**Testing –Forecasting Supply and Demand**

**Git Hub Repo Link:** [**https://github.com/Karpagavinayagam-NoblQ/Test\_PyTest\_Supply\_Demand.git**](https://github.com/Karpagavinayagam-NoblQ/Test_PyTest_Supply_Demand.git)

**Scope:**

This test document outlines the different types of testing and the procedures for executing tests on the Supply and Demand forecasting application, which is powered by the Prophet model. The test cases are designed to validate the app's features by injecting mock data after the model has been trained using known historical data. In the future, the actual data provided by the client will replace the mock data for testing. Additionally, another test case will validate the model's performance using the Pytest framework.

1. **Testing – using the mock data**

**Source Code: test\_supply\_demand-plot-graph.py**

**QA Engineer Testing Process for Supply and Demand Forecasting Model**

After the Data Engineer has trained and validated the forecasting model using the Prophet algorithm, the QA Engineer will follow these steps to test the model's performance and validate the app's functionality.

**Steps for QA Engineer to Execute:**

1. Inject Mock Data into DataFrames (df\_supply and df\_demand):

In the script Create\_DataFrame() located in ../test/test\_supply\_demand-plot-graph.py, the QA Engineer will inject mock supply and demand data to test the model's functionality.

**Example mock data for supply (df\_supply) and demand (df\_demand):**

*df\_supply = {*

*'ds': ['01-12-2024', '12-12-2024', '15-12-2024', '18-12-2024', '21-12-2024', '22-12-2024', '25-12-2024', '26-12-2024'],*

*'y': [71835.71, 86449.29, 70005.61, 86409.22, 81835.71, 86949.29, 51835.71, 87449.29]*

*}*

*df\_demand = {*

*'ds': ['03-12-2024', '08-12-2024', '10-12-2024', '12-12-2024', '16-12-2024', '17-12-2024', '19-12-2024', '21-12-2024'],*

*'y': [93.12, 97.43, 89.89, 87.67, 94.34, 88.45, 90.30, 97.23]*

*}*

The mock data simulates the historical supply and demand values for a given period. These values will be used for training and validating the Prophet model.

1. Execute Streamlit Command:

Once the mock data has been added to the script, the QA Engineer will execute the following Streamlit command to launch the application and visualize the results:

***streamlit run .\test\test\_supply\_demand.py***

1. Launch the Application:

This command will start the application, and the QA Engineer can access it at the following URL:

http://localhost:8502/

1. View the MAE, RMSE, and R² Values:

Once the app is launched, the QA Engineer will see the MAE (Mean Absolute Error), RMSE (Root Mean Squared Error), and R² (R-Squared) values displayed at the top of the page.

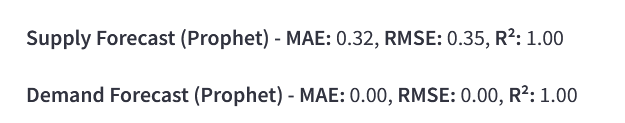
These metrics will help assess the accuracy and performance of the Prophet model. Specifically:

MAE: Measures the average magnitude of errors in the forecast.

RMSE: Provides an indication of how well the model's predictions match actual values, penalizing large errors more heavily.

R²: Indicates how well the model's predictions fit the actual data (ranges from 0 to 1).

Example Output:



1. Forecasted Supply and Demand Values:

* The page will also display the forecasted supply and demand values for the future, based on the trained Prophet model.
* This table will show the forecasted supply (in USD) and demand (as a percentage) for each date in the future period.

Example output:



1. Actual vs Predicted Graphs:

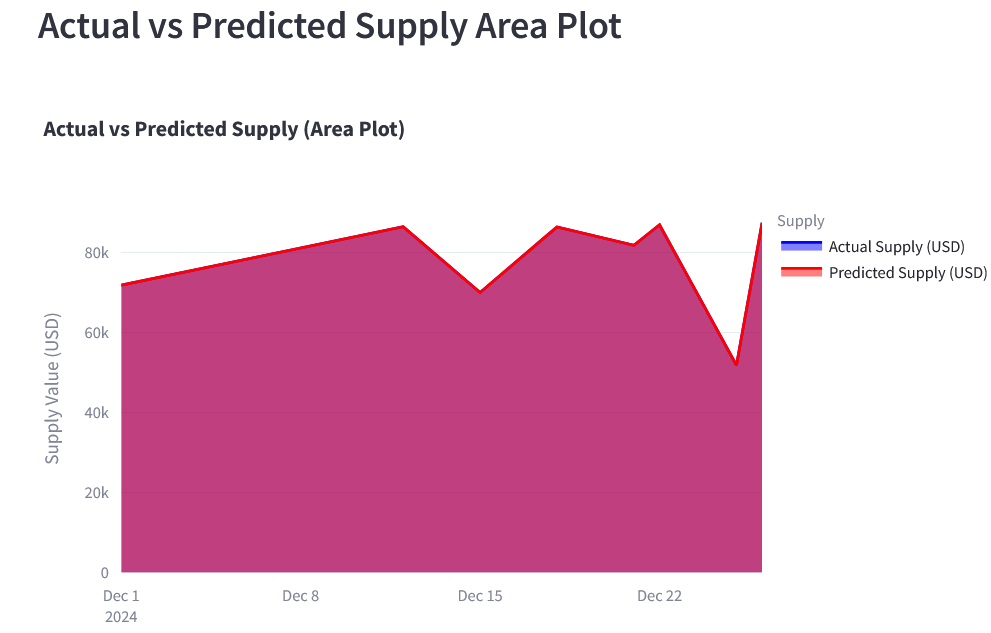
The page will display graphical comparisons between actual and predicted values for both supply and demand. These graphs are visual representations of how well the model's predictions align with the actual values.

Area Plot: A filled line plot that shows the actual and predicted values over time for Supply and Demand.

The blue line represents the actual supply values.

The red line represents the predicted supply values.

Similar plotting is done for Demand, using the green line for actual values and the orange line for predicted values.

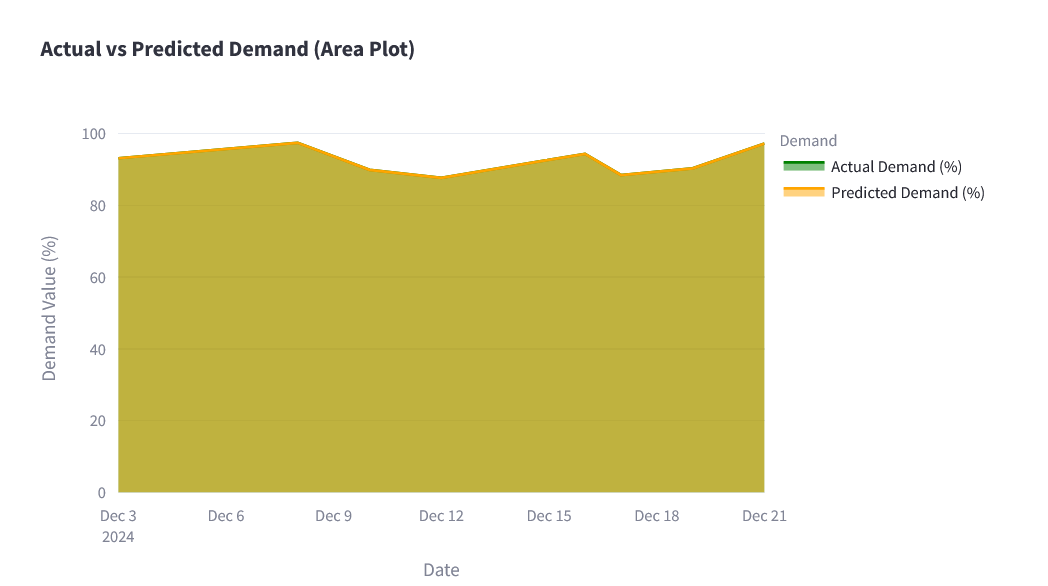


Example of the area plot for Supply:

* Actual Supply (blue) vs Predicted Supply (red)

Example of the area plot for Demand:

* Actual Demand (green) vs Predicted Demand (orange)

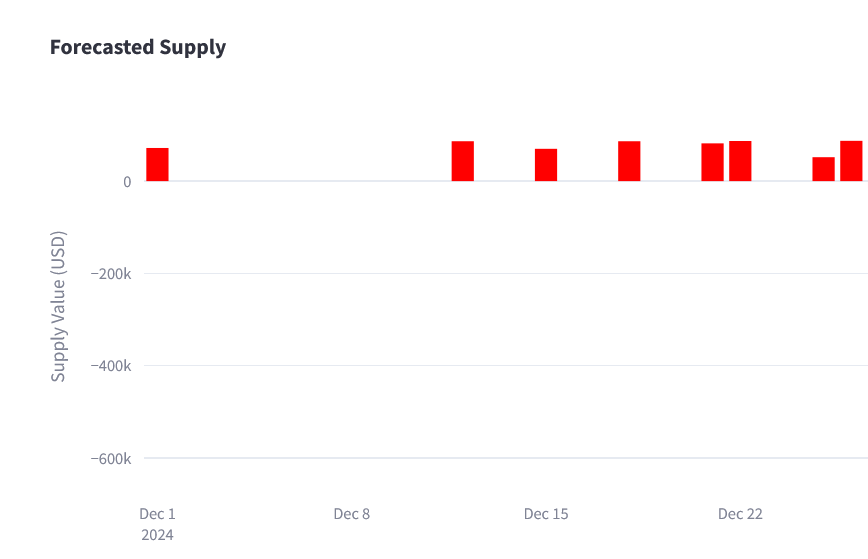


1. Forecasted Supply and Demand Bar Plots:

The page will also display bar plots that visually represent the forecasted supply and demand for the future period.

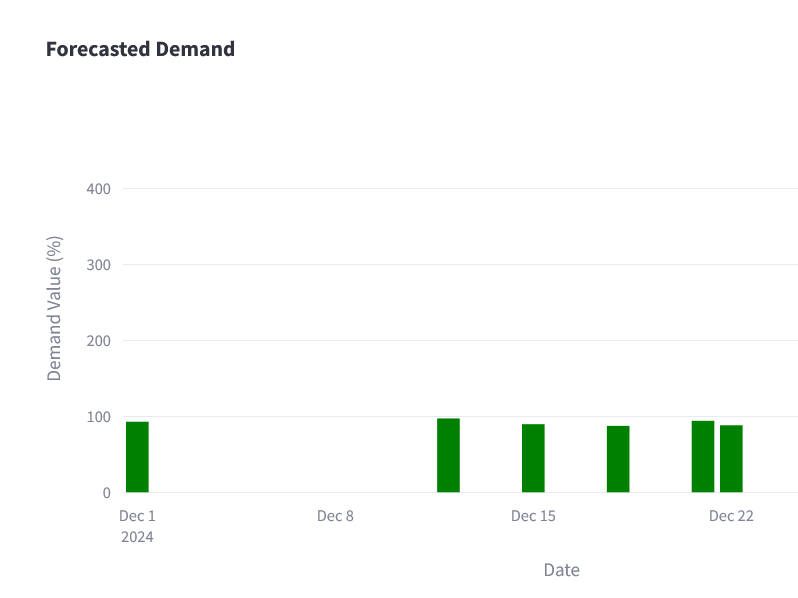
These plots will help the QA Engineer visually inspect the forecast values and check the model's predictions in a clear and simple format.

Example of the bar plot for Supply Forecast:



Bars representing forecasted supply values for each date in the forecasted period.

Example of the bar plot for Demand Forecast:



Bars representing forecasted demand values for each date in the forecasted period.

**Expected Results for QA Testing:**

* MAE, RMSE, and R² values should be displayed accurately.
* Forecasted values for supply and demand should match the expected outputs based on the model's training.
* Graphs and Bar Plots should accurately visualize the actual vs predicted supply and demand values over time.
* Usability: The app should be easy to interact with, and the data should update correctly when new data (e.g., client data) is used in the future. After running the script for training and validation of the model by Data Engineer, the QA Engineer will execute the following steps.

1. **Testing – using PyTest Framework**

**Source Code: test\_supply\_demand.py**

**Test Cases:**

The Pytest framework is a widely used testing framework in Python, known for its simplicity and scalability. It enables developers to write efficient and readable test cases for applications, supporting both unit testing and functional testing. Pytest offers a variety of features that enhance the testing process, such as fixtures, parameterization, and detailed reporting.

Below are the test cases that will be executed using the Pytest framework:

**Test Case for Data Creation:** Verifies that the process of creating or generating data functions correctly.

**Test Case for Fitting Prophet Models:** Ensures that the Prophet models are properly fitted to the provided data.

**Test Case for Forecasting and Metrics Calculation:** Validates the accuracy of the forecasting process and ensures that the metrics calculations produce the expected results.

This framework provides a streamlined and efficient way to ensure the functionality of these critical components.

**Steps for QA Engineer to Execute:**

To execute the test cases in the PyTest framework, follow these steps. This process will allow the QA Engineer to validate the functionality of the Supply and Demand Forecasting App using the automated tests defined in test\_supply\_demand.py.

**Steps to Run PyTest:**

1. Prepare the Environment:

* Ensure that the virtual environment (or the environment where the dependencies are installed) is activated.
* The dependencies, including pytest, prophet, pandas, scikit-learn, and streamlit, should already be installed in your environment.

1. Navigate to the Test Directory:

* Open a terminal or command prompt window and navigate to the directory containing the test\_supply\_demand.py file.

***cd path/to/your/project/test***

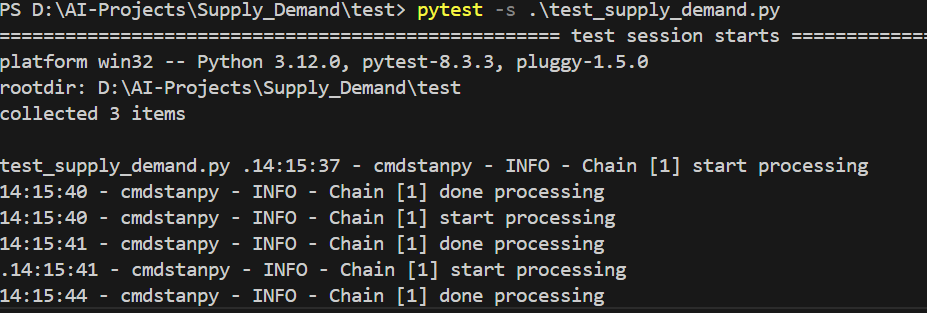
1. Run the PyTest Command:

* Execute the following command to run the test cases in test\_supply\_demand.py:

***pytest -s test\_supply\_demand.py***

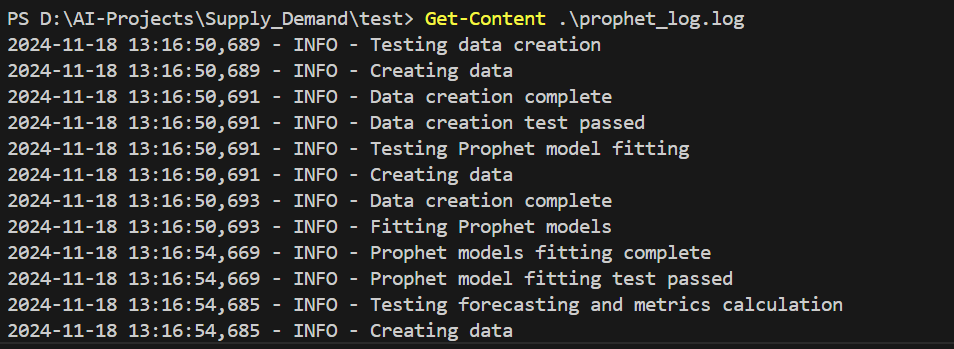
1. Example Output:

The terminal will show the status of each test case in the test suite as it runs. Here is an example of how the output will look:



The logs will be stored in .\prophet\_log.log file,

To view the log use the command **Get-Content .\prophet\_log.log**



PASSED means that the test case executed successfully without errors.

If any of the tests fail, the status will be displayed as FAILED and detailed information about the error will be provided, including the specific assertion or issue that caused the failure.

**What to Check in the PyTest Output:**

* Ensure that all tests pass with the status PASSED.
* If a test fails, review the error message or traceback to identify the cause of the failure.
* It may be due to issues like incorrect data, model misconfiguration, or invalid predictions.

**In case of test failures:**

If any of the tests fail, the error message will give details, such as:

* Assertion errors (e.g., if expected values don't match the actual values).
* Errors during execution (e.g., issues with the Prophet model training or predictions).