Temperature Time Series Analysis and Modelling

A Final Term Project Report submitted to George Washington University in fulfillment of the requirement for the successful completion of the course,

Time Series Analysis and Modelling

DATS 6313

Submitted by

Aditya Kumar

GWID: G46007094

GWmail: aditya_kumar@gwu.edu

Under the guidance of

Dr. Reza Jafari
Professorial Lecturer in Data Science
CCAS, George Washington University

May 2023

ABSTRACT

Temperature is an essential weather component because of its tremendous impact on humans and the environment. As a result, one of the widely researched parts of global climate change study is temperature forecasting. This work analyses trends and forecasts a temperature change to see the transient variations over time using daily temperature data from January 1, 2010 – August 31, 2022, collected from a weather station of the Max Planck Institute for Biogeochemistry in Jena, Germany. The work begins with an extensive cleaning of the unclean data that was retrieved such as duplicate row removal, appropriate aggregations through down sampling, and appropriate data imputations. The following clean dataset is utilized to perform time series-related exploratory data analysis, such as time series decomposition, stationarity tests such as ADF/KPSS tests, and ACF/PACF plots. Following the EDA, various modelling techniques have been applied such as Holt-Winter, Multiple Linear Regression, SARIMA with its three standard processes of identification, diagnosis, and forecasting, and a Deep Learning LSTM Model was developed. Following, the best-performing model was selected. The seasonal autoregressive integrated moving average (SARIMA) (3, 0, 0) (0, 1, 0)365 model is found as appropriate for the studied temperature data on a daily basis. Further, SARIMA model performance is compared with developed base models used for benchmarking. Finally, the selected model is utilized to make h-step predictions on the testing data through a custom-developed forecast function. The software which has been used is PyCharm. The code is written in Python 3.

Keywords: ARIMA, SARIMA, Daily Average Temperature Data, Jena Weather Dataset, Time Series Forecasting, Deep Learning, LSTM, Holt-Winter

LIST OF TABLES

LIST OF FIGURES

Contents

CHAPTER 1
INTRODUCTION

This chapter will provide an overview of the time series analysis and modeling process and an outline of the report.

1.1 Overview of Time Series Analysis and Modelling

A brief Time Series Analysis and Modelling steps can be understood in the figure below.

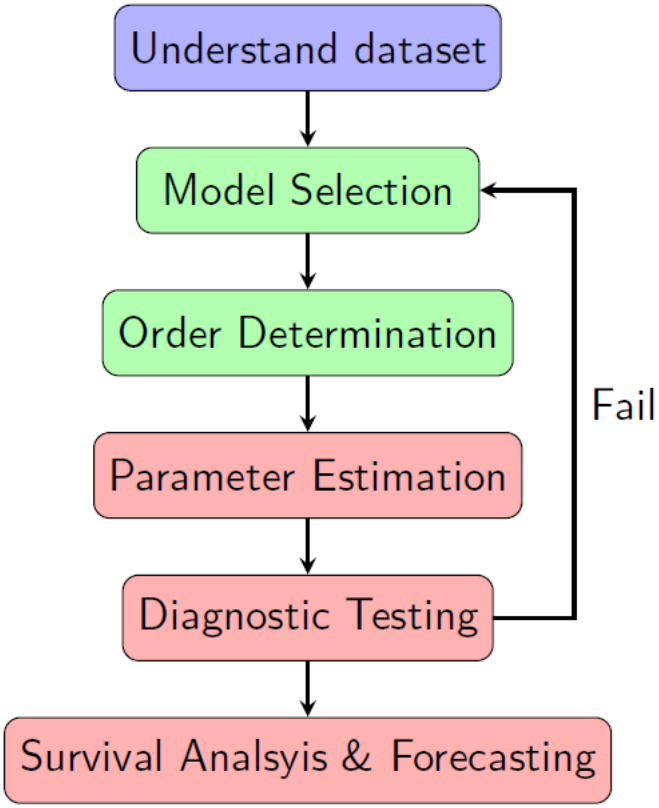


Fig 1.1 Time Series Modelling Steps

Time series analysis and modeling is a statistical approach used to analyze and predict patterns in data that are collected over time. It involves understanding the underlying characteristics of the time series data, identifying trends, seasonality, and other patterns, and using various modeling techniques to make future predictions or forecasts. The steps displayed in the above figure can are summarized below:

1. Understanding the data: This step assumes that the data collection process has been completed successfully and the dataset is of a well-defined frequency / at regular intervals of time. The next important step is dataset cleaning by handling missing values through imputations, removal of duplicate values, etc. Then comes the most important

part of forecasting which is understanding the data. This involves visualizing the data, examining summary statistics, and checking for trends, seasonality, and assessing stationarity. A variety of methods are available to do this such as plotting rolling mean and variance, ADF/KPSS tests, time series decomposition etc. If the data is non-stationary, transformations such as differencing can be applied to make it stationary.

2. Model Selection: The next step is to assess which model to select, whether a linear or non-linear model is suitable based on the characteristics of the data. Common models include Autoregressive Integrated Moving Average (ARIMA), Seasonal ARIMA (SARIMA), Simple Exponential Smoothing (SES), and more advanced models such as Long Short-Term Memory (LSTM) networks. The common strategy is to compare at least two potential models for the dataset.
3. Preliminary Order Detection: This step specifically applies to ARIMA/SARIMA models. Autocorrelations and in turn Partial Auto Correlations are made use of to construct a Generalized Partial Autocorrelation Function (GPAC). A minimum of two orders are selected from this table, the index for a column of constants indicates the Autoregressive (AR) order, and a row of zero adjacent to this column of constant indicates the Moving Average (MA) order.
4. Parameter Estimation: Once the preliminary orders are selected, then a suitable optimization technique such as Levenberg-Marquardt Algorithm can be made use of for parameter estimation by feeding in the identified orders in the function. We fit the selected model (ARIMA/SARIMA) to the time series data and estimate the model parameters using techniques such as maximum likelihood estimation or optimization algorithms.
5. Diagnostic Testing: This is yet another crucial component of Time Series Modelling. The expectation is to see the ARIMA/SARIMA model developed to best fit the underlying data. This best fit is evaluated by performing residual diagnostic testing. This is done by comparing the variance of prediction error versus forecast error indicating whether the model generalizes well, calculating the mean square prediction error, visualizing the scatter plot of the prediction error, and most importantly assessing the autocorrelation plot of the prediction error and performing whiteness tests to understand whether the model captures all the underlying correlation in the time series. Ideally, we want to pick the model with the least number of parameters. If these diagnostic tests fail, we need to repeat the steps from model selection.
6. Survival Analysis and Forecasting: We now use the fitted model to make future predictions or forecasts. This step involves extending the time series beyond the available data to estimate future values.

1.2 Outline of Report

This report is divided into 7 chapters where each section takes a deeper dive into the time series analysis and modeling and finally ends with the best model selection and conclusions from the research. Each part has several sections arranged in a manner for ease of understanding.

Chapter 2 covers the background of the dataset used. Chapter 3 covers the initial data analysis which consists of data cleaning. Chapter 4 covers the exploratory data analysis that is specific to time series data such as stationarity tests and time series decomposition. Chapter 5 covers the pre-processing and modeling, utilizing several modeling techniques. Chapter 6 covers the final model selection and final comparisons between all the built models. Chapter 7 covers the Summary and Conclusions on improvement of the final model.

CHAPTER 2
DATASET

This chapter will discuss important information about the dataset that was used to complete this term project.

2.1 *Dataset Description*

Jena Climate is a weather time-series dataset recorded at the Weather Station of the Max Planck Institute for Biogeochemistry in Jena, Germany. Jena Weather dataset is made up of 19 different numerical features (such as air temperature, atmospheric pressure, humidity, wind direction, and so on) that were recorded every 10 minutes interval, over several years from January 1st, 2010, to August 31st, 2022. The total number of observations in the dataset was 656,957. The dataset is originally sourced from a public forum of the Max Planck Institute for Biogeochemistry through web-scraping and has been uploaded to Kaggle consequently.

Based on the dataset, the project scope was to be defined in terms of target variables and features set through simple initial data analysis. The features are presented below in detail and the description is obtained strategically through independent research.

1. p (m bar): The Pascal SI-derived unit of pressure used to quantify internal pressure. I can take an average here.
2. T (deg C): Temperature in Celsius
3. T pot (K): Temperature in Kelvin
4. T dew (deg C): Temperature in Celsius relative to humidity.
5. VP max (m bar): Saturation vapor pressure
6. V Pact (m bar): Vapor pressure VP def (m bar): Vapor pressure deficit - Average
7. sh (g/kg): Specific humidity
8. H2OC (mmol/mol): Water vapor concentration
9. rho (g/m**3): Airtight
10. wv (m/s): Wind speed
11. max. wv (m/s): Maximum wind speed
12. wd (deg): Wind direction in degrees
13. rain (mm): Rain in millimeters
14. raining (s): Duration of rain in seconds
15. SWDR (W/m*2): Solar Radiation.
16. PAR (mol/m*2/s): The Photo Active Radiation.
17. max PAR (mol/m*2/s): Maximum Photo Active Radiation.
18. T log (deg C): Log of Temperature in Degrees Celsius
19. CO2 (ppm): Carbon Dioxide parts per million

CHAPTER 3
INITIAL DATA ANALYSIS

This chapter will go into detail about the initial data analysis/pre-processing that was performed, including duplicate rows removal, data aggregations, and data imputations.

3.1 *Initial Challenges*

At this stage, the scope of the project was defined, and the variable "T(degC)" was selected as the target variable for the remainder of the project. This section dived into exploring the dataset with a few simple data frame tests such as checking the shape, unique values etc. A few challenges with the dataset I encountered here were:

1. Duplicated Values
2. Missing 10-minute intervals

The duplicated values were found by matching the number of unique date-time stamps with the number of rows. The duplicated rows were removed through simple pandas manipulation. The missing 10-minute intervals were found by creating a program to count the number of unique observations per date. Since the data was at 10-minute intervals, the expectation was to see 144 unique rows per data. This helped to identify missing rows.

3.2 *Data Down Sampling*

A decision to down-sample the data was made at this point to retrieve the data aggregated at the daily level. This was done as a result of encountering missing 10-minute intervals. The features of the dataset were thoroughly researched, and the appropriate aggregation was performed. Most of the features had been averaged up to reflect a single value per day so as to avoid creating bias in the final data. Features such as "rain (mm)" and "raining (s)" were summed up per day.

With the help of an external tool, the expected number of days between the start and end date was obtained and the shape of the aggregated dataset was matched with this value. The final number of observations decreased from 600,000+ to 4562.

3.3 *Data Imputation*

Upon checking for missing values again, it was discovered that two days were missing from the data as shown below.

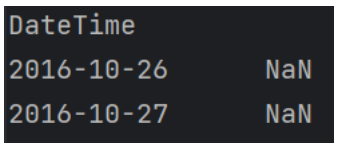


Fig 3.1 Missing dates

Being a time series problem, discarding these missing values was not an option, and therefore, appropriate data imputation techniques were explored.

The first technique was the drift method. The drift method interpolates the first and the last value in the dataset and fills the missing value based on this interpolation line. Prior to performing drift imputation, a train-test split was performed to avoid leakage from the test set. There appeared to be a problem with this drift imputation method. The first and last values of the train set were in the lower range of the target series and therefore the imputed value in the missing dates was very low compared to their neighboring values. The plot below highlights this problem. The target series for the train set and the created drift line for imputation are displayed.

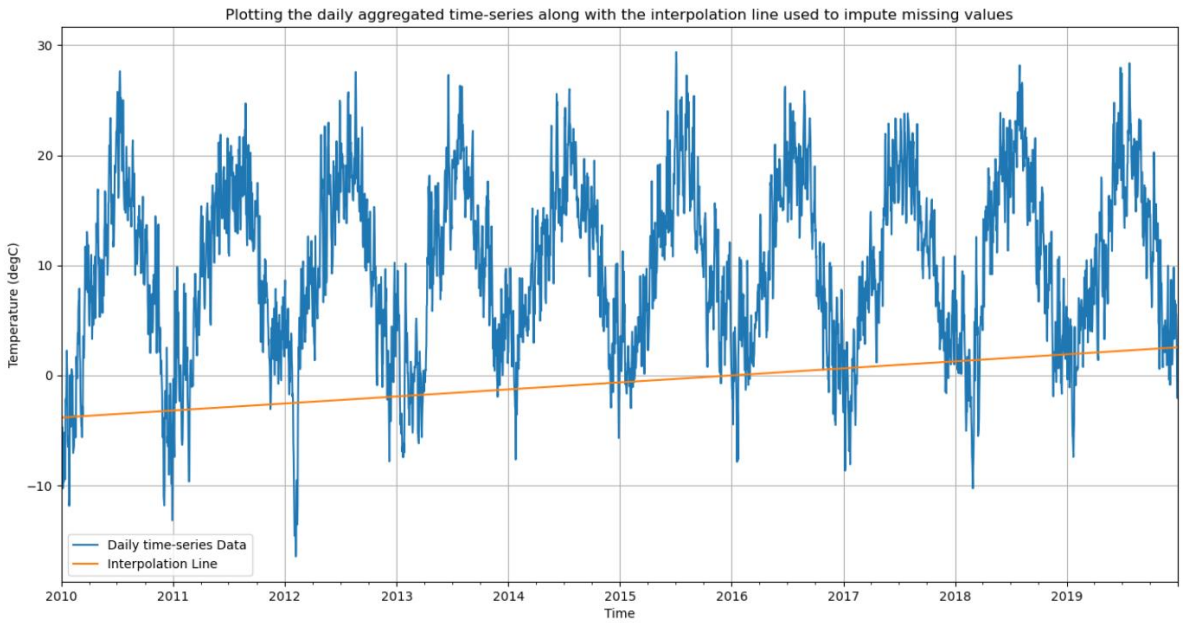


Fig 3.2 Issue with drift imputation

Because of the size of the train set and the end value of the target, an accurate representation of the data through a drift line cannot be retrieved. At this stage, drift imputation was discarded.

The next imputation technique incorporated was a simple linear interpolation through the standard interpolate method of the pandas library. This method works in a similar way as drift except that the start and end values of the drift line are the neighboring values around the missing value.

The target and feature data frames were separated, and data imputation was performed in the following manner. The cleaned versions of the datasets were saved for additional analysis.

CHAPTER 4
EXPLORATORY DATA ANALYSIS

This chapter will go into detail about data exploration and understanding the trends, seasonality, and stationarity of the dataset through appropriate methods.

4.1 *Target Variable Analysis*

The simple time series plot for the target variable is below.

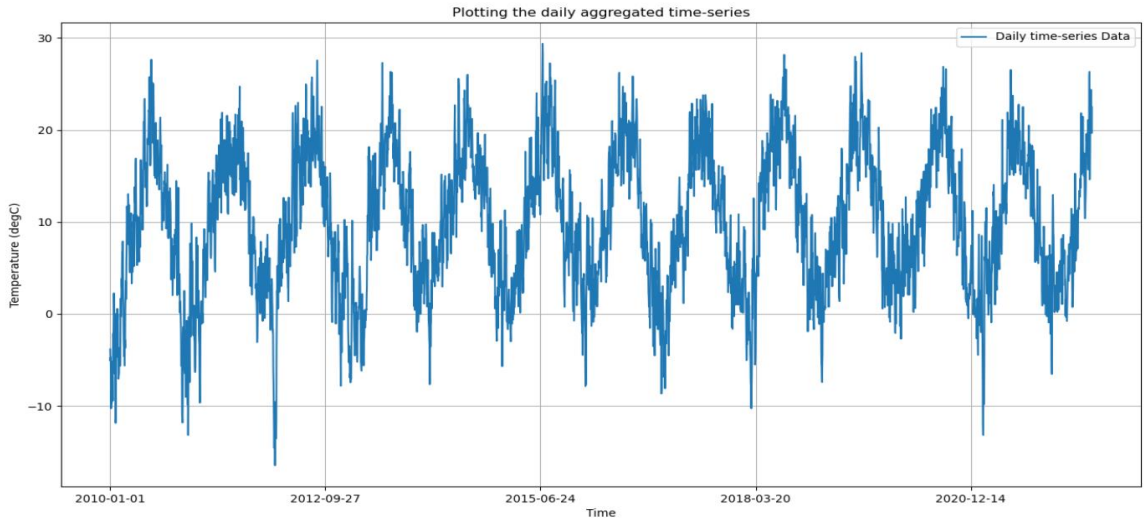


Fig 4.1 Time Series Plot

The time series appears stationary at first glance since there is no apparent trend, only very mildly the temperatures are getting hotter suggesting a very mild upward trend. However, there appears to be extremely high seasonality.

The ACF/PACF plot for the target variable is below. The number of lags (~90) has been calculated using T/50 where T is the total number of observations.

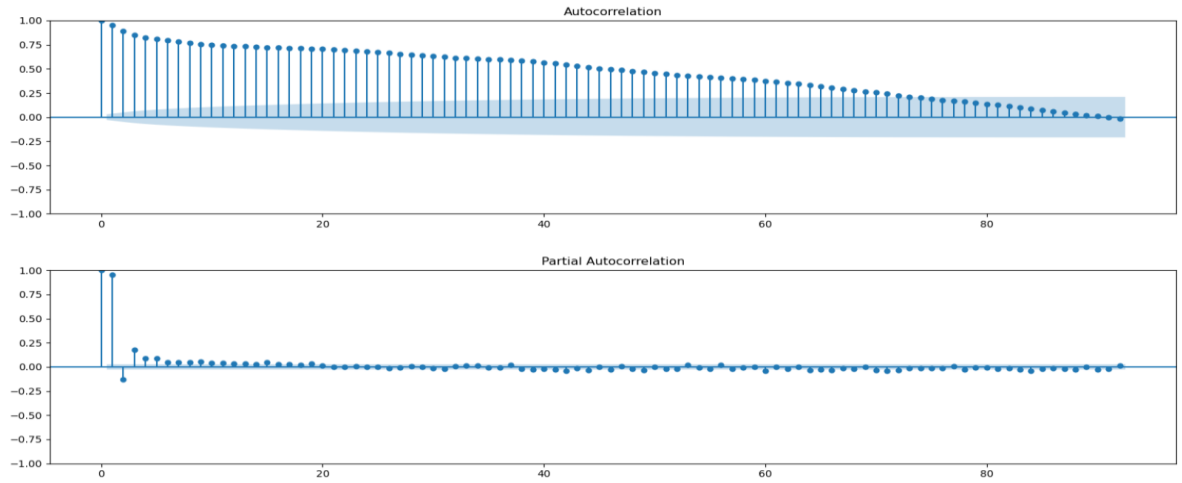


Fig 4.2 ACF/PACF (90 lags) of the target variable

The ACF-PACF plot of the target variable for ~90 lag shows a high correlation between the lagged values judging by the ACF plot. However, judging by the tail-off in ACF and cut-off in PACF after lag = 6, the target variable appears to follow the characteristics of a typical AR process. Figuring out the seasonality of the target variable (Temperature in degree Celsius) is difficult, however, since the data is weather data, the initial assumption was that the seasonality is 365.

To assess the target variable further, the ACF-PACF for an increased number of lags was plotted and can be found below.

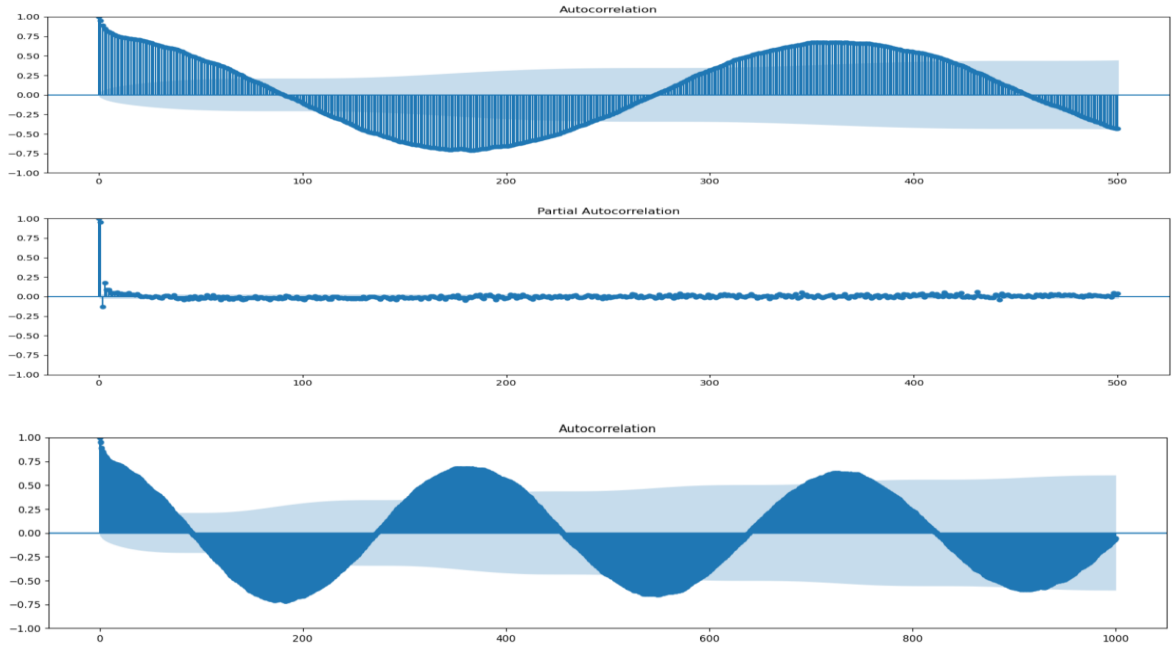


Fig 4.3 ACF/PACF (500/1000 lags) of the target variable

When lags were increased, the seasonality became apparent. The ACF plot seems to have a sinusoidal decay with a seasonality period of 365 (plotted for 500 and 1000 lags).

4.2 *Stationarity Tests*

The target variable's stationarity was assessed by the following tests.

First, the rolling mean and rolling variance were plotted and can be found below.

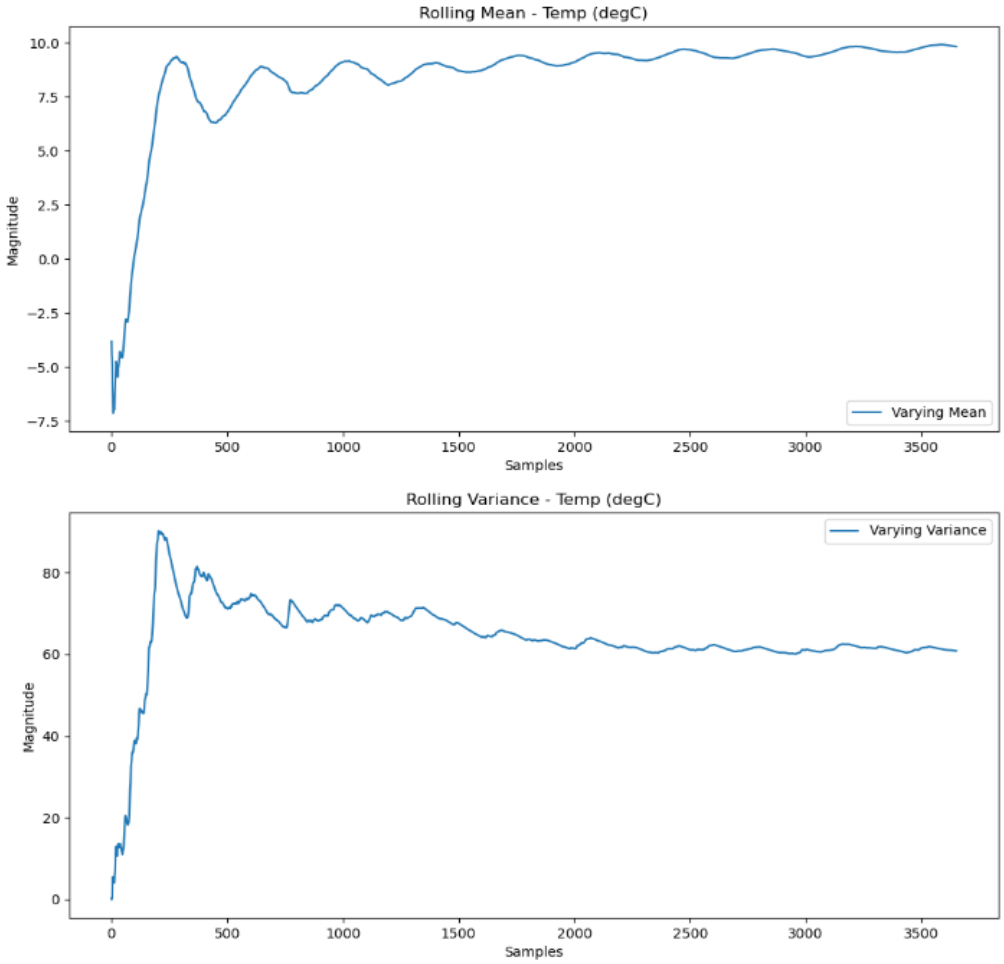


Fig 4.4 Rolling mean and variance

Since the rolling mean and variance of the target variable seem to be stabilizing after all the samples have been included in the calculation, this suggests that the time series is stationary. However, we need to perform a more objective test to ascertain this assumption. Performing the ADF and KPSS tests. The test results can be seen below.

```
ADF Statistic: -4.41
p-value: 0.00
Critical Values:
    1%: -3.43
    5%: -2.86
    10%: -2.57
```

```
Results of KPSS Test:
Test Statistic            0.13
p-value                   0.10
Lags Used                40.00
Critical Value (10%)      0.35
Critical Value (5%)       0.46
Critical Value (2.5%)     0.57
Critical Value (1%)       0.74
```

Fig 4.5 ADF/KPSS test

Based on the small p-value of the ADF statistic, for which the hypothesis is that the process is not stationary, we can reject this hypothesis indicating the process is stationary. Similarly, based on the large p-value of the KPSS statistic, for which the hypothesis is that the is stationary, we are unable to reject this hypothesis indicating that the process is stationary.

Although the ACF of the target variable indicates a very slow decay towards zero, which is a typical characteristic of a non-stationary process, this is due to the presence of high correlations in the process. As a result of all stationarity tests, we can say with a lot of confidence that the target time series is stationary, therefore, no transformations will be required for making the process stationarity.

4.3 *Time Series Decomposition*

The Seasonal-Trend decomposition using LOESS (STL) from the standard library has been used to decompose the target time series. Since the variance of the target series does not vary in magnitude and tends to stabilize, an additive decomposition has been employed as opposed to a multiplicative decomposition. With an additional parameter of seasonality as 365, the time series decomposition can be seen below.

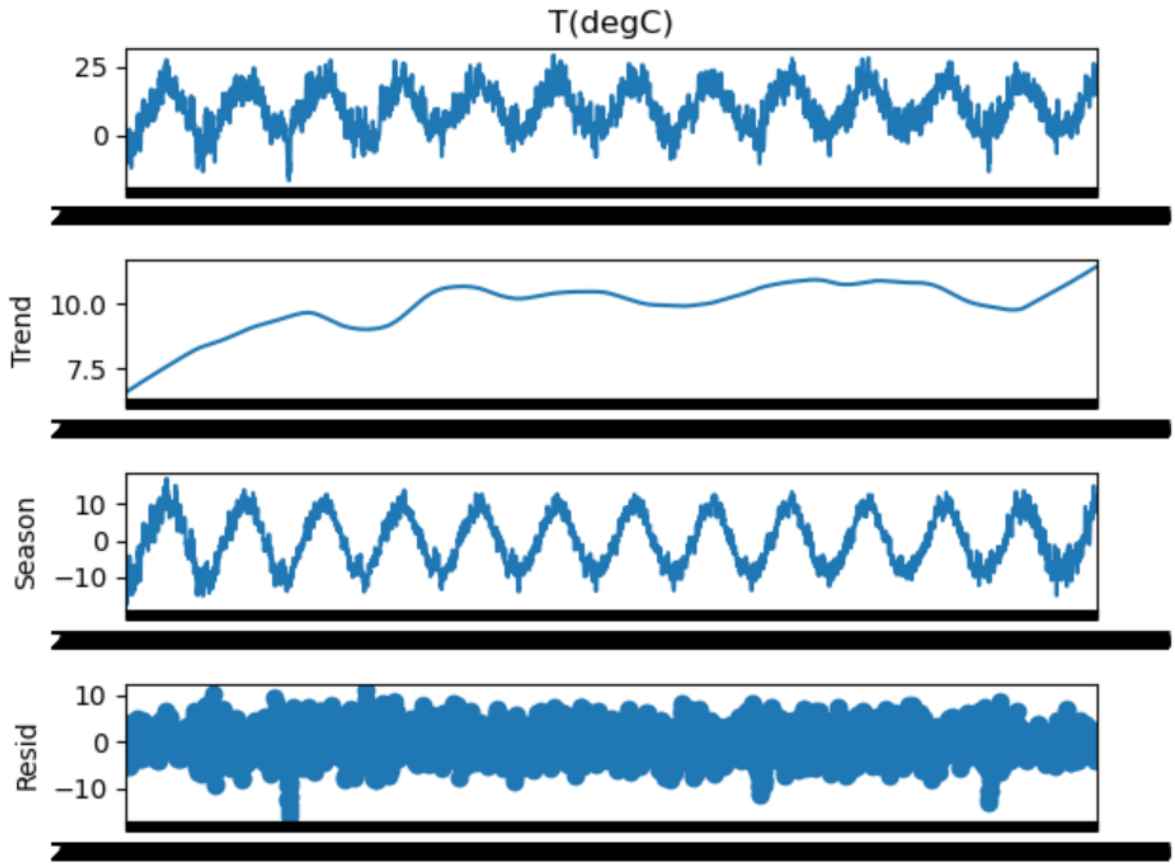


Fig 4.6 Time Series Decomposition

The pattern here is in accordance with the previous observation. The seasonal component appears to be high whereas the trend component is almost flat, with only a very mild increase throughout the years.
Using this decomposition, the de-trended time series plot can be seen below.

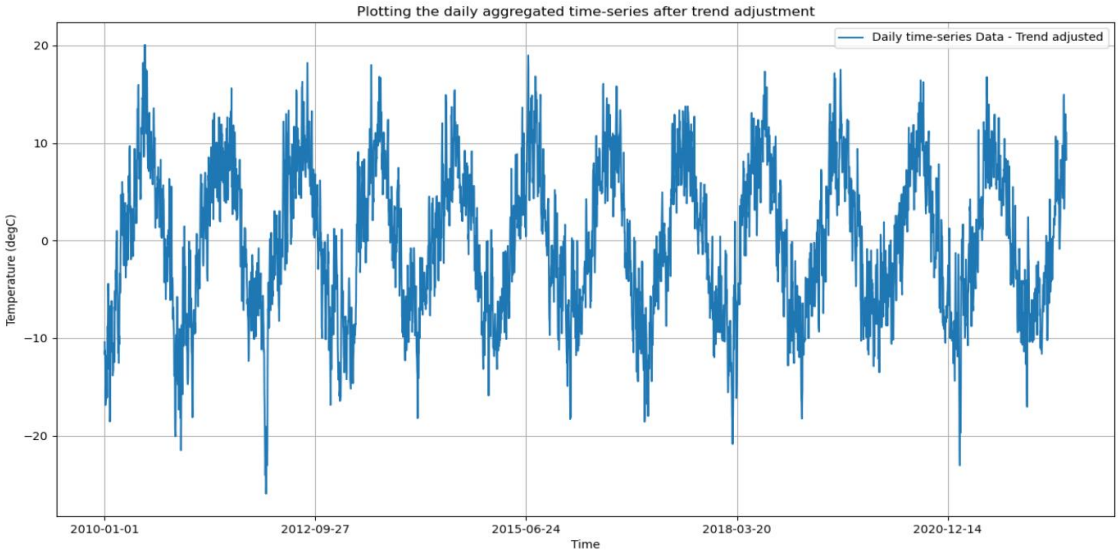


Fig 4.7 De-trended time series plot

As noted above, the de-trended plot appears highly seasonal and much like the original time series plot, indicating that seasonality is the only major component in the data. The seasonally adjusted time series plot is below.

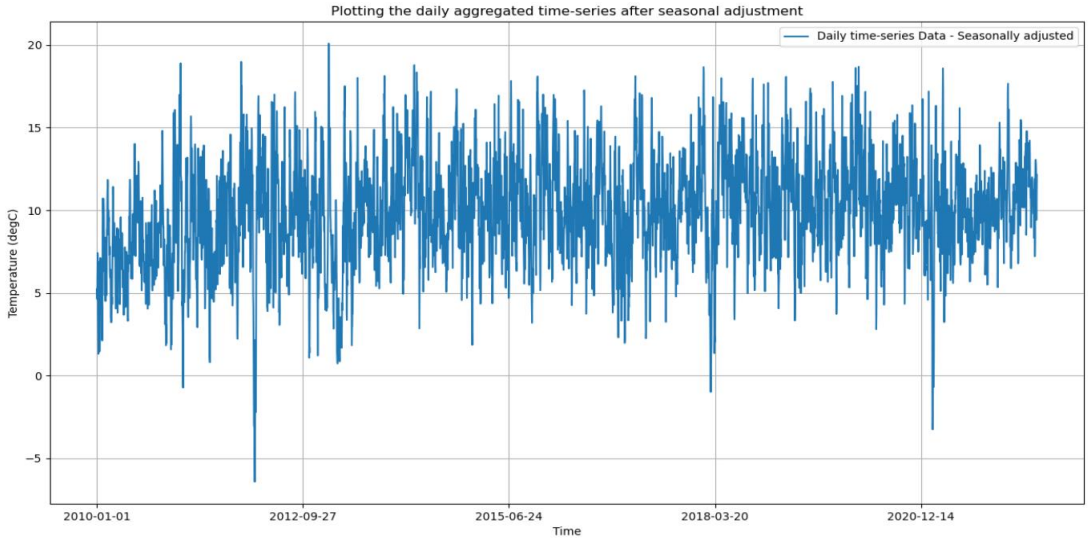


Fig 4.8 Seasonally adjusted time series plot

There appears to be no apparent trend in this plot either. After the seasonality has been removed, the time series plot looks much like noise indicating that all that is left over is residuals of the original data.

To quantify these observations, the following equations have been made use of to calculate the actual Strength of Trend and Strength of Seasonality in the dataset respectively.

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

$$F_S = \max\{0, 1 - \frac{Var(R_t)}{Var(S_t + R_t)}\}$$

Fig 4.9 SOT and SOS Formula

These were the results obtained from the calculations.

```
The Strength of trend in the dataset is: 9.6%
The Strength of seasonality in the dataset is: 84.1%
```

Fig 4.10 SOT and SOS results

These results further validated the visual observations that were made. The dataset was high in seasonality and low in trend.

4.4 *Correlation Matrix*

Further, the correlation matrix with all the features and target set was created and assessed. The result of this correlation matrix is below.

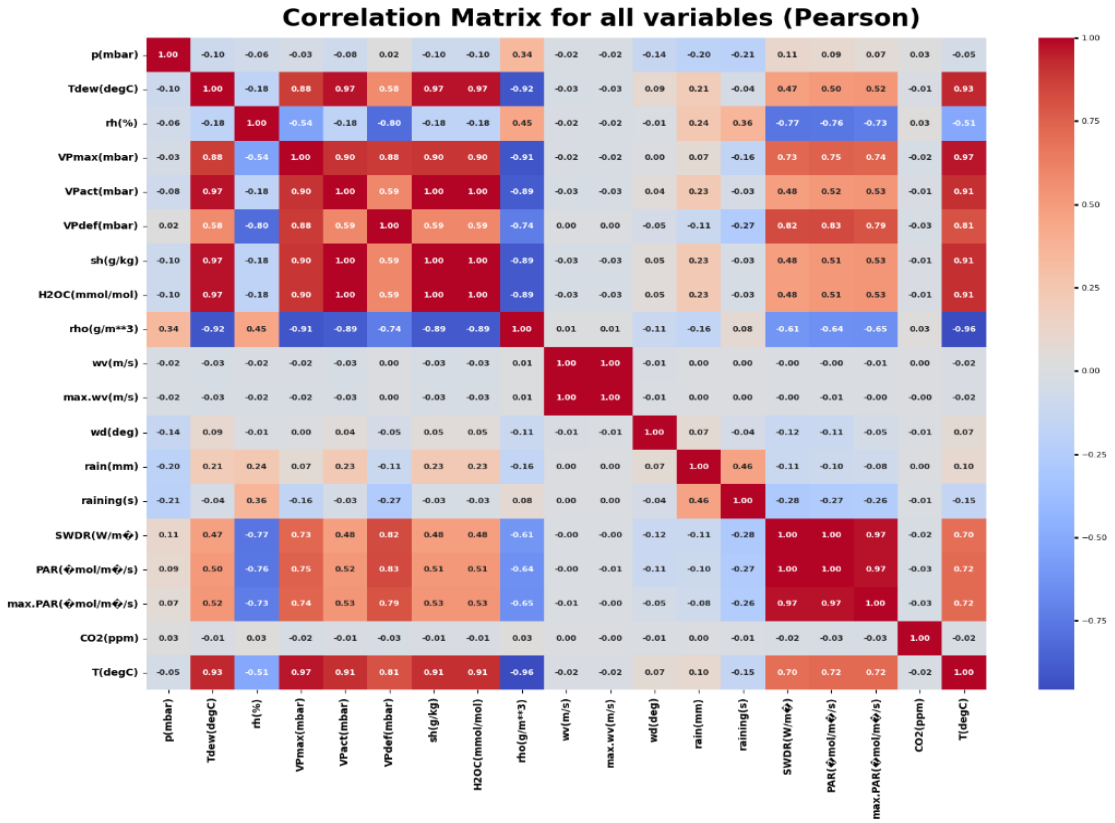
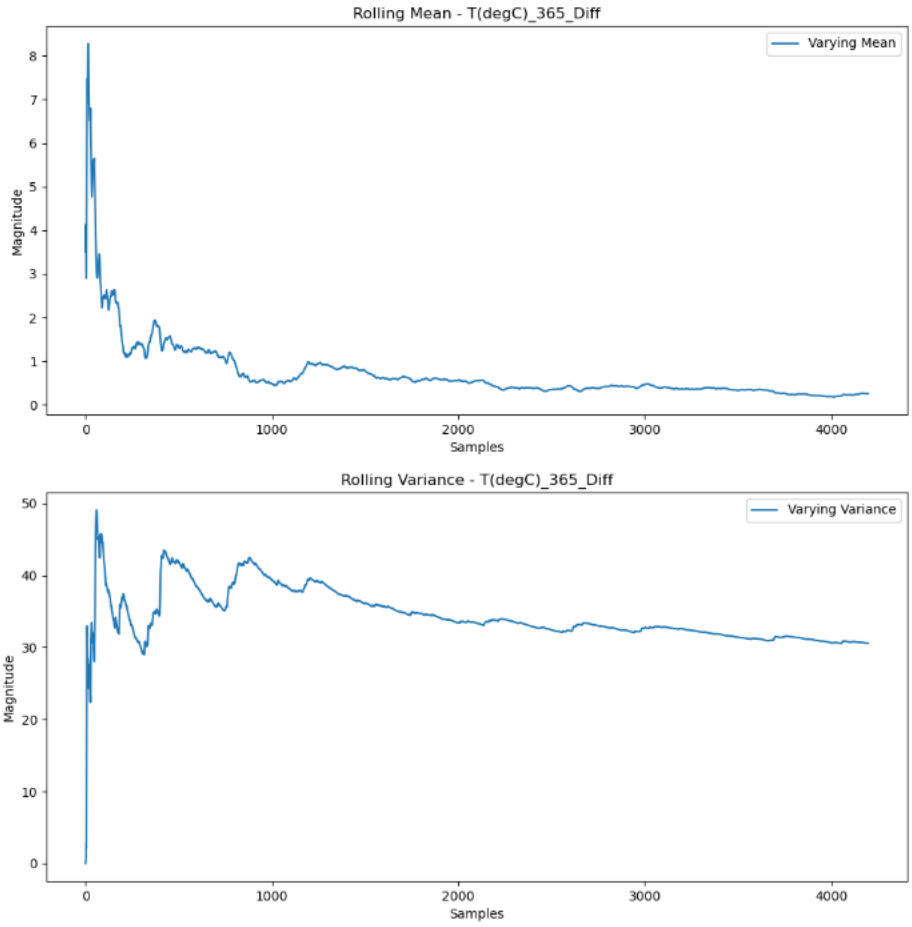


Table 4.1 Correlation Matrix

The presence of a very strong correlation between features, as well as between features and the target variables was observed. However, this high correlation doesn't say anything about the partial correlations between the features and the target variable. Therefore, removing features based on this will be an inaccurate step for modeling.

4.5 *Seasonal Differencing*

Although, it was determined that no differencing was required for making the target variable stationary. However, due to the presence of extremely high correlations in the ACF for increased lag arising due to the high seasonality in the dataset that was reinforced by the Strength of Seasonality test, the decision to seasonally difference the dataset was made. This is because high correlations in the time series can make it difficult to build accurate ARIMA models because the long-term dependencies in the time series are difficult to capture using only a few lags. Consequently, the focus was on investigating the SARIMA model later. There is a trade-off as the total number of observations is lost due to a high seasonal differencing of 365. This section serves as a test to see whether the seasonally differenced dataset was stationary or not, to proceed with SARIMA modeling. After performing one order seasonal differencing of 365, below are the results of stationarity tests.



```
ADF Statistic: -18.29
p-value: 0.00
Critical Values:
    1%: -3.43
    5%: -2.86
    10%: -2.57
```

```
Results of KPSS Test:
Test Statistic            0.22
p-value                   0.10
Lags Used                32.00
Critical Value (10%)      0.35
Critical Value (5%)       0.46
Critical Value (2.5%)     0.57
Critical Value (1%)       0.74
```

Fig 4.11 Stationarity tests for differenced data

The process still remains stationary after seasonal differencing judging by the above tests, in fact, it becomes more stationary based on ADF test statistics. The rolling mean and variance tend to stabilize. KPSS test indicates stationarity as well.

Plotting the ACF/PACF plot of the differenced time series.

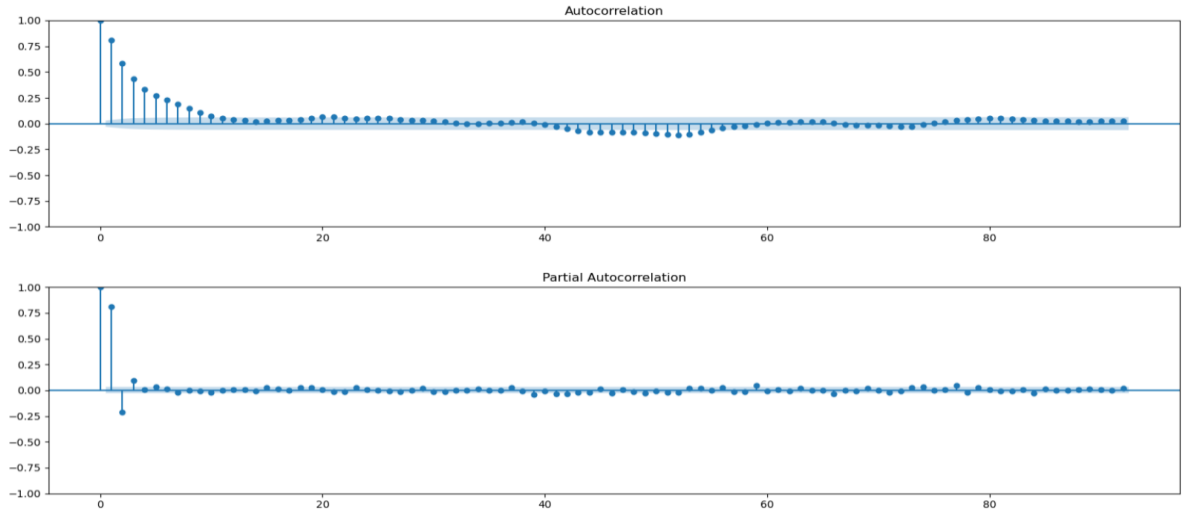


Fig 4.12 ACF/PACF of differenced time series

ACF-PACF plot displays a much cleaner trend with decay in the ACF plot and cut-off in the PACF plot indicating an AR process after differencing. This validates the assumption that the seasonality is 365 true.

CHAPTER 5
PRE-PROCESSING, MODELING, AND DIAGNOSTIC TESTS

This chapter dives into detail on the various modeling techniques that were used to fit the time series. It includes the Holt-Winter model, the Multiple Linear Regression Model, the SARIMA model, the base models (Average, Naïve, Drift, and Simple Exponential Smoothing Methods), and the LSTM Deep Learning network.

A train–test split has been performed at this stage in the analysis. All the training features will be used for developing models and the performance of the models will be tested against the test sets. Further, since dealing with time-series modeling, additional analysis of residual diagnostic testing has been performed for each model to evaluate the fit of the model on the data.

5.1 *Holt-Winter Model*

The first model that was considered was the Holt-Winter model. An additive seasonal and trend component was considered due to the analysis drawn in the Exploratory Data Analysis section. A seasonality of 365 as the argument was additionally added due to the high seasonal behavior of the data and frequency indicating daily values were used. Further, two different variants of the Holt-Winter model were developed, one that considers damping on the trend and another that considers no damping on the trend. Since there isn't much trend in the data, the initial assumption was that these variants will not produce results significantly different. Non-differenced target variable was used since seasonal differencing is only applicable to the SARIMA model.

After the model was built on the train set, a forecast was obtained for the length of the test set. These forecasted values were plotted against the test set to visually evaluate its performance. Below are the results of forecast.

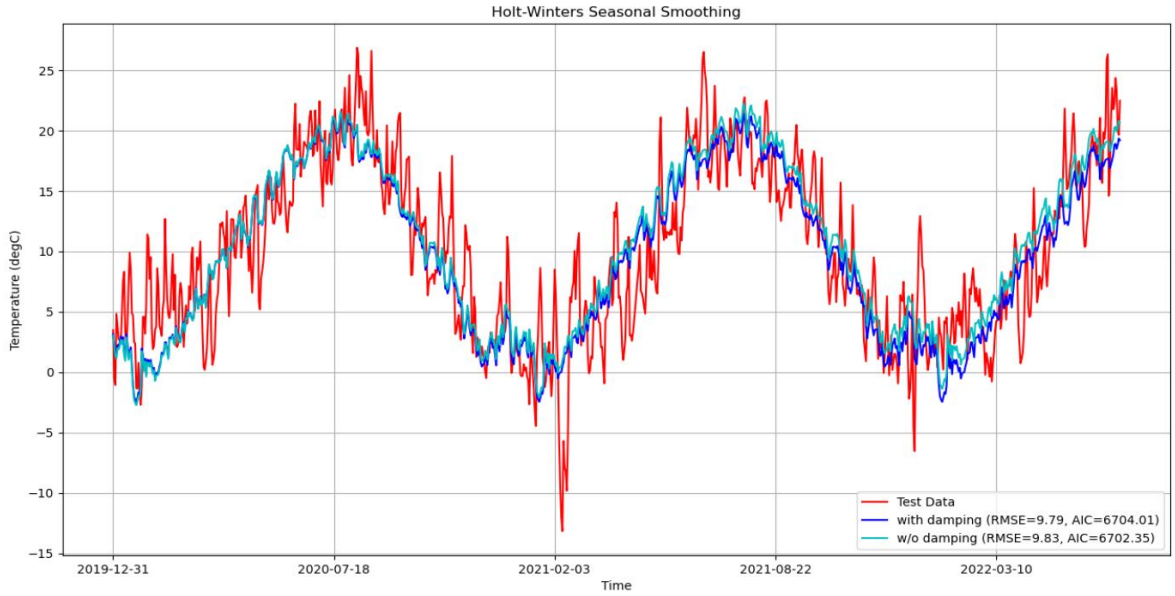


Fig 5.1 Holt-Winter Model Forecast

From a visual standpoint, the results of the Holt-Winter model are only partially satisfactory. The forecasted values are following the actual temperature values of the test set but are not completely able to cover the entire range of these values.

For a more quantified result on the performance of the Holt-Winter model, the following metrics were evaluated: Root Mean Square Error (RMSE) and Mean Absolute Error (MAE). Additionally, AIC and BIC values have been evaluated to assess the performance of the two variants of the model. All results can be seen below.

```
Root mean square error for Holt-Winter with damping is: 9.793
AIC value of this model is: 6704.005
BIC value of this model is: 8991.166
With damping the Mean Absolute Error is: 7.959 and MAPE is: 3.584
```

```
Root mean square error for Holt-Winter without damping is: 9.825
AIC value of this model is: 6702.349
BIC value of this model is: 8991.166
Without damping the Mean Absolute Error: 7.985 and MAPE is: 3.776
```

Fig 5.2 Holt-Winter Forecast Results

Damping doesn't affect the performance drastically as there is not a high trend component. Furthermore, it remains uncertain as to whether damping positively affects the model or not, as the MSE value seems to go up, but the AIC value goes down upon introducing damping.

Since both the variants perform similarly, only of them is used for the final model selection and following steps. The model that is considered is the one without damping. The next step of evaluating the model performance was the residual diagnostic. The residuals of the model were

obtained and the ACF of residuals was plotted. The expectation from a good model is to capture all correlations of the target series and therefore, the ACF plot should have an impulse response. The ACF of the residuals is plotted below.

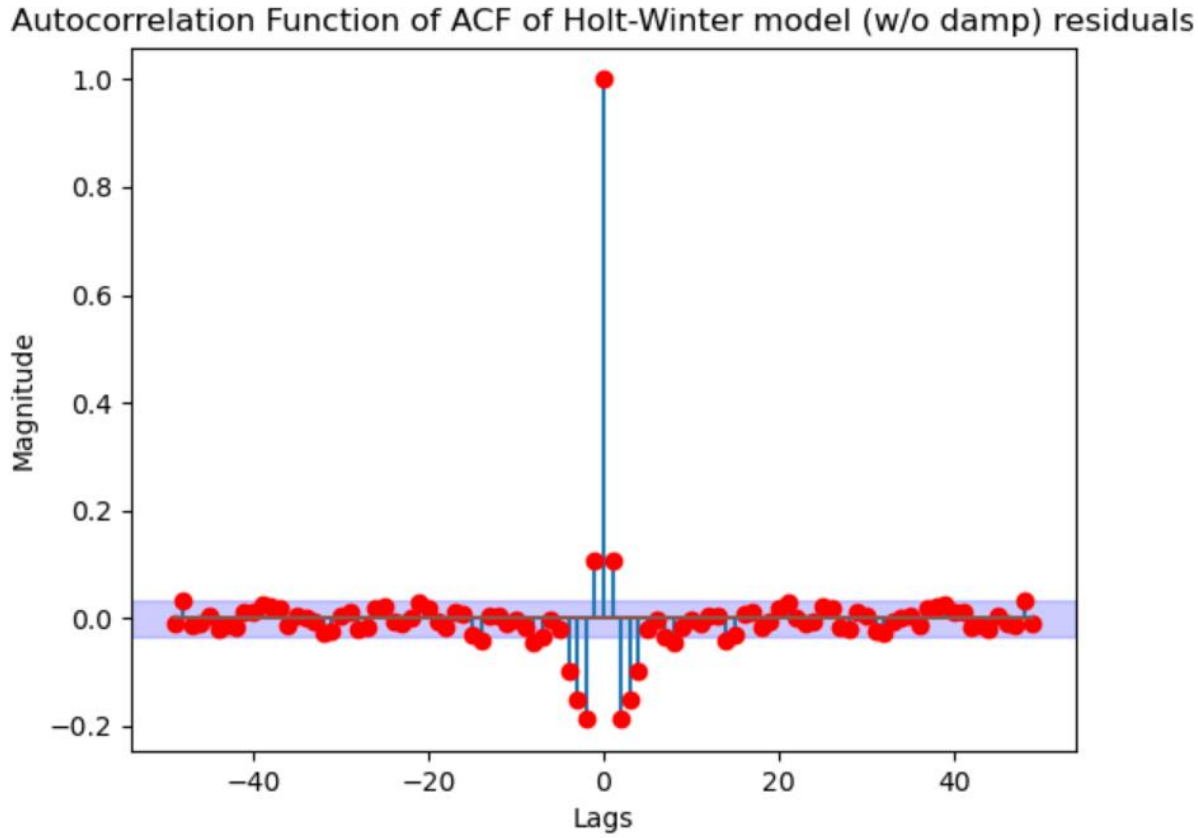


Fig 5.3 Residual Diagnostic for Holt-Winter

The ACF plot does not resemble an impulse, indicating that all the correlations in the train data have not been captured by the Holt-Winter model. Damping does not enhance or degrade the performance of the fitted models as the ACF plot remains identical for the other variant. Non-white residuals indicate that this particular model is not the best fit for the data.

To provide a more objective opinion on the whiteness of these residuals, the Ljung-Box test was performed from the Python package. The results of the Ljung-Box test are below.

	lb_stat	lb_pvalue
365	856.8976	0.0000

Fig 5.4 Residual Diagnostic for Holt-Winter through Ljung-Box test

The hypothesis for the Ljung-Box test states that the process is independent and identically distributed; that the process is white noise. The p-value of 0.000 indicates that we can reject this hypothesis, indicating that the residuals are not white. This reinforces our visual analysis.

5.2 *Multiple Linear Regression Model*

The next model that was built was a Multiple Linear Regression Model (MLR). This model utilizes the additional exogenous feature set that is present in the dataset.

Before the final MLR model is utilized, two additional steps have been performed:

1. Feature Selection
2. Feature Elimination

Feature selection aims to extract only those features from the set that do not possess multi-collinearity. Since MLR models are very sensitive to multi-collinearity, this step becomes very crucial. Three methods have been adopted to investigate and remove multi-collinear features from the dataset:

1. Singular Value Decomposition (SVD)
2. Eigensystem Condition Number
3. Variance Inflation Factor test (VIF)

While the SVD and condition number are useful methods in assessing the presence of multi-collinearity, the VIF tests actually help in identifying the feature that causes multi-collinearity and subsequently help in removing that feature.

Prior to performing these tests, the feature set needed to be standard scaled so as to avoid bias in the results of these tests since the range across features was drastically varying. The target is not scaled with the assumption that the parameter will be large enough to compensate for this. After performing standard scaling, the SVD test was performed. These were the results.

```
SingularValues =  [3.01868782e+04 9.68687457e+03 7.30039526e+03 4.13408588e+03
 4.06357878e+03 3.60646953e+03 2.98521006e+03 1.77486900e+03
 1.30327469e+03 3.73741067e+02 1.99232557e+02 9.57614164e+01
 5.50777613e+00 1.47884518e+00 5.30243616e-01 1.11353071e-01
 7.99469121e-04 9.35787380e-06]
```

Fig 5.5 SVD Analysis

Singular Value Decomposition is a popular technique for dimensionality reduction. This is a linear algebra technique to create a projection of a sparse dataset prior to fitting a model. Any singular values close to zero mean that one or more features are correlated. Judging by this criterion, we see the presence of values close to zero in the singular values matrix above indicating multi-collinearity.

Subsequently, the eigensystem condition number test was performed. The results are below.

```
The Condition Number of the feature space is:  56796.357
```

Fig 5.6 Condition Number

According to this eigensystem analysis, any condition number value above 1000 indicates a high degree of multi-collinearity. Based on this, the presence of multi-collinearity is confirmed in the dataset.

The next test that was used was the Variance Inflation factor. The Variance Inflation Factor was used recursively with the following methodology:

1. Run VIF
2. Identify the feature with the highest VIF value.
3. Drop this feature.
4. Run VIF again.
5. Repeat until all VIF values of the features drop below 5.

The VIF method assesses multi-collinearity by regressing the features one by one based on the other features and determining explainability of this feature. It provides a VIF score against each feature in doing so. A VIF value above 10 is considered highly explainable by the other features, and hence not needed. A strict VIF cut-off of 5 was chosen to make the MLR model more robust. The above process was repeated for all the feature sets until the VIF value went below 5. After doing so, the final number of feature sets was 9. These features and their corresponding VIF values can be seen below.

	VIF Factor	features
0	1.0880	p(mbar)
1	1.7642	Tdew(degC)
2	2.7435	rh(%)
3	1.0023	wv(m/s)
4	1.0630	wd(deg)
5	1.4460	rain(mm)
6	1.4307	raining(s)
7	3.4924	max.PAR(µmol/m²/s)
8	1.0026	CO2(ppm)

Fig 5.7 Final VIF values

After multi-collinearity was removed from the feature set, the next step to be performed was feature elimination. This process aimed at fitting the MLR model with the above features and judging by the significance values of the estimated parameters for the features, a feature was selected to be dropped. An MLR was fit again, and the model performance was evaluated. If dropping the selected feature, enhanced or didn't cause any change in model performance, the new model was retained. The metrics that were utilized to compare the model performance were: Adjusted R-Squared Value, AIC, and BIC. This way, only the least number of features were made use of to build the simplest and best MLR model. The following steps are summarized below:

1. Fit an OLS model with 9 features.
2. Obtain the model summary.
3. Identify feature coefficients with high P-value (indicating insignificance) according to t-tests.
4. Drop the feature and fit MLR again.
5. Compared adjusted-R squared, AIC, and BIC with the previous model.
6. Make a decision based on performance.
7. Repeat until all parameters are significant.

After following these steps, the final features that were retained were 6. The model summary for the final model is provided below.

Dep. Variable:	T(degC)	R-squared:	0.997
Model:	OLS	Adj. R-squared:	0.997
Method:	Least Squares	F-statistic:	2.363e+05
Date:	Tue, 09 May 2023	Prob (F-statistic):	0.00
Time:	23:02:45	Log-Likelihood:	-1788.5
No. Observations:	3651	AIC:	3591.
Df Residuals:	3644	BIC:	3634.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	9.8266	0.007	1502.049	0.000	9.814	9.839
p(mbar)	0.0341	0.007	5.024	0.000	0.021	0.047
Tdew(degC)	6.7207	0.008	794.798	0.000	6.704	6.737
rh(%)	-2.5660	0.011	-238.939	0.000	-2.587	-2.545
wd(deg)	-0.0988	0.007	-14.690	0.000	-0.112	-0.086
raining(s)	0.0424	0.007	5.923	0.000	0.028	0.056
max.PAR(µmol/m²/s)	0.1045	0.012	8.549	0.000	0.081	0.128

Fig 5.8 MLR Model Summary

A very good Adjusted R-squared value of 99.7% is obtained based on this model. Using the features in the training data, the model is able to explain 99.7% of the variability in the target. All the estimated parameters are also within the significant range. Additionally, the F-test statistic indicates that the null hypothesis can be rejected which states that this model and an intercept-only model perform the same indicating, this model with the given features performs much better.

With the selected MLR model, a forecast is made for the test set, which has to be scaled separately, to gauge the model performance on test data. The predicted values are plotted against the test set which can be found below.

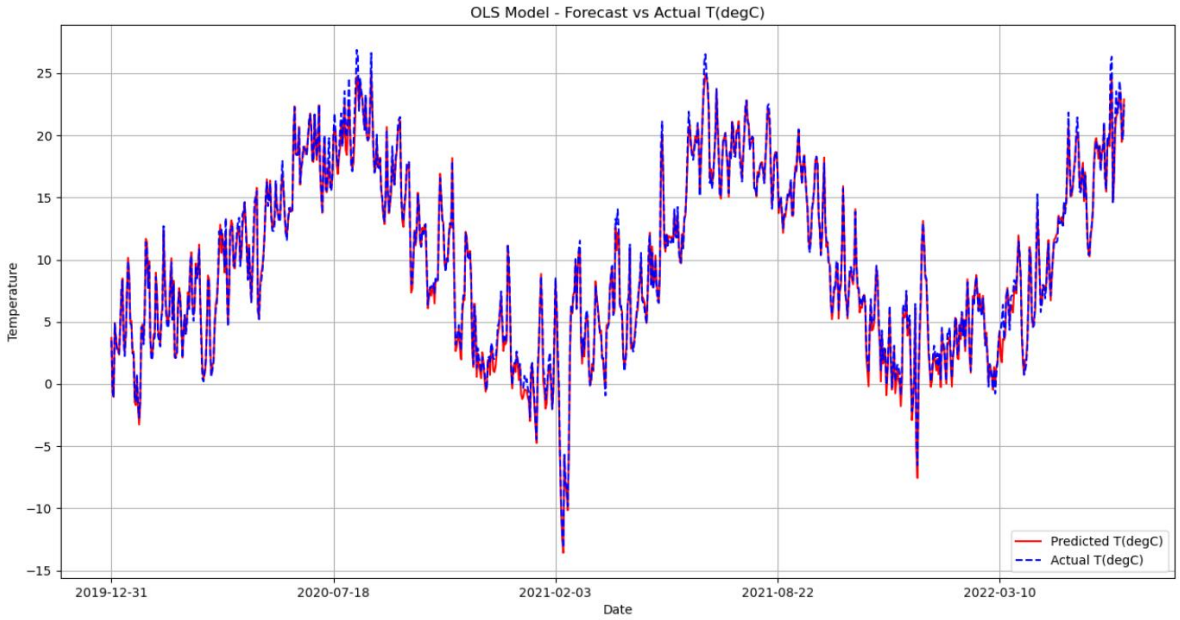


Fig 5.9 MLR Model Forecast

Visually, the test set predictions seem very good and match the test set very accurately. To quantify these results, the following metrics were used to assess the model performance on test data: Root Mean Square Error (RMSE), Mean Absolute Percentage Error, and Mean Absolute Error (MAE). The results are below.

```
The RMSE of this model is: 0.511
The MAE of this model is: 0.361
The MAPE os this model is: 0.18
```

Fig 5.10 MLR Model Results

The model performance seems to be extraordinary on the test set indicating that this model performs well. However, an integral part of time-series analysis is residual diagnostic. The residual diagnostic tests are provided below.

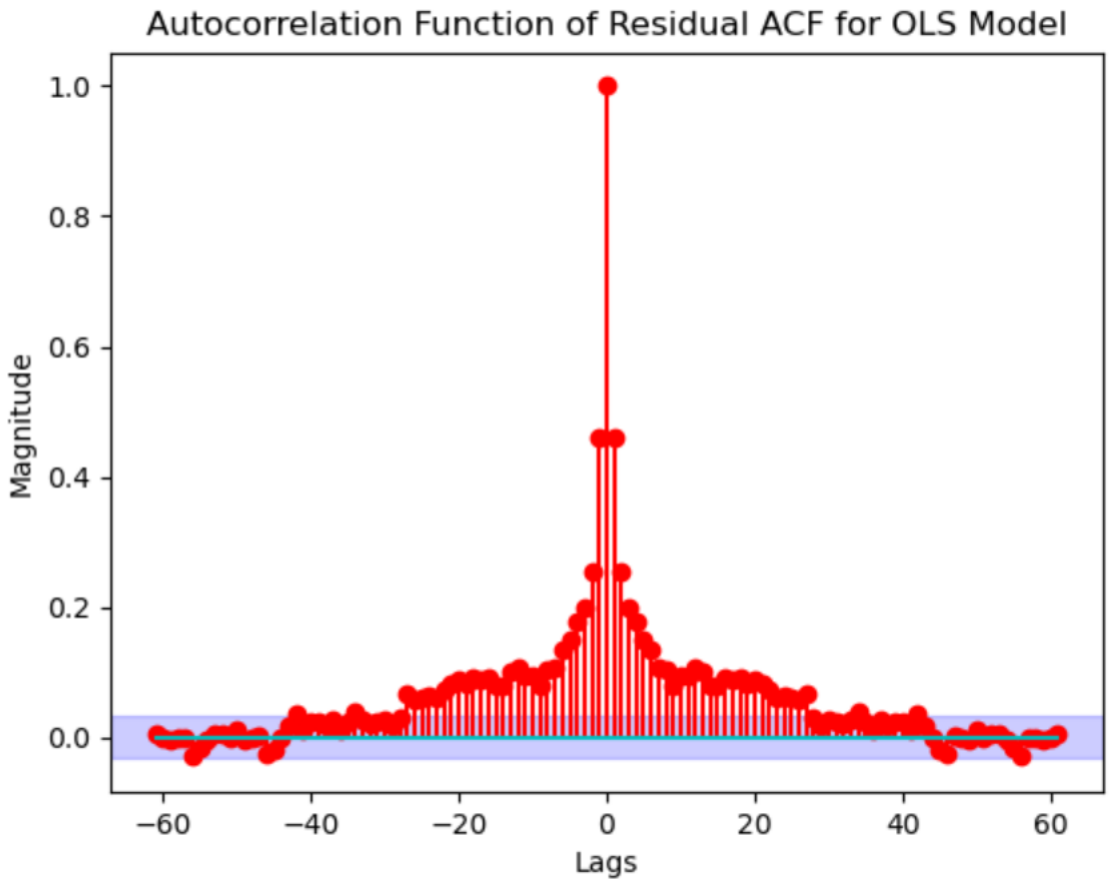


Fig 5.11 MLR Residual ACF

The ACF plot does not resemble an impulse, indicating that all the correlations in the train data have not been captured by the MLR model. Non-white residuals indicate that this particular model is not the best fit for the data. Although the model seems to perform really well on the test set, it fails during the residual diagnostic test.

To provide a more objective opinion on the whiteness of these residuals, the Box-Pierce test was performed which is a variant of the Ljung-Box test. The results of the Box-Pierce test are below.

```
Q is 2023.441312467781 and chi critical is 79.84333812225145
The residual is NOT white
```

Fig 5.12 MLR Residual Diagnostic

The hypothesis for the Box-Pierce test states that the process is independent and identically distributed; that the process is white noise. The high test statistic indicates that there were correlations present in the residual, indicating that the residuals are not white. This reinforces our visual analysis. Furthermore, another aspect of the residual analysis is to obtain a residual with zero mean and constant variance. The results for the MLR model are shown below.

```
The mean of the residuals is: 1.092585956581262e-15
The variance of the residuals is: 0.15596038638769524
```

Fig 5.13 MLR Residual Mean and Variance

Although the residual appears to have a zero mean and constant variance, there are high correlations in them, rendering the MLR model not a good fit for the data for this time series problem.

5.3 *Base Models for Benchmarking*

The first base model was built using the Average Method. Below is the plot for the forecasted values vs the test set for this model.

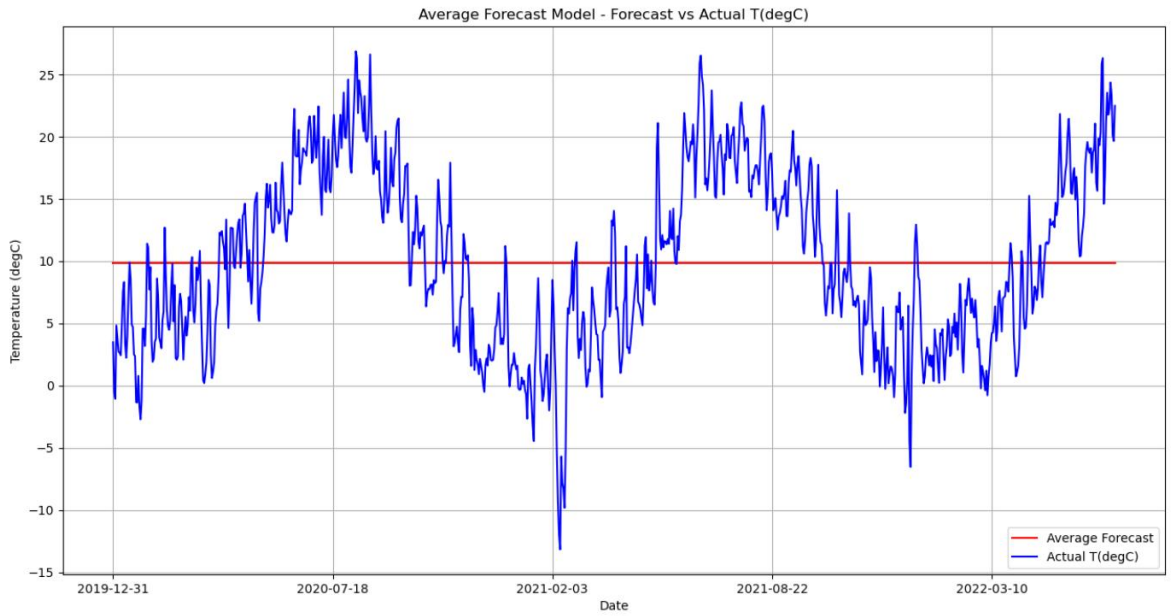


Fig 5.14 Average Model Forecast

To quantify the performance of this model. The metrics used above were calculated. Below are the results.

```
The RMSE of Average model is: 7.087
The MAE of this model is: 6.011
The MAPE of this model is: 10.967
```

Fig 5.15 Average Model Result

The next base model was using the Naïve method. Below is the plot for the forecasted values vs the test set for this model.

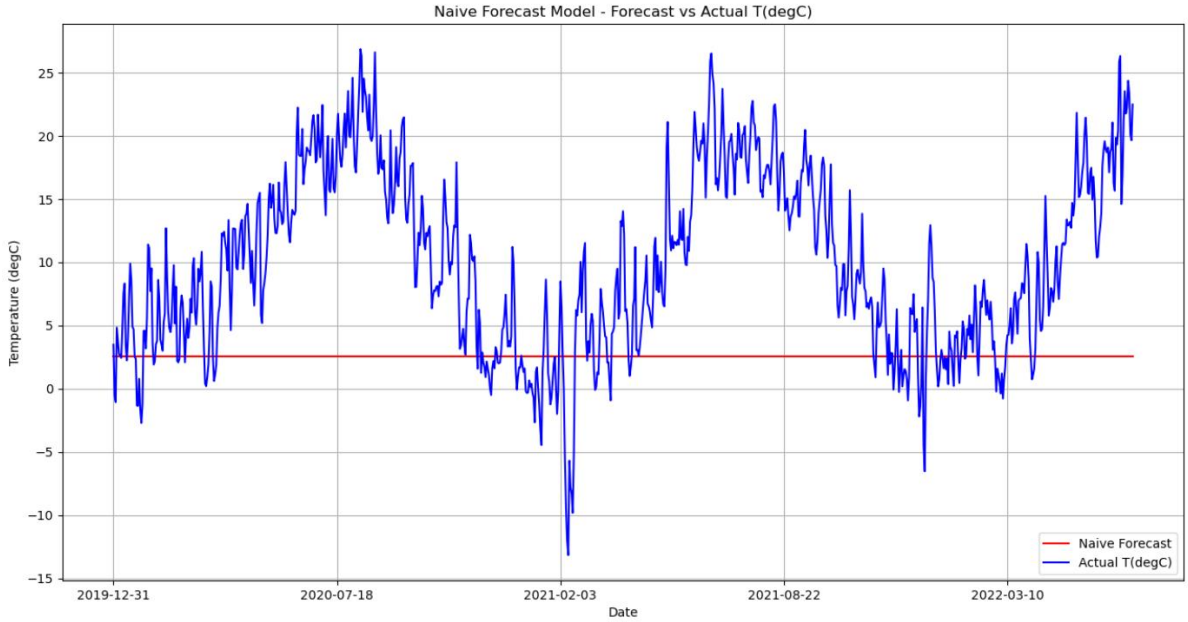


Fig 5.16 Naïve Method Forecast

To quantify the performance of this model. The metrics used above were calculated. Below are the results.

```
The RMSE of Naive model is: 9.722
The MAE of this model is: 7.741
The MAPE of this model is: 10.194
```

Fig 5.17 Naïve Method Result

The next base model was using the Drift method. Below is the plot for the forecasted values vs the test set for this model.

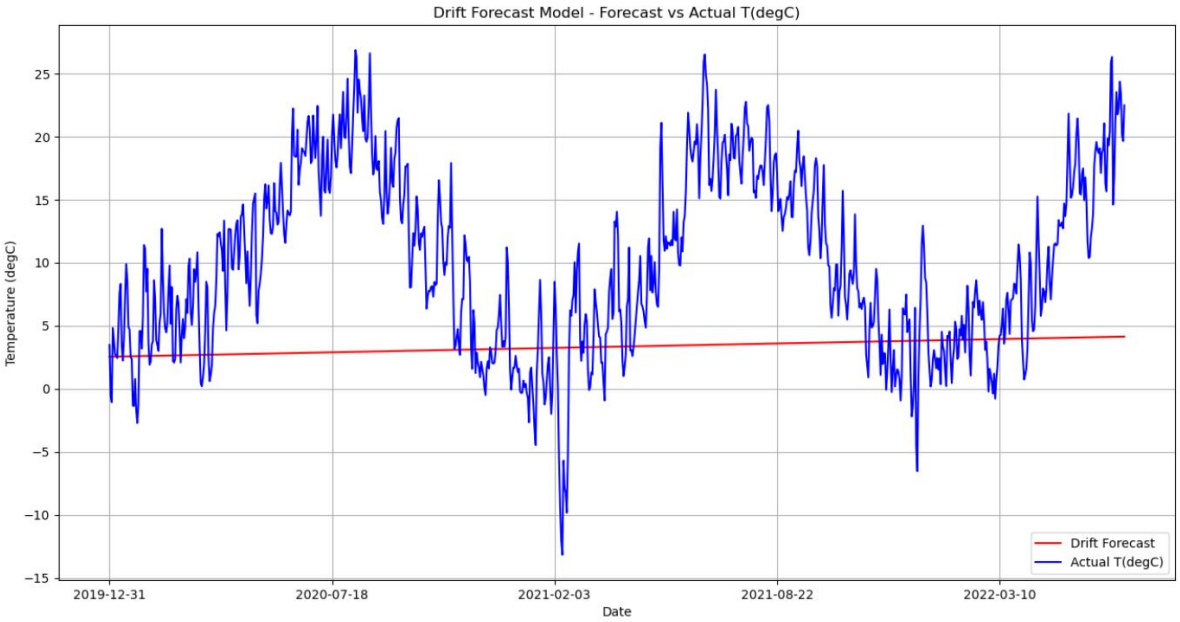


Fig 5.18 Drift Method Forecast

To quantify the performance of this model. The metrics used above were calculated. Below are the results.

```
The RMSE of Drift model is: 9.722
The MAE of this model is: 7.741
The MAPE of this model is: 10.242
```

Fig 5.19 Drift Method Results

The next base model was using the Simple Exponential Smoothing method. Below is the plot for the forecasted values vs the test set for this model.

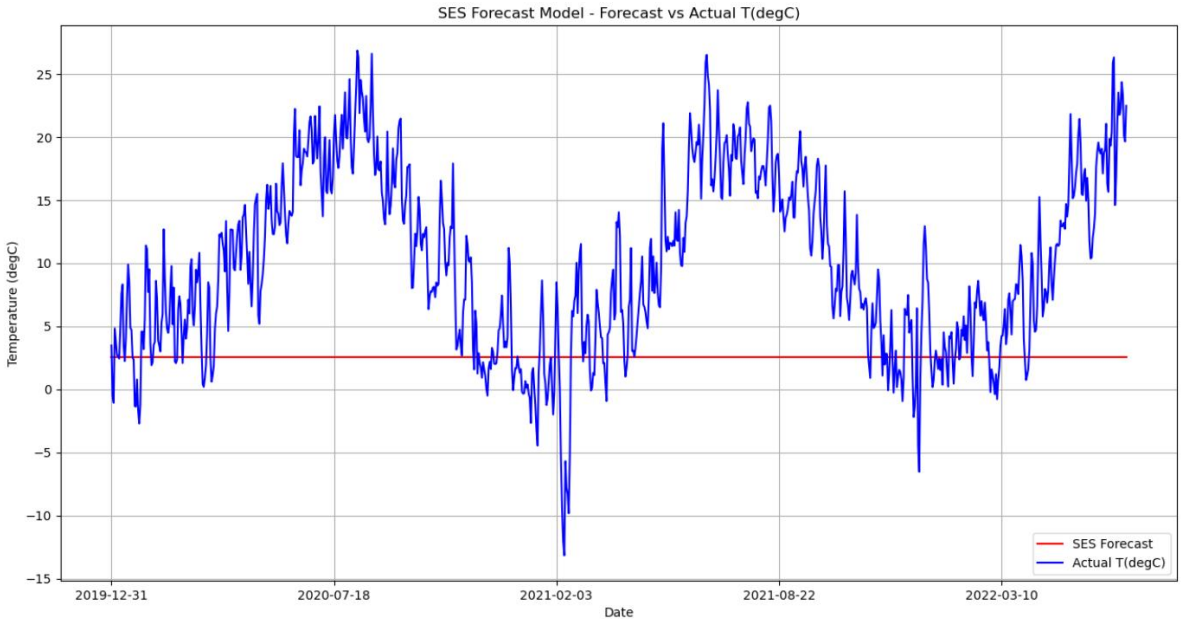


Fig 5.20 SES Method Forecast

To quantify the performance of this model. The metrics used above were calculated. Below are the results.

```
The RMSE of SES model is: 10.28
The MAE of this model is: 8.231
The MAPE of this model is: 10.194
```

Fig 5.21 SES Method Result

The models were built only for the purpose of benchmarking the SARIMA Model. These models are not considered in the final model selection.

5.4 *SARIMA Model*

As identified during the exploratory data analysis stage, owing to the presence of extremely high correlations within the dataset, the model that will be investigated will be SARIMA and not ARIMA. This is because high correlations in the time series can make it difficult to build accurate ARIMA models because the long-term dependencies in the time series are difficult to capture using only a few lags. Therefore, the dataset for modeling that is considered here is the order of one seasonally differenced data with a seasonality of 365. The stationarity assessment had been made previously and will not be repeated here. The seasonal differencing is only performed on the train set.

The SARIMA modeling involved 4 steps, out of which 3 are performed iteratively. The first step is to determine order using the Generalized Partial Autocorrelation Function (GPAC) that is made using the autocorrelations of the series. The next step involves estimating the parameters using the Levenberg-Marquardt algorithm. The next step is to fit the model and perform a residual diagnostic test. If the residual diagnostic is failed, the process needs to be repeated. However, once the model that produces a white residual is obtained, forecasting can be performed.

There is an alternative way of judging the order of the process. This is through the ACF/PACF plot. The ACF/PACF plot for the differenced time series is provided again for ease of reference.

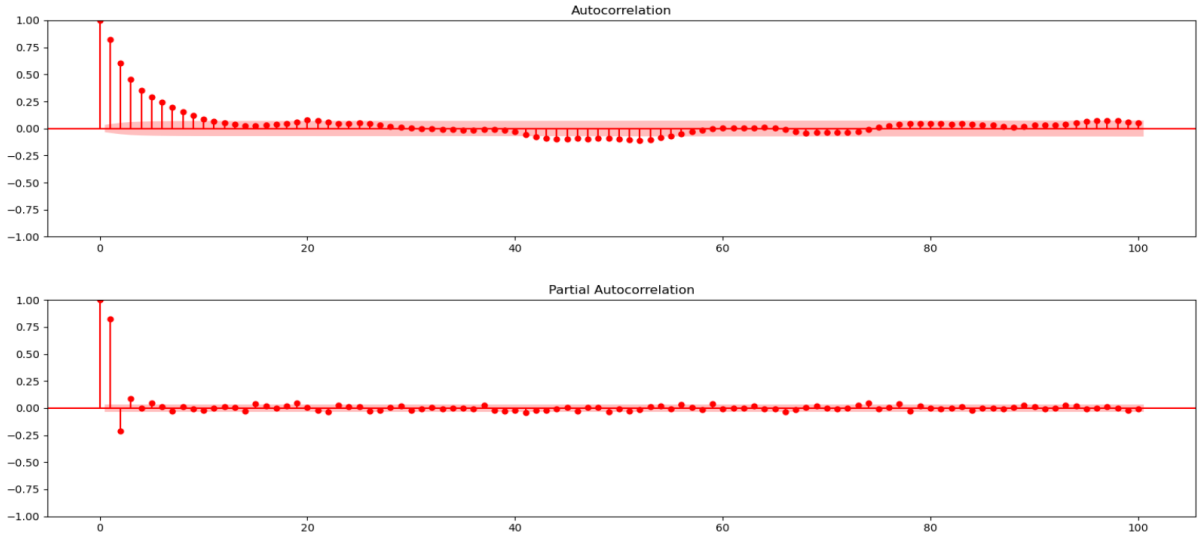


Fig 5.22 ACF/PACF of differenced data

Judging by this ACF/PACF plot, as identified below, the process appears to exhibit an AR behavior with a tail-off in the ACF plot and a cut-off in the PACF plot. The order can then be determined by assessing the lag after which the tail-off occurs. After lag = 3, the PACF values seem to drop significantly. Therefore, a preliminary assumption on the order of the ARIMA portion of the process is that of a (3,0,0) based on the ACF/PACF. However, for more objective and accurate results, the GPAC will be considered.

After performing the first step, the GPAC was obtained which can be seen below.

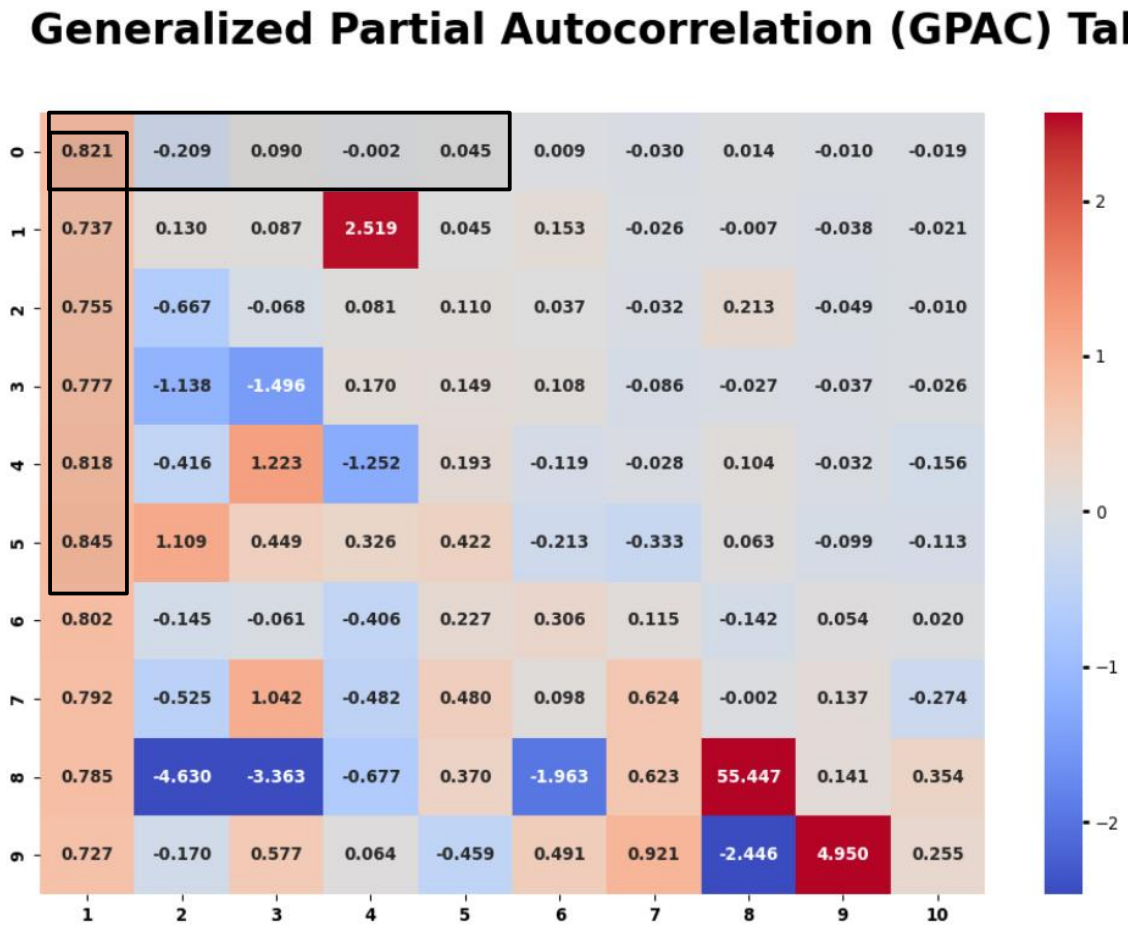


Table 5.1 GPAC of process

From the above GPAC, the first preliminary order that was determined was that of (1,0) indicating that the AR portion had one coefficient and MA had none.

The results of estimating parameters through the custom-developed Levenberg-Marquardt algorithm can be seen below.

```
Algorithm has converged!!!
The Estimated Parameters are: [-0.8213610474338395]
The Covariance matrix is: [[9.90777926e-05]]
The Variance of error is: 10.526537555694125
Confidence interval for AR parameters are:
-0.8412686130900701 < a1 < -0.801453481777609
The roots of the denominator are: [0.82136105]
```

Fig 5.23 SARIMA LM (1,0)

The same order was passed to the python function of ARIMA, and the model summary was obtained below.

SARIMAX Results

Dep. Variable:	T(degC)_365_Diff	No. Observations:	3286
Model:	ARIMA(1, 0, 0)	Log Likelihood	-8529.755
Date:	Wed, 10 May 2023	AIC	17063.510
Time:	22:10:36	BIC	17075.705
Sample:	01-01-2011	HQIC	17067.877
	- 12-30-2019		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.8212	0.010	86.350	0.000	0.803	0.840
sigma2	10.5208	0.254	41.415	0.000	10.023	11.019

Ljung-Box (L1) (Q):	96.06	Jarque-Bera (JB):	2.05
Prob(Q):	0.00	Prob(JB):	0.36
Heteroskedasticity (H):	0.84	Skew:	-0.04
Prob(H) (two-sided):	0.00	Kurtosis:	3.10

Fig 5.24 SARIMAX (1,0)

The parameter values match the custom algorithm and package. The next step of the process is residual diagnostic. This has been performed below.

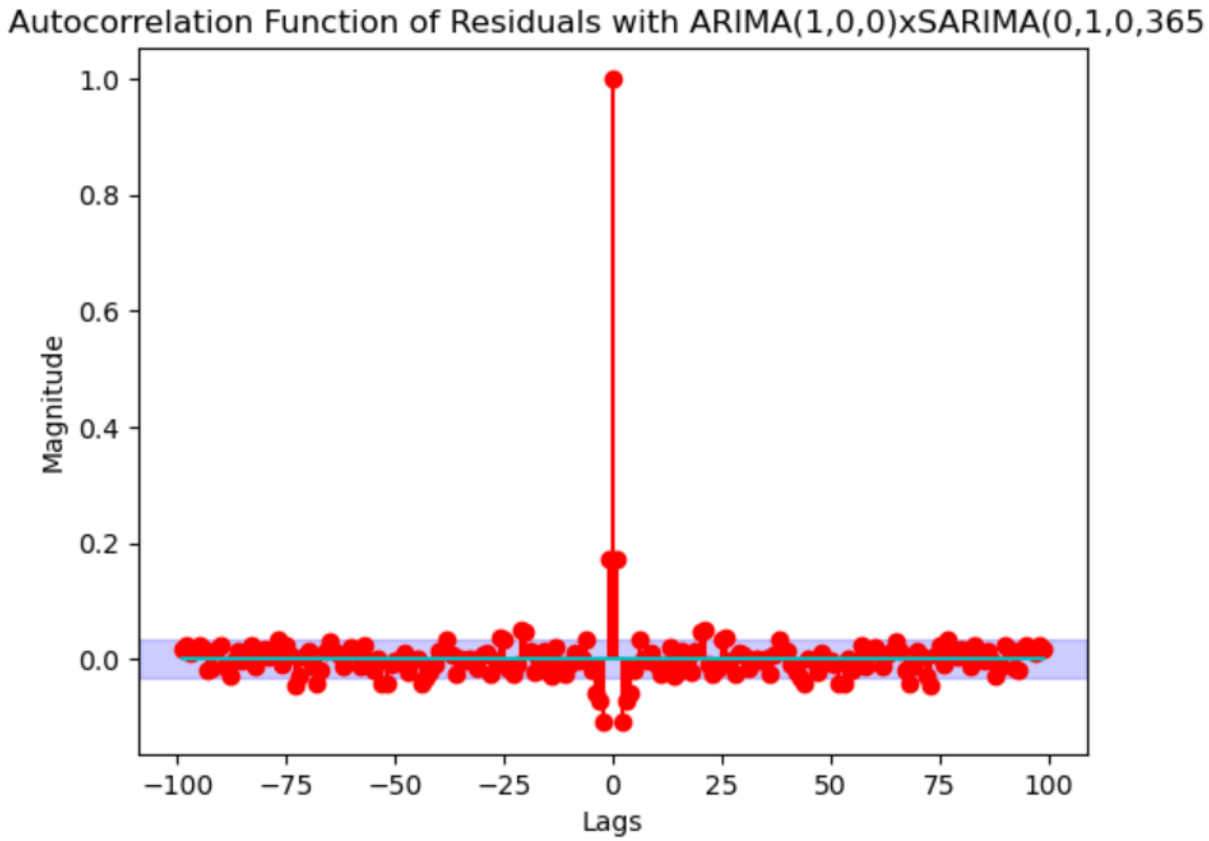


Fig 5.25 SARIMAX (1,0) ACF of Residual

From this ACF plot, we can see that this model has failed the residual diagnostic test as the ACF does not appear to be white. To provide a more objective view, the Box-Pierce test was considered. The results are below.

```
Q is 301.7351579045196 and chi critical is 134.64161685578915
The residual is NOT white
```

Fig 5.26 SARIMAX (1,0) Residual Test

This further reinforces the observation made above. Therefore, this model needs to be discarded.

The process is repeated again for a new order.

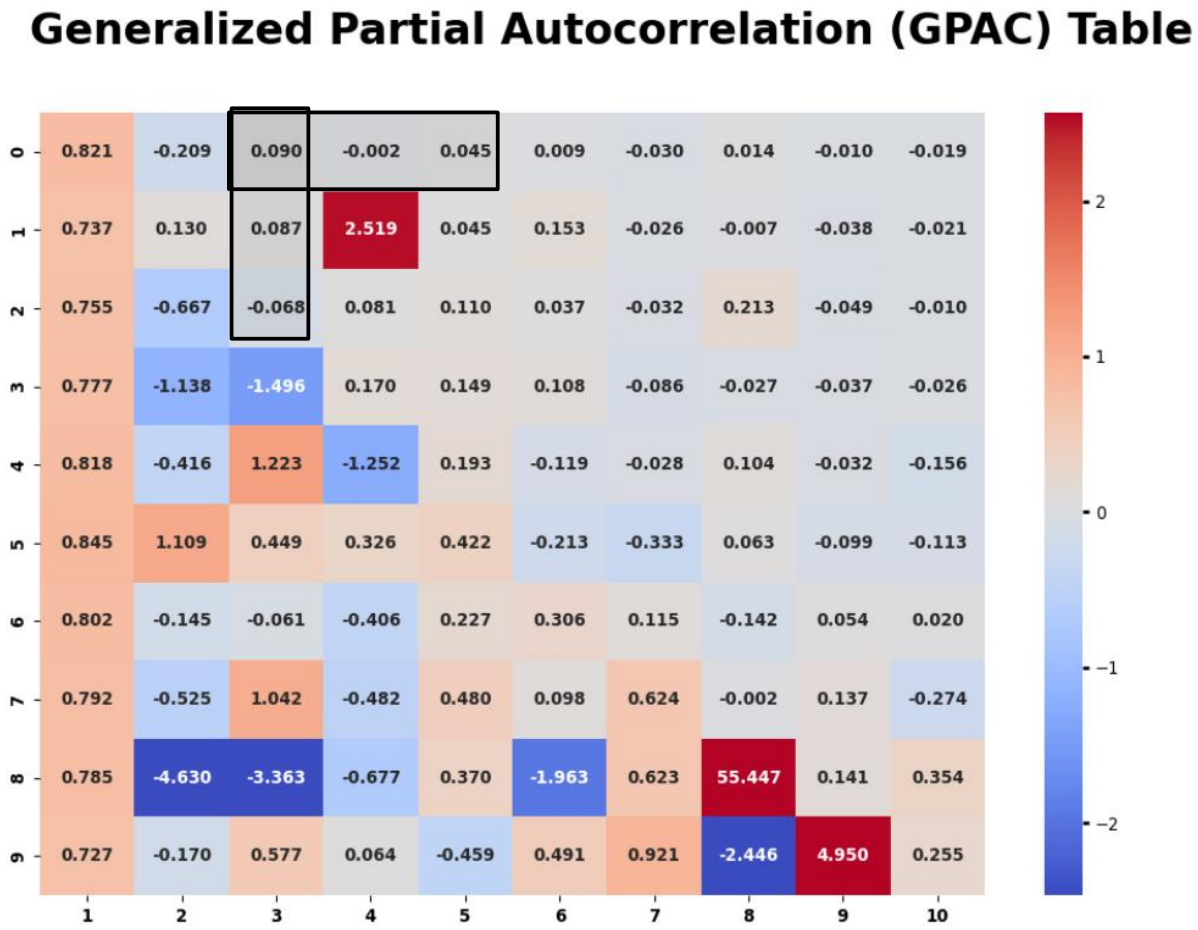


Table 5.2 GPAC of process

This time the order that has been selected is ARIMA (3,0,0). This order matches the order that was identified by the ACF/PACF.

The results of estimating parameters through the custom-developed Levenberg-Marquardt algorithm can be seen below.

```
Algorithm has converged!!!
The Estimated Parameters are: [-1.011398710003359, 0.2983713542290045, -0.09062909533692821]
The Covariance matrix is: [[ 3.02141247e-04 -2.99962460e-04  6.30313817e-05]
 [-2.99962460e-04  5.86977039e-04 -3.00100522e-04]
 [ 6.30313817e-05 -3.00100522e-04  3.02332947e-04]]
The Variance of error is: 9.992928183248585
Confidence interval for AR parameters are:
-1.0461631312799295 < a1 < -0.9766342887267886
0.2499161361914791 < a2 < 0.34682657226652996
-0.12540454340531015 < a3 < -0.05585364726854627
The roots of the denominator are: [0.77757158+0.j         0.11691357+0.32075729j 0.11691357-0.32075729j]
```

Fig 5.27 SARIMAX LM (3,0)

The parameter estimates, the standard deviation of the parameter estimates, and confidence intervals can be referred to from the above plot.

The same order was passed to the Python function of ARIMA, and the model summary was obtained below.

```
                               SARIMAX Results
==============================================================================
Dep. Variable:       T(degC)_365_Diff   No. Observations:                 3286
Model:                 ARIMA(3, 0, 0)   Log Likelihood               -8443.418
Date:                Tue, 09 May 2023   AIC                          16894.837
Time:                        23:19:42   BIC                          16919.226
Sample:                  01-01-2011   HQIC                         16903.569
                       - 12-30-2019
Covariance Type:                 opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.0111      0.016     62.309      0.000       0.979       1.043
ar.L2         -0.2981      0.023    -12.983      0.000      -0.343      -0.253
ar.L3          0.0905      0.017      5.434      0.000       0.058       0.123
sigma2         9.9817      0.238     41.965      0.000       9.515      10.448
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 5.47
Prob(Q):                              0.99   Prob(JB):                         0.07
Heteroskedasticity (H):               0.82   Skew:                            -0.03
```

Fig 5.28 SARIMAX (3,0)

The parameter values match the custom algorithm and package. The next step of the process is residual diagnostic. This has been performed below.

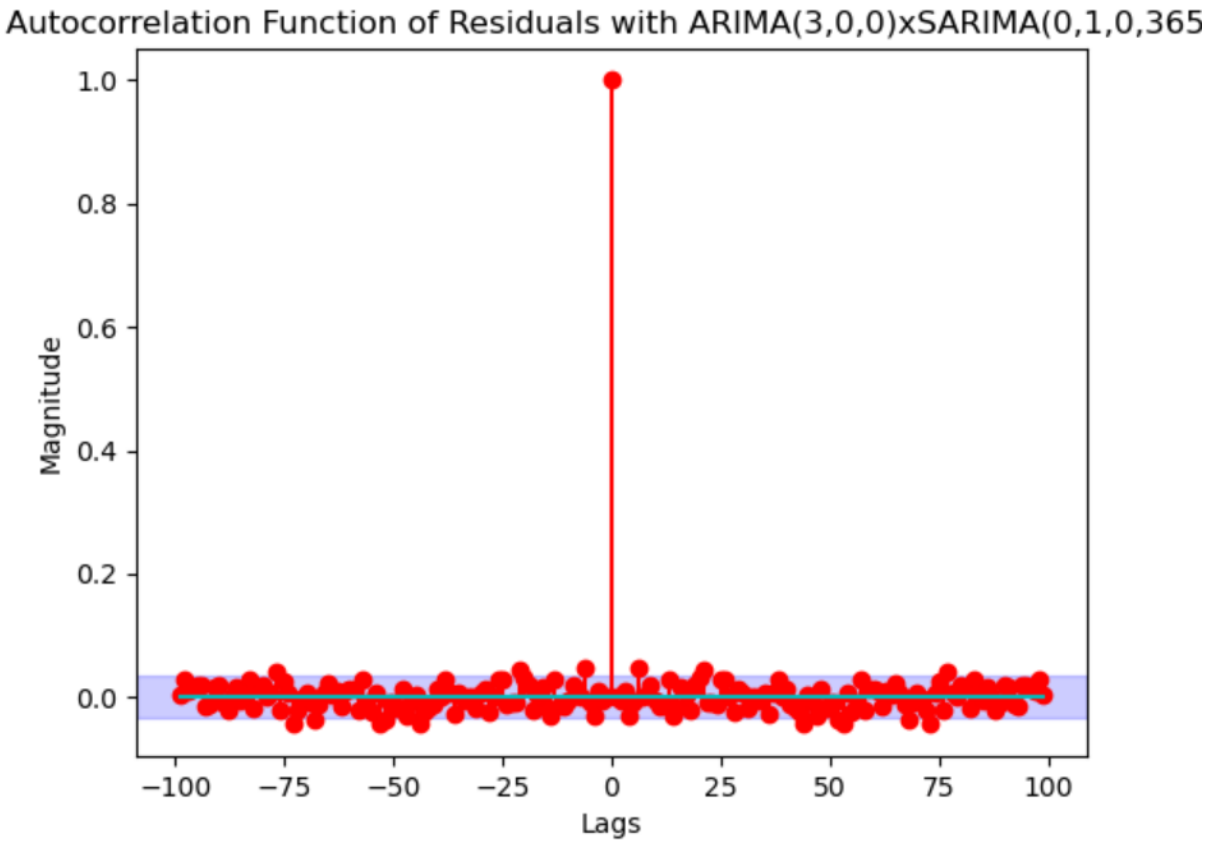


Fig 5.29 SARIMAX (3,0) ACF of Residual

At first glance, this produces an ACF which appears to be white indicating that all the model correlations have been captured in this model. To provide a more objective conclusion, the Box-Pierce test was performed on these residuals. The results are below.

```
Q is 129.62389861782702 and chi critical is 132.30887667181258
The residual is white
```

Fig 5.30 SARIMAX (3,0) Residual Test

The Box-Pierce (chi-square) test concluded that the residual is white. This indicated that the ARIMA (3,0,0) model has passed the diagnostic test. Further assessing the residuals by plotting the distribution. This plot is below.

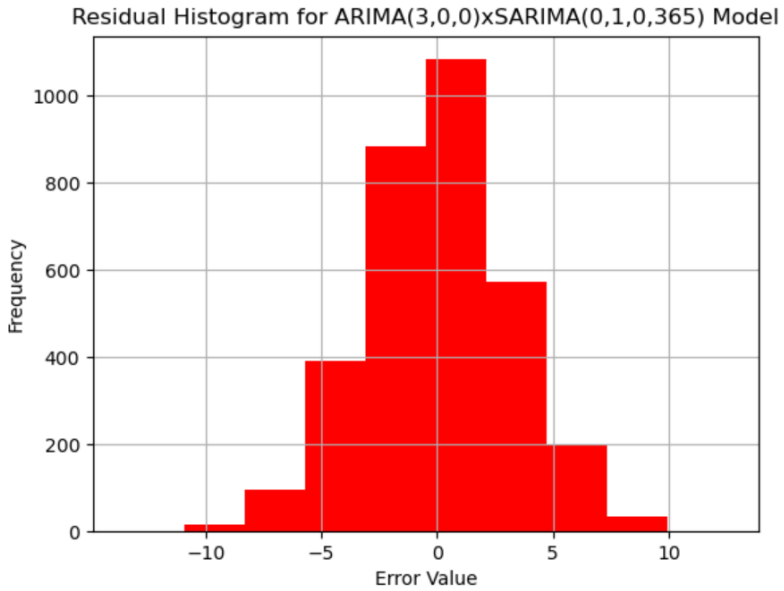


Fig 5.31 SARIMAX (3,0) Residual histogram

This normally distributed residual indicates that the ARIMA (3,0,0) Model is not biased as the mean of the residual of error is close to zero. Furthermore, the error is spread with an equal variance which is desirable. Therefore, the final model that is selected is an ARIMA (3,0,0) x SARIMA (0,1,0,365). There is a SARIMA component present as well since the seasonally differenced data was made use of. Then the non-seasonal order of the differenced data was obtained through the above analysis.

The forecast for the length of the test set is then performed to assess the performance of the model. However, since the differencing was performed on the train set which was used to build the ARIMA (3,0,0) model, a reverse transformation after forecasting was performed to obtain comparable results with test data, which was not differenced. The plot of the forecast vs test set is shown below.

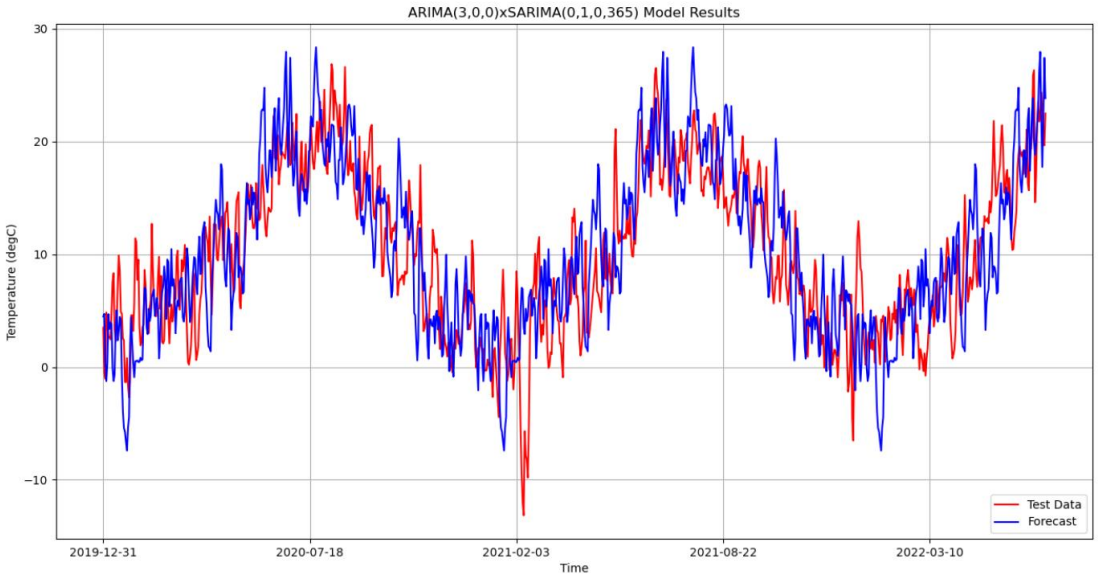


Fig 5.32 SARIMAX Forecast

The model seems to perform well against the test set. To objectively assess the performance, the above-defined metrics are used to assess the performance which can be seen below.

```
The RMSE of ARIMA(3,0,0)xSARIMA(0,1,0,365) is: 4.928
The MAE of ARIMA(3,0,0)xSARIMA(0,1,0,365) is: 3.951
The MAPE of ARIMA(3,0,0)xSARIMA(0,1,0,365) is: 1.565
```

Fig 5.33 SARIMAX Results

To further assess the generalizing capabilities of this model, the variance of residual error was compared against the variance of forecast error. The result of this analysis was: `2.493`
This result indicates that the model does not generalize as well as expected, since ideally the ratio of variance is expected to be 1.

5.4 *LSTM Model*

The final model that was built was the Deep Learning LSTM model. Like the MLR model, the exogenous inputs are made use of, and the features are standard scaled before fitting into the model. The training size shape needs to be manipulated and a shape of 365 continuous features was considered keeping in mind the seasonality in the dataset.
The architecture of the model that was selected is given below.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 365, 64)           21248

 lstm_1 (LSTM)               (None, 50)                23000

 dropout (Dropout)           (None, 50)                0

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 44,299
Trainable params: 44,299
Non-trainable params: 0
_________________________________________________________________
```

Fig 5.34 LSTM Architecture

This architecture was run only for 10 epochs considering the computing capability, with a validation set of size 10% of the train size, with loss function as Mean Square Error and optimizer as Adam.
The Validation Loss and Training Loss were plotted and are depicted below.

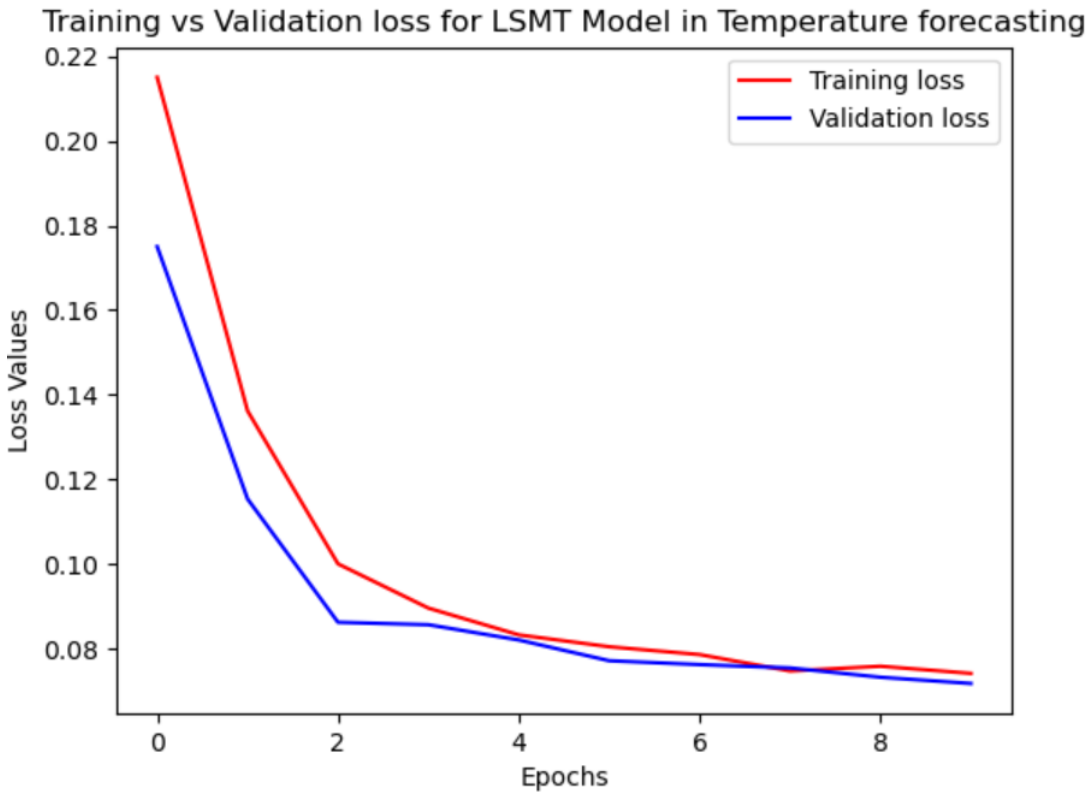


Fig 5.35 LSTM Val-Train Loss

This plot suggests that within only 10 epochs, the LSTM Model was able to converge since there does not appear to be a divergence after 6 epochs. Therefore, the algorithm had reached the global/local minima and stabilized.

The Forecast from the build model was then obtained and plotted against the actual test data which is shown below.

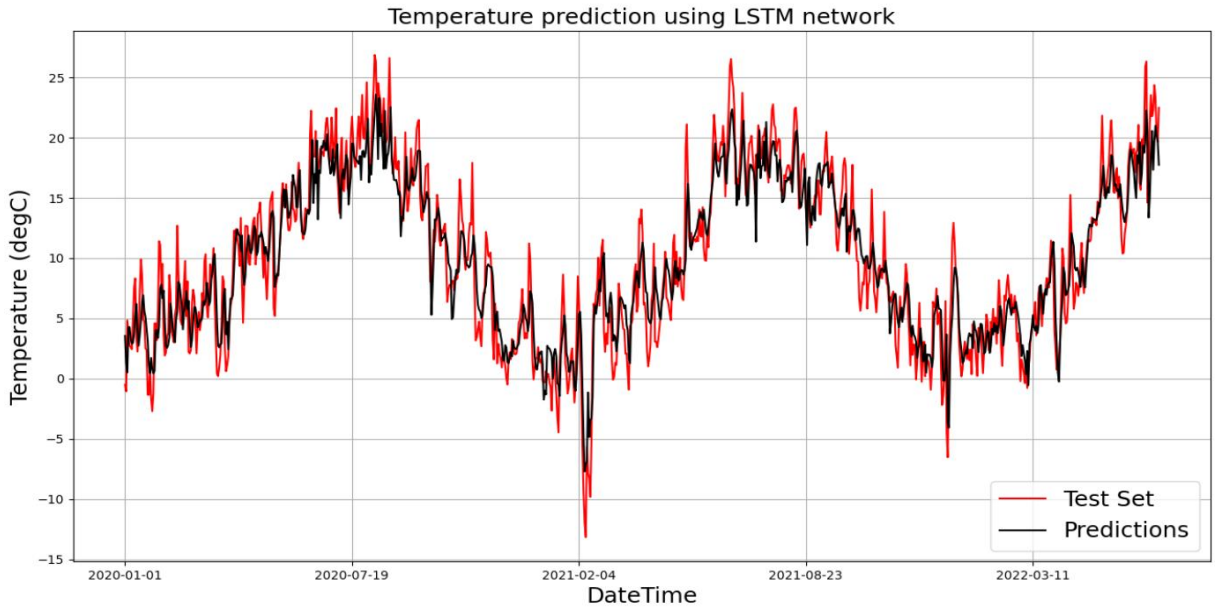


Fig 5.36 LSTM Forecast

The model seems to perform well on the test set. To quantify this performance, the metrics utilized above were calculated and are shown below.

```
The RMSE of LSTM Model is: 2.108
The MAE of LSTM Model is: 1.618
The MAPE of LASTM Model is: 0.788
```

Fig 5.37 LSTM Results

The next step was to perform the residual diagnostic for this model as it is done for all time-series models. It is shown below.

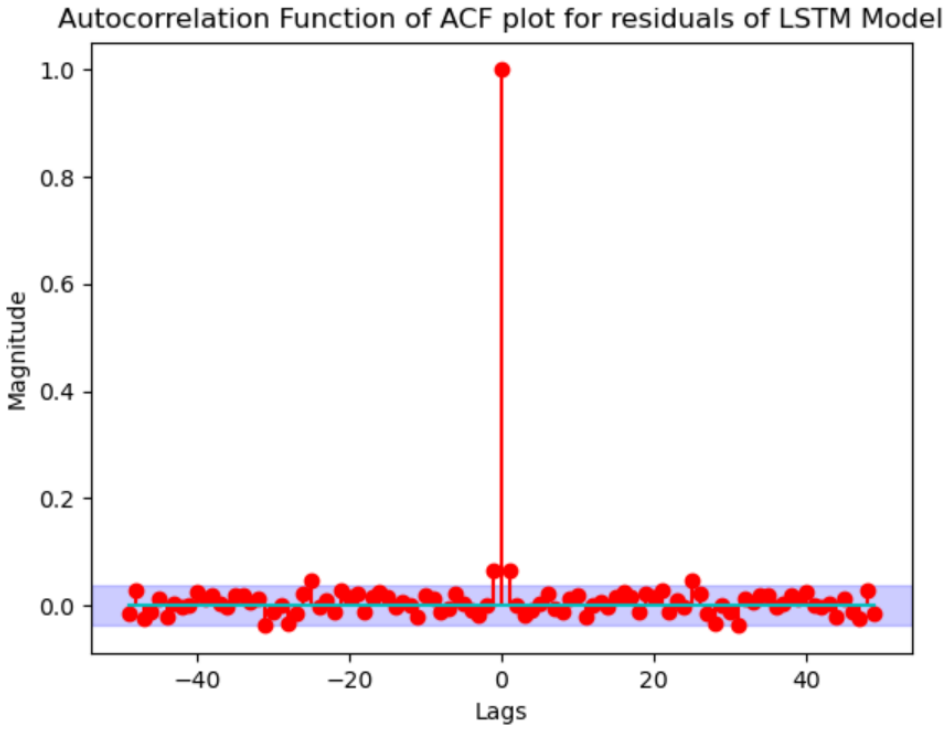


Fig 5.38 LSTM ACF of Residuals

At first glance, the residual analysis seems to fail as there appears to be some value at lag 1 that makes the residual non-white. However, to perform a more objective residual test, the Ljung-Box test was employed. The results are below.

	lb_stat	lb_pvalue
365	378.7741	0.2987

Fig 5.39 LSTM Residual Test

With the results of this statistical test, we can see that the p-value is very high. Therefore, we cannot reject the null hypothesis that the residuals are independent and identically distributed suggesting that the residual could be white.

CHAPTER 6
FINAL MODEL COMPARISONS AND SELECTION

This section dives into the comparison of the models and leads to selecting the best-performing model in terms of pre-determined criteria.

6.1 *Base Model vs SARIMA model*

This section will perform a brief overview of the performance of the ARIMA (3,0,0) x SARIMA (0,1,0,365) model as compared to the developed base models.

The model performance of all models is summarized below.

Model Type	Root-Mean-Square-Error	Mean-Absolute-Error	Mean-Absolute-Percent-Error
Average Method Forecast	7.087	6.011	10.967
Naive Method Forecast	9.722	7.741	10.194
Drift Method Forecast	9.722	7.741	10.242
SES Method Forecast	10.28	8.231	10.194
SARIMA Model Forecast	4.928	3.951	1.565

Table 6.1 Base Model vs SARIMA Results

With these results, it is established that the SARIMA Model performs much better than the base models that are built for benchmarking in terms of the metric displayed.

To get a sense of the forecasting performance visually, the forecasted values by all the base models and the SARIMA model are plotted against the test set and is shown below.

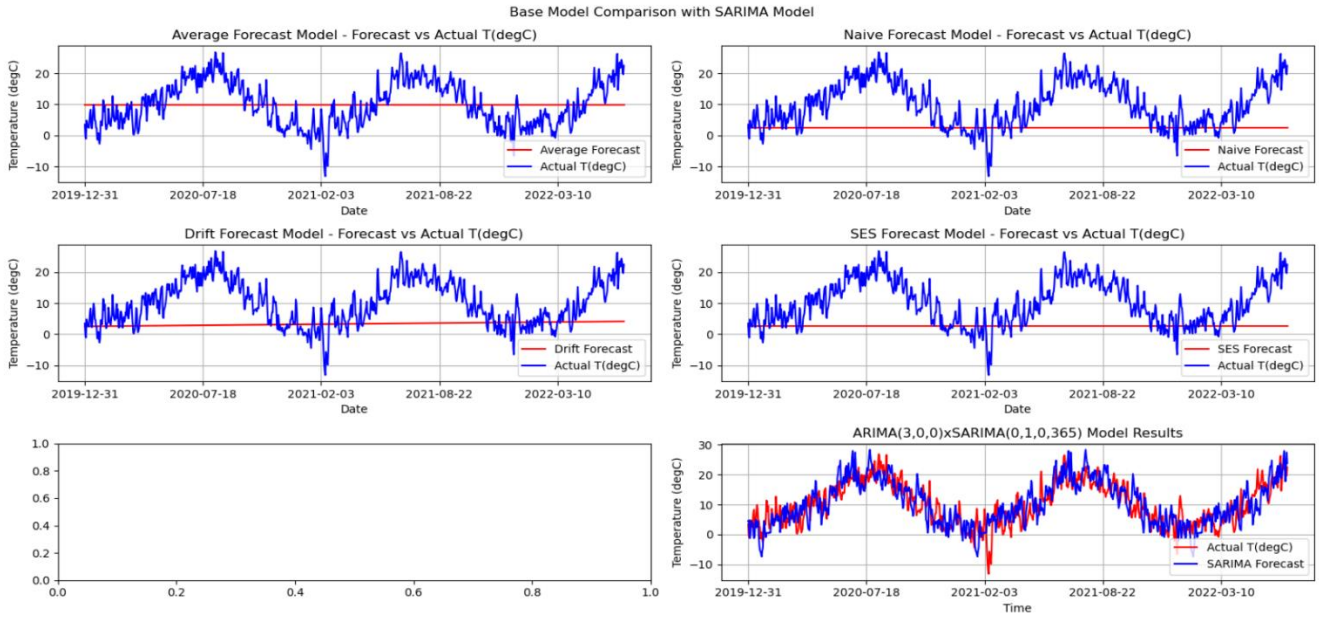


Fig 6.1 Base models vs SARIMA Forecast

Based on these plots, the SARIMA model performs much better than the benchmarking plots.

6.2 *Final Model Selection*

Apart from the base models, the models that were competing are as follows:

1. Holt-Winter Model
2. Multiple Linear Regression Model
3. SARIMA Model
4. LSTM Model

The key comparison criteria that are used to select the best final model are as follows:

1. Residual Diagnostic – Assessment of whiteness of residuals is a key metric that any time-series model needs to strictly adhere to. This metric gives an idea on how well the model represents the data.
2. Performance Metrics – Metrics such as Root Mean Square Error (RMSE), Mean Absolute Percentage Error, and Mean Absolute Error (MAE) will objectively identify the performance of each model.
3. Visual Forecast – Plotting the forecast from each model will help assess how well the model is performing on unseen data.

The first criterion is judged. The summary of ACF of residual plots for all models can be seen below.

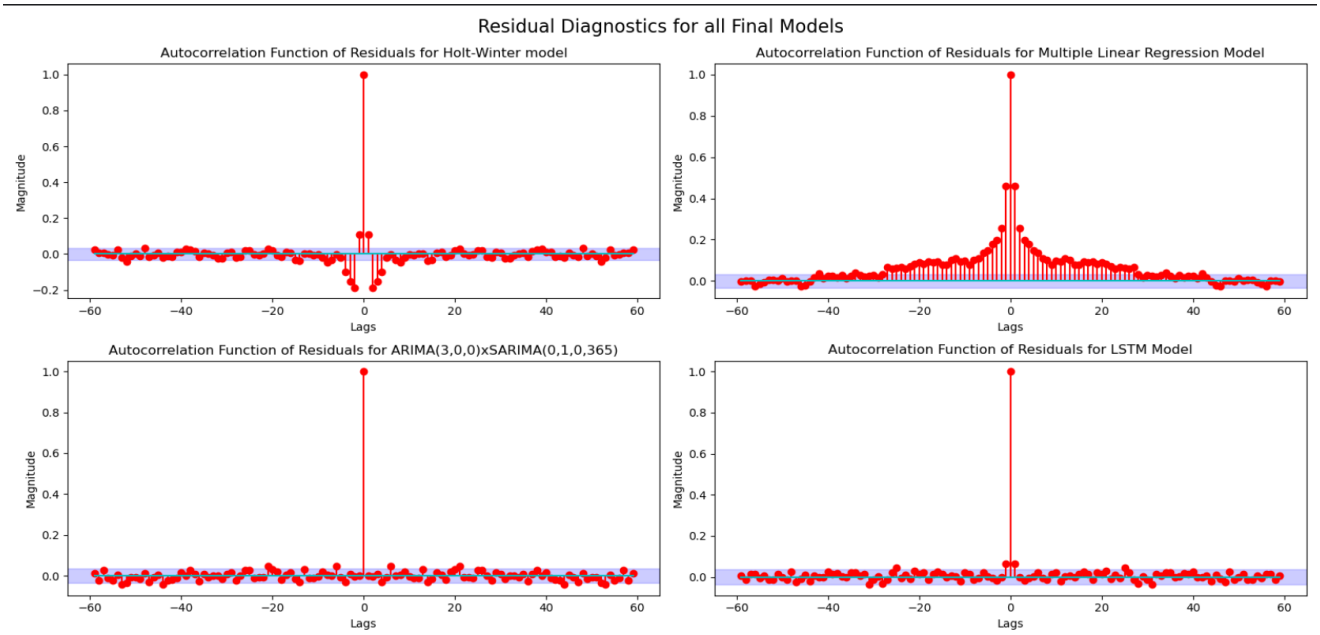


Fig 6.2 Final model Selection Residual Diagnostic

Judging by the residual plots for all models, Holt-Winter and Multiple Linear Regression do not produce a white residual which indicates that these models do not fit the underlying time series data the best. Since this is a very important criteria, these models are out of the

competition already. The competing models are now LSTM and SARIMA. However, the residual of SARIMA appears to be more white as compared to LSTM.

The second criterion is displayed below.

Model Type	Root-Mean-Square-Error	Mean-Absolute-Error	Mean-Absolute-Percent-Error
Holt-Winter Model	9.825	7.985	3.776
MLR Model	0.511	0.361	0.18
SARIMA Model	4.928	3.951	1.565
LSTM Model	2.108	1.618	0.788

Table 6.2 Final Model Selection

Although the MLR model has the best performance, since the residuals of this model are not white, it has not been considered. LSTM performs the second-best followed by SARIMA. The performance of SARIMA and LSTM is comparable.

The third criterion is displayed below.

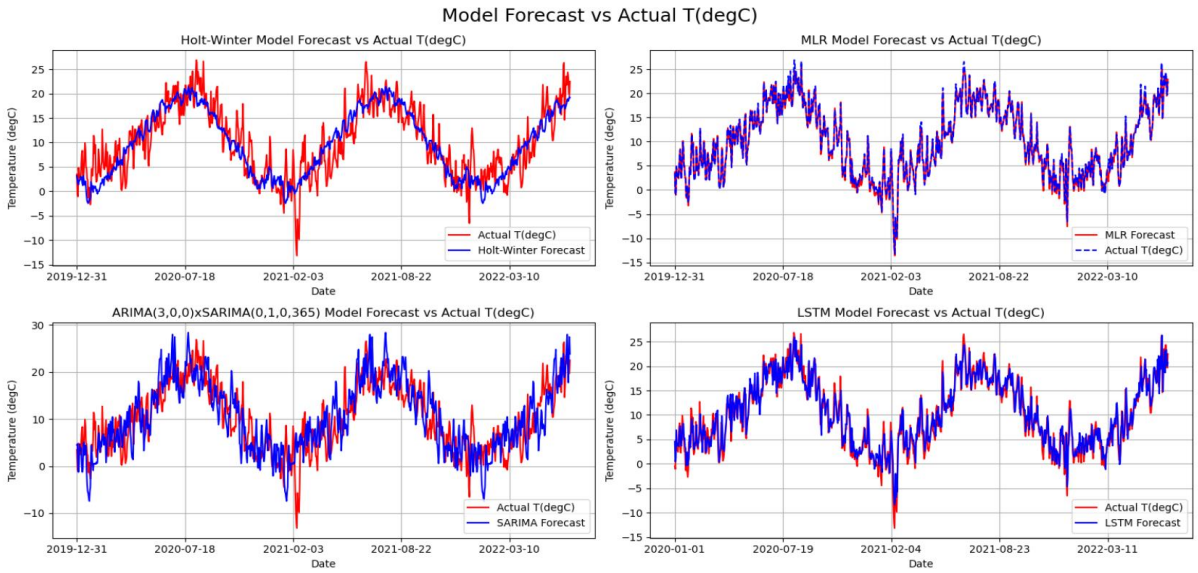


Fig 6.3 Final Model Comparison Forecasts

Based on these plots, the MLR model has the best fit which is also indicated by the lowest performance metric values. This is followed by LSTM and then by SARIMA. However, since the MLR model is out of the competition, only LSTM and SARIMA are considered.

Based on the results of all tests above, the LSTM model appears to have the best final mode. However, judging by the residual analysis of both these models, the SARIMA appears to have a ‘more-white’ residual. Additionally, due to the nature of Ljung-Box test, it cannot be rejected

that the residual of LSTM is white, indicating there isn't sufficient evidence to reject this hypothesis. This test could go either way as the number of samples is increased.
Therefore, the final model that is selected to represent the best fit and have the best performance is ARIMA (3,0,0) x SARIMA (0,1,0,365). The performance of SARIMA and LSTM are comparable as well.

6.3 *Forecast Function and h-step prediction*

The custom forecast function is then developed for the SARIMA model using the estimated parameters. The developed function can be seen below.

```
def custom_forecast_function(data,Step):
    y_hat = []
    for i in range(1, Step+1):
        if i == 1:
            y_hat.append(1.0111 * data.values.ravel().tolist()[-1] - 0.2981 * data.values.ravel().tolist()[-2] +
             0.0905 * data.values.ravel().tolist()[-3])
        elif i == 2:
            y_hat.append(1.0111 * y_hat[0] - 0.2981 * data.values.ravel().tolist()[-1] + 0.0905 * data.values.ravel(
             ).tolist()[-2])
        elif i == 3:
            y_hat.append(1.0111 * y_hat[1] - 0.2981 * y_hat[0] + 0.0905 * data.values.ravel().tolist()[-1])
        else:
            y_hat.append(1.0111 * y_hat[i-2] - 0.2981 * y_hat[i-3] + 0.0905 * y_hat[i-4])
    y_hat = np.array(y_hat)
    return y_hat
```

Fig 6.4 Custom Forecast Function

This code was first developed on paper using the parameters and implemented as it is in Python. Using this function, the h-step predictions are made on the data and are reverse transformed as expected due to differencing. The final forecasted values are plotted against the actual test set below.

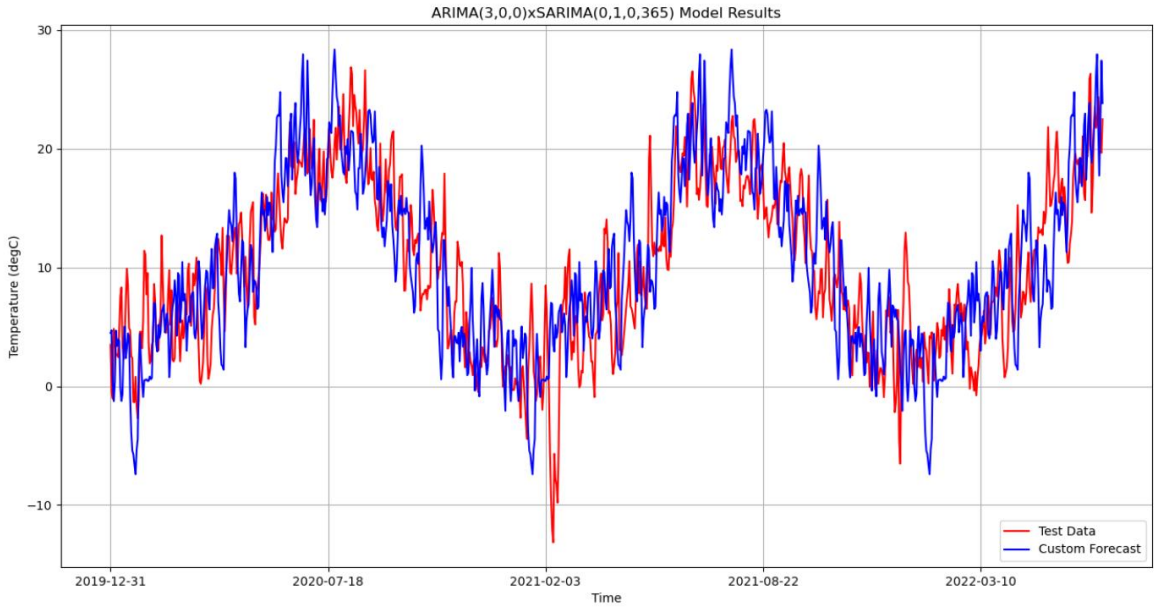
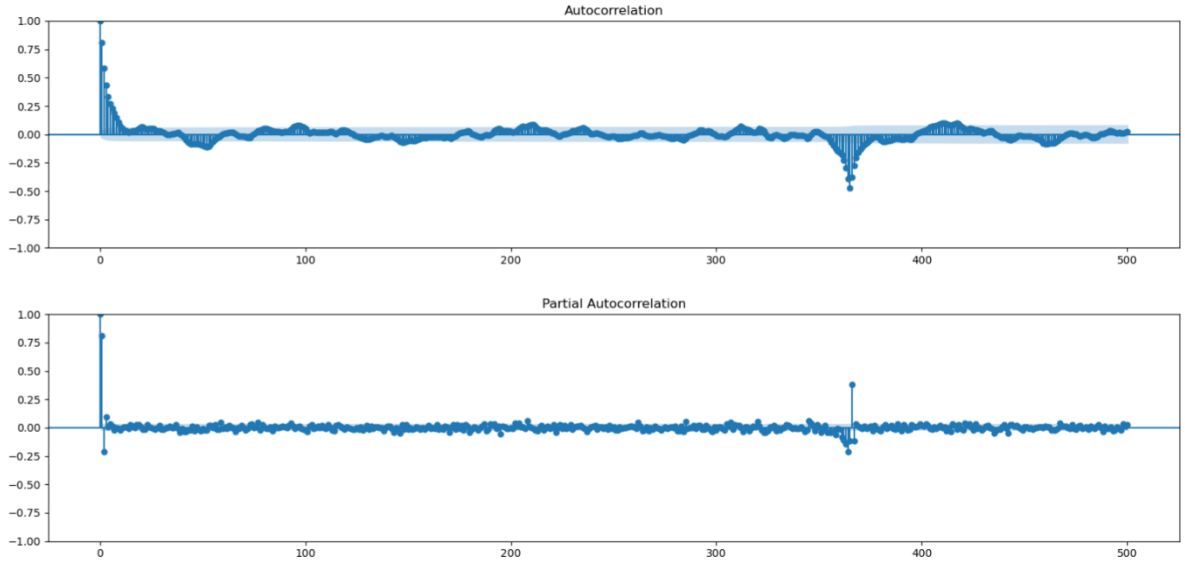


Fig 6.5 Custom Forecast Function Plot

The performance of the SARIMA model through the custom forecast function has a nice fit on the test data. This is also very similar to the forecast plot that is obtained by using the Python package indicating that the custom forecast function is working as expected.
Overall, the SARIMA function fits the data well, and has a good performance on unseen data.

CHAPTER 7
SUMMARY AND CONCLUSION

Although the final SARIMA model performs well on the data and fits the data well. There are certain limitations to it. It was observed during the preliminary assessment that ACF/PACF plot of the differenced data for a greater number of lags produces another spike at 365, the seasonality of the data. This can be seen below.



Judging by this plot, it is seen that there is still some seasonality that has not been captured by the ARIMA (3,0,0) x SARIMA (0,1,0,365) model. Initially, it was attempted to fit the order of SARIMA into the model as well, but the code was not able to run the function with such a high seasonality index. This model development was then discarded.

Future work could include somehow incorporating this seasonal aspect of the dataset. Based on the ACF/PACF plot up to 1200 lags, there appeared to be Seasonal component of MA of order 1 that needs to be incorporated into the final model developed.

REFERENCES

[1] https://article.sciencepublishinggroup.com/pdf/10.11648.j.ijema.20210906.17.pdf

[2] https://www.statsmodels.org/dev/examples/notebooks/generated/stl_decomposition.html

[3] https://stackoverflow.com/questions/72631831/sarima-with-daily-data-determine-saisonal-period

[4] https://barnesanalytics.com/sarima-models-using-statsmodels-in-python/

[5] Time Series Analysis Lecture Notes

[6] https://www.kaggle.com/datasets/harishedison/jena-weather-dataset

[7] https://towardsdatascience.com/time-series-in-python-part-2-dealing-with-seasonal-data-397a65b74051

ANNEXURES

Providing below the code developed by me during this project.

Initial Data Analysis:

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


df = pd.read_csv('Dataset/WeatherJena.csv')

# Looking at all the column names
print(df.columns)

# Target variable is Temperature in degree celsius
print(df.head().to_string())
print(df.tail().to_string())
# One extra observation for a new day. Will remove this observation from
the dataset
df.drop(df.tail(1).index,inplace=True)

print(df.shape)
print(df.nunique())

# I notice the presence of few duplicated dates as the number of unique
dates is less than total number of rows. Dropping duplicated rows

df.drop_duplicates(inplace=True)
print(df.shape)
print(df.nunique())
# Number of date-time now matches the number of rows.

# Getting rid of the white spaces in column names
df.columns = df.columns.str.replace(' ','')

# Looking at the number of rows
print(df.shape)

# Treating DateTime column as index
df.index = pd.to_datetime(df.DateTime,dayfirst=True)
print(df.head())

# We don't require the DateTime column anymore.
df.drop(columns=['DateTime'],axis=1,inplace=True)
print(df.shape)
print(df.head())

# ---------------------------x-----------------------------------
# Result of some external analysis - 2 missing dates + 6 days have
incomplete 10-minute interval observations.

# To count the number of rows for each day in the dataset. Expectation is
to see 144 count for each days as it is a 10 - minute interval. There are
1440 minutes in a day.
j = pd.DataFrame([],columns = ['Count','Year','Month','Day'])
count = 0
for i in range(len(df)):
    if i+1 > len(df)-1:
        break
    if df.iloc[i].name.day == df.iloc[i+1].name.day:
```

```
        count = count + 1
    else:
        count = count + 1
        temp =
pd.DataFrame({'Count':count,'Year':df.iloc[i].name.year,'Month':df.iloc[i].
name.month,'Day':df.iloc[i].name.day},index = [0])
        j = pd.concat([j,temp],ignore_index=True,)
        count = 0

# Now checking how many days don't have exactly 144 observations as is the
expectation
print(j[j['Count']!=144])
# From this analysis we see a few drawbacks in the 10-minute interval data.
9 days don't have all the 10 minute intervals covered while measuring the
data. We cannot use the 10 minute interval as time series.
# To convert into a usable time-series, i will instead aggregate my data to
day level. I will do this by taking the average of all temperatures within
the day so as to avoid bias.

df_target = df[['T(degC)']]
df_target = df_target.resample('D').mean()

# Looking at the number of days that are avilable between this time period
to see if this operation has been performed successfully. Using an external
tool, i've caluclated the number of days between these dates (inclusive) to
be 4564.
print(len(df_target))
# The length matches.

# Now evaluating whether any day is missing from the dataset, expectation
is to not see any NA's after aggregation
print(df_target[df_target.isna().values])
# From this analysis, we see that 2 days don't have any observations in the
original data. Since this is a time-series, I will have to use an
appropriate method to impute this data.
# The method I will be using will be drift method to impute missing values.
I will perform this imputation after the train-test split only using the
train data since with a train test split of 0.2/0.25, only the training
data will have missing values.


# df_target contains only my dependent variable. I need to perform a
similar aggregation for my independent variables.
# Listing down my dependent variables below to see what type of aggregation
will be required for each:
# p (mbar) : The pascal SI derived unit of pressure used to quantify
internal pressure. Meteorological reports typically state - I can take an
average here.
# T (degC) : Temperature in Celsius - Target variable, already used.
# Tpot (K) : Temperature in Kelvin - Redundant variable, will be dropped.
# Tdew (degC) : Temperature in Celsius relative to humidity. Dew Point is a
measure of the absolute amount of water in the air. - Could be highly
correlated to temperature and/or have the same meaning as temperature. Will
have to research on this variable. After Research - The dew point is the
temperature the air needs to be cooled to (at constant pressure) in order
to achieve a relative humidity (RH) of 100%. This feature may be highly
correlated with relative humidity. Will do a correlation test later. Taking
an average of this feature will do.
# rh (%) : Relative Humidity is a measure of how saturated the air is with
water vapor, the %RH determines the amount of water. We can take an average
of this feature as well for a day.
```

```
# VPmax (mbar) : Saturation vapor pressure - Average
# VPact (mbar) : Vapor pressure - Average
# VPdef (mbar) : Vapor pressure deficit - Average
# sh (g/kg) : Specific humidity - Average. Might be correlated with
relative humidity.
# H2OC (mmol/mol) : Water vapor concentration - Average. Might be
correlated with specific humidity. Expressed in different units.
# rho (g/m**3) : Airtight - Average
# wv (m/s) : Wind speed - Average
# max. wv (m/s) : Maximum wind speed - Average
# wd (deg) : Wind direction in degrees - Average
# rain (mm) : Rain in milimetrers - Will have to take a sum of this feature
to aggregate for a day.
# raining (s) : Duration of rain in seconds - Will have to take a sum of
this feature to aggregate for a day.
# SWDR (W/m*2) : The definition of this features wasn't given with the
dataset. After research i found this to be the Solar Radiation. We can take
the Average of this as well.
# PAR (mol/m*2/s) : The definition of this features wasn't given with the
dataset. After research i found this to be the Photo Active Radiation. This
might be correlated with solar radiation. For now, will take an average of
this.
# max PAR (mol/m*2/s) : Maximum Photo Active Radiation. Will take an
average of this.
# Tlog (degC) : Log of Temperature in Degree Celsius - Redundant variable.
Will have to drop this.
# CO2 (ppm) : Carbon Dioxide parts per million - Average of the CO2 content
in a day.

print(df.columns)
df_features = df.drop(columns=['Tpot(K)','Tlog(degC)','T(degC)'],axis=1)

print(df_features.columns)

df_features = df_features.resample('D').agg({'p(mbar)': 'mean',
'Tdew(degC)': 'mean', 'rh(%)': 'mean', 'VPmax(mbar)': 'mean',
'VPact(mbar)': 'mean', 'VPdef(mbar)': 'mean', 'sh(g/kg)': 'mean',
'H2OC(mmol/mol)': 'mean', 'rho(g/m**3)': 'mean', 'wv(m/s)': 'mean',
'max.wv(m/s)': 'mean', 'wd(deg)': 'mean', 'rain(mm)': 'sum', 'raining(s)':
'sum', 'SWDR(W/m²)': 'mean', 'PAR(µmol/m²/s)': 'mean',
'max.PAR(µmol/m²/s)': 'mean', 'CO2(ppm)': 'mean'})

nan = df_features.isnull()
print(df_features[nan.any(axis=1)])

# This feature set also has only those 2 dates worth of missing data. Will
apply the same imputation to these time-series data.

# ---------------------Code no longer in use-------------------------------
---------------
# Prior to train-test split, I'll go ahead and save out these two
dataframes so that they can be used for broaded analyis in other files.
#---- DON'T RUN ----------------------------------------
# df_target.to_csv('Dataset/target_series.csv')
# df_features.to_csv('Dataset/feature_series.csv')
#-------------------------------------------------------------

# # Performing the train-test split here so that data imputation can be
performed prior to proceeding.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df_features, df_target,
```

```
shuffle=False, test_size=0.2, random_state=6313)
#
# # Making sure there aren't any null values in the test set
# print(X_test.isnull().all())
# print(y_test.isnull().all())
#
# # No null values. Will go ahead and save out the test data now
# #-----DON'T RUN-----------------------------
# # X_test.to_csv('Dataset/X_test.csv')
# # y_test.to_csv('Dataset/y_test.csv')
# #-----DON'T RUN-----------------------------
#
# # Confirming the presence of NA's in target train set
print(y_train[y_train.isna().any(axis=1)])
#
# # Data imputation for target train variable using drift method
Vals = np.linspace(y_train.iloc[0,0],y_train.iloc[-1,0],len(y_train))
Temp = pd.Series(Vals,name='T(degC)',index = y_train.index)
Temp = pd.DataFrame(Temp)
y_train = y_train.fillna(Temp)
#
# # Checking for NA's again to see if this worked
print(y_train.isna().sum())
#
# # Checking to see the whether the two known missing days have been
imputed '2016-10-26' & '2016-10-27'
# print(Temp.loc['2016-10-26':'2016-10-27'])
# print(y_train.loc['2016-10-26':'2016-10-27'])
#
# # Repeating the same for feature train set
# # Confirming the presence of NA's in feature train set
# print(X_train[X_train.isna().any(axis=1)])
#
# # Data imputation for feature train variables using drift method
# def linspace(column):
#     return np.linspace(column.iloc[0],column.iloc[-1],len(column))
#
# Temp1 = pd.DataFrame(X_train.apply(linspace, axis=0))
# X_train = X_train.fillna(Temp1)
#
# # Checking for NA's again to see if this worked
# print(X_train.isna().sum())
#
# # Checking to see the whether the two known missing days have been
imputed '2016-10-26' & '2016-10-27'
# print(Temp1.loc['2016-10-26':'2016-10-27'])
# print(X_train.loc['2016-10-26':'2016-10-27'])
#
# # No null values. Will go ahead and save out the train data now
# #-----DON'T RUN-----------------------------
# #X_train.to_csv('Dataset/X_train.csv')
# #y_train.to_csv('Dataset/y_train.csv')
# #-----DON'T RUN-----------------------------

# The following plot and subset of data highlighted problems with drift
method of data imputation
#
# # Plotting the newly aggregated target train data along with the drift
line used for imputation
fig, ax = plt.subplots(figsize=(16,8))
y_train['T(degC)'].plot(label="Daily time-series Data")
```

```
Temp['T(degC)'].plot(label="Interpolation Line")
plt.grid()
plt.legend()
plt.title("Plotting the daily aggregated time-series along with the
interpolation line used to impute missing values")
plt.xlabel("Time")
plt.ylabel("Temperature (degC)")
plt.show()

# print(X_train.loc['2016-10-20':'2016-10-30'])
# print(y_train.loc['2016-10-20':'2016-10-30'])
# --------------------------- Code above not in use-------------------------
-------------------------

# It seems like because of the size of the train set and the end value of
the temperature, we are unable to retrieve an accurate representation of
the data through a drift line. Imputation using drift may not be the best
choice.

# Instead I will do a simple linear interpolation for my dataset for the 2
missing dates.
df_target.interpolate(method='time',axis=0,inplace=True)
print(df_target.loc['2016-10-20':'2016-10-30'])
df_features.interpolate(method='time',axis=0,inplace=True)
print(df_features.loc['2016-10-20':'2016-10-30'].to_string())

# Saving out these clean sets of data of targets and features for analysis
in later files.
df_target.to_csv('Dataset/target_series.csv')
df_features.to_csv('Dataset/feature_series.csv')
```

Exploratory Data Analysis:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from toolbox import cal_autocorr, ADF_Cal, kpss_test, Cal_rolling_mean_var,
ACF_PACF_Plot, diff, order_one_diff
import seaborn as sns
from sklearn.model_selection import train_test_split
from statsmodels.tsa.seasonal import STL
import statsmodels.tsa.holtwinters as ets

# import os
# os.getcwd()

# Retrieving feature set and target set created from previous file.

df_target = pd.read_csv('Dataset/target_series.csv',index_col='DateTime')
df_features =
pd.read_csv('Dataset/feature_series.csv',index_col='DateTime')

# Making a time-series plot for target variable
fig, ax = plt.subplots(figsize=(16,8))
df_target['T(degC)'].plot(label="Daily time-series Data")
plt.grid()
plt.legend()
plt.title("Plotting the daily aggregated time-series")
plt.xlabel("Time")
plt.ylabel("Temperature (degC)")
plt.show()
```

```
# It appears stationary at first glance. Performing an ACF function and
plotting it.
# lags calculation = Number of observations/50 = 92 lags
# df_target.shape
lags = 92
# target = np.array(df_target)
ACF_PACF_Plot(df_target['T(degC)'],lags)
plt.show()

# The ACF-PACF plot of the time-series shows high correlation between the
lagged values. However, judging by the tail-off in ACF and cut-off in PACF
after lag = 6, this appears to be the charateristics of a typical AR
process. However, there does appear to be seasonality in the data judging
by the ACF plot, with mild peaks in the ACF values at regular intervals.
But figuring out the period of this seasonality is difficult due to overall
high correlation between the lags.
# The ACF plot seems to have a sinusoidal decay. with a seasonality period
of 365 (plotted for 500 lags)

# Doing a correlation matrix plot between all the variables
(target+features) in the dataset.
df = pd.concat((df_features,df_target),axis=1)
correlation_matrix = df.corr(method='pearson')
fig, ax = plt.subplots(figsize = (16,12))
sns.heatmap(correlation_matrix,annot=True,cmap='coolwarm',ax=ax,fmt='.2f',a
nnot_kws={"size": 35 /
np.sqrt(len(correlation_matrix)),"fontweight":'bold'},cbar=True)
cbar = ax.collections[0].colorbar
cbar.ax.tick_params(labelsize=35 /
np.sqrt(len(correlation_matrix)),width=2)
fig.suptitle('Correlation Matrix for all variables
(Pearson)',fontweight='bold',size=24)
plt.xticks(weight='bold')
plt.yticks(weight='bold')
plt.tight_layout()
plt.show()

# Here we see the presence of very strong multi-collinearity within
features, as well as high correlation between features and the target
variable.

# Performing a train-test split for further analysis.
X_train, X_test, y_train, y_test = train_test_split(df_features, df_target,
shuffle=False, test_size=0.2, random_state=6313)

# Saving out these train and test datasets for model building later on.
X_test.to_csv('Dataset/X_test.csv')
y_test.to_csv('Dataset/y_test.csv')
X_train.to_csv('Dataset/X_train.csv')
y_train.to_csv('Dataset/y_train.csv')

# Stationarity tests
Cal_rolling_mean_var(df_target["T(degC)"])
# The rolling mean and rolling mean variance tend to stabilize after all
the sample have been added to the calculation

ADF_Cal(df_target['T(degC)'])
# Stationary

kpss_test(df_target['T(degC)'])
# Stationary
```

```
# Looking at above ACF/PACF plots for stationarity we can see a clear AR
pattern, indicating that the process is stationary. However, the ACF seems
to decay very slowly suggesting that the process could be non-stationary,
but this is because of high correlation between the lags.

# Therefore, since process is stationary, no transformations are required.

# Time series decomposition
# Since the magnitude of my variance remains the same and is not changing,
I will consider an additive decomposition for my time series and not a
multiplicative one.

STL = STL(df_target["T(degC)"],period=365)
# Choosing period as 365 since the expectation with weather data is
seasonality of 365 days
res = STL.fit()
T = res.trend
S = res.seasonal
R = res.resid
Seasonally_adjusted = df_target['T(degC)'] - S
# Plotting seasonally adjusted data
fig, ax = plt.subplots(figsize=(16,8))
Seasonally_adjusted.plot(label="Daily time-series Data - Seasonally
adjusted")
plt.grid()
plt.legend()
plt.title("Plotting the daily aggregated time-series after seasonal
adjustment")
plt.xlabel("Time")
plt.ylabel("Temperature (degC)")
plt.show()
# No apparent trend, presence of residual makes it look like noise
trend_adjusted = df_target['T(degC)'] - T
# Plotting trend adjusted data
fig, ax = plt.subplots(figsize=(16,8))
trend_adjusted.plot(label="Daily time-series Data - Trend adjusted")
plt.grid()
plt.legend()
plt.title("Plotting the daily aggregated time-series after trend
adjustment")
plt.xlabel("Time")
plt.ylabel("Temperature (degC)")
plt.show()
# Presence of high seasonality
# Over all breakdown
fig = res.plot()
plt.show()
SOT = max(0,(1-((np.var(R))/(np.var(R+T)))))
print(f"The Strength of trend in the dataset is: {round(SOT,3)*100}%")
# Very less trended
SOS = max(0,(1-((np.var(R))/(np.var(R+S)))))
print(f"The Strength of seasonality in the dataset is:
{round(SOS,3)*100}%")
# Very highly seasonal

# Doubts - If Seasonal data but stationary, do we need to perform
differencing??? Do we need to remove seasonality before GPAC and model
building? Seasonal differencing removes seasonality but its to make time
series stationary, so if its already stationary, do we need to do
differencing??
```

```
# Back-transformation for differencing doubt. When we do differencing,
we're transforming the data so the forecast we make is on this transformed
data. Do we then require a back transformation on the forecasted values?

# For testing purpose, I will perform a 1 order seasonal differencing (365)
in the dataset
df_target_diff = diff(df_target,'T(degC)',365).copy()
# Maintaining original df by removing the new column
df_target.drop(columns=['T(degC)_365_Diff'],axis=1,inplace=True)

Cal_rolling_mean_var(df_target_diff['T(degC)_365_Diff'].dropna())
ADF_Cal(df_target_diff['T(degC)_365_Diff'].dropna())
kpss_test(df_target_diff['T(degC)_365_Diff'].dropna())
# The process still remains stationary judging by above tests, in fact
becomes more stationary based on ADF test statistic.
#ACF_PACF_Plot(df_target_diff['T(degC)_365_Diff'].dropna(),1200)
#plt.show()
# ACF-PACF plot display a much more cleaner trend with decay in ACF plot
and cut-off in PACF plot. This is a typical AR process behavior. This makes
my assumption that the seasonality is 365 correct.
# I can make use of this seasonally differenced dataset for modeling as
well since everything is right. However, one disadvantage is the loss of
data points due to the seasonal differencing of 365. I will first proceed
with my original dataset and then shift to this if that is not producing
good results. Train-test split pending for this differenced data.

# This will be the end for my Exploratory Data Analysis. Continued in pre-
processing and modeling python file

#ACF_PACF_Plot(df_target['T(degC)'],1500)
```

Preprocessing and Modeling:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from toolbox import cal_autocorr, ADF_Cal, kpss_test, Cal_rolling_mean_var,
ACF_PACF_Plot, diff,
Cal_GPAC,lm_param_estimate,autocorrelation,drift_forecast_test
import seaborn as sns
from sklearn.model_selection import train_test_split
from statsmodels.tsa.seasonal import STL
import statsmodels.tsa.holtwinters as ets
from sklearn.preprocessing import StandardScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
import statsmodels.tsa.api as smt
import scipy

X_train = pd.read_csv('Dataset/X_train.csv',index_col='DateTime')
X_test = pd.read_csv('Dataset/X_test.csv',index_col='DateTime')
y_train = pd.read_csv('Dataset/y_train.csv',index_col='DateTime')
y_test = pd.read_csv('Dataset/y_test.csv',index_col='DateTime')
df_target = pd.read_csv('Dataset/target_series.csv',index_col='DateTime')
df_features =
pd.read_csv('Dataset/feature_series.csv',index_col='DateTime')

# Holt-Winter Method
holt1 =
ets.ExponentialSmoothing(y_train['T(degC)'],trend='add',damped_trend=True,s
easonal='add',freq='D',seasonal_periods=365).fit()
```

```
holt2 =
ets.ExponentialSmoothing(y_train['T(degC)'],trend='add',seasonal='add',freq
='D',seasonal_periods=365).fit()
holt1f = holt1.forecast(steps=len(y_test))
holt2f = holt2.forecast(steps=len(y_test))
holt1f = pd.DataFrame(holt1f,columns=['Holt-Winter with
damping']).set_index(y_test.index)
holt2f = pd.DataFrame(holt2f,columns=['Holt-Winter without
damping']).set_index(y_test.index)

RMSE_hw1 =
np.sqrt(np.square(np.subtract(y_test.values,np.ndarray.flatten(holt1f.value
s))).mean())
RMSE_hw2 =
np.sqrt(np.square(np.subtract(y_test.values,np.ndarray.flatten(holt2f.value
s))).mean())
print("Root mean square error for Holt-Winter with damping is ",
RMSE_hw1,"\nAIC value of this model is",holt1.aic,"\nBIC value of this
model is",holt2.bic)
print("Root mean square error for Holt-Winter without damping is ",
RMSE_hw2,"\nAIC value of this model is",holt2.aic,"\nBIC value of this
model is",holt2.bic)

fig, ax = plt.subplots(figsize=(16,8))
y_test['T(degC)'].plot(ax=ax,label='Test Data')
holt1f['Holt-Winter with damping'].plot(ax=ax,label="with damping
(RMSE={:0.2f}, AIC={:0.2f})".format(RMSE_hw1, holt1.aic))
holt2f['Holt-Winter without damping'].plot(ax=ax,label="w/o damping
(RMSE={:0.2f}, AIC={:0.2f})".format(RMSE_hw2, holt2.aic))
plt.legend(loc='lower right')
plt.title(f'Holt-Winters Seasonal Smoothing')
plt.xlabel('Time')
plt.ylabel('Temperature (degC)')
plt.grid()
plt.show()

MAE_hw1 =
np.abs(np.subtract(y_test.values,np.ndarray.flatten(holt1f.values))).mean()
MAE_hw2 =
np.abs(np.subtract(y_test.values,np.ndarray.flatten(holt2f.values))).mean()
MAPE_hw1 =
np.abs(np.subtract(y_test.values,np.ndarray.flatten(holt1f.values))/y_test.
values).mean()
MAPE_hw2 =
np.abs(np.subtract(y_test.values,np.ndarray.flatten(holt2f.values))/y_test.
values).mean()
print(f"Root mean square error for Holt-Winter with damping is:
{round(RMSE_hw1,3)}\nAIC value of this model is: {round(holt1.aic,3)}\nBIC
value of this model is: {round(holt2.bic,3)}\nWith damping the Mean
Absolute Error is: {round(MAE_hw1,3)} and MAPE is: {round(MAPE_hw1,3)}")
print(f"Root mean square error for Holt-Winter without damping is:
{round(RMSE_hw2,3)}\nAIC value of this model is: {round(holt2.aic,3)}\nBIC
value of this model is: {round(holt2.bic,3)}\nWithout damping the Mean
Absolute Error: {round(MAE_hw2,3)} and MAPE is: {round(MAPE_hw2,3)}")

# The results of Holt-Winter are only partially satisfactory. The
forecasted values are more or less following the actual temperatures but
not completely able to cover the correct values.
# Also it remains uncertain as to whether damping positively affects the
model or not. As the MSE value seems to go up but the AIC value goes down
upon introducing damping.
```

```
# I'll evaluate the residuals of the fitted models for further clarity on
model bias.
cal_autocorr(holt1.resid,50,'Holt-Winter model (damped) residuals')
plt.show()
cal_autocorr(holt2.resid,50,'Holt-Winter model (w/o damp) residuals')
plt.show()
# Performing the ljung -box test to mathematically confirm non-whiteness of
the model.
test_results = sm.stats.diagnostic.acorr_ljungbox(holt1.resid, lags=[365])
print(test_results)
# P-value of 0 indicates that we can reject the null hypothesis that the
residual is white.

# We can clearly see that the ACF plot does not resemble an impulse,
indicating that all the correlations in the train data have not been
captured by these Holt-Winter models. Damping does not enhance or degrade
the performance of the fitted models as it is clear with the ACF plot for
residuals.

# Feature Selection/Elimination based on Multi-Collinearity
# From the previous analysis of correlation matrix, we did observe a high
degree of multi-collinearity between the features of the dataset. In this
section i'll try eliminate all such features that have multi collinearity
using SVD, condition number and VIF methods.

# Before performing the multi-collinearity tests, it will be a good idea to
standardise my dataset as the range for features differ drastically.
scaler = StandardScaler()
scaled_xtrain = scaler.fit_transform(X_train)
scaled_xtest = scaler.transform(X_test)
scaled_xtrain_df = pd.DataFrame(scaled_xtrain,columns=X_train.columns)

# Working only with train set to avoid leakage problems
H = np.dot(scaled_xtrain.T,scaled_xtrain)
s, d, v = np.linalg.svd(H)
print("SingularValues = ", d)

# From the SVD analysis, we see that a few singular values are very close
to 0 indicating the presence of multi-collinearity in the feature space.
The features causing these values need to be removed from the feature set.

# Q3b
Cond = np.linalg.cond(scaled_xtrain_df)
print("The Condition Number of the feature space is: ",round(Cond,3))
# From this we can see that the condition number is way above 1000
indicating a high degree of multi-collinearity in the dataset. This needs
to be removed.

# Next I will perform a VIF test to identify the features that result in
high multi-collinearity.
pd.options.display.float_format = '{:.4f}'.format
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(scaled_xtrain_df, i) for i
in range(scaled_xtrain_df.shape[1])]
vif["features"] = X_train.columns
print(vif)

# With this, we can see the presence of extremely large VIF values in few
variables. I'll proceed analysis by dropping the highest VIF value and then
running the VIF test again to see the new VIF values. First identifying
```

```
'VPmax(mbar)' as the column to drop.

scaled_xtrain_df.drop(columns='VPmax(mbar)',axis=1,inplace=True)
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(scaled_xtrain_df, i) for i
in range(scaled_xtrain_df.shape[1])]
vif["features"] = scaled_xtrain_df.columns
print(vif)

# Next identified variable is 'H2OC(mmol/mol)'
scaled_xtrain_df.drop(columns='H2OC(mmol/mol)',axis=1,inplace=True)
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(scaled_xtrain_df, i) for i
in range(scaled_xtrain_df.shape[1])]
vif["features"] = scaled_xtrain_df.columns
print(vif)

# Next identified variable is 'VPact(mbar)'
scaled_xtrain_df.drop(columns='VPact(mbar)',axis=1,inplace=True)
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(scaled_xtrain_df, i) for i
in range(scaled_xtrain_df.shape[1])]
vif["features"] = scaled_xtrain_df.columns
print(vif)

# Next identified variable is 'max.wv(m/s)'
scaled_xtrain_df.drop(columns='max.wv(m/s)',axis=1,inplace=True)
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(scaled_xtrain_df, i) for i
in range(scaled_xtrain_df.shape[1])]
vif["features"] = scaled_xtrain_df.columns
print(vif)

# Next identified variable is 'rho(g/m**3)'
scaled_xtrain_df.drop(columns='rho(g/m**3)',axis=1,inplace=True)
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(scaled_xtrain_df, i) for i
in range(scaled_xtrain_df.shape[1])]
vif["features"] = scaled_xtrain_df.columns
print(vif)

# Next identified variable is 'PAR(µmol/m²/s)'
scaled_xtrain_df.drop(columns='PAR(µmol/m²/s)',axis=1,inplace=True)
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(scaled_xtrain_df, i) for i
in range(scaled_xtrain_df.shape[1])]
vif["features"] = scaled_xtrain_df.columns
print(vif)

# Next identified variable is 'SWDR(W/m²)'
scaled_xtrain_df.drop(columns='SWDR(W/m²)',axis=1,inplace=True)
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(scaled_xtrain_df, i) for i
in range(scaled_xtrain_df.shape[1])]
vif["features"] = scaled_xtrain_df.columns
print(vif)

# Next identified variable is 'sh(g/kg)'
scaled_xtrain_df.drop(columns='sh(g/kg)',axis=1,inplace=True)
vif = pd.DataFrame()
```

```
vif["VIF Factor"] = [variance_inflation_factor(scaled_xtrain_df, i) for i
in range(scaled_xtrain_df.shape[1])]
vif["features"] = scaled_xtrain_df.columns
print(vif)

# After dropping this variable, I see that the VIF values all my features
are now less than 10. However, to make my model more robust, I will choose
a strict cut-off keeping VIF values less than 5 to avoid issues with
multicollinearity in my model.
# Therefore, the next identified variable is 'VPdef(mbar)'
scaled_xtrain_df.drop(columns='VPdef(mbar)',axis=1,inplace=True)
scaled_xtrain_df1 = scaled_xtrain_df.drop(columns='const',axis=1)
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(scaled_xtrain_df1, i) for i
in range(scaled_xtrain_df1.shape[1])]
vif["features"] = scaled_xtrain_df1.columns
print(vif)

# Now all my features have VIF value of less than 5 which indicates I've
dropped all the columns that cause multi-collinearity. Now I can proceed
with other analysis.

# Multiple linear regression analysis.
#-------------------------DECIDED NOT TO DO THIS-------------------------
------------------------
# Before fitting my new scaled train features into a multiple linear
regression model, I will perform a slight data augmentation to treat for
any outlier values that may affect my model predictions drastically. Since
my data is scaled to 0 mean and unit variance, any value less tha  -5 and
beyond 5 will be treated as an outlier that may skew my linear regression.
To deal with such values, I will clip these values to -5 and 5
respectively.
# First inspecting whether such values exist in my dataset
# print(np.min(scaled_xtrain_df))
# print(np.max(scaled_xtrain_df))
# # 3 columns seems to have high outlier values
#
# # Performing data augmentation
# # scaled_xtrain_df = np.clip(scaled_xtrain_df, -5, 5)
# # Checking to see if augmentation was successful
# print(np.min(scaled_xtrain_df))
# print(np.max(scaled_xtrain_df))
# # It worked.
# The training data has been augmented.
#---------------------------------NOT RUNNING ABOVE-----------------------
-------------

# Before proceeding, I will also drop the same columns in X_test that i
dropped from my X_train so that i can make predictions later on
scaled_xtest_df = pd.DataFrame(scaled_xtest,columns=X_train.columns)
scaled_xtest_df.drop(columns=['VPmax(mbar)','H2OC(mmol/mol)','VPact(mbar)',
'max.wv(m/s)','rho(g/m**3)','PAR(mol/m/s)','SWDR(W/m)','sh(g/kg)','VPd
ef(mbar)'],axis=1,inplace=True)

# I am deciding not to standard scale my y_train and y_test as a matter of
choice. Expectation is that the estimated parameters will be sufficiently
large to reproduce the required target values.

# Now fitting an OLS model to my data
scaled_xtrain_df = sm.add_constant(scaled_xtrain_df) # To add constant
intercept
```

```
y_train_df = y_train.reset_index()
y_train_df = y_train_df.drop(columns='DateTime',axis=1)
model = sm.OLS(y_train_df,scaled_xtrain_df).fit()
print(model.params)
print(model.summary())

# I get a very good adjusted R-squared value of 99.7%. Using the features
in my training data, I'm able to explain 99.7% of the variability in my
target.
# However, looking closely at all the parameter values, I do see that 3
parameters have a high P-value > 0.05 and the confidence interval indicates
that these parameters can have a coefficient of 0 as well. Therefore, I'll
remove the first highest P-value and rebuild my model to see the effect on
AIC, BIC and Adjusted-R Squared value.
# The first identified variable is 'CO2(ppm)'
scaled_xtrain_df1 = scaled_xtrain_df.drop(columns='CO2(ppm)',axis=1)
model1 = sm.OLS(y_train_df,scaled_xtrain_df1).fit()
print(model1.params)
print(model1.summary())
# The AIC has gone down along with BIC indicating a better performance,
there appears to be no change in adjusted r-squared value. I'll retain
dropping this feature.
# Another feature that still has an insiginifcant p-value and confidence
interval containing a 0 is 'wv(m/s)'. I'll proceed with dropping this
feature as well.
scaled_xtrain_df2 = scaled_xtrain_df1.drop(columns='wv(m/s)',axis=1)
model2 = sm.OLS(y_train_df,scaled_xtrain_df2).fit()
print(model2.params)
print(model2.summary())
# The AIC has gone down along with BIC indicating a better performance,
there appears to be no change in adjusted r-squared value. I'll retain
dropping this feature.
# Another feature that still has an insiginifcant p-value and confidence
interval containing a 0 is 'rain(mm)'. I'll proceed with dropping this
feature as well.
scaled_xtrain_df3 = scaled_xtrain_df2.drop(columns='rain(mm)',axis=1)
model3 = sm.OLS(y_train_df,scaled_xtrain_df3).fit()
print(model3.params)
print(model3.summary())
# After removing this feature, the model performance has improved further.
The AIC has decreased and BIC has dropped as well. Adjusted R-squared value
remains the same indicating this model performs well! All the parameters
are also withinin the significant range. Additionnally, the F-test
statistic indicates that we can reject the null hypothesis that this model
and an intercept only model perform the same indicating this model with the
given features performs much better. I'll consider this model to be my
final model.

# Removing the same features from my test set to perform forecasts.
scaled_xtest_df = sm.add_constant(scaled_xtest_df)
# Dropping the columns based on above analysis in test set as well.
scaled_xtest_df =
scaled_xtest_df.drop(columns=['CO2(ppm)','rain(mm)','wv(m/s)'],axis=1)
predictions = model3.predict(scaled_xtest_df)

predictions.index = y_test.index
fig, ax = plt.subplots(figsize = (16,8))
predictions.plot(ax=ax,label="Predicted T(degC)")
y_test['T(degC)'].plot(ax=ax,label="Actual T(degC)",linestyle='--')
plt.legend(loc='lower right')
plt.grid()
```

```
plt.title('OLS Model - Forecast vs Actual T(degC)')
plt.xlabel('Date')
plt.ylabel('Temperature')
plt.show()

# Looking at the model performance
RMSE_ols =
np.sqrt(np.square(np.subtract(y_test['T(degC)'],predictions)).mean())
MAE_ols = np.abs(np.subtract(y_test['T(degC)'],predictions)).mean()
MAPE_ols =
(np.abs(np.subtract(y_test['T(degC)'],predictions))/np.abs(y_test['T(degC)'
])).mean()
print(f"The RMSE of this model is: {round(RMSE_ols,3)}\nThe MAE of this
model is: {round(MAE_ols,3)}\nThe MAPE os this model is:
{round(MAPE_ols,3)}")
# According to this metric, the model is very performing well on the unseen
test set.

# Testing out the whiteness of residuals, first visually
residuals = model3.resid
cal_autocorr(residuals,62,'Residual ACF for OLS Model')
plt.show()
# Research: Looking at this ACF plot, we can say that the correlations
within the target series have not been fully captured by this OLS model
since the residuals are not white. Possible reason behind this is
potentially the dataset i'm working with is non-linear and yet i'm trying
to fit into a linear model which may cause the residuals to not be white.
Alternatively, it could also be due the fact that my data has high
autocorrelation which is not being accounted for by this linear model.

# Find out the Q-value and performing box-pierce test
Ry = autocorrelation(residuals,60)
Q = len(residuals) * np.sum(np.square(Ry[60+1:]))
DOF = 60 - 7
alfa = 0.01
chi_critical = scipy.stats.chi2.ppf(1 - alfa, DOF)
print(f"Q is {Q} and chi critical is {chi_critical}")
if Q < chi_critical:
    print("The residual is white ")
else:
    print("The residual is NOT white ")

# Performing the Chi-square test using ljung-box test to mathematically
find out the whiteness of the residuals.
test_results = sm.stats.diagnostic.acorr_ljungbox(residuals, lags=[1])
print(test_results)
# Now checking for lag for 365 since the data has seasonality of 365
test_results1 = sm.stats.diagnostic.acorr_ljungbox(residuals, lags=[365])
print(test_results1)
# Both these test results state that we can reject the null hypothesis that
the residuals come from an independently and identical distibution and
hence reinforces our visual observation that the residuals are not white.

# The mean and variance of the residuals
print(f"The mean of the residuals is: {np.mean(residuals)}")
print(f"The variance of the residuals is: {np.var(residuals)}")
# the residuals do seem to have a mean of 0 and constant variance of 0.155
which is desirable, however, the residuals are not white indicating the
presence of uncaptured correlations in the target series using a simple OLS
model.
```

```
# Building the base models now - Average, Drift, Naive, SES using original
y_train and y_test
# Average method
y_pred1 = np.mean(y_train.values)
y_test_avg = pd.DataFrame(np.array([y_pred1]*len(y_test)),index =
y_test.index,columns = ["Average Forecast"])
y_pred_avg = np.array([y_pred1]*len(y_test))
error_avg = y_test.values.ravel() - y_pred_avg
RMSE_avg = np.sqrt(np.square(error_avg).mean())
MAE_avg = np.abs(error_avg).mean()
MAPE_avg = np.abs(y_test.values.ravel() - y_pred_avg/y_test.values).mean()
print(f"The RMSE of Average model is: {round(RMSE_avg,3)}\nThe MAE of this
model is: {round(MAE_avg,3)}\nThe MAPE of this model is:
{round(MAPE_avg,3)}")
# Plotting the Average forecast vs the test set.
fig, ax = plt.subplots(figsize = (16,8))
y_test_avg["Average Forecast"].plot(ax=ax,label="Average Forecast")
y_test['T(degC)'].plot(ax=ax,label="Actual T(degC)")
plt.legend(loc='lower right')
plt.grid()
plt.title('Average Forecast Model - Forecast vs Actual T(degC)')
plt.xlabel('Date')
plt.ylabel('Temperature (degC)')
plt.show()

# Naive method
y_pred2 = y_train.values[-1]
y_pred_naive = np.array([y_pred2]*len(y_test))
y_test_naive = pd.DataFrame(y_pred_naive,index = y_test.index,columns =
["Naive Forecast"])
error_naive = y_test.values.ravel() - y_pred_naive.ravel()
RMSE_naive = np.sqrt(np.square(error_naive).mean())
MAE_naive = np.abs(error_naive).mean()
MAPE_naive = np.abs(y_test.values.ravel() -
y_pred_naive.ravel()/y_test.values).mean()
print(f"The RMSE of Naive model is: {round(RMSE_naive,3)}\nThe MAE of this
model is: {round(MAE_naive,3)}\nThe MAPE of this model is:
{round(MAPE_naive,3)}")
# Plotting the Naive forecast vs the test set.
fig, ax = plt.subplots(figsize = (16,8))
y_test_naive["Naive Forecast"].plot(ax=ax,label="Naive Forecast")
y_test['T(degC)'].plot(ax=ax,label="Actual T(degC)")
plt.legend(loc='lower right')
plt.grid()
plt.title('Naive Forecast Model - Forecast vs Actual T(degC)')
plt.xlabel('Date')
plt.ylabel('Temperature (degC)')
plt.show()

# Drift method
y_pred3 = drift_forecast_test(y_train.values.ravel(),len(y_test))
y_test_drift = pd.DataFrame(np.array(y_pred3),index = y_test.index,
columns=["Drift Forecast"])
y_pred_drift = np.array([y_pred3]).reshape(-1,1).ravel()
error_drift = y_test.values.ravel() - y_pred_drift.ravel()
RMSE_drift = np.sqrt(np.square(error_drift).mean())
MAE_drift = np.abs(error_drift).mean()
MAPE_drift = np.abs(y_test.values.ravel() -
y_pred_drift.ravel()/y_test.values).mean()
print(f"The RMSE of Drift model is: {round(RMSE_drift,3)}\nThe MAE of this
```

```
model is: {round(MAE_drift,3)}\nThe MAPE of this model is:
{round(MAPE_drift,3)}")
fig, ax = plt.subplots(figsize = (16,8))
y_test_drift["Drift Forecast"].plot(ax=ax,label="Drift Forecast")
y_test['T(degC)'].plot(ax=ax,label="Actual T(degC)")
plt.legend(loc='lower right')
plt.grid()
plt.title('Drift Forecast Model - Forecast vs Actual T(degC)')
plt.xlabel('Date')
plt.ylabel('Temperature (degC)')
plt.show()


# Simple and Exponential Smoothing
SES_model =
ets.ExponentialSmoothing(y_train,trend=None,damped=False,seasonal=None).fit
()
y_pred_ses = SES_model.forecast(steps=len(y_test))
y_test_ses = pd.DataFrame(y_pred_ses,index = y_test.index, columns=["SES
Forecast"])
error_ses = y_test.values.ravel() - y_pred_ses.values.ravel()
RMSE_ses = np.sqrt(np.square(error_ses).mean())
MAE_ses = np.abs(error_ses).mean()
MAPE_ses = np.abs(y_test.values.ravel() -
y_pred_ses.values.ravel()/y_test.values).mean()
print(f"The RMSE of SES model is: {round(RMSE_ses,3)}\nThe MAE of this
model is: {round(MAE_ses,3)}\nThe MAPE of this model is:
{round(MAPE_ses,3)}")
fig, ax = plt.subplots(figsize = (16,8))
y_test_ses["SES Forecast"].plot(ax=ax,label="SES Forecast")
y_test['T(degC)'].plot(ax=ax,label="Actual T(degC)")
plt.legend(loc='lower right')
plt.grid()
plt.title('SES Forecast Model - Forecast vs Actual T(degC)')
plt.xlabel('Date')
plt.ylabel('Temperature (degC)')
plt.show()

# Starting the ARIMA/SARIMA model development
# Although initially i had considered my data to be stationary and
determined that no differencing needs to be applied. Looking at the
ACF/PACF plot of the data again, there appears to be high seasonality in
the data. This is also reinforced by the Strength of seasonality using STL
decomposition previously. A consistent pattern in the ACF/PACF suggests
seasonality. As a consequence of this observation, instead of the ARIMA
model, I will investigate the SARIMA model. There is no need for non-
seasonal differencing since the data is already stationary. However, before
proceeding now (based on several research papers referenced:
https://article.sciencepublishinggroup.com/pdf/10.11648.j.ijema.20210906.17
.pdf, I will perform the seasonal differencing to eliminate this high
seasonality. This is because high correlations in the time series can make
it difficult to build accurate ARIMA models because the long-term
dependencies in the time series are difficult to capture using only a few
lags.

# Hence, i need to perform order one seasonal differencing (seasonality
index as 365 as decided before) here before proceeding and then do my
train-test spit. I will have to trade-off my loss of datapoints before
proceeding.

y_train_diff = diff(y_train,'T(degC)',365).copy()
```

```
# Maintaining original df by removing the new column
y_train.drop(columns=['T(degC)_365_Diff'],axis=1,inplace=True)
# Dropping the previous target column from new dataframe along with the
null rows introduced after differencing
y_train_diff.drop(columns='T(degC)',axis=1,inplace=True)
y_train_diff = y_train_diff.dropna()
# From prior analysis, i already know that this new differenced time series
is also stationary, infact it is more stationary as indicated by ADF test.
Now i'll proceed with the train-test split and further analysis.

# target_train, target_test = train_test_split(df_target_diff,
shuffle=False, test_size=0.2, random_state=6313)
ry = sm.tsa.stattools.acf(y_train_diff['T(degC)_365_Diff'].values,
nlags=100)
ryy = ry[::-1]
Ry = np.concatenate((ryy, ry[1:]))
Cal_GPAC(Ry,10,10)

# Keeping the ACF/PACF plot handy
ACF_PACF_Plot(y_train_diff['T(degC)_365_Diff'].values,100)
# Judging by just the ACF/PACF, i can guess that my AR only process order
is 3 since the PACF cuts off after 3 lags. But i'll consider GPAC for time
being

# A preliminary order i'm deciding to select is ARMA(1,0); Another possible
order i'll select is (3,0)
lm_param_estimate(y_train_diff,1,0)
# Algorithm converges very quickly
lm_param_estimate(y_train_diff,3,0)
# Algorithm converges very quickly

# ARIMA(1,0,0)
arima_model1 = sm.tsa.arima.ARIMA(y_train_diff,order=(1, 0,
0),trend='n',freq='D').fit()
print(arima_model1.summary())
model_hat1 = arima_model1.predict(start=0, end=len(y_train_diff) - 1)
e1 = y_train_diff.reset_index()['T(degC)_365_Diff'] -
model_hat1.reset_index()['predicted_mean']
Re1 = autocorrelation(np.array(e1), 100)
ACF_PACF_Plot(e1,100)
cal_autocorr(e1,100,"Residuals with ARIMA(1,0,0)xSARIMA(0,1,0,365)")
plt.show()
Q = len(e1) * np.sum(np.square(Re1[100+1:]))
DOF = 100 - 1 - 0
alfa = 0.01
chi_critical = scipy.stats.chi2.ppf(1 - alfa, DOF)
print(f"Q is {Q} and chi critical is {chi_critical}")
if Q < chi_critical:
    print("The residual is white ")
else:
    print("The residual is NOT white ")
y_train_diff.index = pd.to_datetime(y_train_diff.index)
model_hat1.index = y_train_diff.index
fig, ax = plt.subplots(figsize=(16,8))
y_train_diff['T(degC)_365_Diff'].plot(ax=ax,label="True data")
model_hat1.plot(ax=ax,label="Fitted data")
plt.xlabel("Samples")
plt.ylabel("Magnitude")
plt.legend()
plt.title(" Train vs One-Step Prediction - ARIMA(1,0,0)xSARIMA(0,1,0,365)")
plt.tight_layout()
```

```
plt.show()
# Judging by the residual analysis, we don't get a white residual hence new
order must be considered.

# ARIMA(3,0,0)
arima_model2 = sm.tsa.arima.ARIMA(y_train_diff,order=(3, 0,
0),trend='n',freq='D').fit()
print(arima_model2.summary())
model_hat2 = arima_model2.predict(start=0, end=len(y_train_diff) - 1)
e2 = y_train_diff.reset_index()['T(degC)_365_Diff'] -
model_hat2.reset_index()['predicted_mean']
test_results_sarima = sm.stats.diagnostic.acorr_ljungbox(e2, lags=[25])
Re2 = autocorrelation(np.array(e2),100)
ACF_PACF_Plot(e2,100)
cal_autocorr(e2,100,"Residuals with ARIMA(3,0,0)xSARIMA(0,1,0,365)")
plt.show()
Q = len(e2) * np.sum(np.square(Re2[100+1:]))
DOF = 100 - 3 - 0
alfa = 0.01
chi_critical = scipy.stats.chi2.ppf(1 - alfa, DOF)
print(f"Q is {Q} and chi critical is {chi_critical}")
if Q < chi_critical:
    print("The residual is white ")
else:
    print("The residual is NOT white ")
model_hat2.index = y_train_diff.index
fig, ax = plt.subplots(figsize=(16,8))
y_train_diff['T(degC)_365_Diff'].plot(ax=ax,label="True data")
model_hat2.plot(ax=ax,label="Fitted data")
plt.xlabel("Samples")
plt.ylabel("Magnitude")
plt.legend()
plt.title(" Train vs One-Step Prediction - ARIMA(3,0,0)xSARIMA(0,1,0,365)")
plt.tight_layout()
plt.show()

# With the residual anlaysis for this model, i can see that the residuals
have become white. Hence the final model that i will select is
ARIMA(3,0,0)xSARIMA(0,1,0,365). Since the model was built on top of
seasonally differenced data, there is a component of SARIMA as well.
lm_param_estimate(y_train_diff,3,0)
# The estimated parameter match between the SARIMAX model and custom-
developed lm_algorithm_estimation
residual_variance = np.var(e2)
forecast_values = arima_model2.forecast(steps=len(y_train_diff))
# The forecasted values that i've gotten from this model are transformed.
I'll have to back-transform these values to compare with my test set.
y_test_orig = []
Num_observations = len(y_train_diff)
s = 365
for i in range(len(y_test)):
    if i < s:
        y_test_orig.append(forecast_values[i] + y_train.iloc[- s + i])
    else:
        y_test_orig.append(forecast_values[i] + y_test_orig[i - s])

y_test_orig = np.array(y_test_orig).ravel()
e_forecast = y_test.reset_index()['T(degC)'].ravel() - y_test_orig
forecast_variance = np.var(e_forecast)
y_testdf = pd.DataFrame(y_test_orig.reshape(-
1,1),columns=["Forecast"],index=y_test.index)
```

```
fig, ax = plt.subplots(figsize=(16,8))
y_test['T(degC)'].plot(ax=ax,label='Test Data')
y_testdf.plot(ax=ax,label="Forecast")
plt.legend(loc='lower right')
plt.title(f'ARIMA(3,0,0)xSARIMA(0,1,0,365) Model Results')
plt.xlabel('Time')
plt.ylabel('Temperature (degC)')
plt.grid()
plt.show()

Generalization = forecast_variance/residual_variance
print(round(Generalization,3))

# Plotting the distribution of the residual error to assess bias in the
model
e2.hist()
plt.title("Residual Histogram for ARIMA(3,0,0)xSARIMA(0,1,0,365) Model")
plt.ylabel("Frequency")
plt.xlabel("Error Value")
plt.show()

# Calculating the Model performance through forecast RMSE
RMSE_sarima = np.sqrt(np.square(e_forecast).mean())
MAE_sarima = np.abs(e_forecast).mean()
MAPE_sarima = np.abs(e_forecast/y_test.values.ravel()).mean()
print(f"The RMSE of ARIMA(3,0,0)xSARIMA(0,1,0,365) is:
{round(RMSE_sarima,3)}\nThe MAE of ARIMA(3,0,0)xSARIMA(0,1,0,365) is:
{round(MAE_sarima,3)}\nThe MAPE of ARIMA(3,0,0)xSARIMA(0,1,0,365) is:
{round(MAPE_sarima,3)}")

# Developing the custom forecast function for SARIMA model
# Retrieving the model parameters from the model summary
lm_param_estimate(y_train_diff,3,0) #[-1.011398710003359,
0.2983713542290045, -0.09062909533692821]
arima_model2.summary() # ar.L1 1.0111     ar.L2 -0.2981   ar.L3 0.0905
# The coefficients obtained from my custom model and package are similar.
I'll utilize the package coefficients as my final model paramter.
def custom_forecast_function(data,Step):
    y_hat = []
    for i in range(1, Step+1):
        if i == 1:
            y_hat.append(1.0111 * data.values.ravel().tolist()[-1] - 0.2981
* data.values.ravel().tolist()[-2] + 0.0905 *
data.values.ravel().tolist()[-3])
        elif i == 2:
            y_hat.append(1.0111 * y_hat[0] - 0.2981 *
data.values.ravel().tolist()[-1] + 0.0905 * data.values.ravel().tolist()[-
2])
        elif i == 3:
            y_hat.append(1.0111 * y_hat[1] - 0.2981 * y_hat[0] + 0.0905 *
data.values.ravel().tolist()[-1])
        else:
            y_hat.append(1.0111 * y_hat[i-2] - 0.2981 * y_hat[i-3] + 0.0905
* y_hat[i-4])
    y_hat = np.array(y_hat)
    return y_hat

# Testing custom function to forecast values
custom_forecast_values = custom_forecast_function(y_train_diff,len(y_test))
# These forecasts are made on the transformed data. I'll have to back
```

```
transform this to perform a comparison with original test data. Utilizing
the reverse transformation used above.

y_test_orig_cust = []
s = 365
for i in range(len(y_test)):
    if i < s:
        y_test_orig_cust.append(custom_forecast_values[i] + y_train.iloc[-
s + i])
    else:
        y_test_orig_cust.append(custom_forecast_values[i] +
y_test_orig_cust[i - s])

# Plotting this against test set
y_test_orig_cust = np.array(y_test_orig_cust).ravel()
e_forecast = y_test.reset_index()['T(degC)'].ravel() - y_test_orig_cust
y_testdf_cust = pd.DataFrame(y_test_orig_cust.reshape(-
1,1),columns=["Custom Forecast"],index=y_test.index)

fig, ax = plt.subplots(figsize=(16,8))
y_test['T(degC)'].plot(ax=ax,label='Test Data')
y_testdf_cust.plot(ax=ax,label="Forecast")
plt.legend(loc='lower right')
plt.title(f'ARIMA(3,0,0)xSARIMA(0,1,0,365) Model Results')
plt.xlabel('Time')
plt.ylabel('Temperature (degC)')
plt.grid()
plt.show()

# Base model Comparisons with SARIMA model

MAPE_avg = np.abs(y_test.values.ravel() - y_pred_avg/y_test.values).mean()
print(f"The RMSE of Average model is: {RMSE_avg}\nThe MAE of this model is:
{MAE_avg}\nThe MAPE of this model is: {MAPE_avg}")
# Plotting the Average forecast vs the test set.
fig, ax = plt.subplots(3,2,figsize = (16,8))
y_test_avg["Average Forecast"].plot(ax=ax[0,0],label="Average Forecast")
y_test['T(degC)'].plot(ax=ax[0,0],label="Actual T(degC)")
ax[0,0].legend(loc='lower right')
ax[0,0].grid()
ax[0,0].set_title('Average Forecast Model - Forecast vs Actual T(degC)')
ax[0,0].set_xlabel('Date')
ax[0,0].set_ylabel('Temperature (degC)')
y_test_naive["Naive Forecast"].plot(ax=ax[0,1],label="Naive Forecast")
y_test['T(degC)'].plot(ax=ax[0,1],label="Actual T(degC)")
ax[0,1].legend(loc='lower right')
ax[0,1].grid()
ax[0,1].set_title('Naive Forecast Model - Forecast vs Actual T(degC)')
ax[0,1].set_xlabel('Date')
ax[0,1].set_ylabel('Temperature (degC)')
y_test_drift["Drift Forecast"].plot(ax=ax[1,0],label="Drift Forecast")
y_test['T(degC)'].plot(ax=ax[1,0],label="Actual T(degC)")
ax[1,0].legend(loc='lower right')
ax[1,0].grid()
ax[1,0].set_title('Drift Forecast Model - Forecast vs Actual T(degC)')
ax[1,0].set_xlabel('Date')
ax[1,0].set_ylabel('Temperature (degC)')
y_test_ses["SES Forecast"].plot(ax=ax[1,1],label="SES Forecast")
y_test['T(degC)'].plot(ax=ax[1,1],label="Actual T(degC)")
ax[1,1].legend(loc='lower right')
ax[1,1].grid()
```

```
ax[1,1].set_title('SES Forecast Model - Forecast vs Actual T(degC)')
ax[1,1].set_xlabel('Date')
ax[1,1].set_ylabel('Temperature (degC)')
y_test['T(degC)'].plot(ax=ax[2,1],label='Actual T(degC)')
y_testdf['Forecast'].plot(ax=ax[2,1],label="SARIMA Forecast")
ax[2,1].legend(loc='lower right')
ax[2,1].set_title(f'ARIMA(3,0,0)xSARIMA(0,1,0,365) Model Results')
ax[2,1].set_xlabel('Time')
ax[2,1].set_ylabel('Temperature (degC)')
ax[2,1].grid()
fig.suptitle("Base Model Comparison with SARIMA Model")
plt.tight_layout()
plt.show()

# Plotting the metrics in a table.
from tabulate import tabulate
basemodel_table = [['Model Type', 'Root-Mean-Square-Error','Mean-Absolute-
Error','Mean-Absolute-Percent-Error'],
         ["Average Method
Forecast",round(RMSE_avg,3),round(MAE_avg,3),round(MAPE_avg,3)],["Naive
Method
Forecast",round(RMSE_naive,3),round(MAE_naive,3),round(MAPE_naive,3)],["Dri
ft Method
Forecast",round(RMSE_drift,3),round(MAE_drift,3),round(MAPE_drift,3)],["SES
Method
Forecast",round(RMSE_ses,3),round(MAE_ses,3),round(MAPE_ses,3)],["SARIMA
Model
Forecast",round(RMSE_sarima,3),round(MAE_sarima,3),round(MAPE_sarima,3)]]
print(tabulate(basemodel_table,headers='firstrow', tablefmt =
'fancy_grid'))

# We can see the SARIMA Model significantly outperforms the base models
that we have.
```

Deep Learning and Final Model Selections:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from tensorflow.keras import Sequential
# from keras.layers import CuDNNLSTM
from tensorflow.keras.layers import Dense, LSTM, Dropout  # ,CuDNNLSTM
from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator
from tensorflow.keras.optimizers import Adam
from toolbox import autocorrelation, cal_autocorr
import statsmodels.api as sm
from tabulate import tabulate

df_target = pd.read_csv('Dataset/target_series.csv',index_col='DateTime')
df_features =
pd.read_csv('Dataset/feature_series.csv',index_col='DateTime')
df = pd.concat((df_features,df_target),axis=1)
data = df.copy()

scaler1 = StandardScaler()
scaled_data = scaler1.fit_transform(data.values)
df_temp = df['T(degC)'].values
dataset = data.values
training_data_len = np.math.ceil(len(df_temp) *.8)
train_data = scaled_data[0:training_data_len,:]
```

```
x_train = []
y_train = []
x_test = []
y_test = []
n_past = len(dataset) - training_data_len

for i in range(n_past,len(train_data)):
    x_train.append(train_data[i-365:i,0:train_data.shape[1]-1])
    y_train.append(train_data[i,train_data.shape[1]-1])

x_train, y_train = np.array(x_train), np.array(y_train)
print(f'trainX shape = {x_train.shape}')
print(f'trainY shape = {y_train.shape}')

model = Sequential()
model.add(LSTM(64,
return_sequences=True,activation='relu',input_shape=(x_train.shape[1],x_tra
in.shape[2])))
model.add(LSTM(50,return_sequences=False))
model.add(Dropout(.2))
model.add(Dense(1))
model.compile(optimizer='adam',loss='mse')
model.summary()
history = model.fit(x_train,y_train, batch_size=16, validation_split = .1,
epochs=10, verbose=1)

plt.figure()
plt.plot(history.history['loss'], 'r',label='Training loss')
plt.plot(history.history['val_loss'], 'b',label='Validation loss')
plt.legend()
plt.title("Training vs Validation loss for LSMT Model in Temperature
forecasting")
plt.ylabel("Loss Values")
plt.xlabel("Epochs")
plt.show()

test_data = scaled_data[training_data_len-365:,:]
x_test = []
y_test = dataset[training_data_len:,-1]

for i in range(365,len(test_data)):
    x_test.append(test_data[i-365:i,0:18])

x_test = np.array(x_test)
predictions = model.predict(x_test)
forecast_copies = np.repeat(predictions, 19, axis=-1)
predictions = scaler1.inverse_transform(forecast_copies)[:,-1]
#
train = data.iloc[:training_data_len]
test = data.iloc[training_data_len:]
test["Predictions"] = predictions

fig = plt.figure(figsize=(16,8))
ax = fig.add_subplot(111)
ax.set_title("Temperature prediction using LSTM network",fontsize=18)
ax.set_xlabel("Date",fontsize=18)
ax.set_ylabel("Temperature (degC)", fontsize=18)
plt.rcParams['axes.prop_cycle'] = plt.cycler(color=['r','b','c'])
test["T(degC)"].plot(ax=ax,color='red',label="Test")
test["Predictions"].plot(ax=ax,color='black',label="Predictions")
```

```
ax.legend(["Test Set", "Predictions"], loc = "lower right", fontsize=18)
ax.grid()
plt.show()

e_forecast = test.reset_index()['T(degC)'] -
test.reset_index()['Predictions']
RMSE_lstm = np.sqrt(np.square(e_forecast).mean())
MAE_lstm = np.abs(e_forecast).mean()
MAPE_lstm = np.abs(e_forecast/test['T(degC)'].values.ravel()).mean()
print(f"The RMSE of LSTM Model is: {round(RMSE_lstm,3)}\nThe MAE of LSTM
Model is: {round(MAE_lstm,3)}\nThe MAPE of LASTM Model is:
{round(MAPE_lstm,3)}")

# Residual diagnostic
fitted_values = model.predict(x_train)
fitted_copies = np.repeat(fitted_values, 19, axis=-1)
fit_values = scaler1.inverse_transform(fitted_copies)[:,-1]

y_train_orig_copies = np.repeat(y_train.reshape(-1,1), 19, axis=-1)
y_train_orig = scaler1.inverse_transform(y_train_orig_copies)[:,-1]

e_residuals = y_train_orig - fit_values
np.var(e_residuals)
np.mean(e_residuals)

# Plotting the distribution of the residual error to assess bias in the
model
pd.DataFrame(e_residuals).hist()
plt.title("Residual Histogram for LSTM Model")
plt.ylabel("Frequency")
plt.xlabel("Error Value")
plt.show()

# It looks normally distributed. Plotting the ACF of the curve and then
performing the ljung-box test to test whiteness.
cal_autocorr(e_residuals,50,"ACF plot for residuals of LSTM Model")
plt.show()
# It's hard to say whether the residuals are white or not visually because
of lag = 1 values
test_results = sm.stats.diagnostic.acorr_ljungbox(e_residuals, lags=[365])
print(test_results)
# Based on the ljung box-test we cannot reject the null hypothesis that the
residuals are white. However, visually lag 1 seems uncertain.

# FINAL MODEL SELECTION -
# NOTE: This section is used to perform a comparative analysis across all
models built till now. It utilizes variables across different files and
might not run if performed on the same console.

# Apart from the base models, which are only used for benchmarking, the
final models i'll compare the performance on this dataset are:
# 1. Holt-winter
# 2. Multiple linear regression
# 3. SARIMA model
# 4. LSTM Model
# Within these models, i will select the best model that i found to fit my
data as i built multiple variations within each model type as well.

# The key criteria in selecting the final model will be the model that
captures all the correlation within my target series. This is judged based
on the residual diagnostic tests. I'll evaluate the residuals of each model
```

```
and determine the whiteness in each model. This way, we'll find the best
model that represents the underlying the data well.
# This whiteness test has been performed for various models already while
building them, I'll aggregate the results in this section.

residuals_ols = model3.resid
fig, ax = plt.subplots(2,2,figsize=(16,8))
cal_autocorr(holt1.resid,60,'Residuals for Holt-Winter model',ax=ax[0,0])
cal_autocorr(residuals_ols,60,'Residuals for Multiple Linear Regression
Model',ax=ax[0,1])
cal_autocorr(e2,60,"Residuals for
ARIMA(3,0,0)xSARIMA(0,1,0,365)",ax=ax[1,0])
cal_autocorr(e_residuals,60,"Residuals for LSTM Model",ax=ax[1,1])
fig.suptitle("Residual Diagnostics for all Final Models",size=16)
plt.tight_layout()
plt.show()

# box-test and ljung box test has been performed separately for all the
models. Accordingly, the only white residuals are present in SARIMA, and
LSTM model.
# However, Sarima model residuals appear to be more white than LSTM
(visually)

# Next i'll plot the performance of the model on the test set in subplots.
fig, ax = plt.subplots(2,2,figsize=(16,8))
y_test['T(degC)'].plot(ax=ax[0,0],label='Actual T(degC)')
holt1f['Holt-Winter with damping'].plot(ax=ax[0,0],label="Holt-Winter
Forecast")
ax[0,0].legend(loc='lower right')
ax[0,0].set_title(f'Holt-Winter Model Forecast vs Actual T(degC)')
ax[0,0].set_xlabel('Date')
ax[0,0].set_ylabel('Temperature (degC)')
ax[0,0].grid()
predictions.plot(ax=ax[0,1],label="MLR Forecast")
y_test['T(degC)'].plot(ax=ax[0,1],label="Actual T(degC)",linestyle='--')
ax[0,1].legend(loc='lower right')
ax[0,1].grid()
ax[0,1].set_title('MLR Model Forecast vs Actual T(degC)')
ax[0,1].set_xlabel('Date')
ax[0,1].set_ylabel('Temperature (degC)')
y_test['T(degC)'].plot(ax=ax[1,0],label='Actual T(degC)')
y_testdf['Forecast'].plot(ax=ax[1,0],label="SARIMA Forecast")
ax[1,0].legend(loc='lower right')
ax[1,0].set_title(f'ARIMA(3,0,0)xSARIMA(0,1,0,365) Model Forecast vs Actual
T(degC)')
ax[1,0].set_xlabel('Date')
ax[1,0].set_ylabel('Temperature (degC)')
ax[1,0].grid()
test["T(degC)"].plot(ax=ax[1,1],color='red',label="Actual T(degC)")
test["Predictions"].plot(ax=ax[1,1],color='blue',label="LSTM Forecast")
ax[1,1].legend(loc = "lower right")
ax[1,1].grid()
ax[1,1].set_title("LSTM Model Forecast vs Actual T(degC)")
ax[1,1].set_xlabel("Date")
ax[1,1].set_ylabel("Temperature (degC)")
fig.suptitle("Model Forecast vs Actual T(degC)",fontsize=18)
plt.tight_layout()
plt.show()

# Judging by these results, the MLR model seems to perform the best on the
test set, followed by the LSTM model, followed by SARIMA model and then the
```

```
Holt-Winter model.
# To better quantify these values, i'll create a table of model performance
metric that i've stored while creating each model

performance_table = [['Model Type', 'Root-Mean-Square-Error','Mean-
Absolute-Error','Mean-Absolute-Percent-Error'],
         ["Holt-Winter
Model",round(RMSE_hw2,3),round(MAE_hw2,3),round(MAPE_hw2,3)],["MLR
Model",round(RMSE_ols,3),round(MAE_ols,3),round(MAPE_ols,3)],["SARIMA
Model",round(RMSE_sarima,3),round(MAE_sarima,3),round(MAPE_sarima,3)],["LST
M Model",round(RMSE_lstm,3),round(MAE_lstm,3),round(MAPE_lstm,3)]]
print(tabulate(performance_table,headers='firstrow', tablefmt =
'fancy_grid'))

# This table helps quantify the results we obtain visually from the
forecast vs actual dataset.

# According to all results obtained the LSTM model performs the best as it
has the second lowest performance errors and also has a white residual.
However, for the sake of forecast function creation, I will go ahead with
SARIMA model as it also has a white residual and comparable model
performance to LSTM.
```