DATA MINING

Project-Comprehensive Forecasting System with User Interface for Multiple Sectors

Submitted to: Sir Saad Munir

Submitted By:
Khadeeja Shah
Nibras Aamir

Introduction:

- Objective
  - The objective of this project is to develop a comprehensive forecasting system capable of implementing and comparing different time series models across various sectors. The system aims to include models such as ARIMA, Artificial Neural Networks (ANN),Long Short-Term Memory (LSTM) ,Support Vector Regression (SVR), Prophet , Exponential Smoothing (ETS), SARIMA (Seasonal ARIMA) and Hybrid ARIMA-ANN. Additionally, the project focuses on providing a user-friendly front-end interface for visualizing data and forecasts.
- Data Sources and Preprocessing
  - Data Sources:
    - Energy Sector: Hourly energy consumption data.
      https://www.kaggle.com/datasets/robikscube/hourly-energy-consumption
      Using: PJM_Load_hourly.csv
    - Environmental Sector: Daily atmospheric CO2 concentrations.
      https://www.co2.earth/co2-datasets
      Using: daily_in_situ_co2_mlo.csv (2000 - 2020)
  - PreProcessing:
    - Cleaning of dataset: null , duplicate , inconsistent records were removed.
    - New csv files were made with the cleaned data, named daily.csv and hour.csv

For Daily atmospheric CO2 concentrations dataset (daily.csv)

- PreProcessing for the application of models:
    - To apply models (arima , sarima , ets , prophet , hybrid), the dataset must undergo several transformations which are as under:
        - Changing format of date:
            - dfd = pd.read_csv("daily.csv")
            - dfd['date'] = pd.to_datetime(dfd['date']) dfd.set_index('date', inplace=True)
            - # Perform differencing dfd_diff = dfd.diff().dropna()
        - Dividing in training and testing:
            - train_size = int(len(dfd_diff) * 0.8) df_train, df_test = dfd_diff.iloc[:train_size], dfd_diff.iloc[train_size:]
        - Normalizing:
            - scaler = StandardScaler() train_data_scaled = scaler.fit_transform(df_train[['cycle']]) test_data_scaled = scaler.transform(df_test[['cycle']])
    - For ann, svr and lstm:
        - # Features and target
            - X_train, X_test, y_train, y_test = train_test_split(df_train[['cycle', 'trend']], df_train['cycle'], test_size=0.2, random_state=42)
        - # Scale data
            - scaler = StandardScaler() X_train_scaled = scaler.fit_transform(X_train) X_test_scaled = scaler.transform(X_test)

For Hourly energy consumption dataset (hour.csv)

- PreProcessing for the application of models:
    - To apply models (arima , sarima , ets , prophet , hybrid), the dataset must undergo several transformations which are as under:
    - dfd = pd.read_csv("PJM_Load_hourly.csv")
    - Change date format
        - dfd['datetime_column'] = pd.to_datetime(dfd['Datetime']) dfd['date'] = dfd['datetime_column'].dt.date dfd['hour'] = dfd['datetime_column'].dt.hour
        - dfd = dfd.drop(columns=['Datetime', 'datetime_column']) dfd.set_index('date', inplace=True)

- Check is dataset is already static or not:
  - from statsmodels.tsa.stattools import adfuller
  - adf_test = adfuller(dfd['PJM_Load_MW'])
  - print(f'p-value: {adf_test[1]}') # not stationary. greater than 0.05
- Apply difference or log
  - #dfd_diff = dfd.diff().dropna()
  - dfd_diff= np.log(dfd)
- # Train-test split
  - train_size = int(len(dfd_diff) * 0.8)
  - df_train, df_test = dfd_diff.iloc[:train_size], dfd_diff.iloc[train_size:]
- # Normalize the data
  - scaler = StandardScaler()
  - scaler3 = MinMaxScaler()
  - Train_data_scaled = scaler3.fit_transform(df_train[['PJM_Load_MW']]) test_data_scaled = scaler3.transform(df_test[['PJM_Load_MW']]
- For ann, svr and lstm:
  - dfd_difff = dfd.diff().dropna() #first difference
  - dfd_difff = np.log(dfd_difff) # then log
  - # Train-test split
    - train_size = int(len(dfd_difff) * 0.8)
    - df_train, df_test = dfd_difff.iloc[:train_size], dfd_difff.iloc[train_size:] df_train.replace([np.inf,-np.inf],np.nan ,inplace=True) df_train.dropna(inplace=True) df_test.replace([np.inf,-np.inf],np.nan ,inplace=True) df_test.dropna(inplace=True)
  - Splitting:
    - X_train, X_test, y_train, y_test = train_test_split(df_train[['PJM_Load_MW', 'hour']], df_train['PJM_Load_MW'], test_size=0.2, random_state=42)
  -
  - Normalize:
    - scaler2 = MinMaxScaler()
    - X_train_scaled = scaler2.fit_transform(X_train) X_test_scaled = scaler2.transform(X_test)

- MODELS:

- All the functions for applying models and plot functions are the same for both dataset. Just the hyperparameters are changed. Choosing hyperparameters depends on the acf, pacf graphs. By looking at the graphs, values of hyperparameters can be determined. Else, built in functions to find the best value of let say arima can be found for eg:
  - import pmdarima as pm
  - auto_arima = pm.auto_arima(df_train['PJM_Load_MW'], stepwise=False, seasonal=False)
- ARIMA (AutoRegressive Integrated Moving Average):
  - Functioning: ARIMA is a time series forecasting method that models the relationship between a series and its lagged values (auto-regression), differences of the series (integration), and the lagged forecast errors (moving average).
  - Working: ARIMA models consist of three main components: Auto-Regressive (AR) terms, Integrated (I) terms, and Moving Average (MA) terms. The ARIMA model is defined by three parameters: p (order of AR terms), d (degree of differencing), and q (order of MA terms). It makes forecasts by analyzing the patterns in the time series data.
  - My Model:

    def arima_model(train_data, test_data):
    model = ARIMA(train_data, order=(0, 1, 2))
    model_fit = model.fit()
    forecast = model_fit.forecast(len(test_data))
    return forecast

- SARIMA (Seasonal AutoRegressive Integrated Moving Average):
  - Functioning: SARIMA is an extension of ARIMA that also considers the seasonality in the time series data. It includes additional seasonal terms to account for recurring patterns at fixed intervals.
  - Working: SARIMA models extend the ARIMA framework by incorporating seasonal AR, MA, and differencing terms. It allows for capturing and forecasting seasonal variations in the data along with the non-seasonal components.
  - MY Model:

    def sarima_model(train_data, test_data):
    model = SARIMAX(train_data, order=(1, 1, 1), seasonal_order=(1, 1, 1, 12))
    model_fit = model.fit()
    forecast = model_fit.forecast(len(test_data))
    return forecast
- ETS (Error-Trend-Seasonality):

        - Functioning: ETS is a time series forecasting method that decomposes the data into three components: Error, Trend, and Seasonality. It models each of these components independently to make forecasts.
        - Working: ETS models capture the underlying patterns in the data by estimating the level, trend, and seasonality components. It uses exponential smoothing techniques to assign weights to past observations, giving more importance to recent data points.
        - My Model:
        def ets_model(train_data, test_data):
        model = ExponentialSmoothing(train_data)
        model_fit = model.fit()
        forecast = model_fit.forecast(len(test_data))
        return forecast

    - Prophet:
        - Functioning: Prophet is an open-source forecasting tool developed by Facebook that is designed for time series forecasting with daily observations that display patterns on different time scales.
        - Working: Prophet decomposes time series data into three main components: trend, seasonality, and holidays. It uses a decomposable time series model with components such as piecewise linear or logistic growth trends, seasonality, and holiday effects.
        - My Model:
        def prophet_model(train_data, test_data):
        model = Prophet()
        df = pd.DataFrame({'ds': train_data.index, 'y': train_data.values})
        model.fit(df)
        Future = model.make_future_dataframe(periods=len(test_data))
        forecast = model.predict(future)
        return forecast['yhat'].tail(len(test_data))

    - ANN (Artificial Neural Network):
        - Functioning: ANN is a machine learning model inspired by the biological neural networks of the human brain. It consists of interconnected nodes (neurons) organized in layers.
        - Working: ANN learns from the data by adjusting the weights between neurons during the training process. It can capture

complex nonlinear relationships in the data and make predictions based on learned patterns.

- My Model:

```
def ann_model(X_train_scaled, y_train, X_test_scaled, y_test ):
model = tf.keras.Sequential([   tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)), tf.keras.layers.Dense(32, activation='relu'), tf.keras.layers.Dense(1)    ])
model.compile(optimizer='adam', loss='mean_squared_error')    model.fit(X_train_scaled, y_train, epochs=100, validation_split=0.2, verbose=0)
predictions = model.predict(X_test_scaled)
return predictions
```

- SVR (Support Vector Regression):
  - Functioning: SVR is a machine learning model used for regression tasks. It works by mapping the input data into a higher-dimensional feature space and finding the optimal hyperplane that best separates the data points.
  - Working: SVR aims to find a hyperplane that maximizes the margin between the support vectors, which are the data points closest to the hyperplane. It uses a kernel function to map the input data into a higher-dimensional space where a linear separation is possible.
  - My Model:

```
def svr_model(X_train_scaled, y_train, X_test_scaled):
model = SVR(kernel='rbf')    model.fit(X_train_scaled, y_train)
predictions = model.predict(X_test_scaled)
return predictions
```

- Hybrid Models:
  - Functioning: Hybrid models combine multiple forecasting techniques to improve prediction accuracy. They leverage the strengths of different models to overcome individual model limitations.
  - Working: Hybrid models can combine traditional time series methods like ARIMA or ETS with machine learning algorithms like ANN or SVR. By blending the forecasts from different models, hybrid models can produce more robust and accurate predictions.
  - My Model:

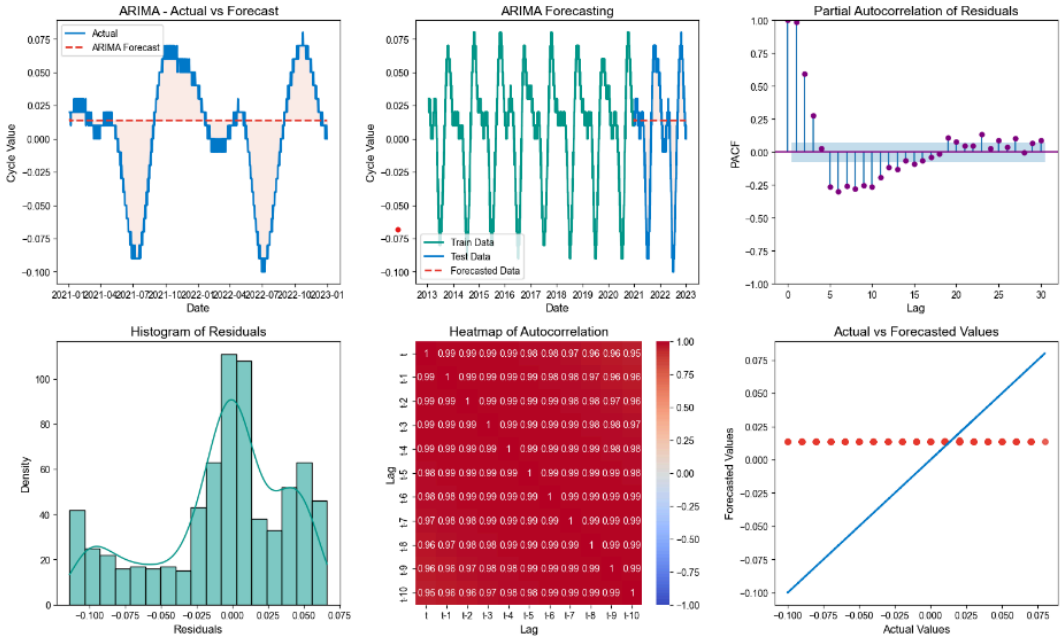
```
def hybrid_model(train_data, test_data):
# Fit ARIMA model
arima_model = ARIMA(train_data, order=(2, 1, 0))
arima_model_fit = arima_model.fit()
# Forecast using ARIMA model
arima_forecast = arima_model_fit.forecast(len(test_data))
# Calculate residuals
arima_residuals = test_data - arima_forecast
# Prepare lagged features and differenced targets for ANN
X_train_ann = train_data.shift(1).dropna().values.reshape(-1,
1)    y_train_ann =
train_data.diff().shift(-1).dropna().values.reshape(-1, 1)
X_test_ann = test_data.shift(1).dropna().values.reshape(-1, 1)
# Scale the data using StandardScaler
scaler_ann = StandardScaler()
X_train_ann_scaled =
scaler_ann.fit_transform(X_train_ann)
y_train_ann_scaled = scaler_ann.transform(y_train_ann)
X_test_ann_scaled = scaler_ann.transform(X_test_ann)
ann_model = tf.keras.Sequential([
tf.keras.layers.Dense(256, activation='relu',
input_shape=(X_train_ann_scaled.shape[1],)),
tf.keras.layers.Dropout(0.3),       tf.keras.layers.Dense(128,
activation='relu'),       tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(64, activation='relu'),
tf.keras.layers.Dense(1)    ])
ann_model.compile(optimizer=tf.keras.optimizers.Adam(learn
ing_rate=0.0005), loss='mse')
ann_model.fit(X_train_ann_scaled, y_train_ann_scaled,
epochs=300, batch_size=128, verbose=0)
ann_residuals_scaled =
ann_model.predict(X_test_ann_scaled)    ann_residuals =
scaler_ann.inverse_transform(ann_residuals_scaled).flatten(
)
if len(ann_residuals) < len(test_data):
ann_residuals = np.append(ann_residuals,
[ann_residuals[-1]])
# Append last value to match length
# Calculate hybrid forecast by combining ARIMA forecast and
ANN residuals
hybrid_forecast = arima_forecast + ann_residuals
return hybrid_forecast
```

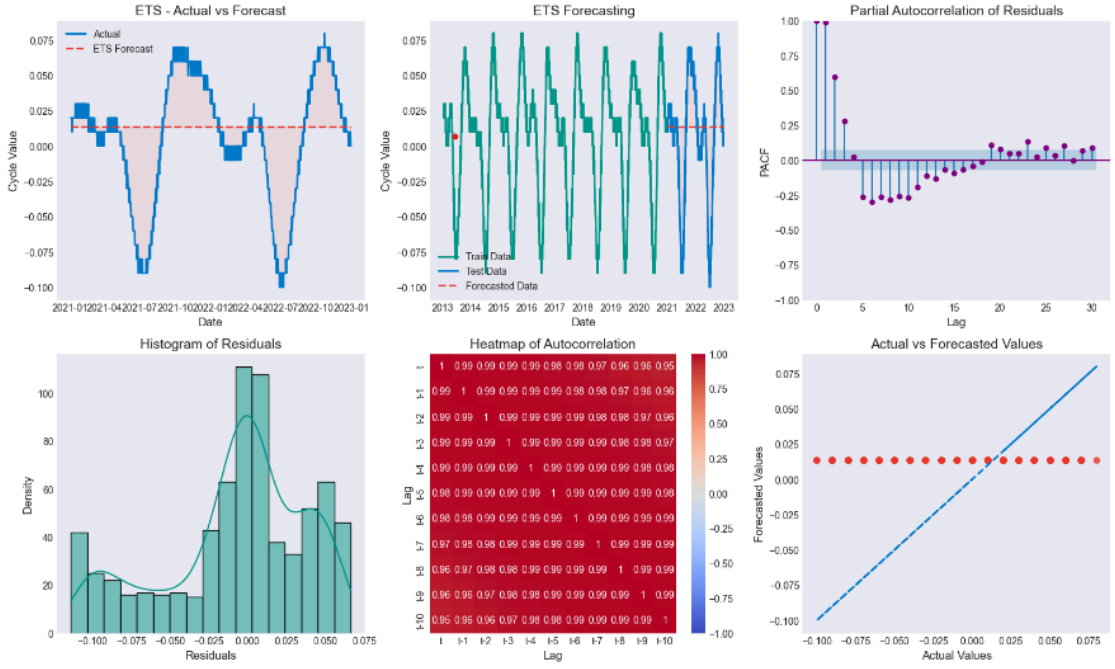
- LSTM (Long Short-Term Memory):
  - Functioning: LSTM is a type of recurrent neural network (RNN) designed to address the vanishing gradient problem in traditional RNNs. It has memory cells that can maintain information over long sequences.
  - Working: LSTM models use gated mechanisms such as input, forget, and output gates to control the flow of information through the network. This allows them to capture long-term dependencies in time series data and make accurate predictions.
  - My Model:

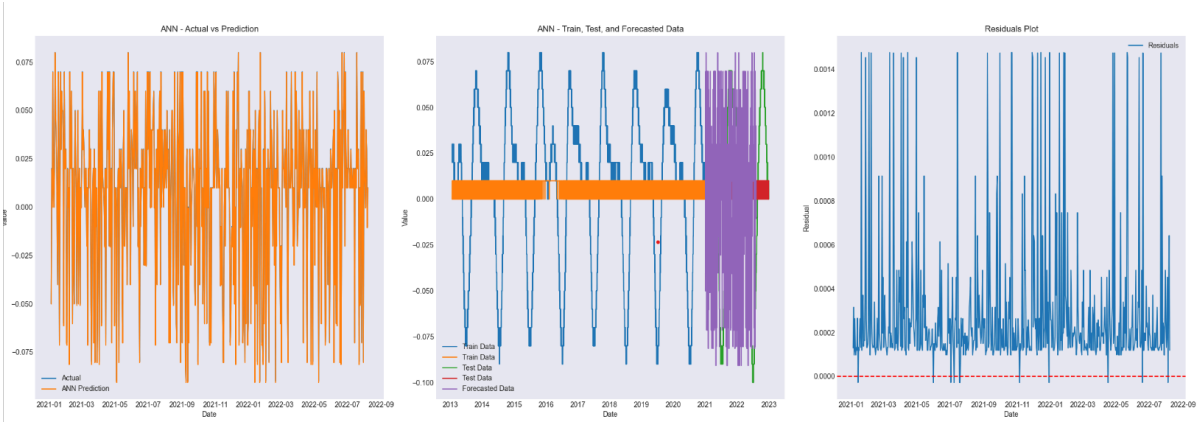
```
def lstm_model(X_train_scaled, y_train, X_test_scaled):    # Reshape input data for LSTM (samples, timesteps, features)
X_train_lstm = X_train_scaled.reshape((X_train_scaled.shape[0], 1, X_train_scaled.shape[1]))    X_test_lstm = X_test_scaled.reshape((X_test_scaled.shape[0], 1, X_test_scaled.shape[1]))        # Define LSTM model    model = Sequential()    model.add(LSTM(50, activation='relu', input_shape=(1, X_train_scaled.shape[1])))
model.add(Dense(1))    model.compile(optimizer='adam', loss='mse')
model.fit(X_train_lstm, y_train, epochs=50, batch_size=16, verbose=0)
lstm_predictions = model.predict(X_test_lstm)
return lstm_predictions.flatten()  # Ensure predictions are flattened
```

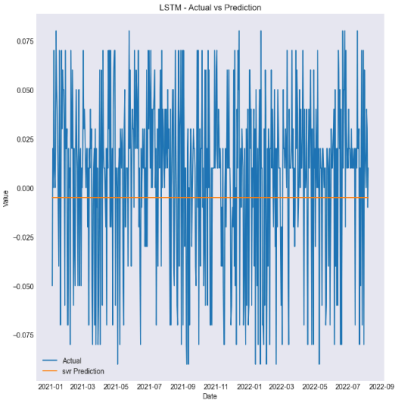
- PLOTS:

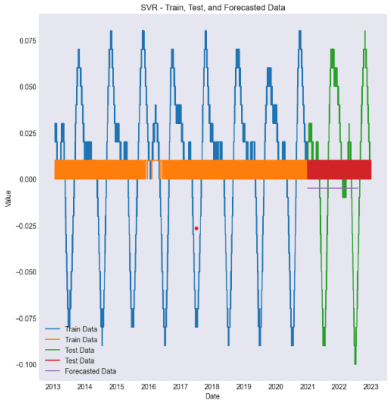
Diff plots were used to show how accurate the model is performing. We made 4 to 6 plots against each model to know how each aspect of a model is working.
Few Plots of co2 data are given below:

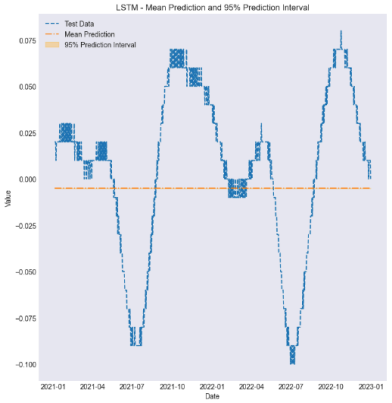

ARIMA - Actual vs Forecast
Actual
ARIMA Forecast
Cycle Value
Date
ARIMA Forecasting
Train Data
Test Data
Forecasted Data
Partial Autocorrelation of Residuals
PACF
Lag
Histogram of Residuals
Density
Residuals
Heatmap of Autocorrelation
Actual vs Forecasted Values
Forecasted Values
Actual Values

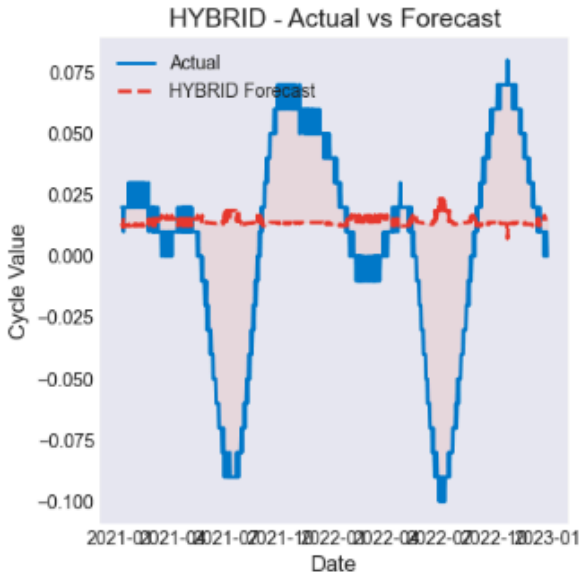

ETS - Actual vs Forecast
Actual
ETS Forecast
Cycle Value
Date
ETS Forecasting
Train Data
Test Data
Forecasted Data
Partial Autocorrelation of Residuals
PACF
Lag
Histogram of Residuals
Density
Residuals
Heatmap of Autocorrelation
Actual vs Forecasted Values
Forecasted Values
Actual Values

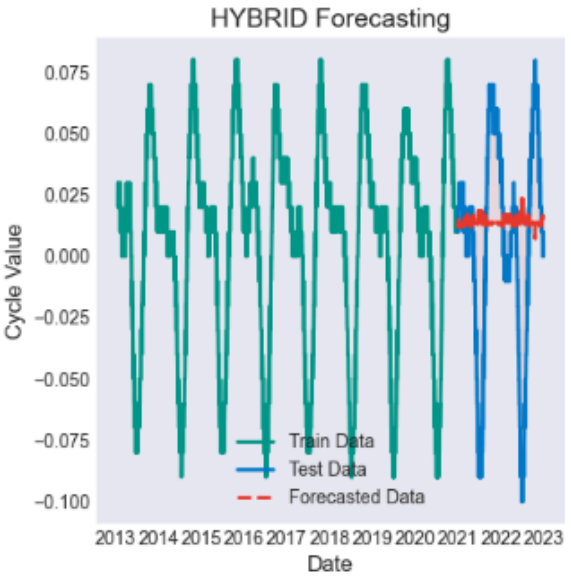

ANN - Actual vs Prediction
Actual
ANN Prediction
Date
ANN - Train, Test, and Forecasted Data
Train Data
Test Data
Forecasted Data
Value
Date
Residuals Plot
Residuals
Residual
Date

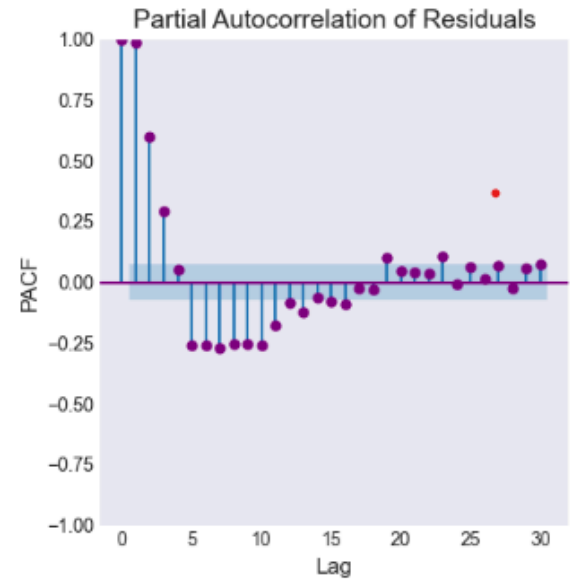

LSTM - Actual vs Prediction
Actual
svr Prediction
Value
Date

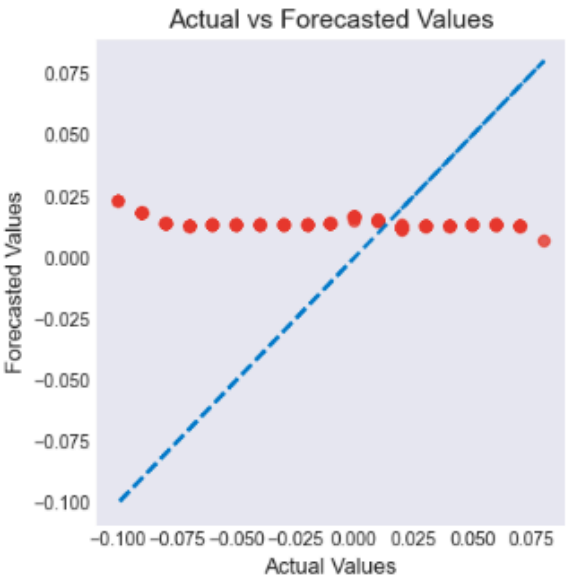

SVR - Train, Test, and Forecasted Data
Train Data
Train Data
Test Data
Test Data
Forecasted Data
Value
Date

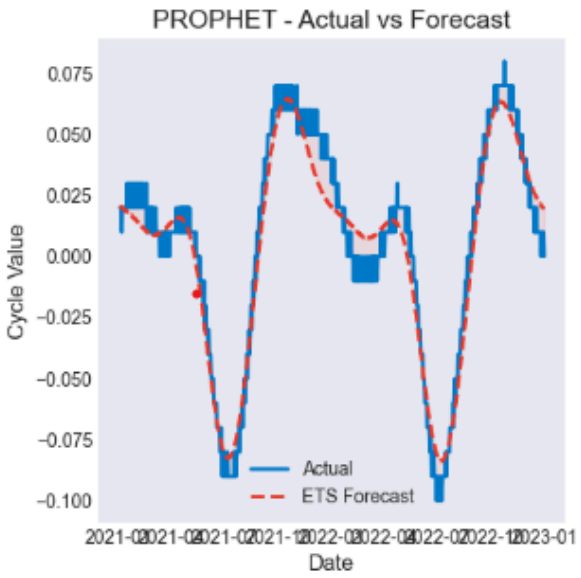

LSTM - Mean Prediction and 95% Prediction Interval
Test Data
Mean Prediction
95% Prediction Interval
Value
Date

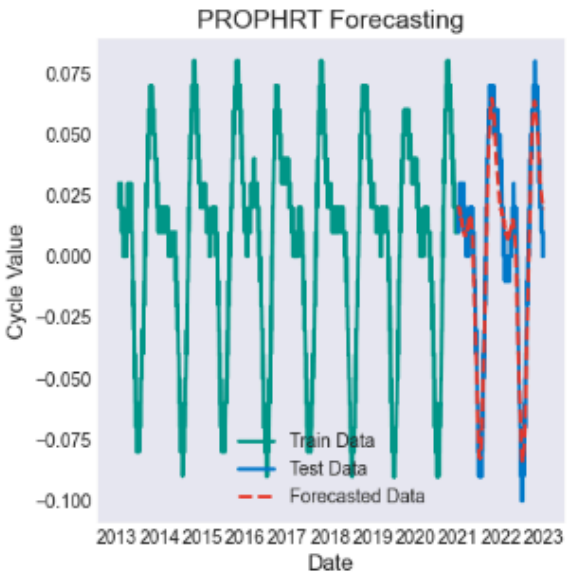

HYBRID - Actual vs Forecast
Actual
HYBRID Forecast
Cycle Value
Date

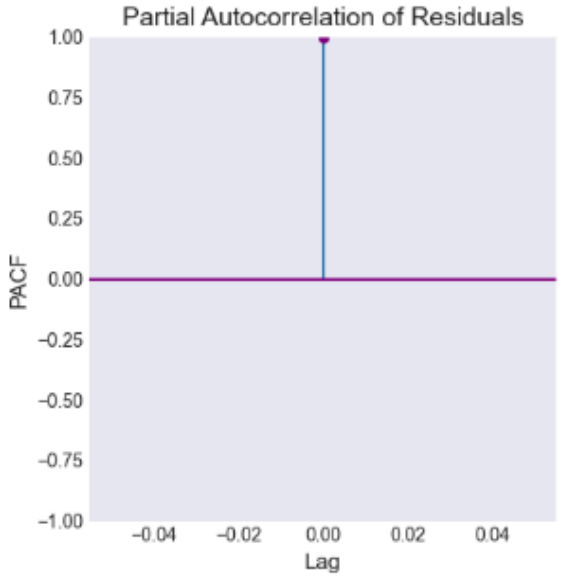

HYBRID Forecasting
Train Data
Test Data
Forecasted Data
Cycle Value
Date

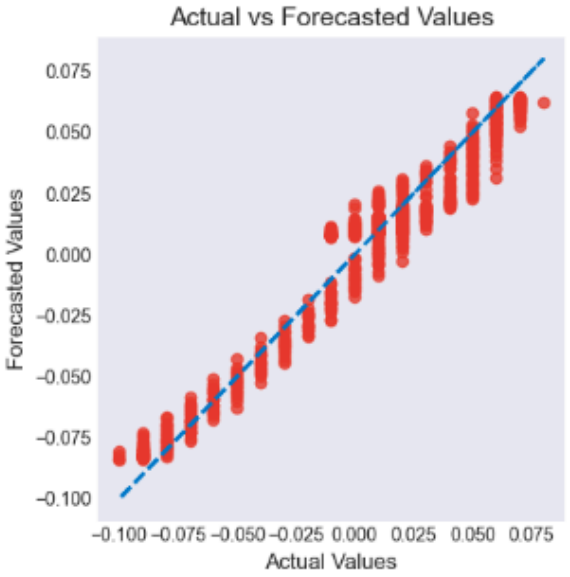

Partial Autocorrelation of Residuals
PACF
Lag


Actual vs Forecasted Values
Forecasted Values
Actual Values









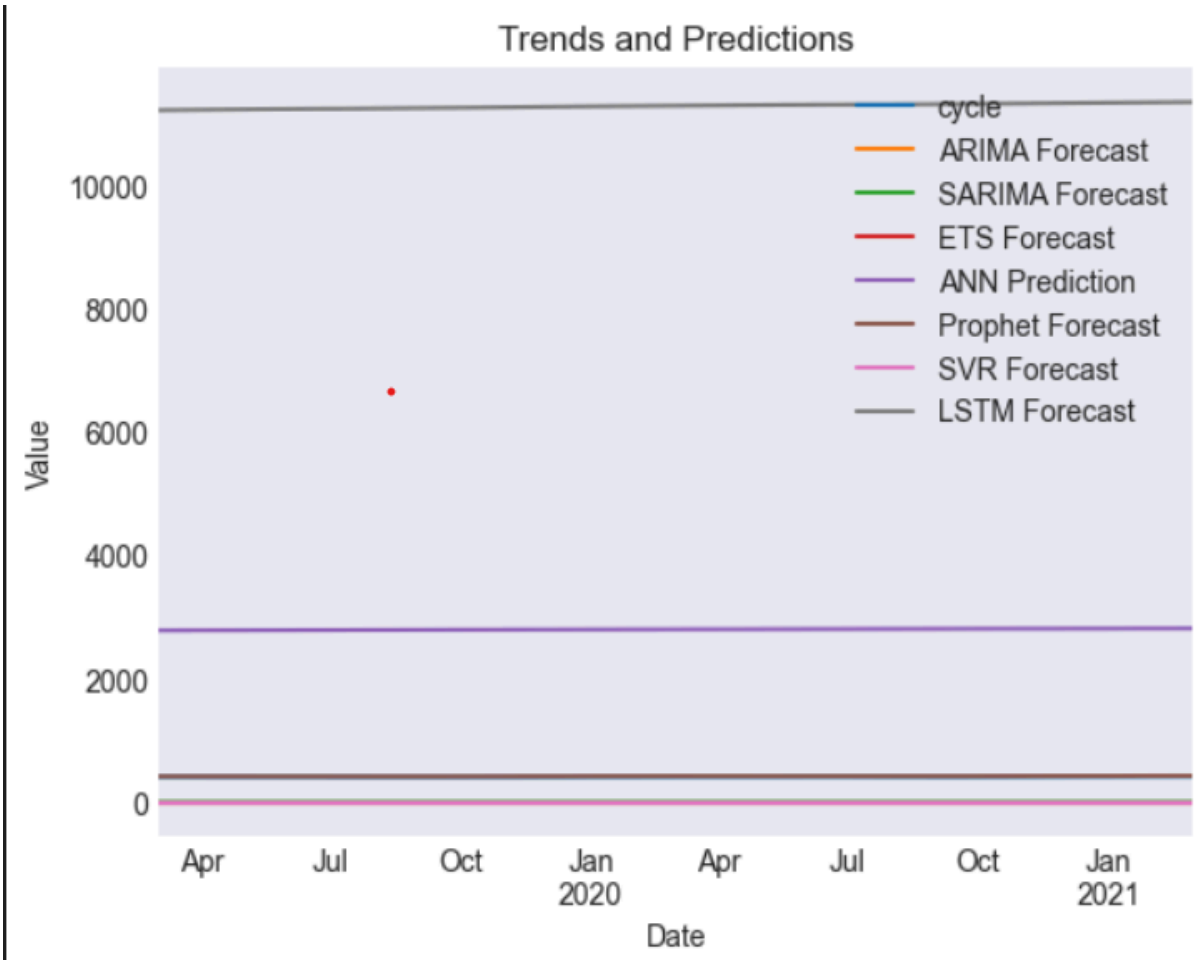
- Accuracy matrix for Co2 data:

```
The best model is LSTM.
Models sorted by RMSE in descending order:
LSTM: 0.00011320097526508696
ANN: 0.000432413945361182
Prophet: 0.009948504684815463
SVR: 0.043986482824343 87
ETS: 0.045591794488982984
ARIMA: 0.04560158101620995
hybrid: 0.04651136319370908
SARIMA: 0.05334104378862111
The best model is LSTM with RMSE of 0.00011320097526508696.
```

- Accuracy matrix of energy consumption data:

```
The best model is ANN.
Models sorted by RMSE in descending order:
ANN: 0.0008816920459519727
LSTM: 0.004568754490376169
SVR: 0.0711122466789092
ARIMA: 0.19633963983842798
ETS: 0.19675042278645913
SARIMA: 0.5962658382860141
Prophet: 0.6435645447622682
hybrid: 1.865332352751255
The best model is ANN with RMSE of 0.0008816920459519727.
```

Since the data of co2 is of daily means all dates are mentioned, we made a plot where the user will select any date and a graph will be plotted mentioning model accuracy of prediction in that time frame: (zoom in to see all models line. Some of them are overlapping because of the same prediction answers.)



FORECASTING APPLICATION

- Introduction:

  The CO2 forecasting web application is designed to provide users with a platform to forecast CO2 levels using various time series forecasting models. This report outlines the development process, functionality, and user interface of the application, as well as its accuracy and utility.

- Development Process:

  The application was developed using Python programming language with Flask web framework for the backend and HTML/CSS for the frontend. Various time series forecasting models such as ARIMA, SARIMA, ETS, Prophet, ANN, SVR, Hybrid, and LSTM were implemented using libraries like statsmodels, scikit-learn, and TensorFlow.

- Functionality:
  - Input Form:

    The application allows users to input parameters such as start date, end date, and choice of forecasting model through an intuitive web form.
  - Model Selection:
    - Users can choose from a range of forecasting models including traditional statistical methods like ARIMA and machine learning techniques like ANN and SVR.
  - Visualization:
    - The application provides interactive visualizations such as ACF and PACF plots, forecast plots based on selected date range, and plots for all forecasting models.
  - Accuracy Metrics:
    - Users can view accuracy metrics for each forecasting model, including Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE).
  - Best Model Selection:
    - The application identifies the best performing model based on RMSE and provides an option to view its forecast plot and accuracy metrics.
  - User Interface:
    - The user interface is designed to be user-friendly and intuitive. It features a clean and simple layout with easy navigation and interactive elements. Users can easily input parameters, select forecasting models, and visualize forecast results with minimal effort.

- How to Use:
    - Access the web application through the provided URL.
    - Input the start date and end date for the forecast period.
    - Choose a forecasting model from the dropdown menu.
    - Select options such as viewing ACF and PACF plots, seeing all RMSE values, and viewing the best performing model.
    - Click on the "Submit" button to generate the forecast and visualize the results.
- Accuracy and Validation:
    - The accuracy of the forecasting models is validated using various performance metrics such as MAE, MSE, and RMSE. The application provides users with detailed accuracy metrics for each model, allowing them to assess the reliability and performance of the forecasts. Additionally, the application identifies the best performing model based on RMSE, ensuring that users can make informed decisions based on forecast accuracy.

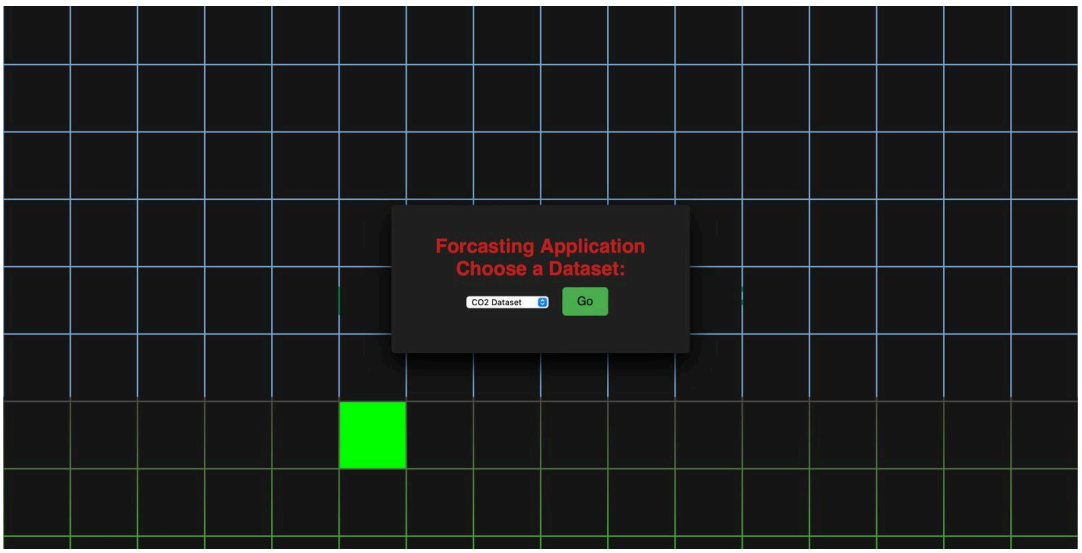
- Conclusion:

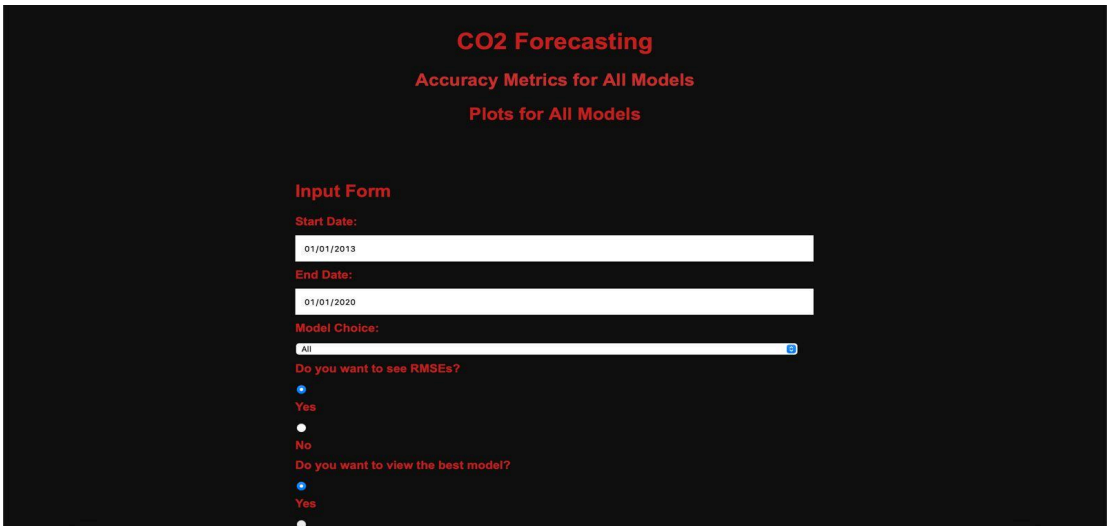
    The CO2 forecasting web application offers users a powerful tool for predicting CO2 levels using state-of-the-art time series forecasting models. With its user-friendly interface, interactive visualizations, and accurate forecasts, the application provides users with valuable insights into future CO2 trends, helping them make informed decisions and take appropriate actions to mitigate environmental impact.

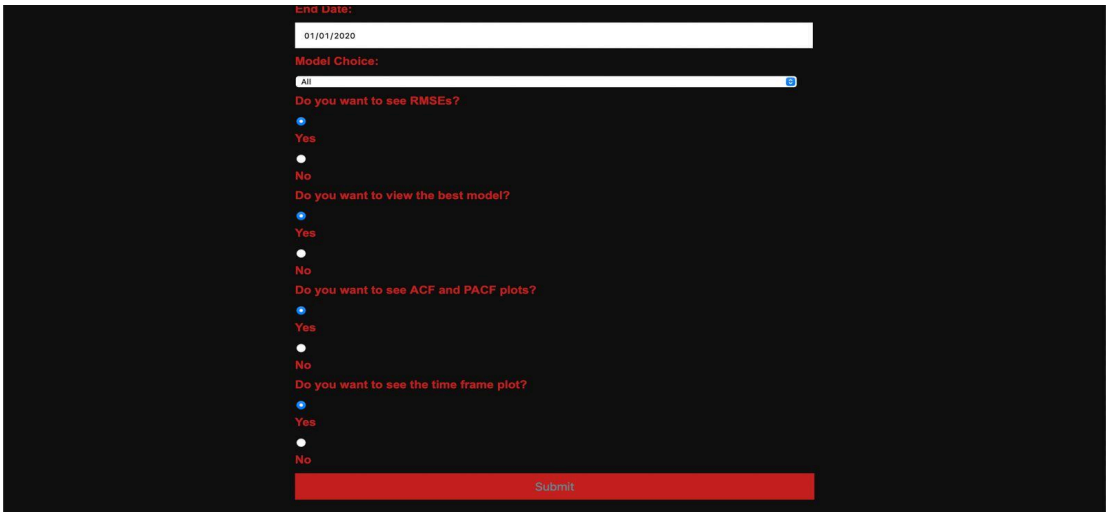
- How to run:

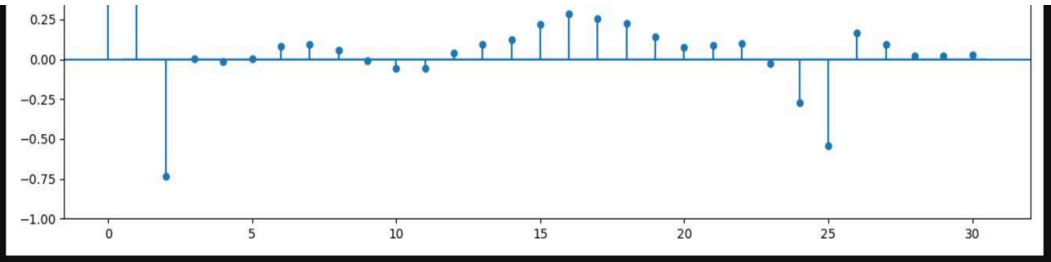
Load the project into visual studio code.
Open 3 terminals:
1st terminal: python verfinal.py
2nd terminal: python verfinal2.py
3rd terminal: python home.py
Open the link you get on the 3rd terminal.
Project is loaded and ready to go.

WEBPAGE IMAGES:


Forcasting Application
Choose a Dataset:
CO2 Dataset
Go


CO2 Forecasting
Accuracy Metrics for All Models
Plots for All Models
Input Form
Start Date:
01/01/2013
End Date:
01/01/2020
Model Choice:
All
Do you want to see RMSEs?
Yes
No
Do you want to view the best model?
Yes


End Date:
01/01/2020
Model Choice:
All
Do you want to see RMSEs?
Yes
No
Do you want to view the best model?
Yes
No
Do you want to see ACF and PACF plots?
Yes
No
Do you want to see the time frame plot?
Yes
No
Submit

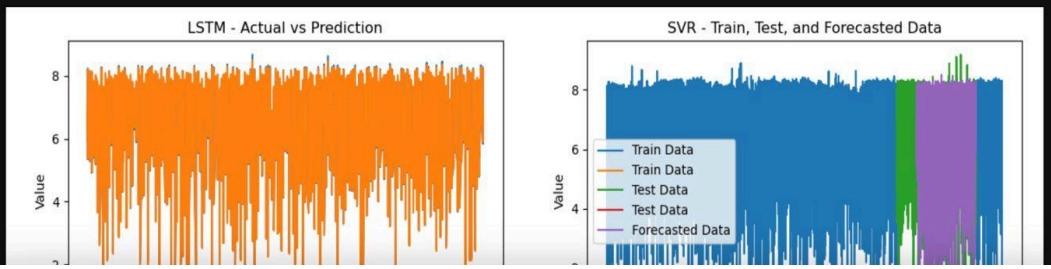


Accuracy Metrics for All Models

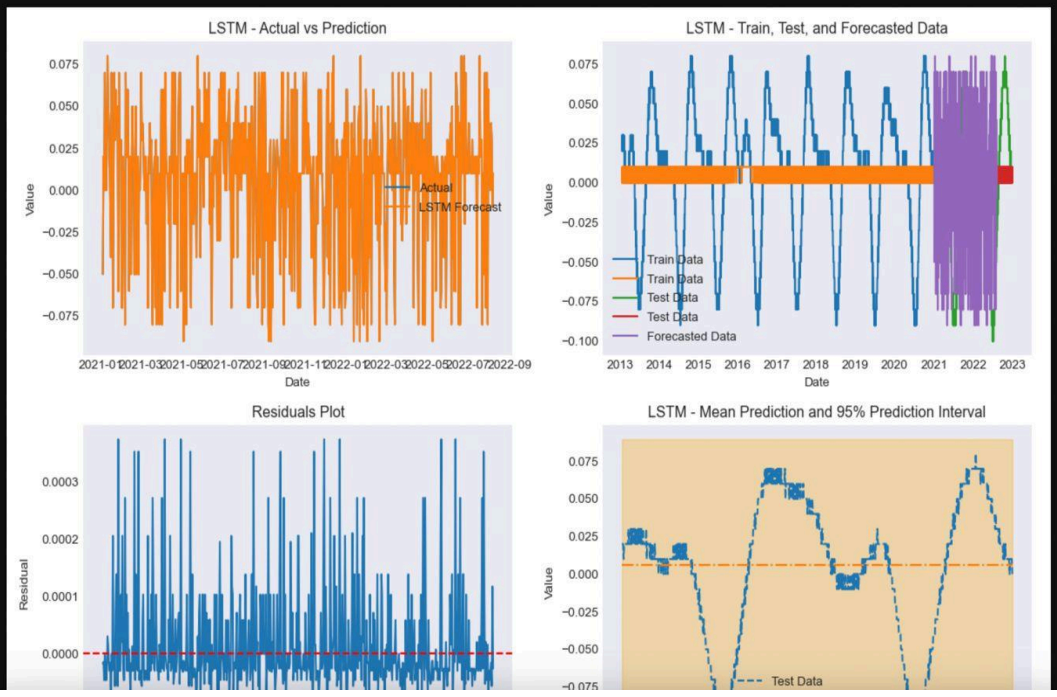
- SVR: MAE: 0.06509333632519175, MSE: 0.00505695162772141, RMSE: 0.07111224667890483

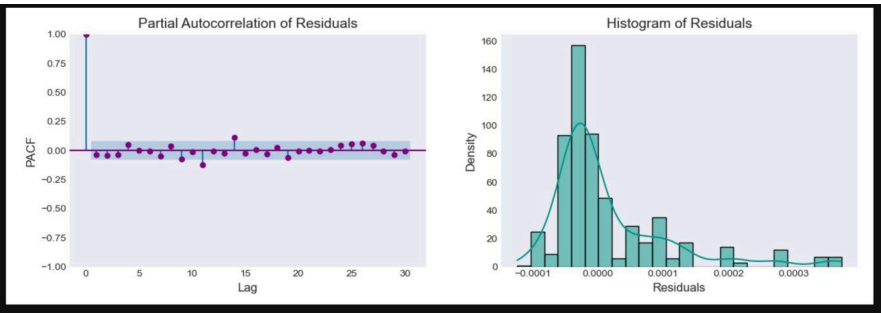
Plots for All Models

SVR



LSTM

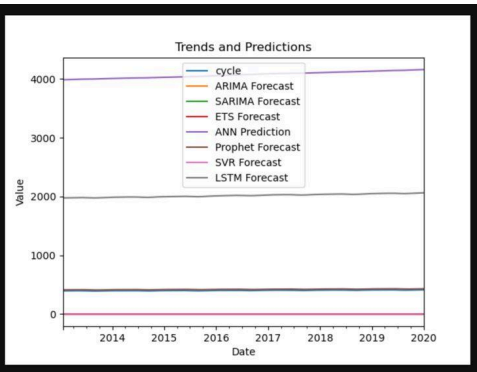




All Models RMSE

RMSE for each model:

Model	RMSE
LSTM	9.44018105794585e-05
ANN	0.0006867797845848511
Prophet	0.00993733126295502
SVR	0.04398648282434387
SARIMA	0.0460617906760102
Hybrid	0.04664816918367882
ETS	0.05637581569068316
ARIMA	0.17085598639712316



Accuracy Metrics for All Models

- ARIMA: MAE: 0.16484474558982645, RMSE: 0.17085598639712316
- SARIMA: MAE: 0.034207603047681265, RMSE: 0.0460617906760102
- ETS: MAE: 0.04116117463628497, RMSE: 0.05637581569068316
- Prophet: MAE: 0.00820042740368562, RMSE: 0.00993733126295502
- ANN: MAE: 0.0005983780637680911, MSE: 8.406461570002127e-07, RMSE: 0.0009168675787703548
- SVR: MAE: 0.03718588640275117, MSE: 0.0019348106712562985, RMSE: 0.04398648282434387
- Hybrid: MAE: 0.035196798439161 36, RMSE: 0.046418306169532804
- LSTM: MAE: 6.053008087859711e-05, RMSE: 9.31347571378772e-05

CO2 Forecasting