Data Mining Project Report

Eman Tahir (i21-1718)

Afaq Alam (i21-1700)

Contribution:

Eman Tahir:

- Dataset Selection
- Preprocessing
- SARIMA
- ARIMA
- ETS
- SVR
- PROPHET
- Pickle file creation
- Front end development
- Report

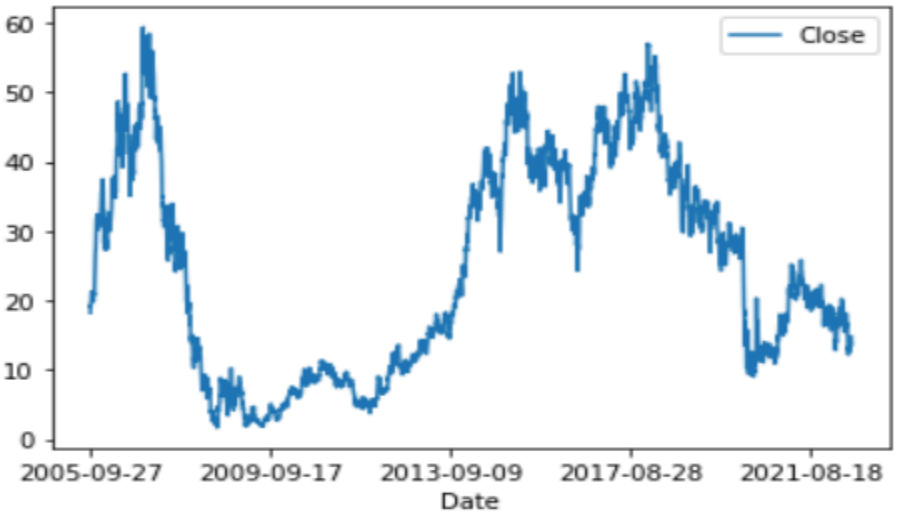
Afaq Alam:

- ANN
- LSTM
- HYBRID
- Pickle file creation
- Integration of models to front end
- Back end development (SQLite)
- Report
- Deployment on GIT

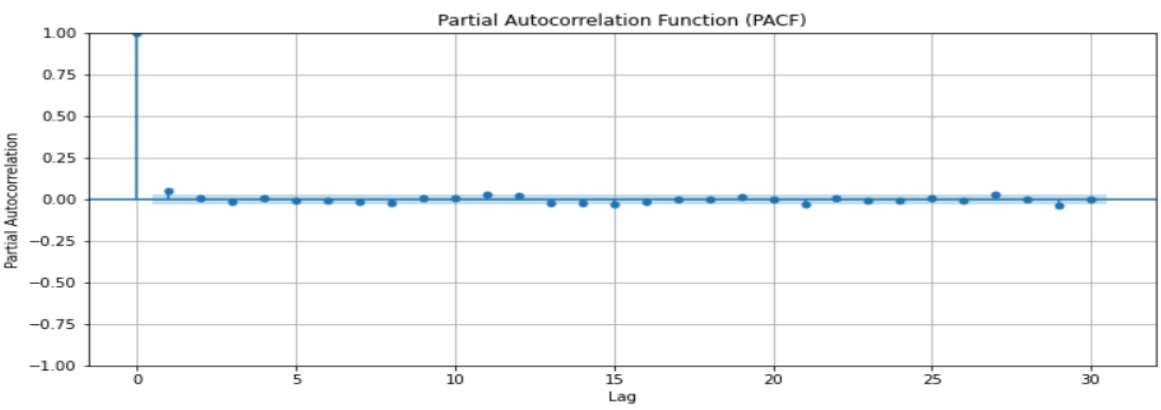
Table of contents:

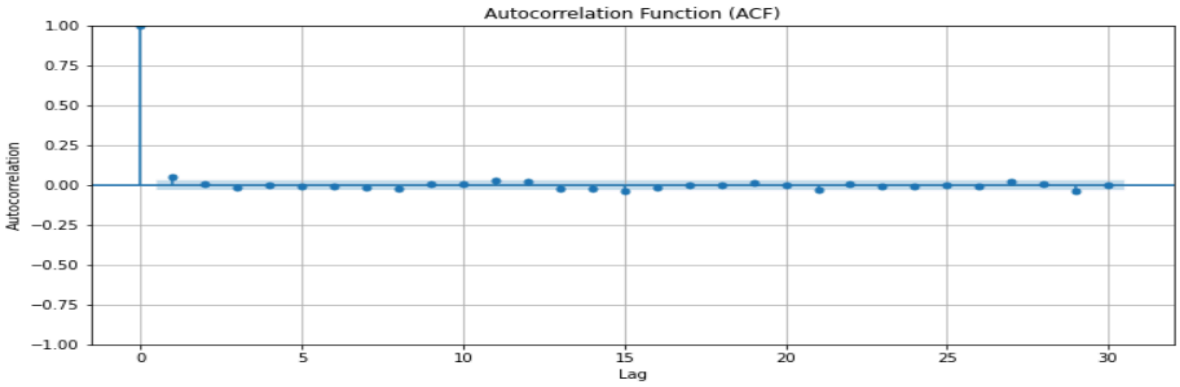
Dataset Selection:

Getting a time series dataset was a task in itself. First I chose a kaggle's dataset from the finance sector. From https://www.kaggle.com/datasets/rprkh15/sp500-stock-prices ,AAL.csv was chosen, however I was not satisfied by it, I wanted to work with seasonal data, but this dataset lacked seasonality.

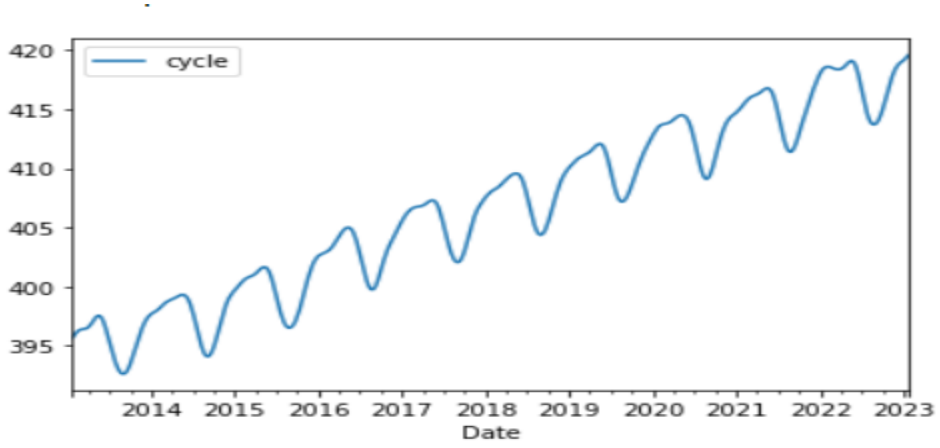


Moreover, the ACF and PACF plots that the dataset showed were almost alike, so our time series model were of waste here .





The next dataset picked showed C02 concentrations, an environmental sector dataset. This too was obtained from kaggle [Daily atmospheric Co2 concentration. (kaggle.com)]. This dataset did have some seasonal trends.



The ACF and PACF plots were suitable for time series analysis, however the models were underperforming. Even after multiple tries to make the dataset stationary, we got the following errors:

```
Mean Squared Error (MSE): 239.05626148793843
Root Mean Squared Error (RMSE): 15.461444353227108
Mean Absolute Error (MAE): nan
```

After a lot of hassle we finally found the dataset that works for us, is satisfactory and worked according to our requirements.

Frome kaggle we used, [Carbon Emissions (kaggle.com)].Now our data had seasonal trends, better ACF and PACF plots due to multiple differencing and seasonal decomposition.

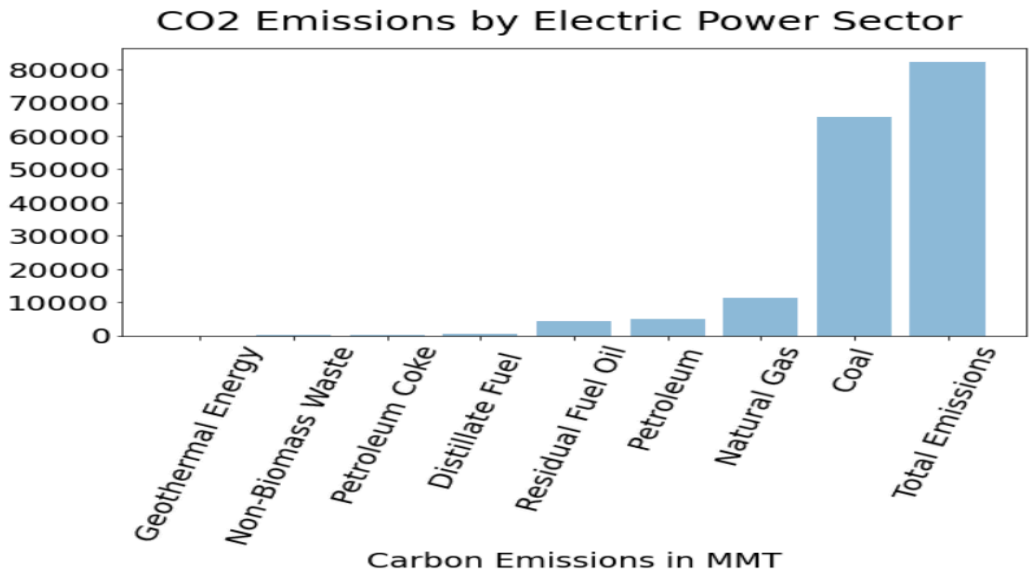
Dataset Description:

We used a public dataset of monthly carbon dioxide emissions from electricity generation available at the Energy Information Administration and Jason McNeill. The dataset includes CO2 emissions from each energy resource starting January 1973 to July 2016.

The dataset has 6 columns where 2 of them are integer data type and 4 objects and 5096 observations.

The dataset has 8 energy sources of CO2 emission:

Geothermal Energy Electric Power Sector CO2 Emissions	10.563
Non-Biomass Waste Electric Power Sector CO2 Emissions	281.367
Petroleum Coke Electric Power Sector CO2 Emissions	338.785
Distillate Fuel, Including Kerosene-Type Jet Fuel, Oil Electric Power Sector CO2 Emissions	404.887
Residual Fuel Oil Electric Power Sector CO2 Emissions	4239.312
Petroleum Electric Power Sector CO2 Emissions	4982.993
Natural Gas Electric Power Sector CO2 Emissions	11295.359
Coal Electric Power Sector CO2 Emissions	65782.393
Total Energy Electric Power Sector CO2 Emissions	82352.676



From the bar chart, we can see that the contribution of coal to the total CO2 emission is significant followed by natural gas.

Stationary Test:

First I conducted the ADF test to check for stationarity. My results were:

Test Statistic 1.831215
p-value 0.998409
#Lags Used 19.000000
Number of Observations Used 503.000000
Critical Value (1%) -3.443418
Critical Value (5%) -2.867303
Critical Value (10%) -2.569840
dtype: float64
The Test Statistic is greater than the critical values with 90%, 95% and 99% confidence levels. Hence, no evidence to reject the null hypothesis. Therefore the series is nonstationary.

After this, the first difference was applied.n this technique, we take the difference of the original observation at a particular instant with that at the previous instant. My results were:

Test Statistic -5.435116
p-value 0.000003
#Lags Used 18.000000
Number of Observations Used 503.000000
Critical Value (1%) -3.443418
Critical Value (5%) -2.867303
Critical Value (10%) -2.569840
dtype: float64
Strong evidence against the null hypothesis, reject the null hypothesis. Data has no unit root, hence it is stationary

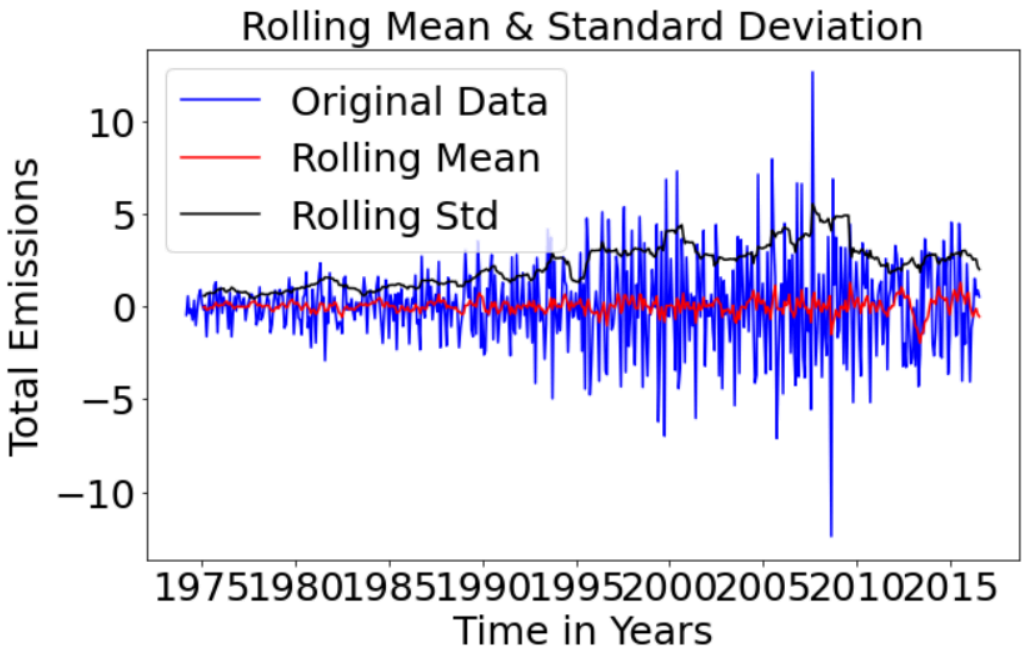
Although this removes the non stationarity of the data a lot, now i removed the seasonal trends from the data.

Test Statistic -4.736434
p-value 0.000072
#Lags Used 12.000000
Number of Observations Used 498.000000
Critical Value (1%) -3.443549
Critical Value (5%) -2.867361
Critical Value (10%) -2.569870
dtype: float64
Strong evidence against the null hypothesis, reject the null hypothesis. Data has no unit root, hence it is stationary

Then by applying first difference on this seasonally decomposed dataset, my results were as follows:

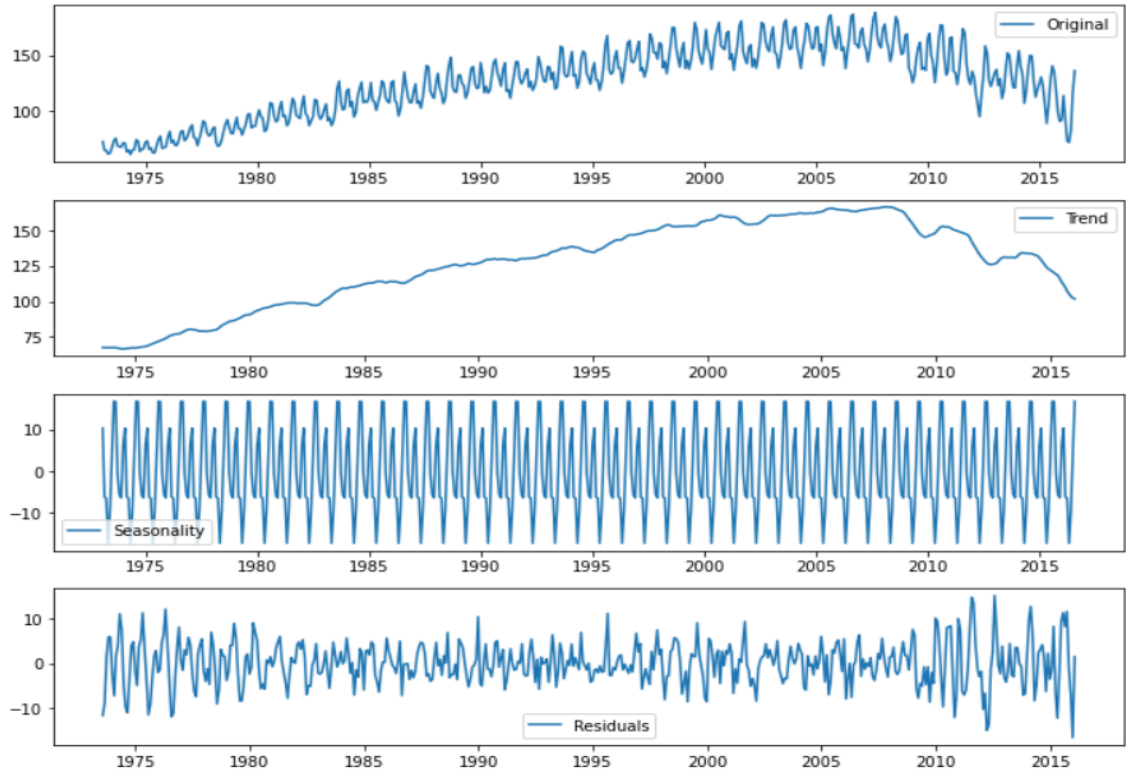
Test Statistic -1.009743e+01
p-value 1.081539e-17
#Lags Used 1.200000e+01
Number of Observations Used 4.970000e+02
Critical Value (1%) -3.443576e+00
Critical Value (5%) -2.867373e+00
Critical Value (10%) -2.569877e+00
dtype: float64
Strong evidence against the null hypothesis, reject the null hypothesis. Data has no unit root, hence it is stationary

I had successfully transformed my time series data to its most stationary form, so the models could perform the best they can, without any effects from trends.



SEASONAL ARIMA:

The first model that I applied on my dataset was sarima. Trend and seasonality was removed from the data as a starting point.

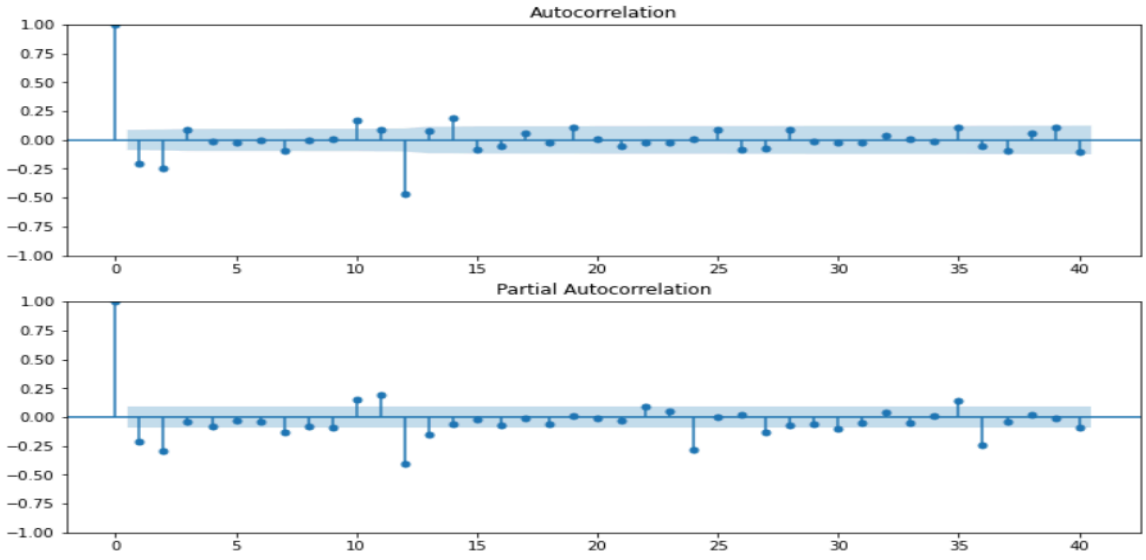


After that I applied the ADF test on my decomposed model , my results were:

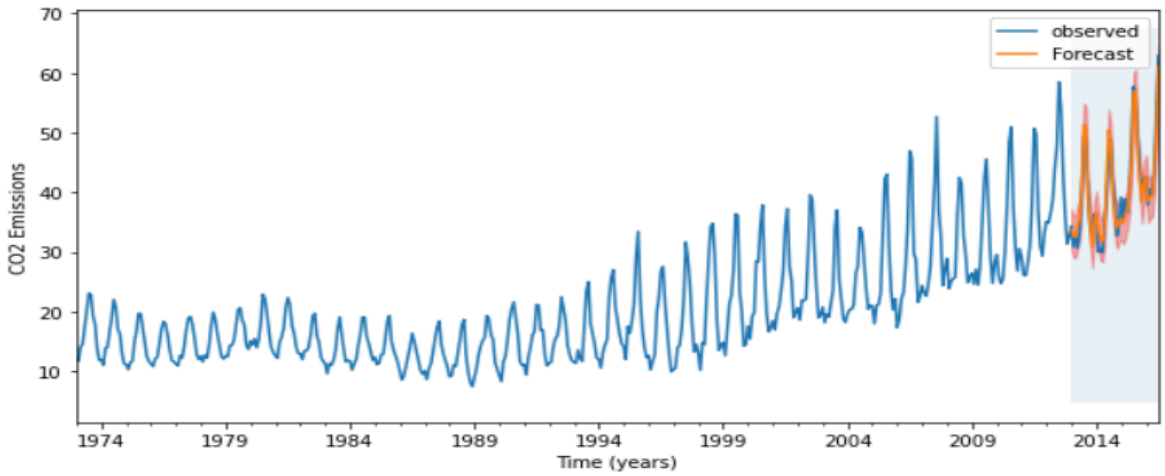
Test Statistic -1.030709e+01
p-value 3.257381e-18
#Lags Used 1.400000e+01
Number of Observations Used 4.960000e+02
Critical Value (1%) -3.443603e+00
Critical Value (5%) -2.867385e+00
Critical Value (10%) -2.569883e+00
dtype: float64
Strong evidence against the null hypothesis, reject the null hypothesis. Data has no unit root, hence it is stationary

Then I made my ACF and PACF plots to determine the order for my sarima model.
ACF plot shows the correlation of the series with its own lags. In our ACF plot:The significant spike at lag 1 that quickly cuts off is typical of a differenced series, which suggests the differencing term d d is correctly set when preparing the data for SARIMA modeling. The lag at which the ACF cuts off can suggest the q q parameter for the MA (moving average) component of an SARIMA model. In our

case, since the ACF cuts off after the first lag, q = 1 q=1 might be a good starting point. Partial Autocorrelation Function (PACF) The PACF plot shows the partial correlation of a series with its own lag, excluding correlations of the intervening lags. From our PACF plot:The significant spike at lag 1 and then a cutoff also supports the use of p = 1 p=1 for the AR (autoregressive) component of the SARIMA model.



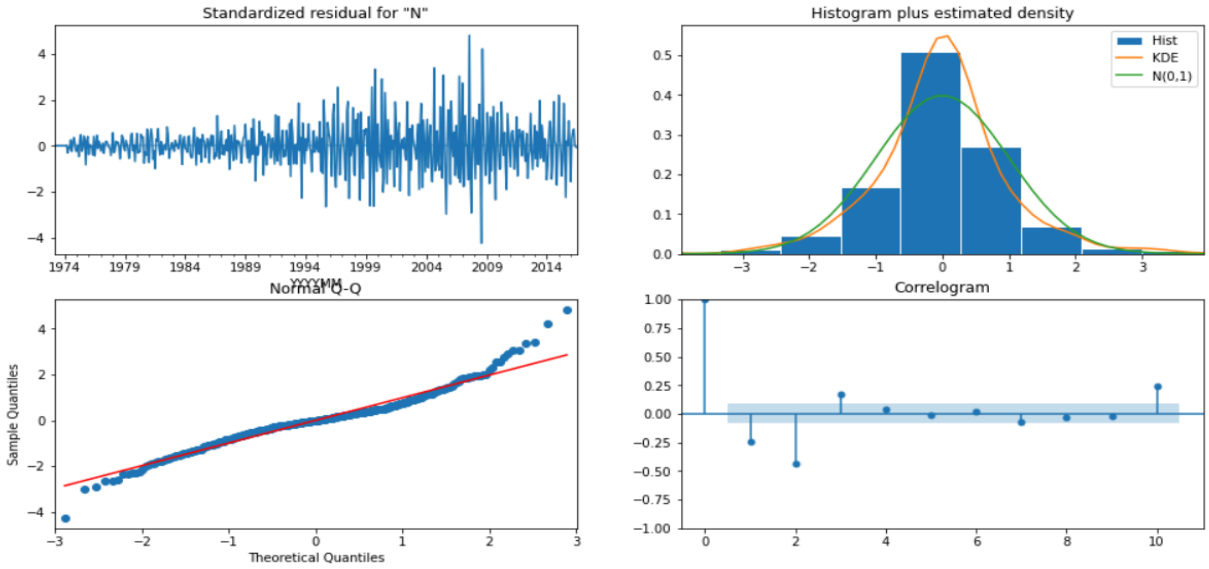
The forecast for my sarima model is as follows:



The Mean Squared Error (MSE) of the forecast is 56.41
The Root Mean Square Error (RMSE) of the forcast: 7.5106

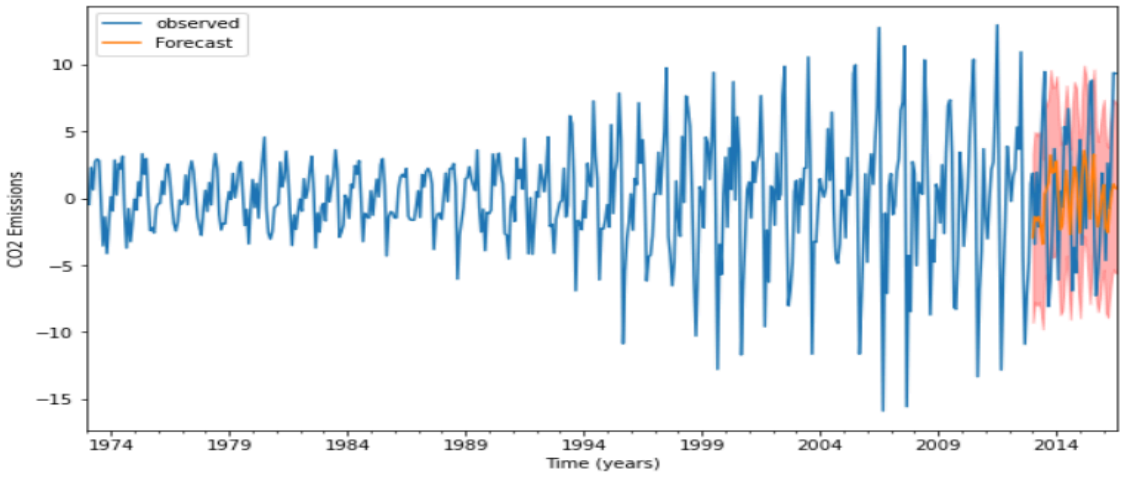
ARIMA:

Arima was applied on the dataset that I made stationary, by applying the seasonal difference. My results are as aforementioned:



Forecasting:

It begins by generating predictions using the ARIMA model's get_prediction method, specifying a start and end point for the forecasted range. The dynamic=False parameter indicates a static forecasting approach, where the model uses actual historical data without updating predictions based on forecasted values. Confidence intervals for the forecasted values are computed using the pred.conf_int() method, creating a range of uncertainty around each predicted value. These forecasted values, along with their confidence intervals, are then plotted on a graph.



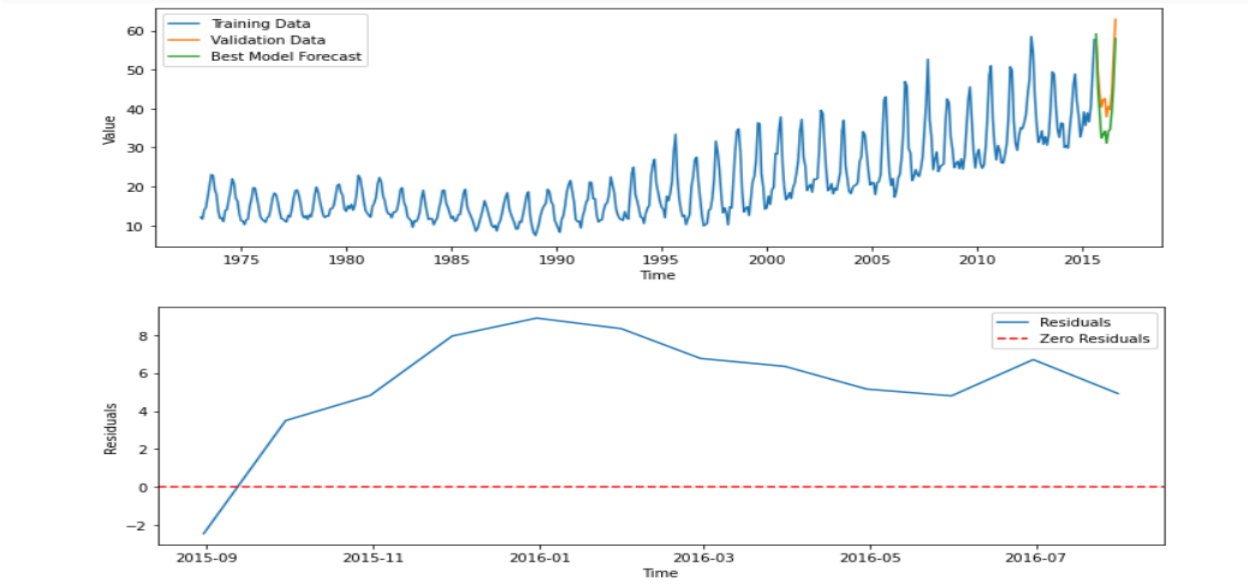
Mean Squared Error (MSE): 1694.6087285290866
Root Mean Squared Error (RMSE): 41.16562556950989

ETS:

We performed an exhaustive search for the best-fitting Exponential Smoothing (ETS) model by iterating through various configurations based on error type, trend type, and seasonal type. It first splits the data into training and validation sets. The iterative process creates ETS models with different combinations of error, trend, and seasonal components using statsmodels.api.ExponentialSmoothing. Each model is then fitted to the training data, and evaluation metrics such as Bayesian Information Criterion (BIC), Mean Squared Error (MSE), Mean Absolute Percentage Error (MAPE), and Root Mean Squared Error (RMSE) are calculated for the validation data.

The best model is determined based on the lowest BIC value, indicating a trade-off between model complexity and goodness of fit. Once the best model is identified, it is used to forecast values for the validation data. Evaluation metrics are calculated again for the forecasted values to assess the model's predictive accuracy.

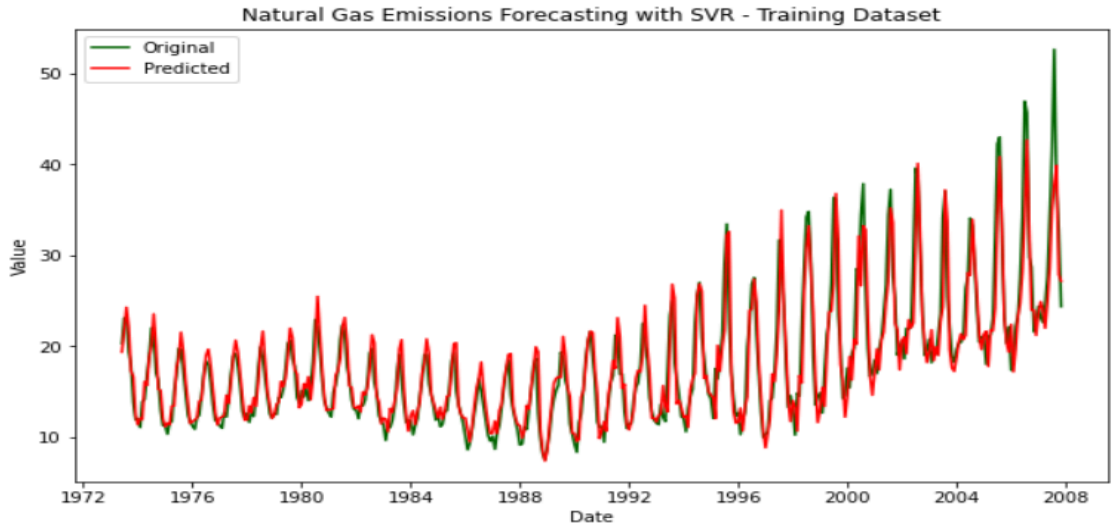
Finally, the code plots the training and validation data, the forecasted values from the best model, and the residuals (the difference between actual and forecasted values). These visualizations help in understanding the model's performance and the behavior of residuals over time, aiding in model validation and interpretation.

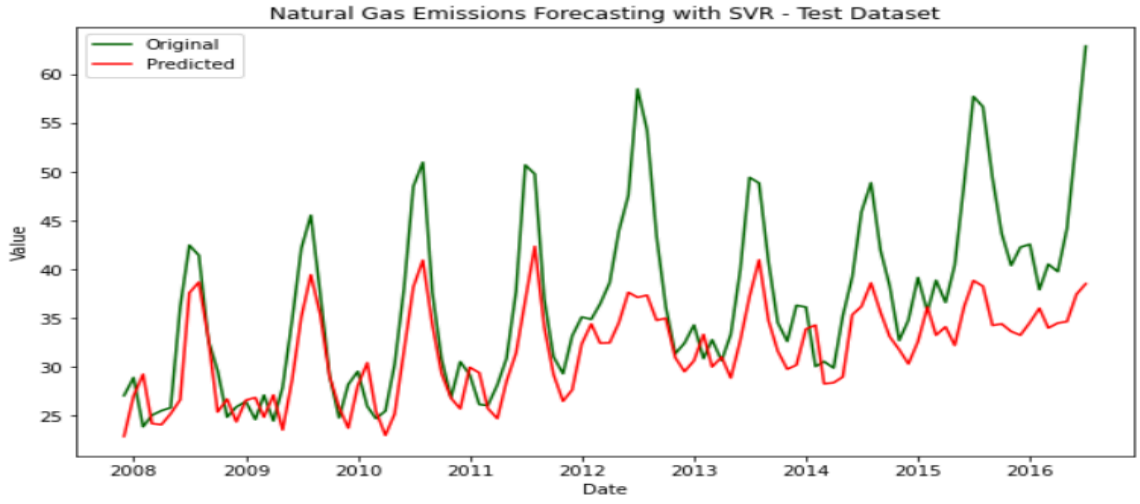


Best Model BIC: 703.422166790702
Best Model BIC: 703.422166790702
Mean Squared Error (MSE) - Best Model: 38.16606714788859
Mean Absolute Percentage Error (MAPE) - Best Model: 13.366514939978838
Root Mean Squared Error (RMSE) - Best Model: 6.177869142988429

SVR:

SVR is applied to forecast natural gas emissions, where the data is first scaled using MinMaxScaler to normalize it between 0 and 1. The create_dataset function prepares the data for time series forecasting by creating input-output pairs with a specified time window. The SVR model is trained on a portion of the data and evaluated using mean squared error (MSE) on both training and testing sets. The inverse scaling is then applied to interpret the predictions in their original scale. Finally, the results are visualized using matplotlib, showing the original and predicted values over time for both the training and testing datasets, providing insights into the SVR model's forecasting performance.





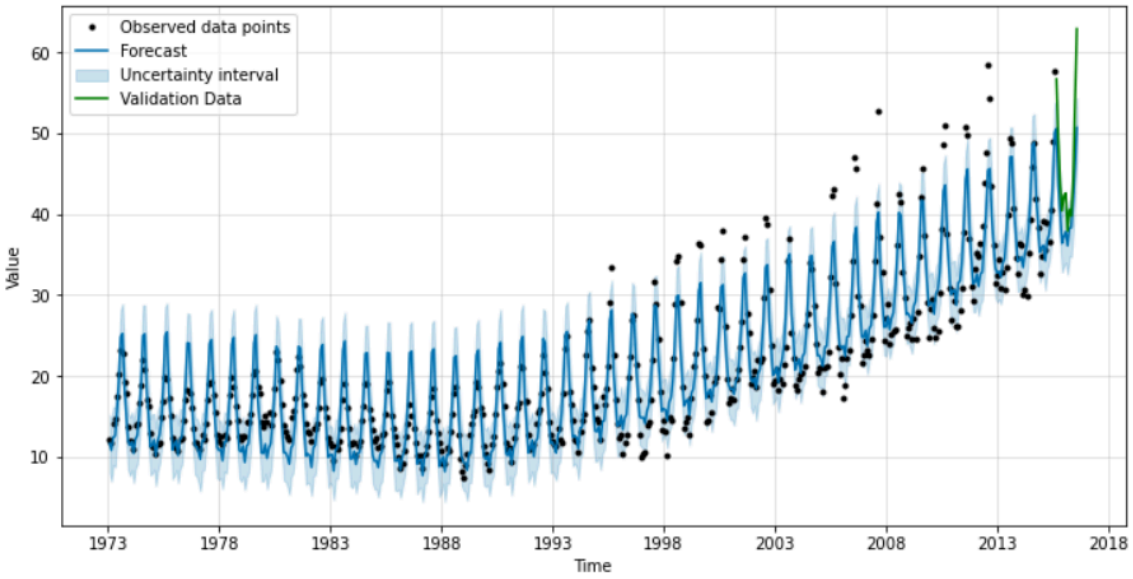
Mean Squared Error on Training Data: 7.741365212055047
Mean Squared Error on Testing Data: 52.60066192444773

PROPHET:

We start by preparing the data in a format suitable for Prophet, with a DataFrame containing timestamps ('ds') and corresponding values ('y'). The data is then split into training and validation sets, with the last 12 months assumed as the validation period. The Prophet model is initialized with various seasonal components, including yearly, weekly, and daily patterns, along with additional options like holidays and changepoint prior scale for flexibility in trend detection. The model is then trained on the training data, automatically optimizing its parameters.

Next, the model is used to make predictions for the validation period using make_future_dataframe to extend the time range. Evaluation metrics such as Mean Squared Error (MSE), Mean Absolute Percentage Error (MAPE), and Root Mean Squared Error (RMSE) are calculated to assess the model's performance on the validation data. The results, including the model object, forecast, validation data, and evaluation metrics, are stored in a dictionary and optionally saved to a pickle file.

Finally, the code plots the forecasted values alongside the validation data using matplotlib. This visualization helps in understanding how well the Prophet model captures the underlying patterns and trends in the data, providing a clear comparison between actual and predicted values over time.

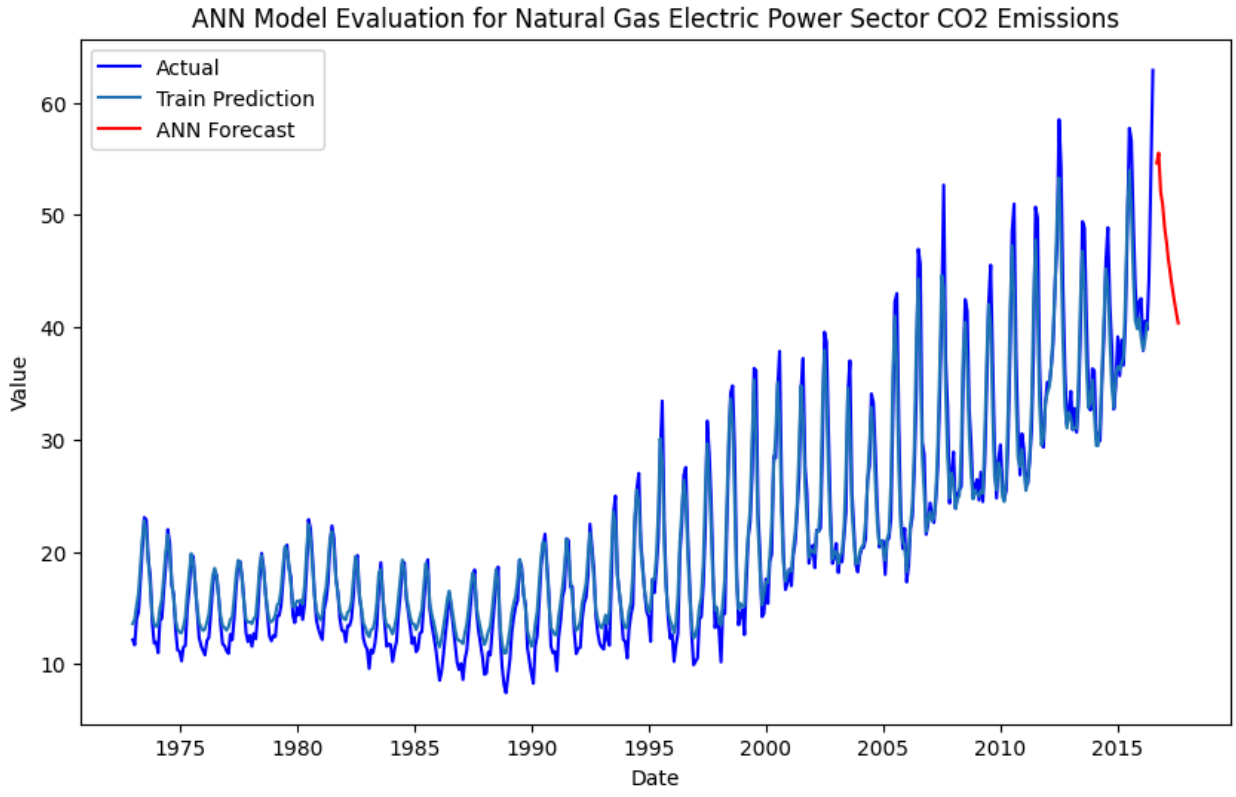


Mean Squared Error (MSE): 31.5644203774323
Mean Absolute Percentage Error (MAPE): 9.853121324073118
Root Mean Squared Error (RMSE): 5.6182221723096974

ANN:

I have implemented an Artificial Neural Network (ANN) model using sklearn's MLPRegressor for time series forecasting. It begins by scaling the input data using MinMaxScaler. The `create_dataset` function prepares the data for training by creating input-output pairs based on a specified look-back window. The ANN model is then defined with one hidden layer of 100 neurons, ReLU activation, and Adam optimizer. Training involves fitting the model to the training data. Predictions are made for a forecast horizon by iteratively predicting one step ahead and updating the input sequence. After inverting the scaling, predictions for both training and testing data are obtained.

The `evaluate_ann_model` function assesses the model's performance by converting the 'Value' column to numeric and handling missing values. It then calls `ann_model` to obtain predictions and calculates the Root Mean Squared Error (RMSE) for the last 30 days of data. The plot visualizes the actual data, training predictions, and ANN forecasts.



ANN RMSE: 9.350252141651637
Next Year's Forecasted Values:
[[54.6146397]
 [55.50788511]
 [52.02847073]]

[50.9642221]
[48.90549023]
[47.53784475]
[46.00552552]
[44.72429277]
[43.47780301]
[42.36272394]
[41.31846365]
[40.36486274]]

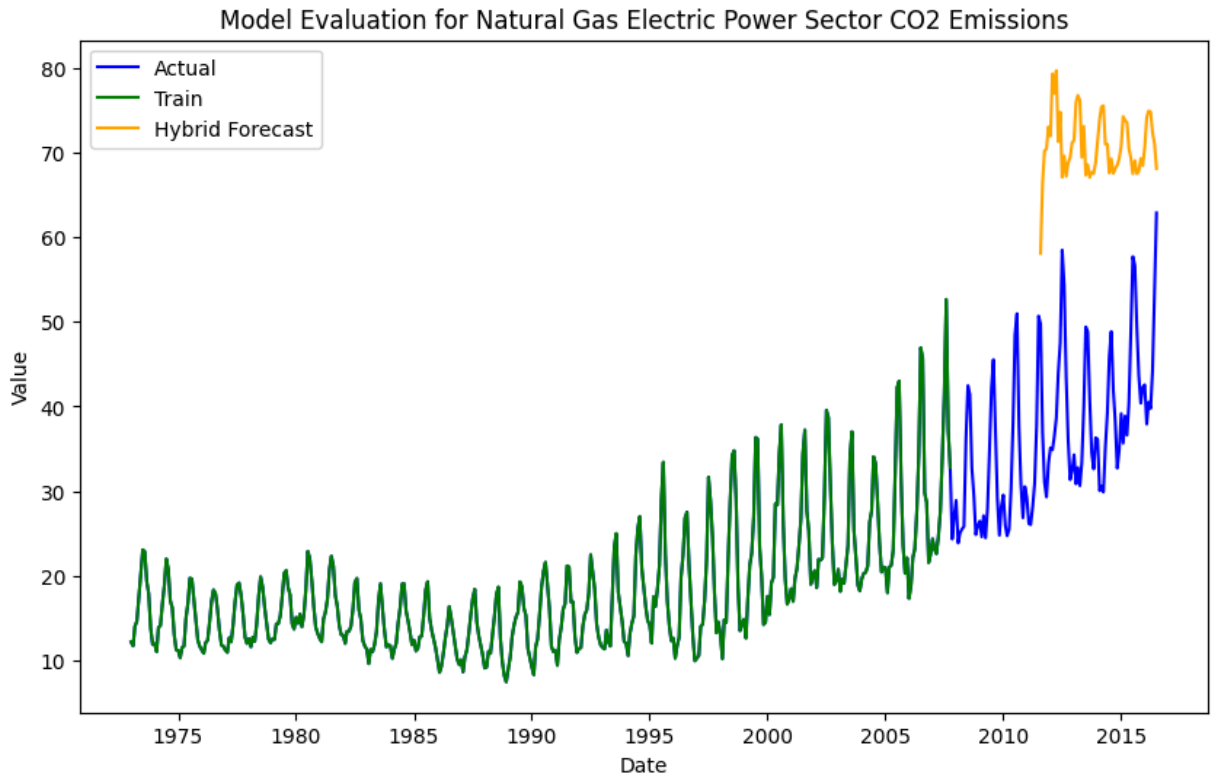
HYBRID-ANN:

I have defined three functions: ann_model for an Artificial Neural Network (ANN) model, arima_model for an ARIMA model, and hybrid_model that combines both models' forecasts using a weighted average. The evaluate_hybrid_model function evaluates the hybrid model's performance by comparing its forecast with actual values and plotting them. It also calculates the Root Mean Squared Error (RMSE) for the forecast horizon, saves the hybrid model to a file, and prints the RMSE.

The ann_model function scales the input data, creates input-output pairs for training, fits an MLPRegressor model to the training data, and generates forecasts iteratively. The arima_model function fits an ARIMA model to the data and makes predictions for the specified forecast horizon. The hybrid_model function combines the forecasts from both models using a weighted average based on the ann_weight parameter.

In evaluate_hybrid_model, the data is preprocessed, split into training and testing sets, and the hybrid model is used to generate forecasts. The actual values, training values, and hybrid model forecasts are plotted over time. The RMSE is calculated to assess the hybrid model's accuracy, and the trained hybrid model is saved to a file named 'hybrid_model.pkl'.

Overall, this code integrates ANN and ARIMA models into a hybrid forecasting approach, evaluates the hybrid model's performance, and saves the trained model for future use.



Hybrid RMSE: 32.30041270149407

Hybrid RMSE (% of actual range): 96.20949185802303 %

LSTM:

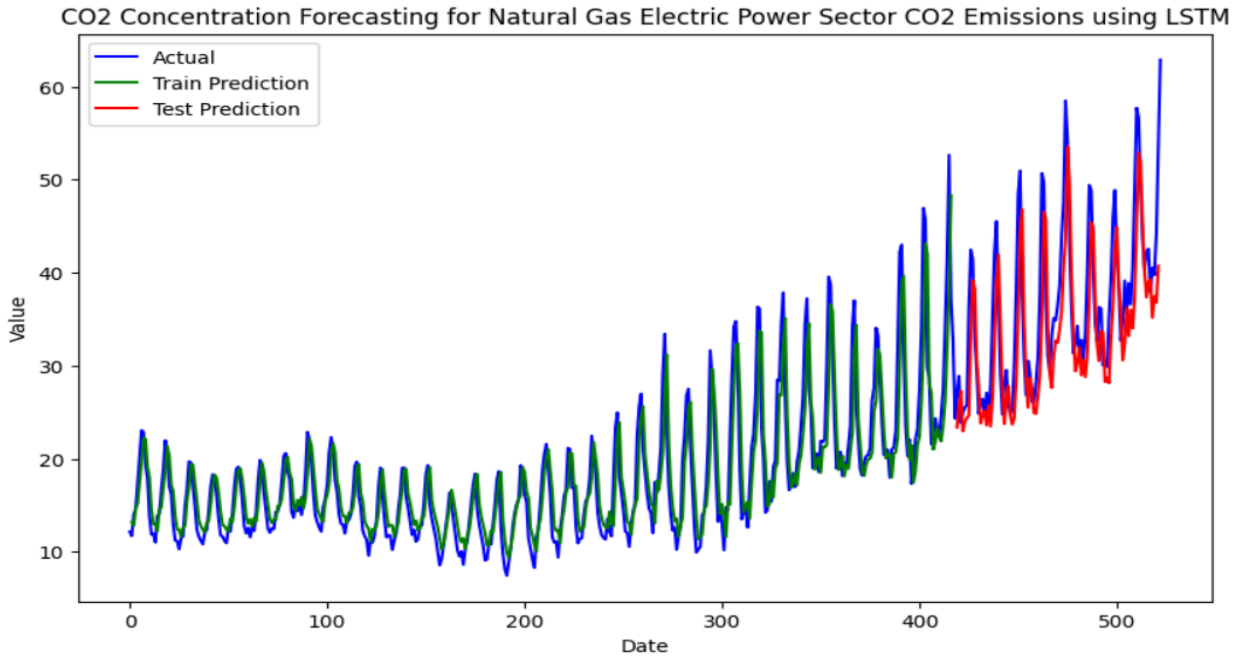
I have utilized a Long Short-Term Memory (LSTM) neural network, implemented through Keras, for time series forecasting of CO2 emissions in the natural gas electric power sector..

The plot_data function is defined to visualize the time series data, focusing on the 'Value' column. Another function, create_dataset, prepares the data for LSTM modeling by creating input-output pairs with a specified look-back window and normalizes the dataset using MinMaxScaler.

The dataset is split into training and testing sets, reshaped to fit the LSTM input requirements, and the LSTM model is designed with one LSTM layer containing 50 units followed by a dense

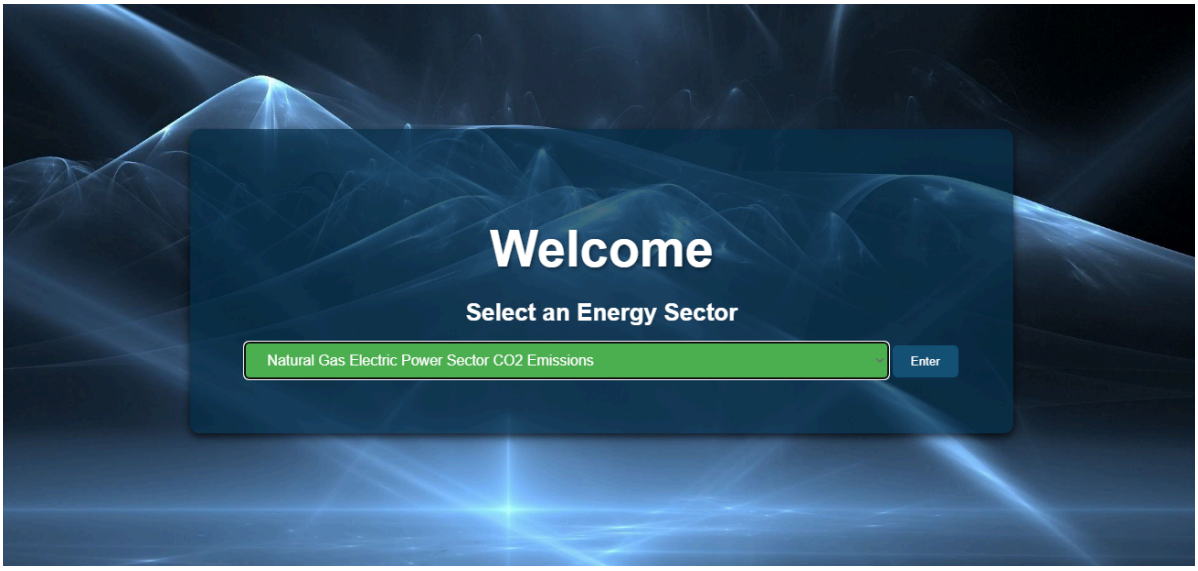
layer for output. It is then compiled with the Adam optimizer and trained on the training data for 10 epochs.

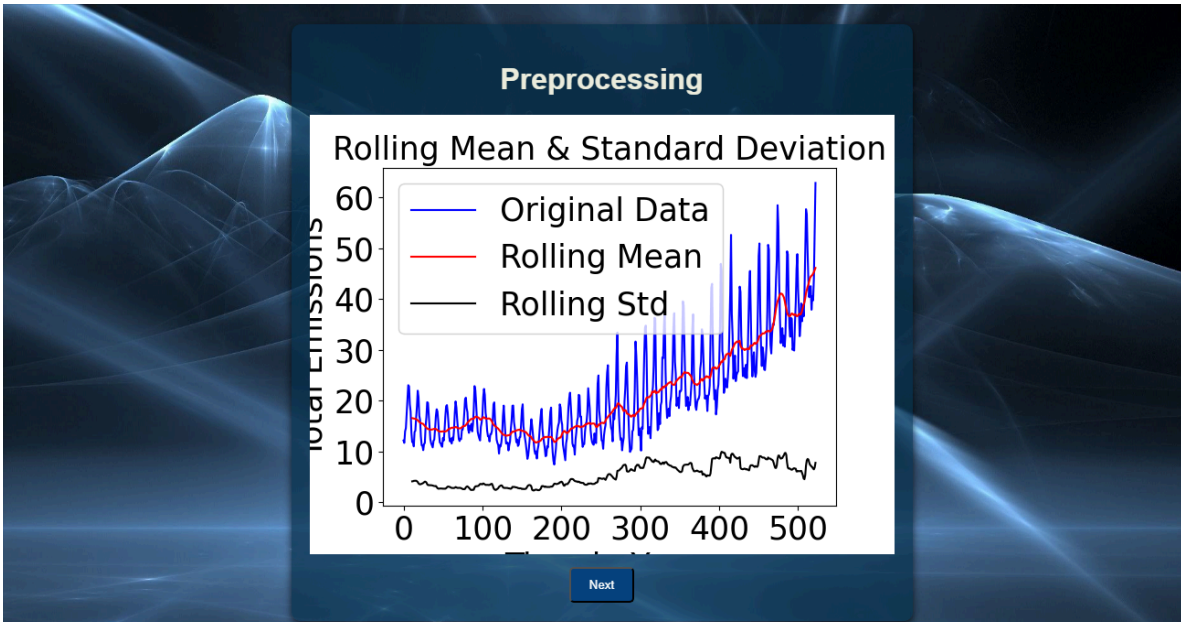
After training, predictions are made on both the training and testing sets, which are then inverted to their original scales for evaluation. Root Mean Squared Errors (RMSE) are calculated for both the training and testing predictions to assess model performance.

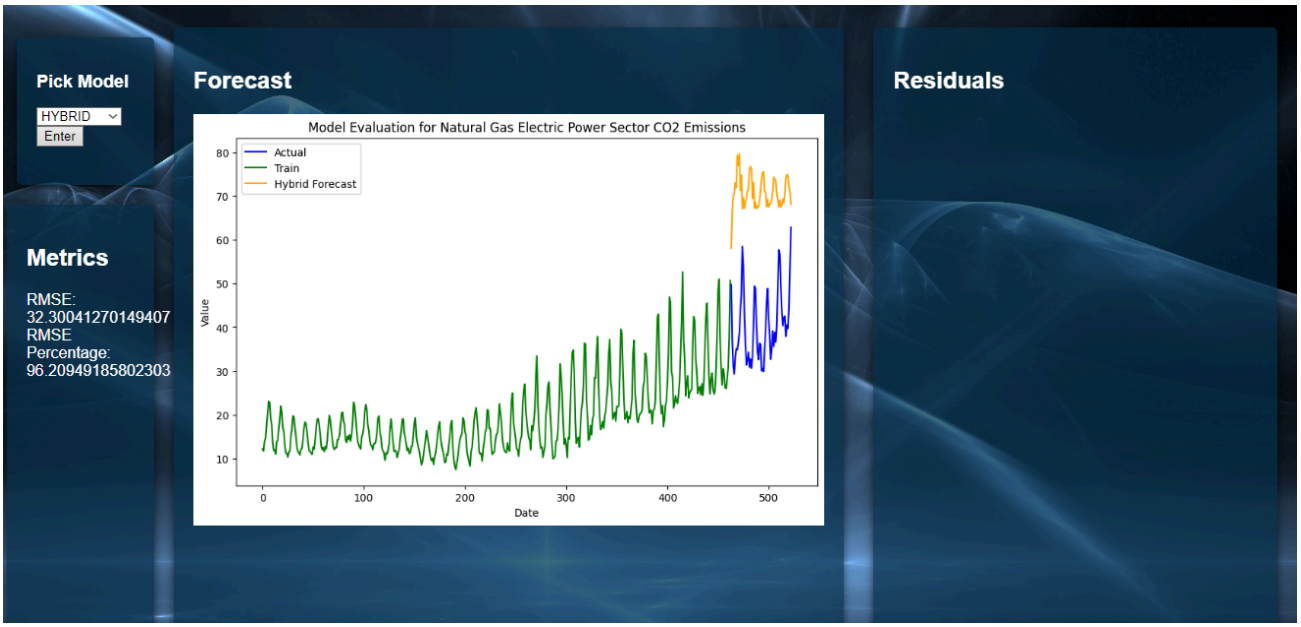


PredictionsTrain Score: 3.52 RMSE
Test Score: 5.78 RMSE

Front End:


Welcome
Select an Energy Sector
Natural Gas Electric Power Sector CO2 Emissions
Enter


Preprocessing
Rolling Mean & Standard Deviation
Original Data
Rolling Mean
Rolling Std
60
50
40
30
20
10
0
0
100
200
300
400
500
Next

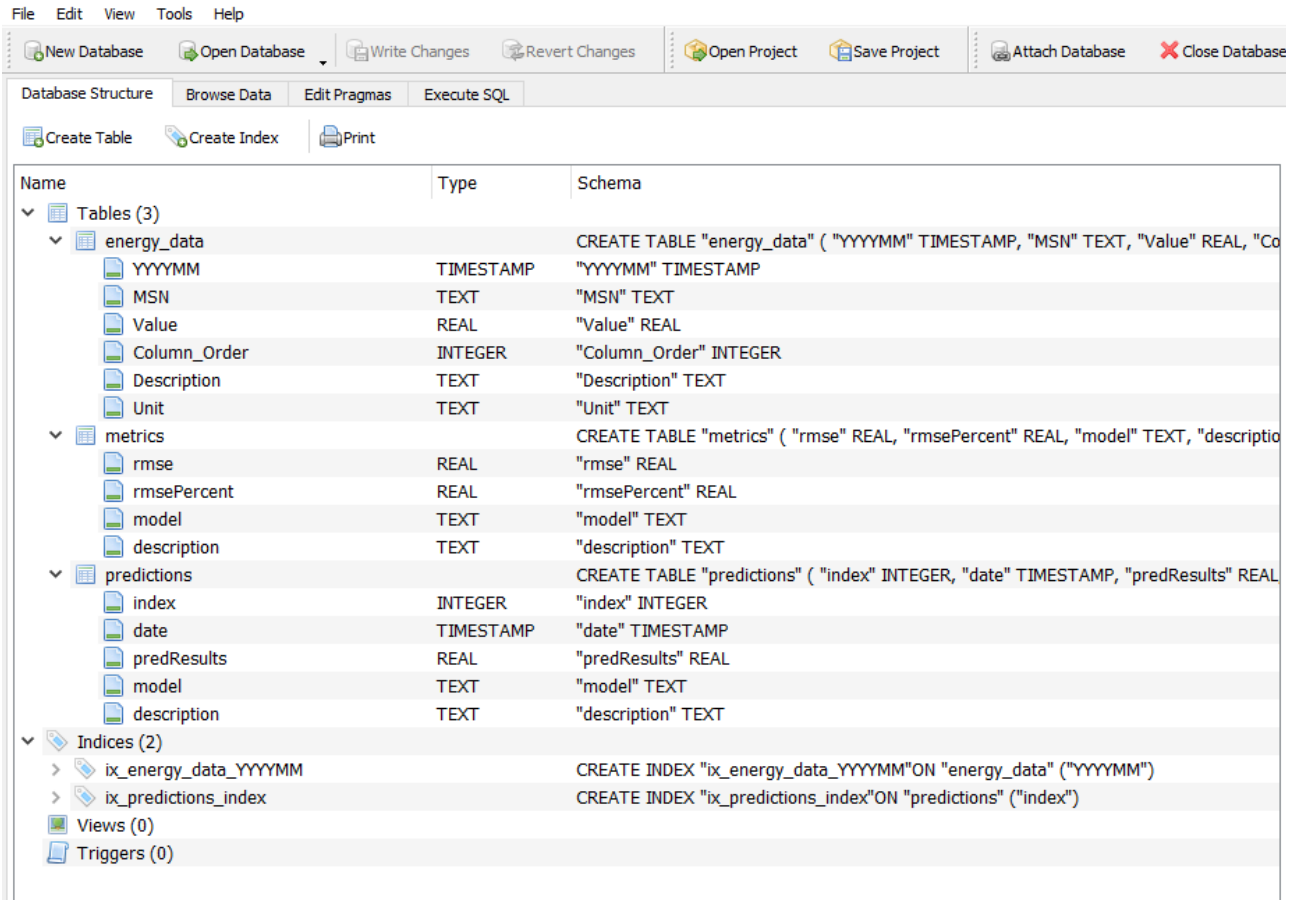


Database:

To store processed data and results in a database for quick retrieval for visualization on a web app, we used an appropriate database like SQLite.

Next, we designed the database schema, creating tables that represent the structure of our processed data and model results logically. Each table has columns corresponding to the attributes of our data.

Once the database schema is set up, we insert our processed data and model results into the database using SQLite.


File Edit View Tools Help
New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Create Table
Create Index
Print
Name
Type
Schema
Tables (3)
energy_data
CREATE TABLE "energy_data" ("YYYYMM" TIMESTAMP, "MSN" TEXT, "Value" REAL, "Co
YYYYMM
TIMESTAMP
"YYYYMM" TIMESTAMP
MSN
TEXT
"MSN" TEXT
Value
REAL
"Value" REAL
Column_Order
INTEGER
"Column_Order" INTEGER
Description
TEXT
"Description" TEXT
Unit
TEXT
"Unit" TEXT
metrics
CREATE TABLE "metrics" ("rmse" REAL, "rmsePercent" REAL, "model" TEXT, "descriptio
rmse
REAL
"rmse" REAL
rmsePercent
REAL
"rmsePercent" REAL
model
TEXT
"model" TEXT
description
TEXT
"description" TEXT
predictions
CREATE TABLE "predictions" ("index" INTEGER, "date" TIMESTAMP, "predResults" REAL
index
INTEGER
"index" INTEGER
date
TIMESTAMP
"date" TIMESTAMP
predResults
REAL
"predResults" REAL
model
TEXT
"model" TEXT
description
TEXT
"description" TEXT
Indices (2)
ix_energy_data_YYYYMM
CREATE INDEX "ix_energy_data_YYYYMM"ON "energy_data" ("YYYYMM")
ix_predictions_index
CREATE INDEX "ix_predictions_index"ON "predictions" ("index")
Views (0)
Triggers (0)