Approach Notes

Step 1: Data Generation, Features Used for Training, and Training Classifier Model

The dataset used for this entire process is the sample data provided in the hackathon.

Initial Approach:

I divided the problem into four formats: hourly, daily, weekly, and monthly, to align with the dataset's format. Consequently, I needed to build four classifier models to predict these four different formats. I had several reasons to believe that this approach would be more effective than a combined model approach:

1. Each classifier is trained on a specific data format, allowing them to specialize in identifying the best time series model for that particular format. This specialization can lead to more accurate predictions for each format.
2. Different data formats may require distinct sets of features or preprocessing steps. By training separate classifiers, I could customize the feature engineering process for each format, potentially improving model performance. Additionally, it allows me to identify which features are most important for each format.
3. Real-world scenarios often exhibit different patterns across time formats. Therefore, having separate classifiers could make it easier to predict the best time series model for each format.

However, it's important to acknowledge a limitation of this approach:

- If we want to predict a format (e.g., yearly) that is not within the four predefined formats, the model may not perform well since it was not trained on that specific dataset.
- Employing multiple classifiers increases the complexity of the system. If simplicity is a priority, a single classifier might be a more suitable choice.

Therefore, I assumed that users would try forecasting for the predefined formats. If other formats are required, we can try to aggregate values and convert them into one of the four predefined formats as needed.

__Data Generation Process:__

I utilized four time series models: 'ARIMA,' 'XGBoost,' 'Prophet,' and 'ETS.' For all the formats, For particular format i combined all the samples and performed data preprocessing, which involved removing duplicate outliers and interpolating missing values.

After preprocessing, use that preprocessed dataset to generate samples to determine data points resulting in the best time series model with low MAPE. I employed two approaches to achieve this:

1. __Rolling window samples:__ I used a fixed rolling window size, such as 25 data points. For each window, I trained and fitted all the models. Subsequently, I selected the best model based on low Mean Absolute Percentage Error (MAPE) for test values. For **feature extraction**: I extracted various statistical features, including mean, variance, skew, and kurtosis, as well as two temporal features: seasonality trend residual. These features were calculated for each rolling window, and the mean values across the features were considered as one data point. This formed the basis of my training sampling.

Mean	**Variance**	**Kurtosis**	**Trend Mean**	**Seasonal Mean**	**Residual Mean**	**Best Model**

After generating data for one format, I noticed that it was predicting only one model for the entire dataset, which is not ideal for sampling.

I then employed a technique called TimeSeriesSplit, which is similar to rolling windows but includes past data cumulatively. I followed the same process as the rolling window approach. As a result, I obtained some data points for all the time series formats. However, the dataset was not balanced.

Now, we have a labeled dataset containing all the classes, but it's unbalanced. To address this, we need to balance it by undersampling the more frequent class labels and oversampling the less frequent data points. Additionally, during the preprocessing step, I observed that there was no correlation between skewness and the best model. Therefore, I removed that particular feature for all the formats.

Subsequently, I utilized a Random Forest classifier with multiple parameters in a grid search to obtain the best model:

```
Format hourly Index(['Mean', 'Variance', 'Seasonal Mean',
'Residual Mean', 'Best Model',
       'model'],
      dtype='object')
Class distribution before undersampling:
Counter({0: 83, 3: 43, 2: 40, 1: 34})
Class distribution after undersampling:
Counter({0: 30, 1: 30, 2: 30, 3: 30})
Class distribution after upsampling:
Counter({0: 30, 1: 30, 2: 30, 3: 30})
```

```
Best Parameters: {'max_depth': 20, 'n_estimators': 200}
Classification Report:
              precision    recall  f1-score  support
0              0.400000  0.333333  0.363636     6.00
1              0.285714  0.285714  0.285714     7.00
2              0.333333  0.142857  0.200000     7.00
3              0.111111  0.250000  0.153846     4.00
accuracy       0.250000  0.250000  0.250000     0.25
macro avg      0.282540  0.252976  0.250799    24.00
weighted avg   0.299074  0.250000  0.258217    24.00
-----------------------------------------------------------------
----
Format monthly Index(['Mean', 'Variance', 'Kurtosis', 'Trend
Mean', 'Seasonal Mean',
       'Residual Mean', 'Best Model', 'model'],
      dtype='object')
Class distribution before undersampling:
Counter({1: 36, 0: 30, 3: 23, 2: 11})
Class distribution after undersampling:
Counter({0: 25, 1: 25, 3: 23, 2: 11})
Class distribution after upsampling:
Counter({0: 25, 1: 25, 2: 25, 3: 25})

Classifier: RandomForest
Best Parameters: {'max_depth': 10, 'n_estimators': 200}
Classification Report:
              precision    recall  f1-score  support
0              0.333333  0.166667  0.222222     6.00
1              0.500000  0.500000  0.500000     6.00
```

```
2               0.400000  1.000000  0.571429     2.00
3               0.166667  0.166667  0.166667     6.00
accuracy        0.350000  0.350000  0.350000     0.35
macro avg       0.350000  0.458333  0.365079    20.00
weighted avg    0.340000  0.350000  0.323810    20.00

-------------------------------------------------------------------
----
Format weekly Index(['Mean', 'Variance', 'Kurtosis', 'Trend
Mean', 'Best Model', 'model'], dtype='object')
Class distribution before undersampling:
Counter({1: 27, 2: 19, 3: 16, 0: 13})
Class distribution after undersampling:
Counter({1: 25, 2: 19, 3: 16, 0: 13})
Class distribution after upsampling:
Counter({0: 25, 1: 25, 2: 25, 3: 25})

Classifier: RandomForest
Best Parameters: {'max_depth': 30, 'n_estimators': 200}
Classification Report:
              precision    recall  f1-score  support
0               0.800000  0.888889  0.842105     9.00
1               0.666667  0.400000  0.500000     5.00
2               1.000000  0.800000  0.888889     5.00
3               0.333333  1.000000  0.500000     1.00
accuracy        0.750000  0.750000  0.750000     0.75
macro avg       0.700000  0.772222  0.682749    20.00
weighted avg    0.793333  0.750000  0.751170    20.00
```

Combined Approach:

Once the data is generated, I have four models. However, I thought , since I had prepared the data schema to be similar for all four formats, why not join the data and use it to create a single classifier? This approach could potentially address the limitations I encountered in the previous method.

By combining all the data into one dataset, it allows the classifier to learn from a broader range of examples. This can be advantageous when the decision boundary between formats is not clear-cut or if there are similarities between the formats. While this approach makes modeling easier, it's important to note that most real-world scenarios would likely have distinct decision boundaries between the formats.

I proceeded to combine all the preprocessed data into a single file and utilized it to classify the model. Similar to the previous approach, I needed to balance the dataset. I employed a Random Forest classifier in a grid search to find the best parameters. The results were as follows:

```
Class distribution before undersampling:
Counter({0: 266, 1: 132, 3: 96, 2: 81})
Class distribution after undersampling:
Counter({0: 100, 1: 100, 3: 96, 2: 81})
Class distribution after upsampling:
Counter({0: 100, 1: 100, 2: 100, 3: 100})

Classifier: RandomForest
Best Parameters: {'max_depth': 10, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 100}
Classification Report:
              precision    recall  f1-score   support
0              0.600000  0.346154  0.439024   26.000
1              0.333333  0.333333  0.333333   18.000
2              0.555556  0.652174  0.600000   23.000
3              0.400000  0.615385  0.484848   13.000
accuracy       0.475000  0.475000  0.475000    0.475
macro avg      0.472222  0.486761  0.464302   80.000
weighted avg   0.494722  0.475000  0.468971   80.000
```

Step 2: Building Backend API and Frontend

In the previous process, I pickled the classifier models. In the FastAPI, I created two endpoints to access models:

```
POST
/predict?date_from=2021-08-01&date_to=2021-08-03&period=0
POST
/predict_with_format?format=daily&date_from=2021-08-01&date_to=2021-08-03&p
eriod=0
```

For the frontend, I designed a streamlined interface consisting of three pages. Two of these pages resemble a Postman-like interface, allowing users to access the endpoints easily. The third page serves as a user-friendly interface, where users can upload a CSV file along with the desired forecasting periods. This page returns a chart and predicted values as output.

Future Works and Decisions that Would Have Been Avoided:

What can we do if we want to add a new time series model?

If you want to add a new time series model to your existing system, you can follow these steps:

1. Decide on the new time series forecasting model you want to add to your system. It could be a well-established model or a custom one.
2. Ensure that your new model can work with the **same data schema** and format that you have used for your existing models. If the new model requires different data features or preprocessing steps, you may need to modify your data generation and preprocessing process to accommodate these changes.
3. Train the new time series model using your existing dataset alongside the other models. This means that you would include the new model in your ensemble of classifiers. assess the performance of the new model by comparing Mean Absolute Percentage Error (MAPE) and ensure that the new model provides meaningful improvements or complements the existing models.

When generating data, ensure that the training dataset is specific to each model. Feature engineering should include the **normalization** of features. This would make the features consistent across all models, reducing bias in the modeling process and potentially improving model performance.

In the feature extraction step, more effort could have been put into identifying and selecting features that are **highly relevant** and **representative** of the underlying patterns in the data. This may require **domain knowledge** or further data analysis to uncover important features, as we assumed that we will be using only univariate data.

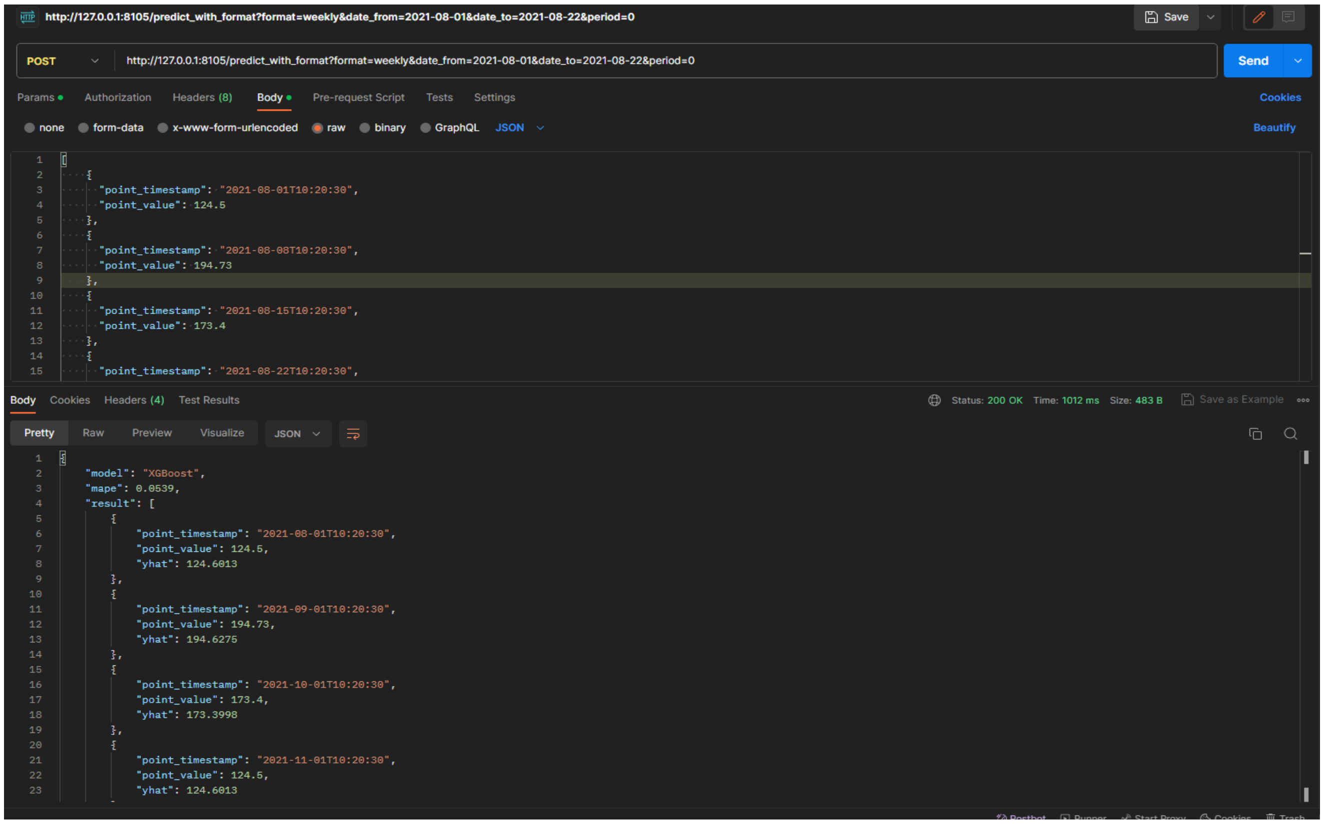
By taking these steps, the addition of a new time series model could have been smoother, and the models could have been more robust and adaptable to various formats. These decisions would have enhanced the flexibility and accuracy of the forecasting system.

We can continuously capture new data points in the real-world application and utilize them to understand the behavior of the new time series model. This data-driven approach allows us to finetune the model, enhancing its accuracy over time.

Furthermore, it's crucial to tune the hyperparameters of the time series models meticulously. This hyperparameter tuning process aims to ensure that each model is optimized for its specific task, ultimately leading to improved classification accuracy.

By following these practices, we can adapt and enhance our forecasting system to perform better as it encounters new data and challenges in the real world.

<u>Application Screenshots:</u>



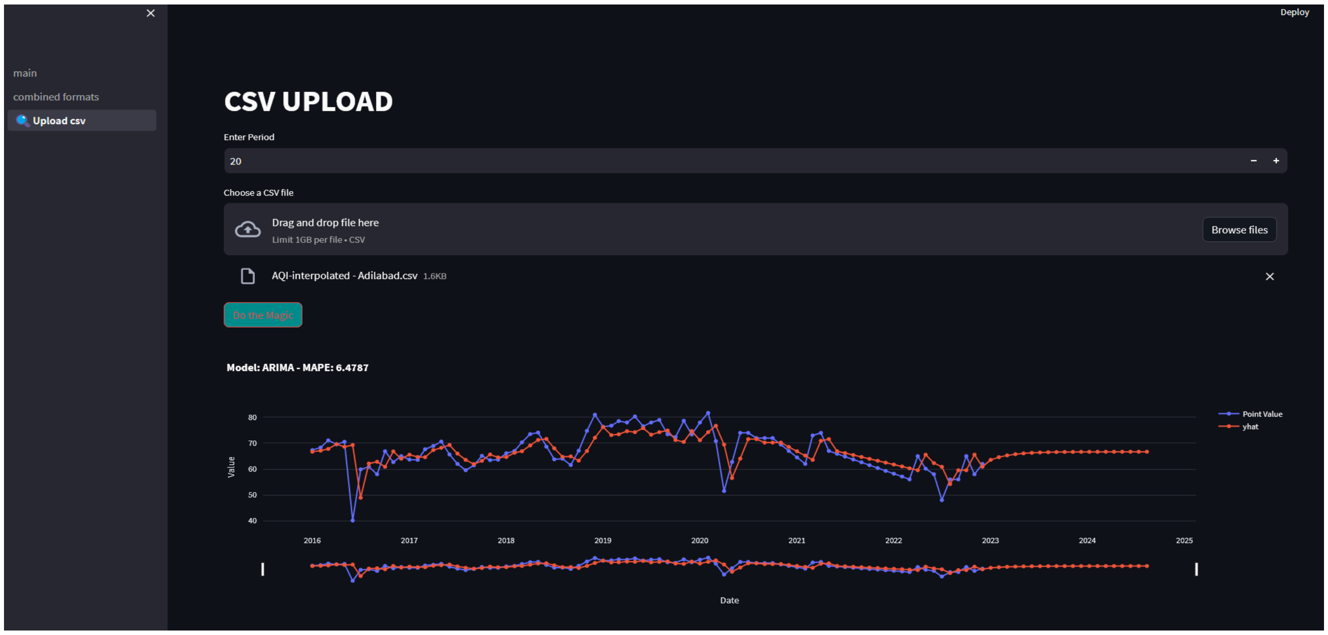
POST http://127.0.0.1:8105/predict_with_format?format=weekly&date_from=2021-08-01&date_to=2021-08-22&period=0

```
[
    {
        "point_timestamp": "2021-08-01T10:20:30",
        "point_value": 124.5
    },
    {
        "point_timestamp": "2021-08-08T10:20:30",
        "point_value": 194.73
    },
    {
        "point_timestamp": "2021-08-15T10:20:30",
        "point_value": 173.4
    },
    {
        "point_timestamp": "2021-08-22T10:20:30",
```

Status: 200 OK Time: 1012 ms Size: 483 B

```
{
    "model": "XGBoost",
    "mape": 0.0539,
    "result": [
        {
            "point_timestamp": "2021-08-01T10:20:30",
            "point_value": 124.5,
            "yhat": 124.6013
        },
        {
            "point_timestamp": "2021-09-01T10:20:30",
            "point_value": 194.73,
            "yhat": 194.6275
        },
        {
            "point_timestamp": "2021-10-01T10:20:30",
            "point_value": 173.4,
            "yhat": 173.3998
        },
        {
            "point_timestamp": "2021-11-01T10:20:30",
            "point_value": 124.5,
            "yhat": 124.6013
```

Fig : Accessing the FAST API via POSTMAN



CSV UPLOAD

Enter Period

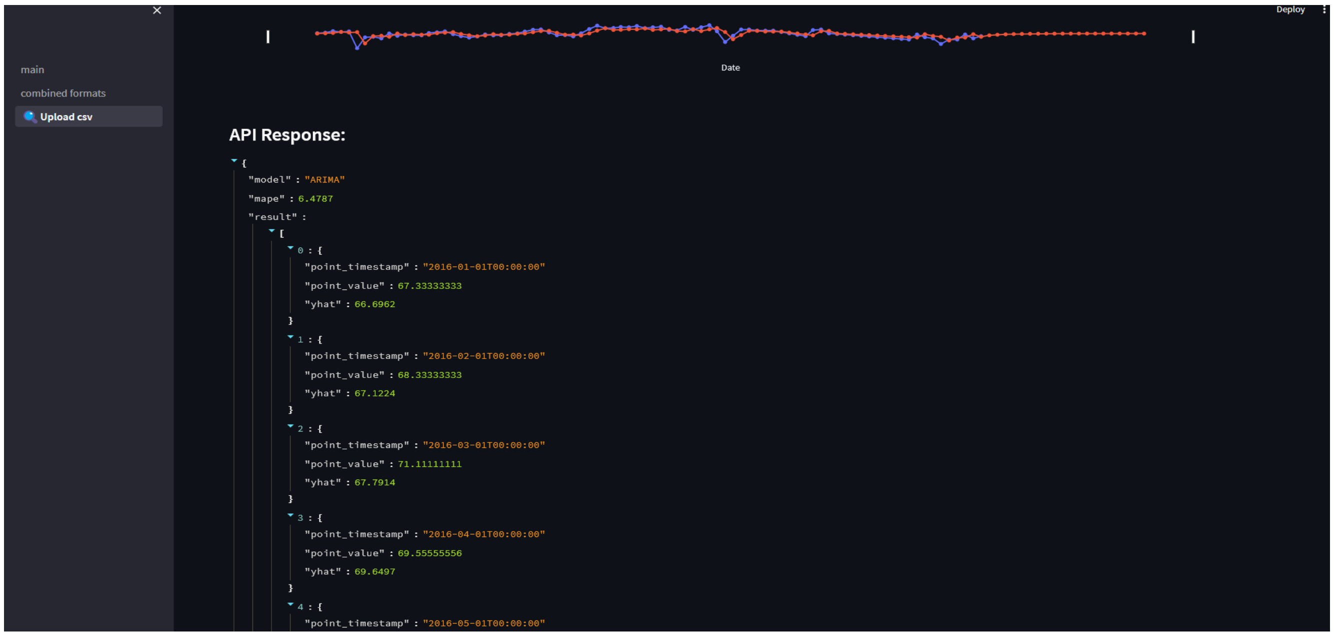
20

Choose a CSV file

Drag and drop file here

Limit 1GB per file • CSV

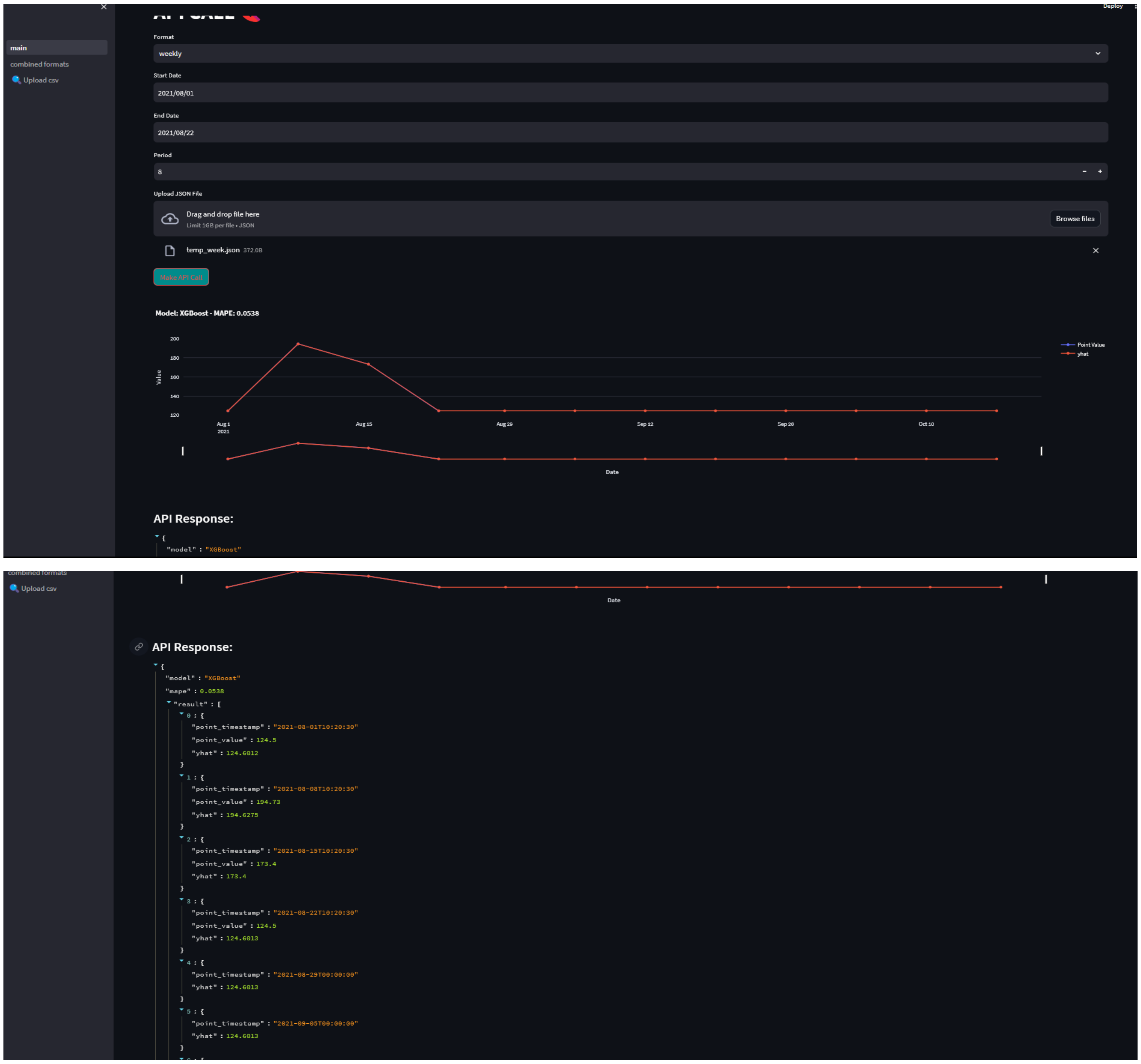
Model: ARIMA - MAPE: 6.4787



API Response:

```
{
  "model" : "ARIMA"
  "mape" : 6.4787
  "result" :
    [
      0 : {
        "point_timestamp" : "2016-01-01T00:00:00"
        "point_value" : 67.33333333
        "yhat" : 66.6962
      }
      1 : {
        "point_timestamp" : "2016-02-01T00:00:00"
        "point_value" : 68.33333333
        "yhat" : 67.1224
      }
      2 : {
        "point_timestamp" : "2016-03-01T00:00:00"
        "point_value" : 71.11111111
        "yhat" : 67.7914
      }
      3 : {
        "point_timestamp" : "2016-04-01T00:00:00"
        "point_value" : 69.55555556
        "yhat" : 69.6497
      }
      4 : {
        "point_timestamp" : "2016-05-01T00:00:00"
```

Fig : Streamlit CSV upload feature



Format
weekly

Start Date
2021/08/01

End Date
2021/08/22

Period
8

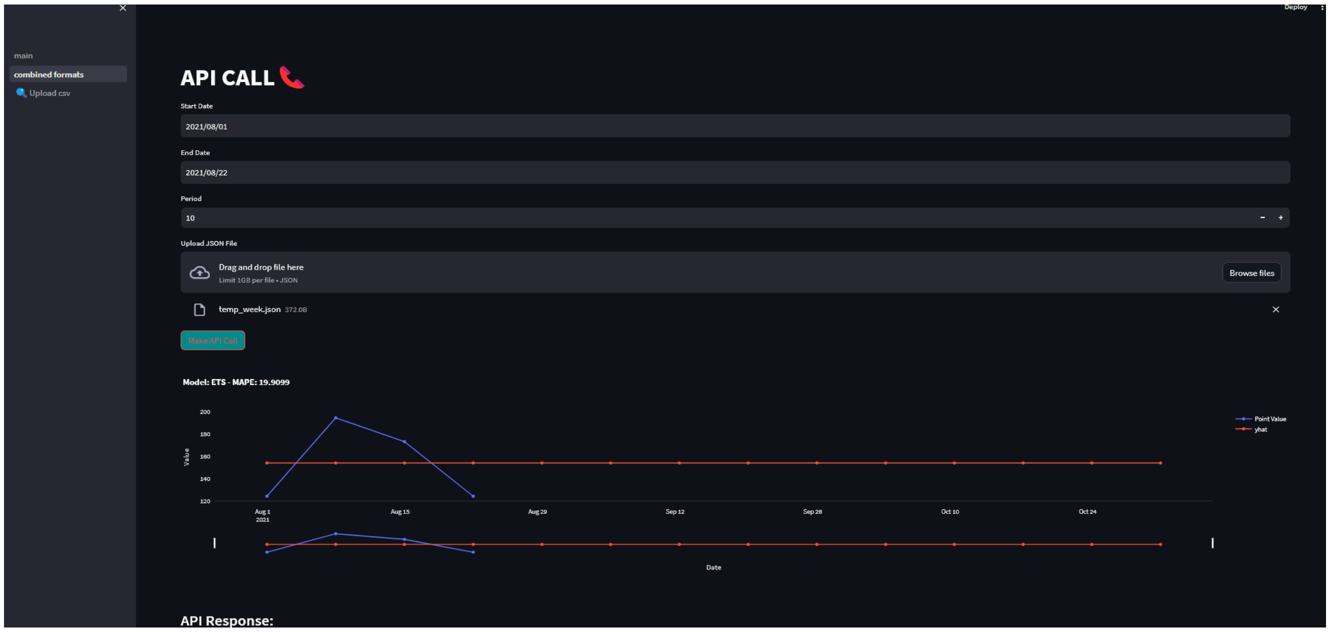
Upload JSON File
temp_week.json 372.0B

Model: XGBoost - MAPE: 0.0538

API Response:

```
{
  "model" : "XGBoost"
  "mape" : 0.0538
  "result" : [
    0 : {
      "point_timestamp" : "2021-08-01T10:20:30"
      "point_value" : 124.5
      "yhat" : 124.6012
    }
    1 : {
      "point_timestamp" : "2021-08-08T10:20:30"
      "point_value" : 194.73
      "yhat" : 194.6275
    }
    2 : {
      "point_timestamp" : "2021-08-15T10:20:30"
      "point_value" : 173.4
      "yhat" : 173.4
    }
    3 : {
      "point_timestamp" : "2021-08-22T10:20:30"
      "point_value" : 124.5
      "yhat" : 124.6013
    }
    4 : {
      "point_timestamp" : "2021-08-29T00:00:00"
      "yhat" : 124.6013
    }
    5 : {
      "point_timestamp" : "2021-09-05T00:00:00"
      "yhat" : 124.6013
    }
```

Fig : Streamlit Weekly prediction with 4 models classifier models



API CALL

Start Date
2021/08/01

End Date
2021/08/22

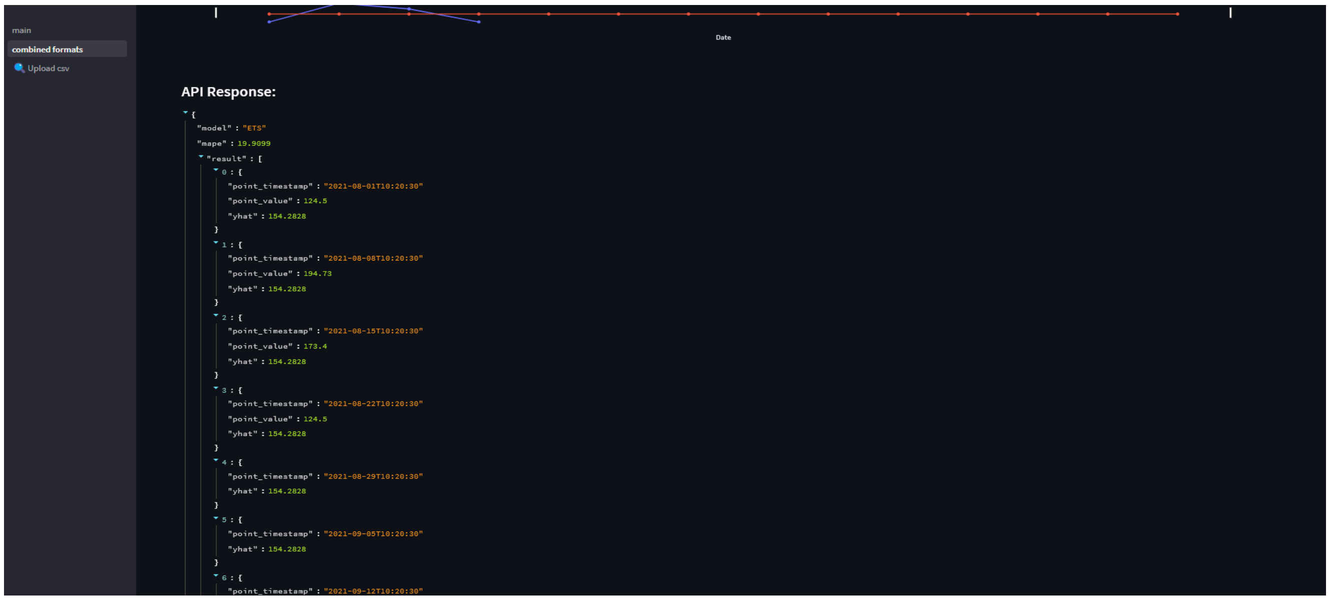
Period
10

Upload JSON File
Drag and drop file here
Limit 1GB per file • JSON

temp_week.json 372.0B

Model: ETS - MAPE: 19.9099

API Response:



API Response:

```
{
  "model" : "ETS"
  "mape" : 19.9099
  "result" : [
    0 : {
      "point_timestamp" : "2021-08-01T10:20:30"
      "point_value" : 124.5
      "yhat" : 154.2828
    }
    1 : {
      "point_timestamp" : "2021-08-08T10:20:30"
      "point_value" : 194.73
      "yhat" : 154.2828
    }
    2 : {
      "point_timestamp" : "2021-08-15T10:20:30"
      "point_value" : 173.4
      "yhat" : 154.2828
    }
    3 : {
      "point_timestamp" : "2021-08-22T10:20:30"
      "point_value" : 124.5
      "yhat" : 154.2828
    }
    4 : {
      "point_timestamp" : "2021-08-29T10:20:30"
      "yhat" : 154.2828
    }
    5 : {
      "point_timestamp" : "2021-09-05T10:20:30"
      "yhat" : 154.2828
    }
    6 : {
      "point_timestamp" : "2021-09-12T10:20:30"
```

Fig : Streamlit Weekly prediction with 1 models classifier model