DATAGENIE HACKATHON 2024 - MY APPROACH

Checkpoint 1: Generate the data, preprocess the data, feature extraction and train a model to predict the forecastability score

The dataset I used for this hack is the sample time series data provided.

Outline of my approach:

The dataset consists of daily, weekly, hourly and monthly format data. I decided to work on each format of data separately and not together as a whole. The reason I choose this approach is due to the following reasons:

- Each of the dataset: hourly, daily, weekly and monthly were handled separately as in real world scenarios, they exhibit different patterns across different time formats. Hence it only made sense to train the model separately for each dataset.
- Each type of dataset had its own trend and seasonality. So focusing on them separately also improved my feature engineering process and allowed me to have a brief analysis of each format.

Assumptions of my approach:

The below listed are a few assumptions that I made with respect to this problem statement :

- In the frontend part, I have made an assumption that the user is trying to predict only for a particular format of data (given in dataset). The user will not be able to predict for a format like minutely or yearly kind of data.
- I assumed 1.5 deviations , typically it implies using 1.5 times the standard deviation as threshold in anomaly detection, capturing data points that deviate significantly from the mean and encompassing approximately 95%% of the data if normally distributed. The reason I

chose this threshold is because it is more lenient than using 3 deviations and may identify only the most extreme anomalies.

Forecastibility score:

Before calculating the scores, preprocessing steps like duplicate removal and interpolating nan values were performed.

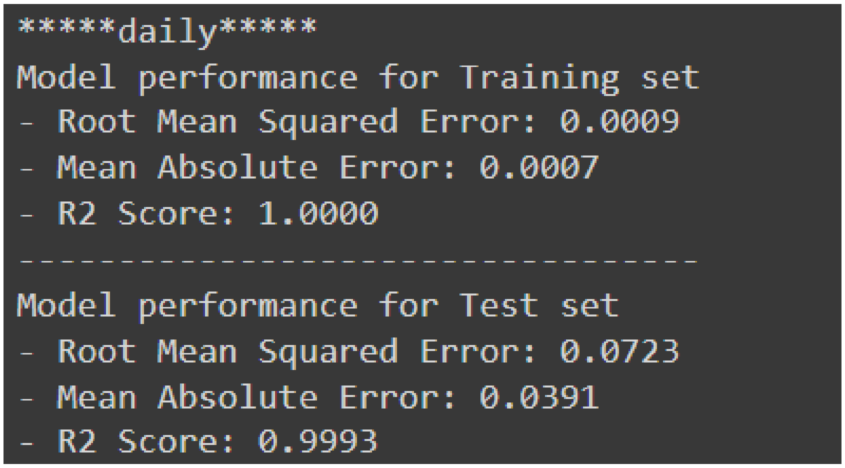
Features were extracted from the dataset like mean, variance, skewness, kurtosis, trend mean, seasonal mean and residual mean.

The forecastability score must range between 0 to 10, 10 being highly forecastable and 0 indicating white noise. I used the concept of **Approximate entropy (Apen)** as a relative measure to determine forecastability of a time series.

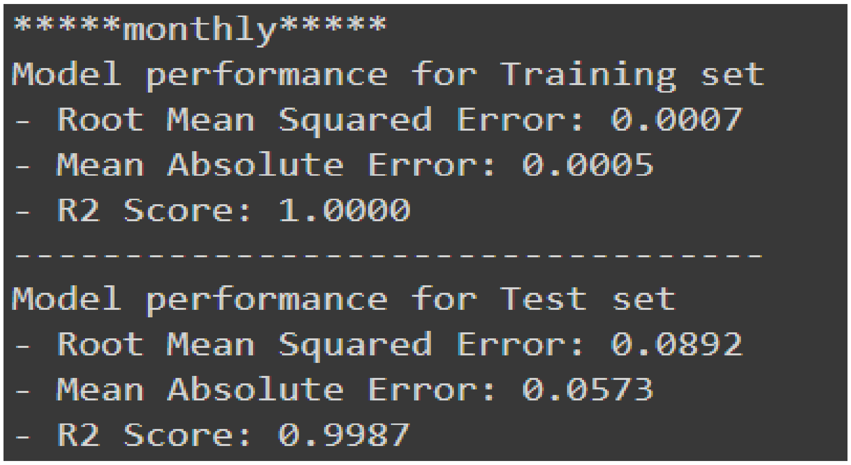
The reason is chose this approach is because, every time series is made up of 3 components: Trent, Seasonality and randomness. If the data exhibits a strong trend and or is highly seasonal, the prediction will be easy. If the data is mostly random, then by definition you can' predict anything. This approach was inspired from an article titled "Entropy as an A Priori indicator of forecastibility"

I used a technique called TimeSeriesSplit, which is somewhat like a rolling window approach, but the reason i chose this one is because it cumulatively includes the past data points also. Hence this approach was applied to all the 4 formats. The optimal threshold for the time series data was calculated based on mean plus 1.96 number of standard deviations which will give us 95% confidence and the dataset was filtered to include only those data points with a forecastability score greater than threshold.

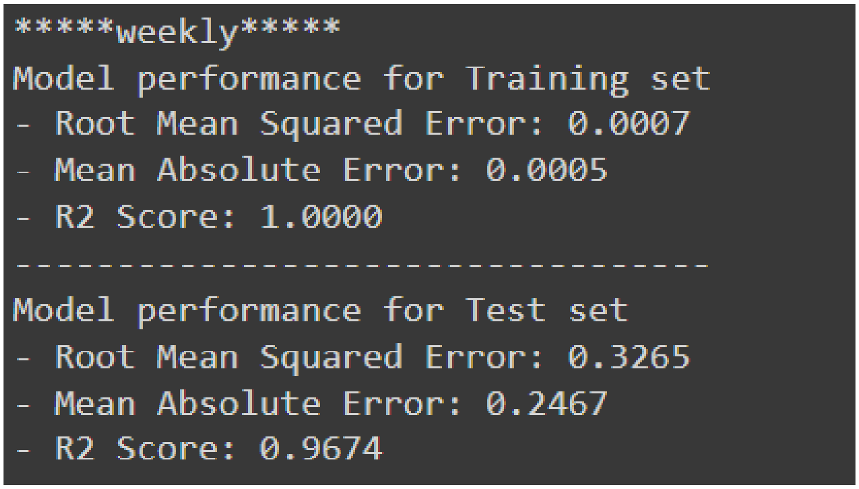
The dataset was then passed onto a XGBoost machine learning algorithm to predict the forecastability score and the results are shown below:



```
*****daily*****
Model performance for Training set
- Root Mean Squared Error: 0.0009
- Mean Absolute Error: 0.0007
- R2 Score: 1.0000
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 0.0723
- Mean Absolute Error: 0.0391
- R2 Score: 0.9993
```



```
*****monthly*****
Model performance for Training set
- Root Mean Squared Error: 0.0007
- Mean Absolute Error: 0.0005
- R2 Score: 1.0000
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 0.0892
- Mean Absolute Error: 0.0573
- R2 Score: 0.9987
```



```
*****weekly*****
Model performance for Training set
- Root Mean Squared Error: 0.0007
- Mean Absolute Error: 0.0005
- R2 Score: 1.0000
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 0.3265
- Mean Absolute Error: 0.2467
- R2 Score: 0.9674
```

The above results imply that all formats perform exceptionally well on training data achieving perfect R2 scores, indicating perfect fit.

On the testing data, the formats generally exhibited good performance with high R2 scores indicating accurate prediction. However there are slight variation in performance across different datasets, with the weekly dataset showing relatively lower performance compared to others.

Checkpoint 2: Batch anomaly detection using Prophet

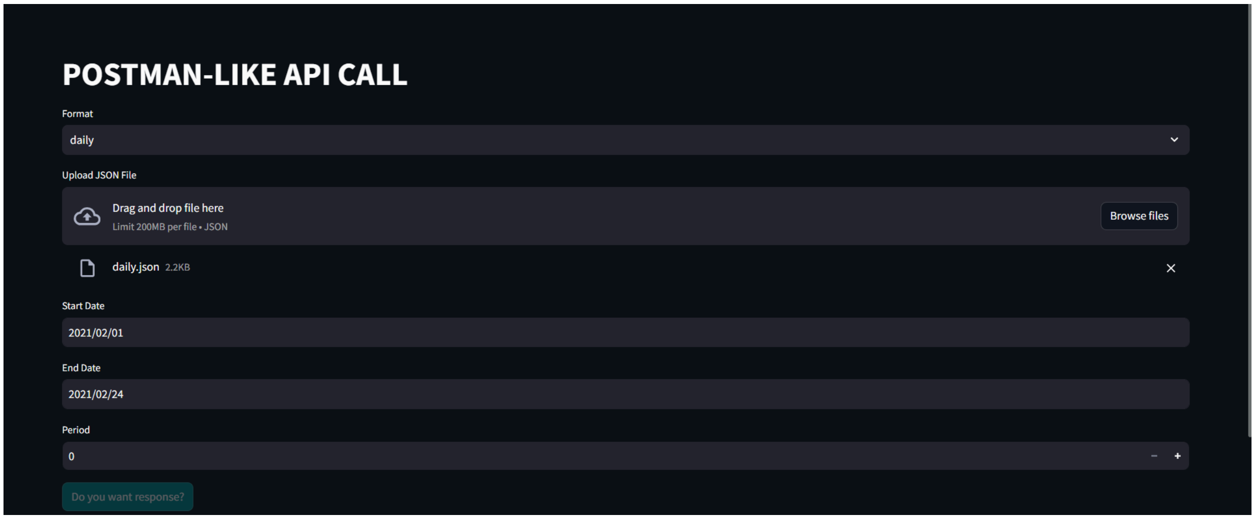
This algorithm, splits the input data as batches and applies the Prophet machine learning algorithm.
I assumed to take the number of batches in window for daily and hourly as 7 whereas for monthly and hourly to be as 4.
The reason for this choice is because a larger window size is suitable for higher granularity data(e.g., daily or hourly), as it allows the model to capture more complex patterns. On the other hand, for lower granularity data(e.g., weekly or monthly) a smaller window size is more appropriate, as the patterns in data are less complex and less prone to anomalies.
Also, each batch at a time only had 10% of the total datapoints.
This method uses a static window approach. It can be modified to as a rolling window or sliding window, but usually for time series data we prefer rolling window and that is what I also used here.

As the algorithm starts a timer also starts and the average time per fit is also calculated and returned along with the number of batch fits.

Checkpoint 3: Backend FASTAPI , Frontend - Streamlit

The input from and to date were created as two endpoints to access the API and perform anomaly detection.
For the frontend , I created a postman like API interface, where the user will upload a json file along with format and the desired outcomes like forecastability score, MAPE values, batch fits, average time per fit were displayed. Along with that the actual vs forecasted , trends and seasonality plots for the dataset were also plotted.

Frontend results:



POSTMAN-LIKE API CALL

Format: daily

Upload JSON File

Drag and drop file here
Limit 200MB per file • JSON

Browse files

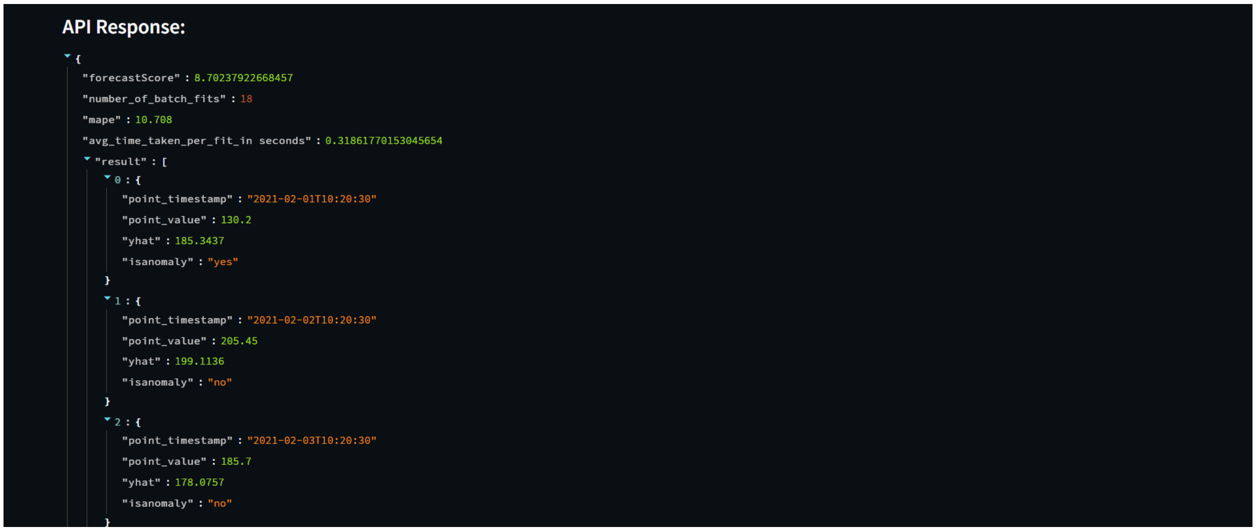
daily.json 2.2KB

Start Date: 2021/02/01

End Date: 2021/02/24

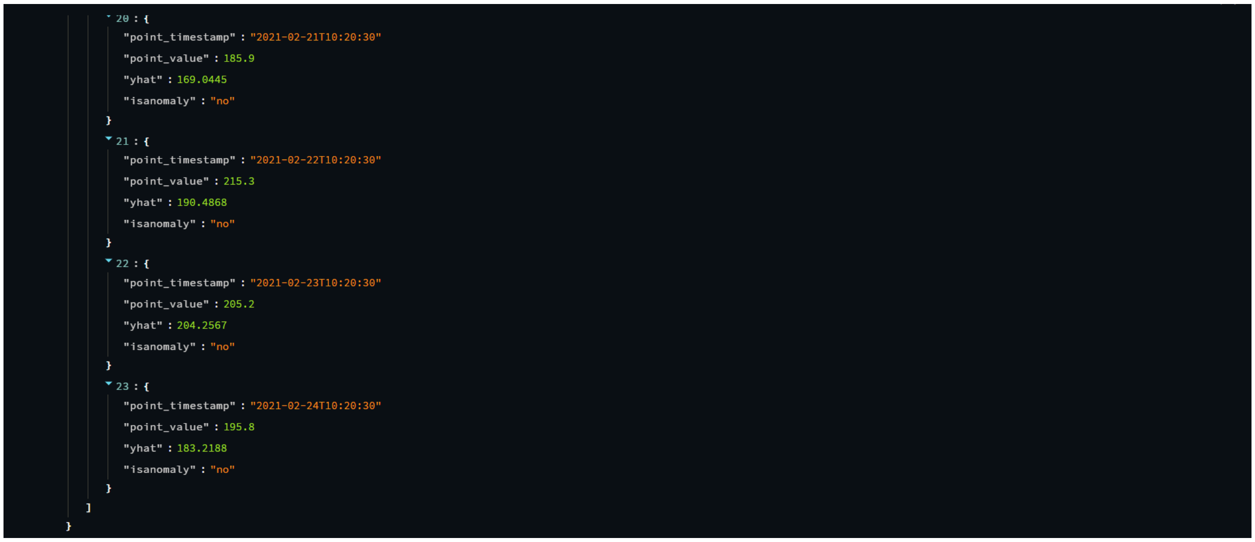
Period: 0

Do you want response?

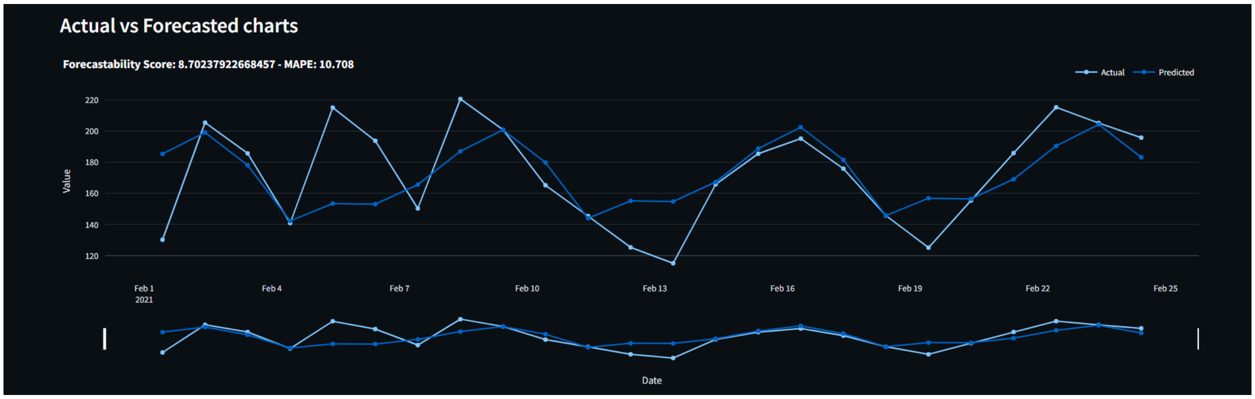


API Response:

```
{
  "forecastScore" : 8.70237922668457
  "number_of_batch_fits" : 18
  "mape" : 10.708
  "avg_time_taken_per_fit_in seconds" : 0.31861770153045654
  "result" : [
    0 : {
      "point_timestamp" : "2021-02-01T10:20:30"
      "point_value" : 130.2
      "yhat" : 185.3437
      "isanomaly" : "yes"
    }
    1 : {
      "point_timestamp" : "2021-02-02T10:20:30"
      "point_value" : 205.45
      "yhat" : 199.1136
      "isanomaly" : "no"
    }
    2 : {
      "point_timestamp" : "2021-02-03T10:20:30"
      "point_value" : 185.7
      "yhat" : 178.0757
      "isanomaly" : "no"
    }
```

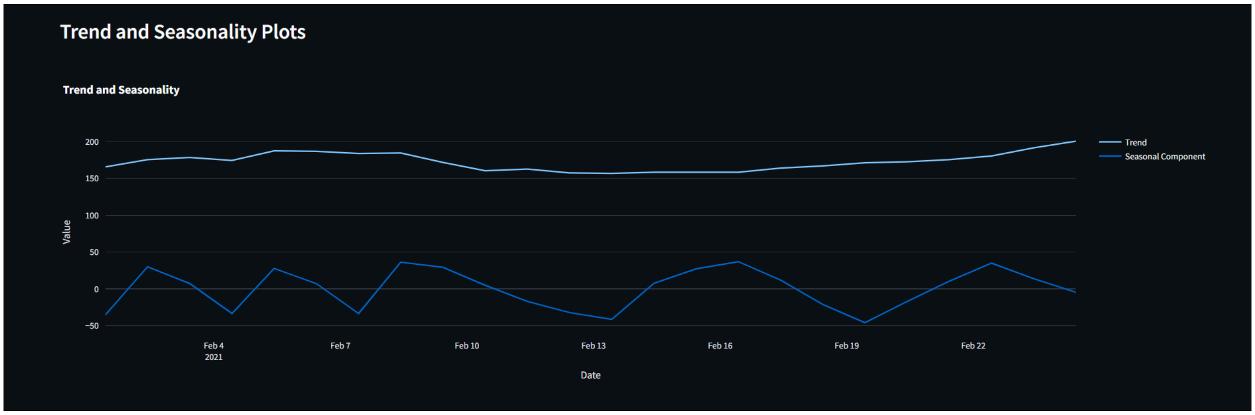


```
      20 : {
        "point_timestamp" : "2021-02-21T10:20:30"
        "point_value" : 185.9
        "yhat" : 169.0445
        "isanomaly" : "no"
      }
      21 : {
        "point_timestamp" : "2021-02-22T10:20:30"
        "point_value" : 215.3
        "yhat" : 190.4868
        "isanomaly" : "no"
      }
      22 : {
        "point_timestamp" : "2021-02-23T10:20:30"
        "point_value" : 205.2
        "yhat" : 204.2567
        "isanomaly" : "no"
      }
      23 : {
        "point_timestamp" : "2021-02-24T10:20:30"
        "point_value" : 195.8
        "yhat" : 183.2188
        "isanomaly" : "no"
      }
    ]
}
```



Actual vs Forecasted charts

Forecastability Score: 8.70237922668457 - MAPE: 10.708



Trend and Seasonality Plots

Trend and Seasonality

Checkpoint 4: Optimize the model:

The main objective here is to prejudice the model’s per fit training time and reduce the overall execution time.
My approach here was to reduce the average batch fit time by parallelizing the model fitting process, which can be useful when dealing with large datasets.
This approach also uses a rolling window, the size of each batch is 10% of whole dataset and based on the format, the window will contain various number of batches. The multiprocessing library in python creates fixed number of processes each of which fits a time series model to a single batch of data.
However the limitation of this approach is that, the code requires multiple CPU cores to work efficiently and hence I wasn’t able to run this code in my system.

Checkpoint 5: Batch reduction algorithm: