])
  for j in range(3,len(l)+2):
```

```
        difference_value = l[j]-l[j-1]
        difference.append((difference_value))
    return Series(difference)


def correlation_coefficient_cal(x,y):
    n=len(x)
    sum_1=0
    sum_2=0
    for i in x:
        sum_1+=i

    mean__x= sum_1/n

    for j in y:
        sum_2+=j

    mean__y =sum_2/n
    difference=0
    denominator_1=0
    denominator_2=0
    for i,j in zip(x,y):
```

```
        difference+=(i-mean__x)*(j-mean__y)
        denominator_1+=(i-mean__x)**2
        denominator_2+=(j-mean__y)**2

    d=(denominator_1)**0.5
    d1=(denominator_2)**0.5
    r=difference/(d*d1)
    return r

def ACF(x,lags):
    mean_acf =np.mean(x)
    T=len(x)
    list_1=[]
    prod_1=0
    deno_1=0
    for t in range(0, T):
        deno_1 += (x[t] - mean_acf) ** 2

    for l in range(0,lags+1):
        for t in range(l,T):
            prod_1+=(x[t]-mean_acf)*(x[t-l]-mean_acf)
```

```
        acf=float(prod_1/deno_1)

        prod_1 = 0

        list_1.append(acf)

    print("ACF:",list_1)

    return list_1



def estimated_var(x,k):

    sum=0

    for i in range(0,len(x)):

        sum+=i**2

    denominator = 1/(len(x)-k-1)

    variance = np.sqrt((denominator*sum))

    return variance



def movingaverage(data):

    m=int(input("Enter m:"))

    if m%2!=0: #odd

       j = 0

       ma_list =[]

       while (j+m)!=len(data)+1:

           sum = 0
```

```
        mean=0
        for i in range(j,j+m):
            sum+=data[i]
        mean=sum/m
        ma_list.append(mean)
        j+=1
        k = int((m - 1) / 2)
    return ma_list,k


elif m%2==0: #even
    fold=int(input("Folding order:"))
    if fold%2!=0:
        print("Invalid fold value(Fold should be even)")
    else:
        j=0
        ma_list = []
        while (j + m) != len(data) + 1:
            sum = 0
            mean = 0
            for i in range(j, j + m):
                sum += data[i]
```

```
            mean = sum / m
            ma_list.append(mean)
            j += 1
        j=0
        final=[]
        while (j + fold) != len(ma_list) + 1:
            sum = 0
            mean = 0
            for i in range(j, j + fold):
                sum += ma_list[i]
            mean = sum / fold
            final.append(mean)
            j += 1
        k=int(m/2)
        return final,k


def armaprocess_GPAC():
    T=int(input("Enter number of samples: "))
    mean=int(input("Enter the mean of white noise: "))
    var=int(input("Enter variance of white noise: "))
    na=int(input("Enter AR process order: "))
```

```
    nb=int(input("Enter MA process order: "))
    naparam=[0]*na
    nbparam=[0]*nb
    for i in range(0,na):
        naparam[i]=float(input(f"Enter the coefficient of AR:a{i+1}: "))
    for i in range(0,nb):
        nbparam[i]=float(input(f"Enter the coefficient of MA:b{i+1}: "))
    ar=np.r_[1,naparam]
    ma=np.r_[1,nbparam]
    arma_process = sm.tsa.ArmaProcess(ar, ma)
    mean_y = mean* (1 + np.sum(nbparam)) / (1 + np.sum(naparam))
    y = arma_process.generate_sample(T, scale=np.sqrt(var) + mean_y)
    return y


def qvalue_cal(y,an,bn):
    deno=[]
    k=an
    j=bn
    for a in range(k):
        deno.append([])
        for b in range(k):
```

```
            deno[a].append(y[np.abs(j+b)])

        j=j-1

    ddeno=round(np.linalg.det(deno),5)

    j=bn

    num=deno[:k-1]

    num.append([])

    for a in range(k):

        num[k-1].append(y[j+a+1])

    dnum=round(np.linalg.det(num),5)

    if ddeno==0:

        return float('inf')

    else:

        qval=dnum/ddeno

        return round(qval,4)



def GPAC(y,k,j):

    q=[]

    for b in range(j):

        q.append([])

        for a in range(1,k+1):

            q[b].append(qvalue_cal(y,a,b))
```

```
    gpac=np.array(q).reshape(j,k)

    gpactable=pd.DataFrame(gpac)

    c=np.arange(1,k+1)

    gpactable.columns=c

    print(gpactable)

    sns.heatmap(gpactable,annot=True)

    plt.xlabel('k')

    plt.ylabel('j')

    plt.title('GPAC table')

    plt.show()



    # ACF plot

def acf_plot(lags, acf, samples):

    x = np.arange(0, lags)

    m = 1.96 / np.sqrt(len(samples))

    plt.stem(x, acf, linefmt='r-', markerfmt='bo', basefmt='b-')

    plt.stem(-1 * x, acf, linefmt='r-', markerfmt='bo', basefmt='b-')

    plt.title("ACF")

    plt.axhspan(-m, m, alpha=.2, color='yellow')

    plt.xlabel('Lags')

    plt.ylabel('ACF')
```

```
    plt.show()



def acf_pacf_plot_gpac(lags,samples,process):

    x = np.arange(0,lags)

    m = 1.96 / np.sqrt(samples)

    p=pacf(process,lags-1)

    acf=ACF(process,lags)

    fig,((ax1,ax2))=plt.subplots(nrows=2, ncols=1,figsize=(8,8))

    ax1.stem(x,acf, linefmt='r-', markerfmt='bo', basefmt='b-')

    ax1.stem(-1 *x,acf,linefmt='r-',markerfmt='bo', basefmt='b-')

    ax1.set_title("ACF")

    ax1.axhspan(-m,m,alpha=.1,color='yellow')

    ax2.stem(x,p, linefmt='r-', markerfmt='bo', basefmt='b-')

    ax2.stem(-1 * x,p, linefmt='r-', markerfmt='bo', basefmt='b-')

    ax2.set_title("PACF")

    ax2.axhspan(-m, m, alpha=.1, color='yellow')

    fig.supxlabel('Lags')

    fig.supylabel('ACF/PACF')

    fig.tight_layout()

    plt.show()
```

```
from statsmodels.graphics.tsaplots import plot_acf , plot_pacf

def ACF_PACF_Plot(y,lags):

acf = sm.tsa.stattools.acf(y, nlags=lags)

pacf = sm.tsa.stattools.pacf(y, nlags=lags)



fig = plt.figure()

fig.suptitle(f'ACF/PACF of the raw data : {lags}')

plt.subplot(211)

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()



#LM algorithm

def WN(teta,na,y):

   numerator=[1]+list(teta[na:])

   denominator=[1]+list(teta[:na])

   if len(numerator)!=len(denominator):

       while len(numerator)<len(denominator):

           numerator.append(0)
```

```
        while len(denominator)<len(numerator):
            denominator.append(0)
    system=(denominator,numerator,1)
    t,e=signal.dlsim(system,y)
    e=[i[0] for i in e]
    return np.array(e)


def step0(na,nb):
    teta_o=np.zeros(shape=(na+nb,1))
    return teta_o.flatten()


def step1(delta,na,nb,teta,y):
    e_teta=WN(teta,na,y)
    SSE_O=np.dot(e_teta.T,e_teta)
    X=[]
    for i in range(na+nb):
        teta_delta = teta.copy()
        teta_delta[i]=teta[i]+delta
        en=WN(teta_delta,na,y)
        Xi=(e_teta-en)/delta
        X.append(Xi)
```

```
    Xfinal=np.transpose(X)

    A=np.dot(Xfinal.T,Xfinal)

    G=np.dot(Xfinal.T,e_teta)

    return A,G,SSE_O



def step2(A,G,mu,na,nb,teta,y):

    n=na+nb

    I=np.identity(n)

    dteta1=A+(mu*I)

    dteta_inv=np.linalg.inv(dteta1)

    delta_teta=np.dot(dteta_inv,G)

    teta_new=teta+delta_teta

    e=WN(teta_new,na,y)

    SSE_new=np.dot(e.T,e)

    if np.isnan(SSE_new):

        SSE_new=10**10

    return SSE_new,delta_teta,teta_new



def step3(max_iter,mu,delta,epsilon,mu_max,na,nb,y):

    num_iter=0

    teta=step0(na,nb)
```

```
SSE=[]

while num_iter<max_iter:

    A,G,SSE_O=step1(delta,na,nb,teta,y)

    if num_iter == 0:

        SSE.append(SSE_O)

    SSE_new,delta_teta,teta_new=step2(A,G,mu,na,nb,teta,y)

    SSE.append(SSE_new)

    if SSE_new<SSE_O:

        if np.linalg.norm(delta_teta)<epsilon:

            teta_hat=teta_new

            var=SSE_new/(len(y)-A.shape[0])

            A_inv=np.linalg.inv(A)

            cov=var*A_inv

            return SSE,cov,teta_hat,var

        else:

            teta=teta_new

            mu=mu/10

    while SSE_new>=SSE_O:

        mu=mu*10

        if mu>mu_max:

           print('Mu\'s maximum limit is exceeded')
```

```
            return None,None,None,None

        SSE_new, delta_teta, teta_new = step2(A, G, mu, na,nb, teta, y)

    num_iter+=1

    teta = teta_new

    if num_iter>max_iter:

        print('Maximum iterations exceeded')

        return None,None,None,None



#Confidence interval

with np.errstate(divide='ignore'):

   np.float64(1.0) / 0.0

def conf_int(cov,params,na,nb):

   print("Confidence Interval:")

   for i in range(na):

       pos=params[i]+2*np.sqrt(cov[i][i])

       neg=params[i]-2*np.sqrt(cov[i][i])

       print(neg,f'<a{i+1}<',pos)

   for i in range(nb):

       pos=params[na+i]+2*np.sqrt(cov[na+i][na+i])

       neg=params[na+i]-2*np.sqrt(cov[na+i][na+i])

       print(neg,f'<b{i+1}<',pos)
```

```
#zero-poles cancellation
def zero_poles(params,na):
    y_den=[1]+list(params[:na])
    e_num=[1]+list(params[na:])
    zeros=np.roots(e_num)
    poles=np.roots(y_den)
    print("The roots of numerator are",zeros)
    print("The roots of denominator are",poles)


#Plot of SSE
def plotSSE(SSE):
    iter=np.arange(0,len(SSE))
    plt.plot(iter,SSE,label='SSE')
    plt.xlabel('Number of iterations')
    plt.ylabel('SSE')
    plt.title('SSE vs. #of Iterations')
    plt.legend()
    plt.show()


#Plot one step ahead prediction plot
```

```
def onestepplot(y,y_hat):

   plt.plot(y,label='Actual/Train')

   plt.plot(y_hat,label='one step predictions')

   plt.title('Plot of Actual/Train vs. One step prediction')

   plt.legend()

   plt.xlabel('Samples')

   plt.ylabel('Value')

   plt.show()



#Chi-square test

from scipy.stats import chi2

def chi_test(na,nb,lags,Q,e):

   chi_statistic = (Q ** 2).sum().sum()

   dof = (Q.shape[na] - nb) * (Q.shape[na] - nb)

   alpha = 0.01

   chi_critical = chi2.ppf(1 - alpha, dof)

   if chi_statistic > chi_critical:

       print('The residuals are white')

   else:

       print('The residual is not white')
```

```
def difference(y,interval):
    diff=[]
    for i in range(interval,len(y)):
        value=y[i]-y[i-interval]
        diff.append(value)
    return diff


def sarima_model():
    T=int(input('Enter number of samples: '))
    mean=eval(input('Enter mean of white nosie: '))
    var=eval(input('Enter variance of white noise: '))
    na = int(input("Enter AR process order: "))
    nb = int(input("Enter MA process order: "))
    naparam = [0] * na
    nbparam = [0] * nb
    for i in range(0, na):
        naparam[i] = float(input(f"Enter the coefficient of AR:a{i + 1}: "))
    for i in range(0, nb):
        nbparam[i] = float(input(f"Enter the coefficient of MA:b{i + 1}: "))
    while len(naparam) < len(nbparam):
        naparam.append(0)
```

```
    while len(nbparam) < len(naparam):
        nbparam.append(0)
    ar = np.r_[1, naparam]
    ma = np.r_[1, nbparam]
    e=np.random.normal(mean,np.sqrt(var),T)
    system=(ma,ar,1)
    t,process=signal.dlsim(system,e)
    return process


#Base models
#average
def avg_one(x):
    train=[]
    for i in range(0,len(x)):
        mean=np.mean(x[0:i])
        train.append(mean)
    return train
def avg_hstep(train,test):
    forecast=np.mean(train)
    pred=[]
    for i in range(len(test)):
```

```
        pred.append(forecast)

    return pred
```

Ponswarnalaya_ravichandran_project_TS_2023:

```
import matplotlib.pyplot as plt

from sklearn.metrics import mean_squared_error

from sklearn.model_selection import train_test_split, GridSearchCV

import statsmodels.tsa.holtwinters as ets

from sktime.forecasting.exp_smoothing import ExponentialSmoothing

from statsmodels.stats.diagnostic import acorr_ljungbox

from sklearn.decomposition import PCA

from sklearn.linear_model import Ridge

from sklearn.preprocessing import StandardScaler

from statsmodels.stats.outliers_influence import variance_inflation_factor

from numpy import linalg as la

from Toolbox import *

import pandas as pd

import numpy as np

import statsmodels.api as sm

import warnings

warnings.filterwarnings('ignore')
```

```
data = pd.read_csv("OG.csv", encoding = 'unicode_escape')

print(data.info())

#6

#checking for null values

print(data.isnull().sum())



#ffilling the dta

# Example using forward-fill for missing values

data['Temp (°C)'].fillna(method='ffill', inplace=True)

print(data.isnull().sum())



#renaming a column which has '#'

data.rename(columns={'#': 'INDEX'}, inplace=True)

print(data.info(), data.head())







data['Date'] = pd.to_datetime(data['Date'] + ' ' + data['Time'])



print('DATA AFTER RENAMING THE DATE')

print(data.info())
```

```
#PLOT OF TIME VS TEMP(C)

plt.plot(data['Date'], data['Temp (°C)'], label='Temperature')

plt.xlabel('Time')

plt.ylabel('Temp (°C)')

plt.title('Time vs Temperature')

plt.tight_layout()

plt.legend()

plt.show()



#onehot

df = pd.get_dummies(data, columns=['Site'], prefix='Site')

print(df.info())



#downsampling

df = df.set_index('Date').resample('H').mean()

df = df.reset_index()

print(df.info())



#plotting the downsampled data

plt.plot(df['Date'], df['Temp (°C)'], label='Temperature')

plt.xlabel('Time')
```

```
plt.ylabel('Temp (°C)')

plt.title('Time vs Temperature')

plt.tight_layout()

plt.legend()

plt.show()



cor = df.corr()

plt.figure(figsize=(12, 10))

import seaborn as sns

sns.heatmap(cor,vmin=-1,vmax=1,center=0,cmap='PiYG')

plt.title("Heatmap of dataset", fontsize =15)

plt.show()

print(cor)



print("The dataset is df now")

train,test=train_test_split(df,test_size=0.2,shuffle=False)




#7

#stationarity check

cal_rolling_mean_var(df['Temp (°C)'], df['Date']) #rollingmean
```

```
ADF_cal(df["Temp (°C)"]) #adf of raw data

kpss_test(df["Temp (°C)"])



acf = ACF(df['Temp (°C)'],50)

x=np.arange(0,51)

plt.stem(x,acf,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.stem(-1*x,acf,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.title("ACF plot")

plt.ylabel("ACF")

plt.xlabel("Lags")

plt.show()



ACF_PACF_Plot(df['Temp (°C)'], 3000)







print("The data is not stationary at this point")







#first order differencing

diff1=difference(df['Temp (°C)'],interval=1260)
```

```
diff_df=pd.DataFrame(diff1, index=df.index[1260:])

cal_rolling_mean_var(diff_df, np.arange(len(diff1)))

ADF_cal(diff_df)

kpss_test(diff_df)



acf2=ACF(diff1,50)

x= np.arange(0,51)

plt.stem(x,acf2,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.stem(-1*x,acf2,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.title("ACF plot")

plt.ylabel("ACF")

plt.xlabel("Lags")

plt.show()



ACF_PACF_Plot(diff1, 3000)



print("The data is not stationary yet")



#diff2

diff2 = difference(diff1, 1)

diff_df1 = pd.DataFrame(diff1, df.index[1260:])  # Adjust the index range
```

```
diff_df2 = pd.DataFrame(diff2, df.index[1261:])



diff_df21 = pd.DataFrame(diff2, index=df.index[1261:], columns=["Temp (°C)"])

# Adjust column name if needed



# Plotting code

plt.plot(diff_df21.index, diff_df21["Temp (°C)"], label='Differenced')

plt.xlabel('Time')

plt.ylabel('Temp (°C)')

plt.title('Time vs Temperature')

plt.tight_layout()

plt.legend()

plt.show()



cal_rolling_mean_var(diff_df2, np.arange(len(diff2)))



ADF_cal(diff2)

kpss_test(diff2)



acf3=ACF(diff2,50)

x= np.arange(0,51)

plt.stem(x,acf3,linefmt='r-', markerfmt='bo', basefmt='b-')
```

```
plt.stem(-1*x,acf3,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.title("ACF plot")

plt.ylabel("ACF")

plt.xlabel("Lags")

plt.show()



ACF_PACF_Plot(diff2,3000)




#8

#time series decomposition

from statsmodels.tsa.seasonal import STL

index = pd.date_range('2013-02-20 11:40:02', periods=len(df), freq='1H')   #
Assuming data is recorded every 1 hour



Temperature = pd.Series(np.array(df['Temp (°C)']), index=index)



STL= STL(Temperature)

res=STL.fit()

fig=res.plot()

plt.xlabel("Iterations")

plt.tight_layout()
```

```
plt.show()



T=res.trend

S=res.seasonal

R=res.resid




plt.plot(T,label="Trend")

plt.plot(R,label="Residual")

plt.plot(S,label="Seasonal")

plt.xlabel("Iterations")

plt.ylabel("STL")

plt.legend()

plt.title("Trend, Seasonality and residuals of data")

plt.show()



#Strength of trend

var=1-(np.var(R)/np.var(T+R))

Ft=np.max([0,var])

print("Strength of trend:",Ft)
```

```
#Strength of seasonality

var1=1-(np.var(R)/np.var(S+R))

Fs=np.max([0,var1])

print("Strength of seasonality:",Fs)



#seasonally adjusted data

seasonally_adj= Temperature-S

plt.figure(figsize=(12, 10))

plt.plot(index, df['Temp (°C)'],label="Original")

plt.plot(index, seasonally_adj,label="adjusted")

plt.xlabel("Time")

plt.xticks(rotation=45, ha='right')

plt.ylabel("Temperature")

plt.title("Seasonally adjusted vs. Original")

plt.legend()

plt.show()

#

#detrended data

detrended=Temperature-T

plt.figure(figsize=(12, 10))

plt.plot(index, df['Temp (°C)'],label="Original")
```

```
plt.plot(index,detrended,label="Detrended")

plt.xlabel("Time")

plt.xticks(rotation=45, ha='right')

plt.ylabel("Temperature")

plt.title("Detrended vs. Original Data")

plt.legend()

plt.show()



# 9

# Holt-Winters method

print(df.columns)

df['Date'] = pd.to_datetime(df['Date'])

df.set_index('Date', inplace=True)

X = df[['INDEX', 'Latitude', 'Longitude', 'Depth', 'Site_Ilha Deserta',
'Site_Ilha da Galé', 'Site_Ilha do Coral',

       'Site_Ilha dos Lobos', 'Site_Moleques do Sul', 'Site_Parcel da
Pombinha', 'Site_Parcel do Xavier (Alalunga)',

       'Site_Tamboretes', 'Site_lha do Xavier']]

y = df['Temp (°C)']



x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
shuffle=False)
```

```
# Fitting Holt-Winters model

holtw1 = ets.ExponentialSmoothing(y_train, damped_trend= True,trend='add',
seasonal='add',seasonal_periods=1260)

holtw = holtw1.fit()



# HW prediction on train set

holtw_pred_train = holtw.predict(start=y_train.index[0], end=y_train.index[-
1])

holtw_df_train = pd.DataFrame(holtw_pred_train, columns=['Temp (°C)'],
index=y_train.index)



# HW prediction on test set

holtw_pred_test = holtw.predict(start=y_test.index[0], end=y_test.index[-1])

holtw_df_test = pd.DataFrame(holtw_pred_test, columns=['Temp (°C)'],
index=y_test.index)



# Plot of HW model

plt.figure(figsize=(12, 8))

plt.plot(y_train.index, y_train, label='Train')

plt.plot(y_test.index, y_test, label='Test')

plt.plot(holtw_df_test.index, holtw_df_test['Temp (°C)'], label='Holts winter
prediction (Test)')
```

```
plt.legend()

plt.xlabel("Time")

plt.ylabel("Temperature")

plt.title("Holts winter model")

plt.show()



#Model performance on train and test data



#MSE

HW_train_mse=mean_squared_error(y_train,holtw_df_train[['Temp (°C)']])

print('MSE of Holts Winter method on train data:',HW_train_mse)

HW_test_mse=mean_squared_error(y_test,holtw_df_test['Temp (°C)'])

print('MSE of Holts Winter method on test data:',HW_test_mse)



#residual error

HW_reserror=y_train-holtw_df_train['Temp (°C)']



#Forecast error

HW_foerror=y_test-holtw_df_test['Temp (°C)']



#ACF
```

```
acf_hw_res=ACF(HW_reserror.values,60) #train

x=np.arange(0,61)

m=1.96/np.sqrt(len(y_test))

plt.stem(x,acf_hw_res,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.stem(-1*x,acf_hw_res,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.xlabel('Lags')

plt.ylabel('ACF')

plt.title('ACF Plot of Residuals (HW)')

plt.axhspan(-m,m,alpha = .1, color = 'yellow')

plt.tight_layout()

plt.show()




acf_hw_fore=ACF(HW_foerror.values,60) #test

x=np.arange(0,61)

m=1.96/np.sqrt(len(y_test))

plt.stem(x,acf_hw_fore,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.stem(-1*x,acf_hw_fore,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.xlabel('Lags')

plt.ylabel('ACF')

plt.title('ACF Plot of Forecast error (HW)')
```

```
plt.axhspan(-m,m,alpha = .1, color = 'yellow')

plt.tight_layout()

plt.show()



#Q-value

hotl_q_t=sm.stats.acorr_ljungbox(HW_reserror, lags=5,return_df=True)

print('Q-value (residual):',hotl_q_t)

lbvalue=sm.stats.acorr_ljungbox(HW_foerror,lags=5,return_df=True)

print('Q-value (Forecast):\n',lbvalue)



#Error mean and variance

print('Holts winter: Mean of residual error is',np.mean(HW_reserror),'and

Forecast error is',np.mean(HW_foerror))

print('Holts winter: Variance of residual error is',np.var(HW_reserror),'and

Forecast error is',np.var(HW_foerror))



#RMSE

HW_train_rmse=mean_squared_error(y_train,holtw_df_train['Temp

(°C)'],squared=False)

print('RMSE of Holts Winter method on train data:',HW_train_rmse)

HW_test_rmse=mean_squared_error(y_test,holtw_df_test['Temp

(°C)'],squared=False)
```

```
print('RMSE of Holts Winter method on test data:',HW_test_rmse)



#10: Feature selection and collinearity

# Assuming X contains your predictor variables and y is the target variable

X_mat = x_train.values

Y = y_train.values

X_svd =sm.add_constant(X_mat)

H= np.matmul(X_svd.T,X_svd)

s,d,v= np.linalg.svd(H)

print('Singular Values: ',d)



#Condition number

print("The condition number is ",la.cond(X_svd))



#Feature selection

x_train_ols=sm.add_constant(x_train)

model=sm.OLS(y_train,x_train_ols).fit()

print(model.summary())



#collinearity removal process

vif_data = pd.DataFrame()
```

```
vif_data["Variable"] = X.columns

vif_data["VIF"]   =   [variance_inflation_factor(X.values,   i)   for   i   in
range(X.shape[1])]

print(vif_data)



scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(x_train)

X_test_scaled = scaler.transform(x_test)



# Ridge Regression with Cross-Validated Grid Search

ridge = Ridge()

param_grid = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}

grid_search            =            GridSearchCV(ridge,            param_grid,
scoring='neg_mean_squared_error', cv=5)

grid_search.fit(X_train_scaled, y_train)



# Best hyperparameters

best_alpha = grid_search.best_params_['alpha']



# Ridge Regression with optimal alpha

ridge_optimal = Ridge(alpha=best_alpha)

ridge_optimal.fit(X_train_scaled, y_train)
```

```
# Predictions on train and test sets
y_train_pred = ridge_optimal.predict(X_train_scaled)
y_test_pred = ridge_optimal.predict(X_test_scaled)

# Evaluate MSE
mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)

# VIF Calculation
vif = pd.DataFrame()
vif["Variable"] = X.columns
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in
range(X.shape[1])]

# Print results
print(f'MSE of Ridge regression on train data: {mse_train}')
print(f'MSE of Ridge regression on test data: {mse_test}')
print('Optimal alpha:', best_alpha)
print('VIF values:\n', vif)

# Standardize the features
```

```
scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)

#pca

pca=PCA(n_components='mle',svd_solver='full')

pca.fit(X)

x_pca=pca.transform(X)

print('Explained variance ratio: Original Feature space vs.Reduced Feature
space\n',pca.explained_variance_ratio_)

ev = pca.explained_variance_ratio_

cv = np.cumsum(ev) * 100

num_com_95 = np.argmax(cv >= 95) + 1

num_com_to_remove = pca.n_components_ - num_com_95

explained_variance_ratio = pca.explained_variance_ratio_

cumulative_explained_variance = explained_variance_ratio.cumsum()

num_components = np.argmax(cumulative_explained_variance >= 0.95) + 1



# Calculate VIF for the new features after PCA

vif_data = pd.DataFrame()

vif_data["Variable"] = range(1, x_pca.shape[1] + 1)  # Use range as index for
VIF calculation

vif_data["VIF"]    =    [variance_inflation_factor(x_pca,    i)    for    i    in
range(x_pca.shape[1])]
```

```
print("VIF values after PCA:")

print(vif_data)



#11

#Base models

#Average method

train_pred_avg=avg_one(y_train)

test_pred_avg=avg_hstep(y_train,y_test)



#Plot of average method

plt.plot(y_train.index,y_train,label='Train')

plt.plot(y_test.index,y_test,label='Test')

plt.plot(y_test.index,test_pred_avg,label='Predicted')

plt.xlabel('Time')

plt.xticks(rotation='vertical')

plt.ylabel('Temperature')

plt.title('Average method predictions')

plt.legend()

plt.show()



#Plot of test vs predicted
```

```
plt.plot(y_test.index,y_test,label='Test')

plt.plot(y_test.index,test_pred_avg,label='Forecasted')

plt.xlabel('Time')

plt.xticks(rotation='vertical')

plt.ylabel('Temperature')

plt.title('Average method Forecast')

plt.legend()

plt.show()



#residual and forecast error

avg_res=y_train-train_pred_avg

avg_fore=y_test-test_pred_avg



#ACF of residual

acf_avg_train=ACF(avg_res.values[1:],40)

x=np.arange(0,41)

m=1.96/np.sqrt(len(y_train))

plt.stem(x,acf_avg_train,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.stem(-1*x,acf_avg_train,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.xlabel('Lags')

plt.ylabel('ACF')
```

```
plt.title('ACF Plot of Residuals')

plt.axhspan(-m,m,alpha = .1, color = 'yellow')

plt.tight_layout()

plt.show()



#forecastedavg acf

acf_avg_test=ACF(avg_fore.values,40)

x=np.arange(0,41)

m=1.96/np.sqrt(len(y_test))

plt.stem(x,acf_avg_test,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.stem(-1*x,acf_avg_test,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.xlabel('Lags')

plt.ylabel('ACF')

plt.title('ACF Plot of Forecast (Average)')

plt.axhspan(-m,m,alpha = .1, color = 'yellow')

plt.tight_layout()

plt.show()



#Model performance on train and test data



#MSE
```

```
avg_train_mse=mean_squared_error(y_train[1:],train_pred_avg[1:])

print('MSE of Average on train data:',avg_train_mse)

avg_test_mse=mean_squared_error(y_test,test_pred_avg)

print('MSE of Average on test data:',avg_test_mse)



#Q-value

q_avg_train=acorr_ljungbox(avg_res.values[1:],                              lags=5,
boxpierce=True,return_df=True)

print('Q-value (residual):',q_avg_train)

q_avgtest=sm.stats.acorr_ljungbox(avg_fore.values[1:],lags=5,boxpierce=True,r
eturn_df=True)

print('Q-value (Forecast):\n',q_avgtest)



#Error mean and variance

print('Average: Mean of residual error is',np.mean(avg_res),'and Forecast error
is',np.mean(avg_fore))

print('Average: Variance of residual error is',np.var(avg_res),'and Forecast
error is',np.var(avg_fore))



#RMSE

avg_train_rmse=
mean_squared_error(y_train[1:],train_pred_avg[1:],squared=False)
```

```
print('RMSE of Average method on train data:',avg_train_rmse)

avg_test_rmse= mean_squared_error(y_test,test_pred_avg,squared=False)

print('RMSE of Average method on test data:',avg_test_rmse)



#Naive method

train_naive=[]

for i in range(len(y_train[1:])):

   train_naive.append(y_train.values[i-1])



test_naive=[y_train.values[-1] for i in y_test]

naive_fore= pd.DataFrame(test_naive).set_index(y_test.index)



#Plot of naive method

plt.plot(y_train.index,y_train,label='Train')

plt.plot(y_test.index,y_test,label='Test')

plt.plot(y_test.index,test_naive,label='Predicted')

plt.xlabel('Time')

plt.xticks(rotation='vertical')

plt.ylabel('Temperature')

plt.title('Naive method predictions')

plt.legend()
```

```
plt.show()



#Plot of test vs predicted

plt.plot(y_test.index,y_test,label='Test')

plt.plot(y_test.index,test_naive,label='Predicted')

plt.xlabel('Time')

plt.xticks(rotation='vertical')

plt.ylabel('Temperature')

plt.title('Naive method predictions')

plt.legend()

plt.show()



#residual and forecast error

naive_res= y_train[1:]-train_naive

naive_fore1 = y_test-test_naive



#ACF

acf_naive_train=ACF(naive_res.values,40)



x=np.arange(0,41)

m=1.96/np.sqrt(len(y_train))
```

```
plt.stem(x,acf_naive_train,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.stem(-1*x,acf_naive_train,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.xlabel('Lags')

plt.ylabel('ACF')

plt.title('ACF Plot of Residuals(Naive)')

plt.axhspan(-m,m,alpha = .1, color = 'yellow')

plt.tight_layout()

plt.show()



acf_naive_test=ACF(naive_fore1.values,40)



x=np.arange(0,41)

m=1.96/np.sqrt(len(y_test))

plt.stem(x,acf_naive_test,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.stem(-1*x,acf_naive_test,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.xlabel('Lags')

plt.ylabel('ACF')

plt.title('ACF Plot of Forecast (Naive)')

plt.axhspan(-m,m,alpha = .1, color = 'yellow')

plt.tight_layout()

plt.show()
```

```
#Model performance on train and test data


#MSE
naive_train_mse=mean_squared_error(y_train[1:],train_naive)
print('MSE of Naive on train data:',naive_train_mse)
naive_test_mse=mean_squared_error(y_test,test_naive)
print('MSE of Naive on test data:',naive_test_mse)


#Q-value
q_naive_train=acorr_ljungbox(naive_res,         lags=5,         boxpierce=True,
return_df=True)
print('Q-value (residual):',q_naive_train)
q_naivetest=acorr_ljungbox(naive_fore1,lags=5,boxpierce=True,return_df=True)
print('Q-value (Forecast):\n',q_naivetest)


#Error mean and variance
print('Naive: Mean of residual error is',np.mean(naive_res),'and Forecast error
is',np.mean(naive_fore))
print('Naive: Variance of residual error is',np.var(naive_res),'and Forecast
error is',np.var(naive_fore))
```

```
#RMSE

naive_train_rmse=mean_squared_error(y_train[1:],train_naive,squared=False)

print('RMSE of Naive method on train data:',naive_train_rmse)

naive_test_rmse=mean_squared_error(y_test,test_naive,squared=False)

print('RMSE of Naive method on test data:',naive_test_rmse)



#Drift method

train_drift = []

value = 0

for i in range(len(y_train)):

   if i > 1:

       slope_val = (y_train[i - 1]-y_train[0]) / (i-1)

       y_predict = (slope_val * i) + y_train[0]

       train_drift.append(y_predict)

   else:

       continue



test_drift= []

for h in range(len(y_test)):

   slope_val = (y_train.values[-1] - y_train.values[0] ) /( len(y_train) - 1 )

   y_predict= y_train.values[-1] + ((h +1) * slope_val)
```

```
    test_drift.append(y_predict)



#Plot of drift method

plt.plot(y_train.index,y_train,label='Train')

plt.plot(y_test.index,y_test,label='Test')

plt.plot(y_test.index,test_drift,label='Predicted')

plt.xlabel('Time')

plt.xticks(rotation='vertical')

plt.ylabel('Temperature')

plt.title('Drift method predictions')

plt.legend()

plt.show()



#Plot of test vs predicted

plt.plot(y_test.index,y_test,label='Test')

plt.plot(y_test.index,test_drift,label='Forecasted')

plt.xlabel('Time')

plt.xticks(rotation='vertical')

plt.ylabel('Temperature')

plt.title('Drift method forecast')

plt.legend()
```

```
plt.show()



#residual and forecast error

drift_res=y_train[2:]-train_drift

drift_fore=y_test-test_drift



#RESIDUALS ACF

acf_drift_train=ACF(drift_res.values,40)

x=np.arange(0,41)

m=1.96/np.sqrt(len(y_train))

plt.stem(x,acf_drift_train,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.stem(-1*x,acf_drift_train,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.xlabel('Lags')

plt.ylabel('ACF')

plt.title('ACF Plot of Residuals')

plt.axhspan(-m,m,alpha = .1, color = 'yellow')

plt.tight_layout()

plt.show()



#Forecast acf

acf_drift_test=ACF(drift_fore.values,40)
```

```
x=np.arange(0,41)

m=1.96/np.sqrt(len(y_test))

plt.stem(x,acf_drift_test,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.stem(-1*x,acf_drift_test,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.xlabel('Lags')

plt.ylabel('ACF')

plt.title('ACF Plot of Forecast (Drift)')

plt.axhspan(-m,m,alpha = .1, color = 'yellow')

plt.tight_layout()

plt.show()



#Model performance on train and test data



#MSE

drift_train_mse=mean_squared_error(y_train[2:],train_drift)

print('MSE of Drift on train data:',drift_train_mse)

drift_test_mse=mean_squared_error(y_test,test_drift)

print('MSE of Drift on test data:',drift_test_mse)



#Q-value

q_drift_train=acorr_ljungbox(drift_res,         lags=5,         boxpierce=True,

return_df=True)
```

```
print('Q-value (residual):',q_drift_train)

q_drifttest=acorr_ljungbox(drift_fore,lags=5,boxpierce=True,return_df=True)

print('Q-value (Forecast):\n',q_drifttest)



#Error mean and variance

print('Drift: Mean of residual error is',np.mean(drift_res),'and Forecast error
is',np.mean(drift_fore))

print('Drift: Variance of residual error is',np.var(drift_res),'and Forecast
error is',np.var(drift_fore))



#RMSE

drift_train_rmse=mean_squared_error(y_train[2:],train_drift,squared=False)

print('RMSE of Drift method on train data:',drift_train_rmse)

drift_test_rmse=mean_squared_error(y_test,test_drift,squared=False)

print('RMSE of Drift method on test data:',drift_test_rmse)




#SES

ses=
ets.ExponentialSmoothing(y_train,trend=None,damped_trend=False,seasonal=None)
.fit(smoothing_level=0.5)

train_ses= ses.forecast(steps=len(y_train))
```

```
train_ses=pd.DataFrame(train_ses).set_index(y_train.index)



test_ses= ses.forecast(steps=len(y_test))

test_ses=pd.DataFrame(test_ses).set_index(y_test.index)



#Plot of SES method

plt.plot(y_train.index,y_train,label='Train')

plt.plot(y_test.index,y_test,label='Test')

plt.plot(y_test.index,test_ses[0],label='Predicted')

plt.xlabel('Time')

plt.xticks(rotation='vertical')

plt.ylabel('Temperature')

plt.title('SES method predictions')

plt.legend()

plt.show()



#Plot of test vs predicted

plt.plot(y_test.index,y_test,label='Test')

plt.plot(y_test.index,test_ses[0],label='Forecasted')

plt.xlabel('Time')

plt.xticks(rotation='vertical')
```

```
plt.ylabel('Temperature')

plt.title('SES method forecast')

plt.legend()

plt.show()



#residual and forecast error

ses_res= y_train[2:]-train_ses[0]

ses_fore= y_test-test_ses[0]



#ACF

acf_ses_train=ACF(ses_res.values[2:],40)

x=np.arange(0,41)

m=1.96/np.sqrt(len(y_train))

plt.stem(x,acf_ses_train,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.stem(-1*x,acf_ses_train,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.xlabel('Lags')

plt.ylabel('ACF')

plt.title('ACF Plot of Residuals')

plt.axhspan(-m,m,alpha = .1, color = 'yellow')

plt.tight_layout()

plt.show()
```

```
acf_ses_test=ACF(ses_fore.values,40)

x=np.arange(0,41)

m=1.96/np.sqrt(len(y_test))

plt.stem(x,acf_ses_test,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.stem(-1*x,acf_ses_test,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.xlabel('Lags')

plt.ylabel('ACF')

plt.title('ACF Plot of Forecast (SES)')

plt.axhspan(-m,m,alpha = .1, color = 'yellow')

plt.tight_layout()

plt.show()



#Model performance on train and test data



#MSE

ses_train_mse=mean_squared_error(y_train,train_ses)

print('MSE of SES on train data:',ses_train_mse)

ses_test_mse=mean_squared_error(y_test,test_ses)

print('MSE of SES on test data:',ses_test_mse)
```

```
#Q-value

q_ses_train=acorr_ljungbox(ses_res[2:],        lags=5,        boxpierce=True,
return_df=True)

print('Q-value (residual):',q_ses_train)

q_sestest=acorr_ljungbox(ses_fore,lags=5,boxpierce=True,return_df=True)

print('Q-value (Forecast):\n',q_sestest)



#Error mean and variance

print('SES: Mean of residual error is',np.mean(ses_res),'and Forecast error
is',np.mean(ses_fore))

print('SES: Variance of residual error is',np.var(ses_res),'and Forecast error
is',np.var(ses_fore))



#RMSE

ses_train_rmse=mean_squared_error(y_train,train_ses,squared=False)

print('RMSE of SES method on train data:',ses_train_rmse)

ses_test_rmse=mean_squared_error(y_test,test_ses,squared=False)

print('RMSE of SES method on test data:',ses_test_rmse)



# 12. Multiple Linear Regression

#Prediction on train data

import numpy as np
```

```
from sklearn.linear_model import LinearRegression

import statsmodels.api as sm



X_train_sm = sm.add_constant(x_train)

model = sm.OLS(y_train, X_train_sm).fit()



# Make predictions on the test set

X_test_sm = sm.add_constant(x_test)

y_pred = model.predict(X_test_sm)



pred_train = model.predict(X_train_sm)

#12.b

print("F-test:")

print("F value:",model.fvalue)

print("P-value:",model.f_pvalue)





print(model.summary())



residuals = y_test - y_pred

fore =y_train-pred_train
```

```
plt.plot(y_train.index, y_train, label='Train')

plt.plot(y_test.index, y_test, label='Test')

plt.plot(y_test.index, y_pred, label='Predicted')

plt.legend()

plt.xlabel("Time")

plt.ylabel("Dependent Variable")

plt.title("Train vs. Test vs. Predicted Values")

plt.show()

#residual and forecast error




#ACF of residual

acf_avg_train=ACF(residuals,40)

x=np.arange(0,41)

m=1.96/np.sqrt(len(y_train))

plt.stem(x,acf_avg_train,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.stem(-1*x,acf_avg_train,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.xlabel('Lags')

plt.ylabel('ACF')

plt.title('ACF Plot of Residuals')
```

```
plt.axhspan(-m,m,alpha = .1, color = 'yellow')

plt.tight_layout()

plt.show()




#Model performance on train and test data



#MSE

ML_train_mse=mean_squared_error(y_train,pred_train)

print('MSE of MLR on train data:',ML_train_mse)

ML_test_mse=mean_squared_error(y_test,y_pred)

print('MSE of MLR on test data:',ML_test_mse)



#Q-value

q_ml_train=sm.stats.acorr_ljungbox(residuals, lags=5, return_df=True)

print('Q-value (residual):',q_ml_train)

q_mltest=sm.stats.acorr_ljungbox(residuals,lags=5,return_df=True)

print('Q-value (Forecast):\n',q_mltest)



#Error mean and variance

print('MLR: Mean of residual error is',np.mean(residuals),'and Forecast error

is',np.mean(fore))
```

```
print('MLR: Variance of residual error is',np.var(residuals),'and Forecast
error is',np.var(fore))



#RMSE

ml_train_rmse=mean_squared_error(y_train,pred_train,squared=False)

print('RMSE of MLR method on train data:',ml_train_rmse)

ml_test_rmse=mean_squared_error(y_test,y_pred,squared=False)

print('RMSE of MLR method on test data:',ml_test_rmse)



#13: ARMA models



#Order determination

diff_train,diff_test=train_test_split(diff_df21,test_size=0.2,shuffle=False)




#ACF

ACF_PACF_Plot(diff_df21,40)

acf_gpac=ACF(diff_train['Temp (°C)'].values,40)

x=np.arange(0,41)

m=1.96/np.sqrt(len(diff_df21))

plt.stem(x,acf_gpac,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.stem(-1*x,acf_gpac,linefmt='r-', markerfmt='bo', basefmt='b-')
```

```
plt.xlabel('Lags')

plt.ylabel('ACF')

plt.title('ACF Plot for GPAC')

plt.axhspan(-m,m,alpha = .1, color = 'yellow')

plt.tight_layout()

plt.show()



#GPAC

GPAC(acf_gpac,10,10)



#order ARMA(1,0)



#14: LMA algorithm

na=1

nb=0

order = (na, 0, nb)  # ARIMA order: (p, d, q)

arma_1_0 = sm.tsa.ARIMA(y_train, order=order, trend=None).fit()



#Estimated parameters

for i in range(na):

   print(f"The AR coefficient a{i} is:",arma_1_0.params[i])
```

```
for i in range(nb):

   print(f"The MA coefficient a{i} is:",arma_1_0.params[i + na])

print(arma_1_0.summary())



#initialise values used in function

mu=0.01

delta=10**-6

epsilon=0.001

mu_max=10**10

max_iter=100

SSE,cov,params,var=step3(max_iter,mu,delta,epsilon,mu_max,na,nb,y)

print('The estimated parameter of AR is ',params)



#Prediction on train set

total_length = 18487

train_length = int(0.8 * total_length)

test_length = total_length - train_length



# For training set

arma_1_0_train = arma_1_0.predict(start=0, end=train_length - 1)

arma_1_0_res = y_train - arma_1_0_train
```

```
# For test set

arma_1_0_test = arma_1_0.predict(start=train_length, end=total_length - 1)

arma_1_0_fore = y_test - arma_1_0_test



#residual ACF

acf_arma_1_0_train=ACF(arma_1_0_res,40)

x=np.arange(0,41)

m=1.96/np.sqrt(len(y_train))

plt.stem(x,acf_arma_1_0_train,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.stem(-1*x,acf_arma_1_0_train,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.xlabel('Lags')

plt.ylabel('ACF')

plt.title('ACF Plot of Residuals')

plt.axhspan(-m,m,alpha = .1, color = 'yellow')

plt.tight_layout()

plt.show()



#forecasted acf

acf_arma_1_0_test=ACF(arma_1_0_fore.values,40)

x=np.arange(0,41)
```

```
m=1.96/np.sqrt(len(y_test))

plt.stem(x,acf_arma_1_0_test,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.stem(-1*x,acf_arma_1_0_test,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.xlabel('Lags')

plt.ylabel('ACF')

plt.title('ACF Plot of Forecast (ARMA(1,0))')

plt.axhspan(-m,m,alpha = .1, color = 'yellow')

plt.tight_layout()

plt.show()



#Model performance on train and test data



#MSE

arma10_train_mse=mean_squared_error(y_train,arma_1_0_train)

print('MSE of ARMA(1,0) on train data:',arma10_train_mse)

arma10_test_mse=mean_squared_error(y_test,arma_1_0_test)

print('MSE of ARMA(1,0) on test data:',arma10_test_mse)



#Q-value

q_arma10_train=acorr_ljungbox(arma_1_0_res,      lags=5,      boxpierce=True,
return_df=True)

print('Q-value (residual):',q_arma10_train)
```

```
q_ar10test=acorr_ljungbox(arma_1_0_fore,lags=5,boxpierce=True,return_df=True)

print('Q-value (Forecast):\n',q_ar10test)



#Error mean and variance

print('ARMA(1,0): Mean of residual error is',np.mean(arma_1_0_res),'and

Forecast error is',np.mean(arma_1_0_fore))

print('ARMA(1,0): Variance of residual error is',np.var(arma_1_0_res),'and

Forecast error is',np.var(arma_1_0_fore))



#RMSE

ar10_train_rmse=mean_squared_error(y_train,arma_1_0_train,squared=False)

print('RMSE of ARMA(1,0) method on train data:',ar10_train_rmse)

ar10_test_rmse=mean_squared_error(y_test,arma_1_0_test,squared=False)

print('RMSE of ARMA(1,0) method on test data:',ar10_test_rmse)



#16

#Covariance matrix

print('Covariance matrix\n',arma_1_0.cov_params())



#confidence interval

print('Confidence interval:\n',arma_1_0.conf_int())
```

```
#standard error

print('Standard error:',arma_1_0.bse)



#Display the estimated variance of error

print('The estimated variance of error is',var)



#chitest

lags = 40

Q1 = q_arma10_train

error1 = arma_1_0_test

chi_test(na,nb,lags,Q1,error1)



#POLE CANCELLATION

zero_poles(params,na)



#Plot of ARMA(1,0) method

plt.plot(y_train.index,y_train,label='Train')

plt.plot(y_test.index,y_test,label='Test')

plt.plot(y_test.index,arma_1_0_test,label='Predicted')

plt.xlabel('Time')

plt.xticks(rotation='vertical')
```

```
plt.ylabel('Temperature')

plt.title('ARMA(1,0) method predictions')

plt.legend()

plt.show()



#Plot of test vs forecasted

plt.plot(y_test.index,y_test,label='Test')

plt.plot(y_test.index,arma_1_0_test,label='Forecasted')

plt.xlabel('Time')

plt.xticks(rotation='vertical')

plt.ylabel('Temperature')

plt.title('ARMA(1,0) method forecast')

plt.legend()

plt.show()



#Plot of train vs predicted

plt.plot(y_train.index,y_train,label='Train')

plt.plot(y_train.index,arma_1_0_train,label='Predicted')

plt.xlabel('Time')

plt.xticks(rotation='vertical')

plt.ylabel('Temperature')
```

```
plt.title('ARMA(1,0) method predictions')

plt.legend()

plt.show()



#sarima

#SARIMA(0,0,1)(0,1,1260)

#order = (1, 0, 0)

#seasonal_order = (0, 1, 1260, 1)  # SARIMA(1,0,0)(0,1,1260)




import statsmodels.api as sm

import math

# Assuming your time series data is stored in the variable 'y_train'



sarima = sm.tsa.statespace.SARIMAX(y_train, order=(1, 0, 0), seasonal_order=(0,

1, 0, 21),

                                   enforce_stationarity=False,

enforce_invertibility=False)



results = sarima.fit(disp=0)

print(results.summary())
```

```
#predictions on train data

sarima_train      =      results.get_prediction(start=0,      end=len(y_train),

dynamic=False)

Sarima_pred = sarima_train.predicted_mean

Sarima_res = y_train-Sarima_pred.values[1:]



#forecast

sarima_test=results.predict(start=0, end=(len(y_test)))

sarima_fore=y_test-sarima_test.values[1:]



#ACF



#ACF of rresiduals

acf_sarima_train=ACF(Sarima_res.values,40)

x=np.arange(0,41)

m=1.96/np.sqrt(len(y_train))

plt.stem(x,acf_sarima_train,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.stem(-1*x,acf_sarima_train,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.xlabel('Lags')

plt.ylabel('ACF')

plt.title('ACF Plot of Residuals')

plt.axhspan(-m,m,alpha = .1, color = 'yellow')
```

```
plt.tight_layout()

plt.show()



#acf of forecast

acf_sarima_test=ACF(sarima_fore.values,40)

x=np.arange(0,41)

m=1.96/np.sqrt(len(y_test))

plt.stem(x,acf_sarima_test,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.stem(-1*x,acf_sarima_test,linefmt='r-', markerfmt='bo', basefmt='b-')

plt.xlabel('Lags')

plt.ylabel('ACF')

plt.title('ACF Plot of Forecast (SARMA(1,0,0)X(0,1,0,12))')

plt.axhspan(-m,m,alpha = .1, color = 'yellow')

plt.tight_layout()

plt.show()



#Model performance on train and test data

#MSE

sarima_train_mse=mean_squared_error(y_train,Sarima_pred[1:])

print('MSE of SARIMA on train data:',sarima_train_mse)

sarima_test_mse=mean_squared_error(y_test,sarima_test[1:])
```

```
print('MSE of SARIMA on test data:',sarima_test_mse)



#Q-value

q_sarima_train=acorr_ljungbox(Sarima_res,        lags=5,        boxpierce=True,
return_df=True)

print('Q-value (residual):',q_sarima_train)

q_sarimatest=acorr_ljungbox(sarima_fore,lags=5,boxpierce=True,return_df=True)

print('Q-value (Forecast):\n',q_sarimatest)




#Error mean and variance

print('SARIMA: Mean of residual error is',np.mean(Sarima_res),'and Forecast
error is',np.mean(sarima_fore))

print('SARIMA: Variance of residual error is',np.var(Sarima_res),'and Forecast
error is',np.var(sarima_fore))



#Covariance matrix

print('Covariance matrix\n',results.cov_params())



#standard error

print('Standard error:',results.bse)
```

```
#RMSE

sarima_train_rmse=mean_squared_error(y_train,Sarima_pred[1:],squared=False)

print('RMSE of SARIMA method on train data:',sarima_train_rmse)

sarima_test_rmse=mean_squared_error(y_test,sarima_test[1:],squared=False)

print('RMSE of SARIMA method on test data:',sarima_test_rmse)



#Plot of SARIMA method

plt.plot(y_train.index,y_train,label='Train')

plt.plot(y_test.index,y_test,label='Test')

plt.plot(y_test.index,sarima_test[1:],label='Predicted')

plt.xlabel('Time')

plt.xticks(rotation='vertical')

plt.ylabel('Temperature')

plt.title('SARIMA method predictions')

plt.legend()

plt.show()



#Plot of test vs forecasted

plt.plot(y_test.index,y_test,label='Test')

plt.plot(y_test.index,sarima_test[1:],label='Forecasted')

plt.xlabel('Time')
```

```
plt.xticks(rotation='vertical')

plt.ylabel('Temperature')

plt.title('SARIMA method forecast')

plt.legend()

plt.show()



#Plot of train vs predicted

plt.plot(y_train.index,y_train,label='Train')

plt.plot(y_train.index,Sarima_pred[1:],label='Predicted')

plt.xlabel('Time')

plt.xticks(rotation='vertical')

plt.ylabel('Temperature')

plt.title('SARIMA method predictions')

plt.legend()

plt.show()



#According to mse metrics, Naive was the best model, but if we consider all the

other metrics,



#18

# sarima was more compeitive and most fitted model.

#saarima model is my final model
```

```
#15 forecast function

def  holt_winters_forecast(train_data,  seasonal_type='add',  trend_type='add',
seasonal_periods=21):

   model   =   sm.tsa.ExponentialSmoothing(train_data,   seasonal=seasonal_type,
trend=trend_type,

                                        seasonal_periods=seasonal_periods,
initialization_method='estimated')

   fit_model = model.fit()

   forecast_values = fit_model.forecast()

   return forecast_values



forecast_result = holt_winters_forecast(y_train)

print(forecast_result)




#19: H-step ahead prediction

H = len(y_test)

h_step_prediction = holtw.forecast(steps= H)

print(h_step_prediction)
```

```
#predictions of test data for MLR was performed previously (see at MLR part of
code)

#Let's plot the test vs forecasted values of MLR



plt.plot(y_test.index,y_test,label='Test')

plt.plot(y_test.index, h_step_prediction,label='Forecasted')

plt.xlabel('Time')

plt.xticks(rotation='vertical')

plt.ylabel('Temperature')

plt.title(' H_step prediction')

plt.legend()

plt.show()
```

REFERENCES

- **https://www.seanoe.org/data/00510/62120/**
- **https://www.kaggle.com/code/ashutoshp2002/underwater-temperature-prediction**
- **https://towardsdatascience.com/time-series-forecasting-with-arima-sarima-and-sarimax-ee61099e78f6**
- **https://github.com/imkhoa99/Time-Series-Analysis-and-Weather-Forecast-/blob/master/Time_Series_Project.ipynb**
- **Class lectures, lecture videos**
- **Class homeworks, Labs**