Team ID	Team-592401
Project Name	Time Series Analysis For Bitcoin Price Prediction Using Prophet

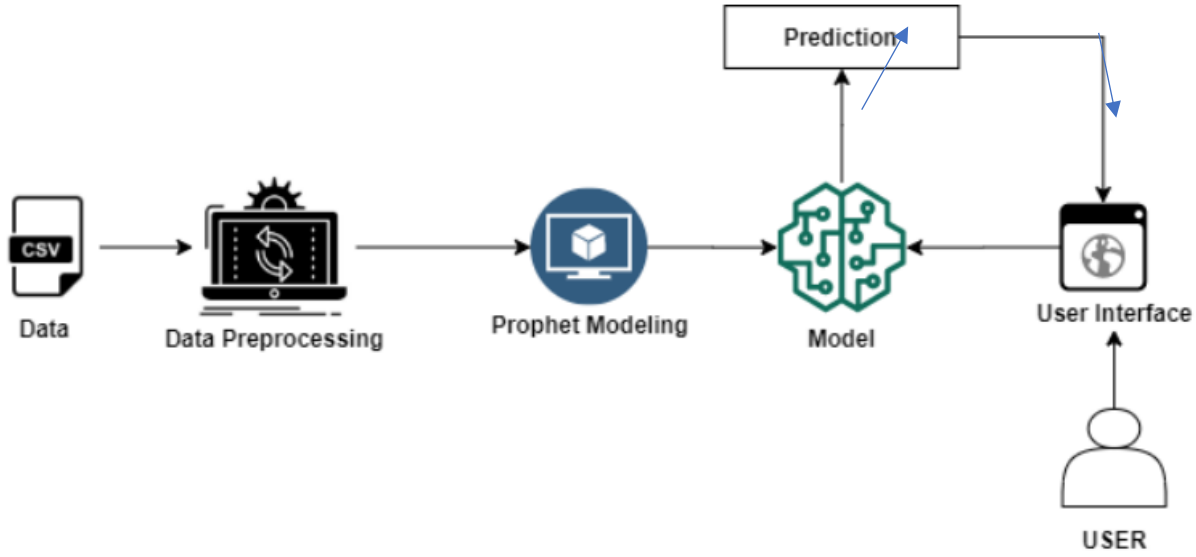
Crypto Price Prediction using FbProphet

Project Idea:

Bitcoin, the pioneering cryptocurrency created in January 2009, has emerged as the most valuable and widely traded digital asset, with a presence on over 40 cryptocurrency exchanges worldwide. It stands out for its acceptance of more than 30 different fiat currencies, making it a global financial instrument. Bitcoin's unique characteristics, notably its high volatility, set it apart from traditional currencies, making it an intriguing subject for price forecasting. Notably, in January 2017, the price of Bitcoin was $1,000 USD, and it reached a remarkable peak of $16,000 USD by the end of December 2017. As of June 2022, its value stands at $25,711 USD. The extreme volatility within the cryptocurrency market, coupled with Bitcoin's role as a flagship digital currency, renders it a subject of interest for investors, with its allure stemming from both anonymity and transparency within its blockchain system.

This project aims to develop a predictive system for Bitcoin price utilizing Facebook Prophet, a versatile time series forecasting tool. Given Bitcoin's susceptibility to multiple factors, including market sentiment, technological advancements, regulatory changes, and macroeconomic events.

Architecture:



Learning Outcomes:

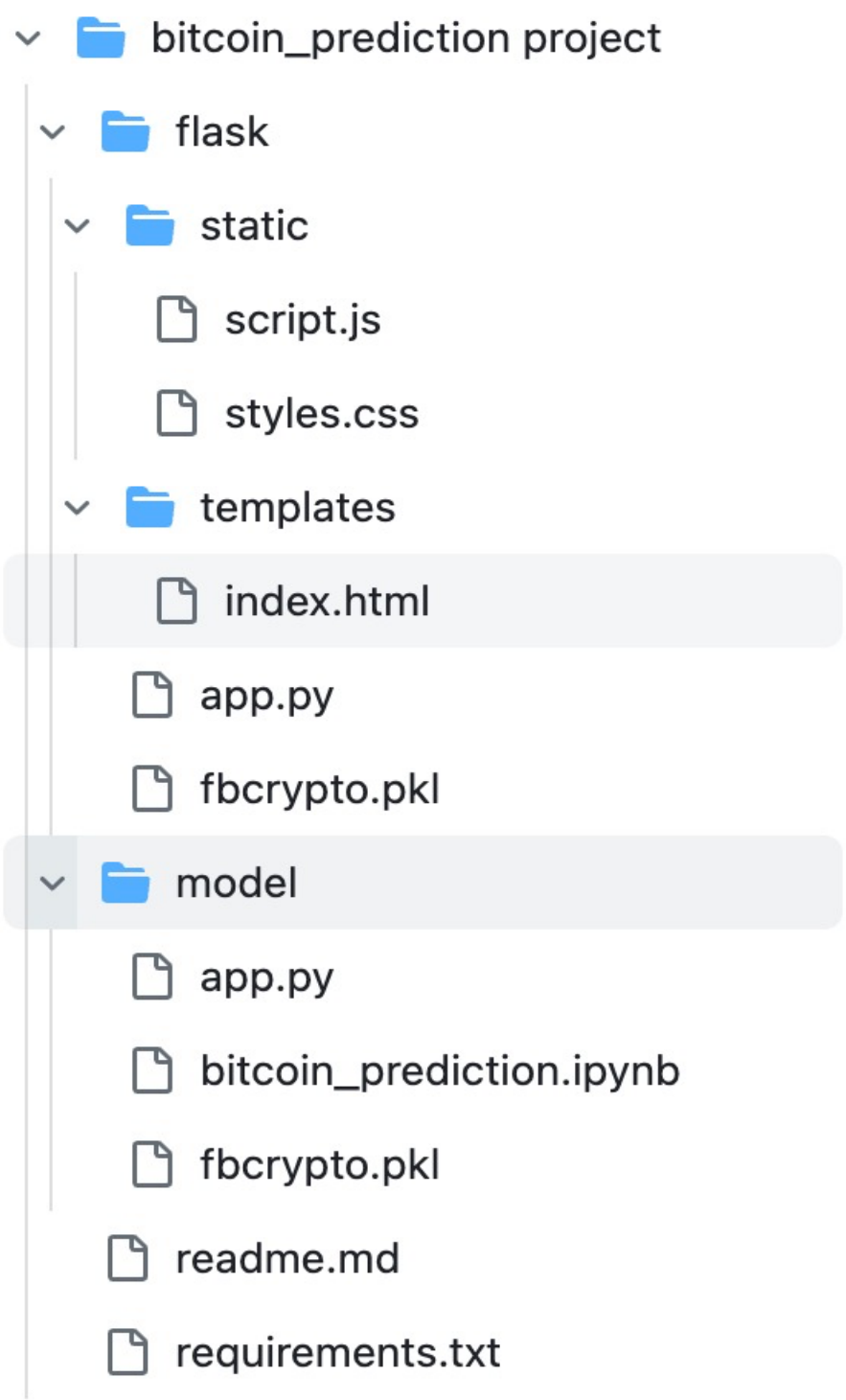
● You'll gain a solid understanding of the essential principles behind time series forecasting.
● You'll develop the skills to analyze data effectively and extract valuable insights using visualization techniques.
● You'll acquire the knowledge and proficiency to construct a web application using the Flask framework.

Project Flow:

● The user engages with the user interface (UI) to choose a date as their input.
● The model, seamlessly integrated, processes the selected date input values.
● After the model completes its analysis, the predicted output is displayed on the user interface.
To accomplish this, complete all the milestones & activities listed below.
*Install and Import Libraries
*Get Bitcoin Price Data
*Train Test Split
* Train Time Series Model using Prophet
*Use Prophet Model To make Prediction

* Time Series Decomposition
* Identify Change Points
* Cross Validation
* Prophet Model performance Evaluation

Project Structure:
Create a Project folder that contains files as shown below



All the mentioned files will be utilized in the creation of a Flask application.

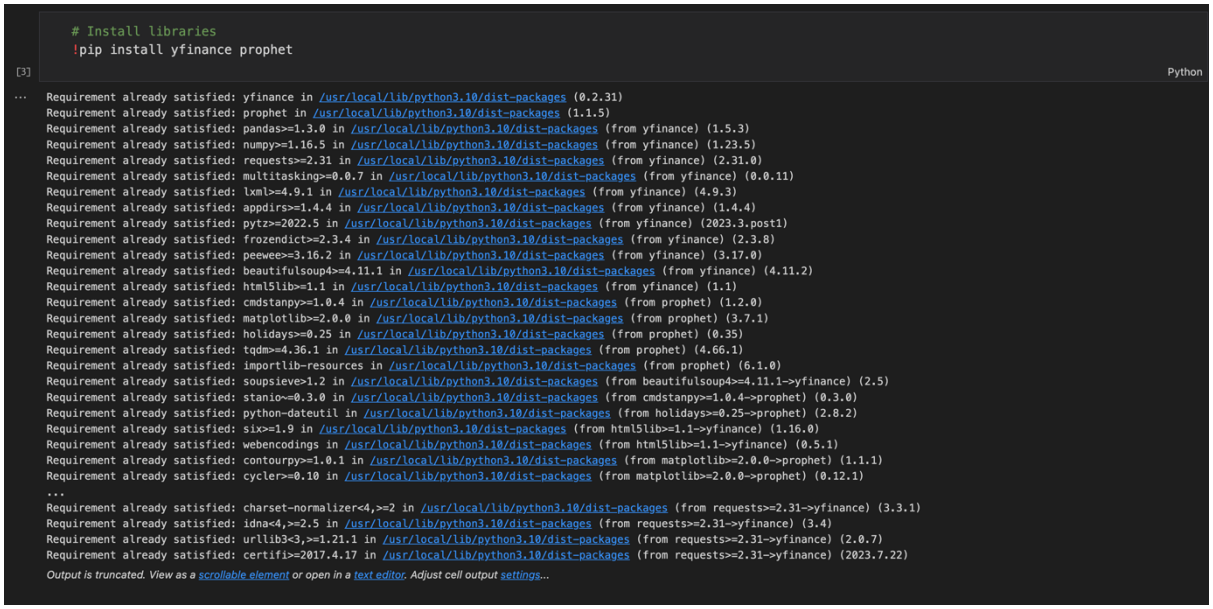
- Within the static folder, there is a CSS file.
- We are in the process of constructing a Flask Application that requires HTML pages, which are stored in the templates folder, along with a Python script named app.py for server-side scripting.

● Inside the Model training folder, you'll find the training file named FB_prophet_Bitcoin_forecasting.ipynb.
● The file fbcrypto.pkl is a saved model file.

- Prophet is a Python time series forecast library developed by Facebook.
- Prophet automatically detects yearly, weekly, and daily seasonality.
- It can quickly decompose the trend and seasonality effects.

Step 1: Install And Import Libraries

- In the first step, we will install and import libraries.
- Two Python packages need to be installed, `yfinance` and `prophet`.



```
# Install libraries
!pip install yfinance prophet
```

[3] Python

```
Requirement already satisfied: yfinance in /usr/local/lib/python3.10/dist-packages (0.2.31)
Requirement already satisfied: prophet in /usr/local/lib/python3.10/dist-packages (1.1.5)
Requirement already satisfied: pandas>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from yfinance) (1.5.3)
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.10/dist-packages (from yfinance) (1.23.5)
Requirement already satisfied: requests>=2.31 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.31.0)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.10/dist-packages (from yfinance) (0.0.11)
Requirement already satisfied: lxml>=4.9.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (4.9.3)
Requirement already satisfied: appdirs>=1.4.4 in /usr/local/lib/python3.10/dist-packages (from yfinance) (1.4.4)
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2023.3.post1)
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.3.8)
Requirement already satisfied: peewee>=3.16.2 in /usr/local/lib/python3.10/dist-packages (from yfinance) (3.17.0)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (4.11.2)
Requirement already satisfied: html5lib>=1.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (1.1)
Requirement already satisfied: cmdstanpy>=1.0.4 in /usr/local/lib/python3.10/dist-packages (from prophet) (1.2.0)
Requirement already satisfied: matplotlib>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from prophet) (3.7.1)
Requirement already satisfied: holidays>=0.25 in /usr/local/lib/python3.10/dist-packages (from prophet) (0.35)
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.10/dist-packages (from prophet) (4.66.1)
Requirement already satisfied: importlib-resources in /usr/local/lib/python3.10/dist-packages (from prophet) (6.1.0)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4>=4.11.1->yfinance) (2.5)
Requirement already satisfied: stanio~=0.3.0 in /usr/local/lib/python3.10/dist-packages (from cmdstanpy>=1.0.4->prophet) (0.3.0)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from holidays>=0.25->prophet) (2.8.2)
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.10/dist-packages (from html5lib>=1.1->yfinance) (1.16.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from html5lib>=1.1->yfinance) (0.5.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (1.1.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (0.12.1)
...
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (3.3.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (2023.7.22)
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

- After the package installation, we need to import libraries.
- `numpy` and `pandas` are for data processing. `yfinance` is for pulling the data.
- Prophet is for building the time series forecast. `.plot` is for model output visualization, and `diagnostics` is for model performance evaluation.
- `plotly` is imported to visualize the Bitcoint price trend.

```python
# Data processing
import numpy as np
import pandas as pd

# Get time series data
import yfinance as yf

# Prophet model for time series forecast
from prophet import Prophet
from prophet.plot import add_changepoints_to_plot, plot_cross_validation_metric
from prophet.diagnostics import cross_validation, performance_metrics

# Visualization
import plotly.graph_objs as go
```

[4] Python

Step 2: Get Bitcoin Price Data

- In the 2nd step, the Bitcoin price data is downloaded from the Yahoo finance API.

- Yahoo finance downloads data with the date as an index. Using `reset_index`, we created a new index and used the date as a column. This is because Prophet requires the date-time variable to be a column for the model input.

- By default, the date is string type, `pd.to_datetime` changes it to a DateTime format.

```python
# Download Bitcoin data
data = yf.download(tickers='BTC-USD', start='2020-01-01', end='2023-11-06', interval = '1d')

# Reset index and have date as a column
data.reset_index(inplace=True)

# Change date to datetime format
data['Date'] = pd.to_datetime(data['Date'])

# Take a look at the data
data.head()
```

[5] Python

```
[**********100%%**********]  1 of 1 completed
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2020-01-01	7194.892090	7254.330566	7174.944336	7200.174316	7200.174316	18565664997
1	2020-01-02	7202.551270	7212.155273	6935.270020	6985.470215	6985.470215	20802083465
2	2020-01-03	6984.428711	7413.715332	6914.996094	7344.884277	7344.884277	28111481032
3	2020-01-04	7345.375488	7427.385742	7309.514160	7410.656738	7410.656738	18444271275
4	2020-01-05	7410.451660	7544.497070	7400.535645	7411.317383	7411.317383	19725074095

- Get trend chart.

```python
# Declare a figure
fig = go.Figure()

# Candlestick chart
fig.add_trace(go.Candlestick(x=data.Date,
                open=data['Open'],
                high=data['High'],
                low=data['Low'],
                close=data['Close'],
                name = 'Bitcoin Data'))
```

- In this model, we will forecast the Bitcoin close price. Prophet takes two columns as inputs, a datetime column called ds and a value column called y. Therefore, we need to drop all the other columns, rename Date to ds and Close to y.

```python
# Keep only date and close price
df = data.drop(['Open', 'High', 'Low', 'Adj Close', 'Volume'], axis=1)

# Rename date to ds and close price to y
df.rename (columns={'Date': 'ds', 'Close': 'y'}, inplace=True)

# Take a look at the data
df.head ()
```

	ds	y
0	2020-01-01	7200.174316
1	2020-01-02	6985.470215
2	2020-01-03	7344.884277
3	2020-01-04	7410.656738
4	2020-01-05	7411.317383

```python
# Data information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1404 entries, 0 to 1403
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   ds      1404 non-null   datetime64[ns]
 1   y       1404 non-null   float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 22.1 KB
```

Step 3: Train Test Split

- In step 3, a training and a testing dataset are created. We cannot use random split for time series data because it causes data leakage from the future dates to the past

dates. Usually, a cutoff date is selected. The data before the cutoff date is the training dataset, and the data after the cutoff date is used as the testing dataset.

```
# Train test split
df_train = df[df['ds']<='2023-09-30']
df_test = df[df['ds']>'2023-09-30']

# Print the number of records and date range for training and testing dataset.
print('The training dataset has', len(df_train), 'records, ranging from', df_train['ds'].min(), 'to', df_train['ds'].max())
print('The testing dataset has', len(df_test), 'records, ranging from', df_test['ds'].min(), 'to', df_test['ds'].max())
```

```
The training dataset has 1369 records, ranging from 2020-01-01 00:00:00 to 2023-09-30 00:00:00
The testing dataset has 35 records, ranging from 2023-10-01 00:00:00 to 2023-11-04 00:00:00
```

Step 4: Train Time Series Model Using Prophet

- In step 4, we will train the time series model using the training dataset.

```
# Create the prophet model with confidence internal of 95%
m = Prophet(interval_width=0.95, n_changepoints=7)

# Fit the model using the training dataset
m.fit(df_train)
```

```
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpiy56emqa/by06435u.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpiy56emqa/9hr6g2y9.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=52142', 'data', 'file=/tmp/tmpiy56emqa/by06435u.json', 'init=
12:51:30 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
12:51:30 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
<prophet.forecaster.Prophet at 0x7aae904f4d30>
```

- The yearly seasonality and daily seasonality are automatically disabled.
- This is because Prophet detects that the dataset we are using does not have full multiple years of data and does not have units smaller than a day.

Step 5: Use Prophet Model To Make Prediction

- Step 5 uses the trained Prophet model to make the prediction.
- This is the same as using the testing dataset we created above.
- The prediction output contains lots of information. We kept the predicted value `yhat` and its prediction interval upper and lower bound value.

```
# Create a future dataframe for prediction
future = m.make_future_dataframe(periods=31)

# Forecast the future dataframe values
forecast = m.predict(future)

# Check the forecasted values and upper/lower bound
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

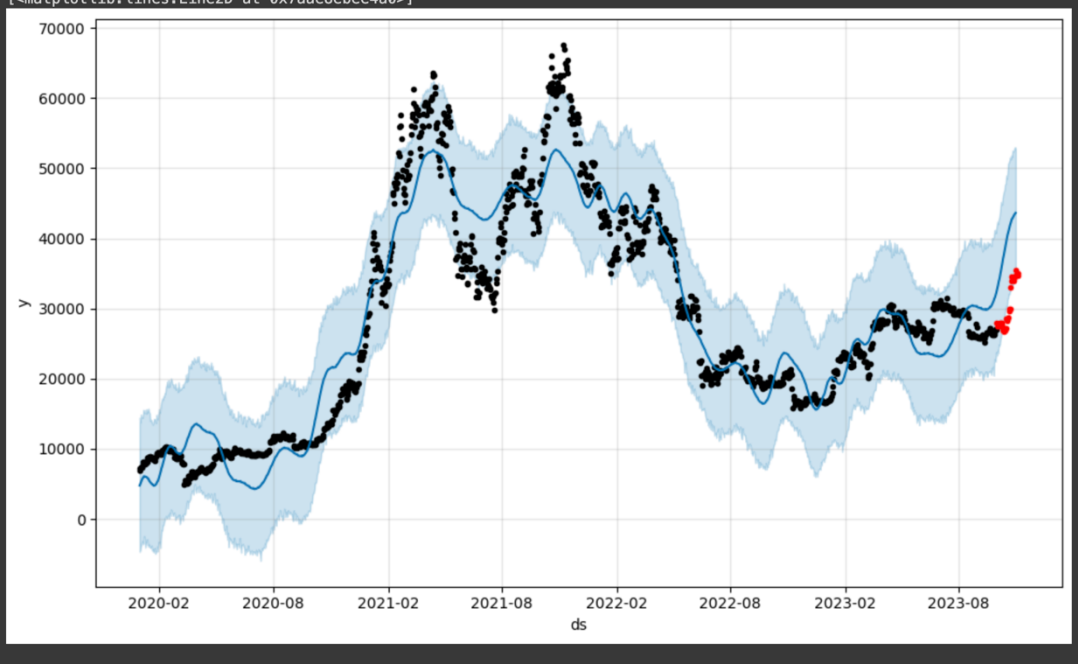
	ds	yhat	yhat_lower	yhat_upper
1395	2023-10-27	43049.914463	33823.316006	52339.159332
1396	2023-10-28	43221.581413	34111.610597	52488.602259
1397	2023-10-29	43389.428472	34018.211349	52850.472656
1398	2023-10-30	43502.986768	34797.681066	52998.656552
1399	2023-10-31	43620.977000	34479.518683	52769.251016

- The x-axis is the date in the forecast visualization, and the y axis is the Bitcoin close price. The black dots are the actual prices in the training dataset, and the red dots

are the actual forecast prices. The blue line is the time series model prediction. The shaded area is the 95% prediction interval.

```
# Visualize the forecast
fig = m.plot(forecast)
ax = fig.gca()
ax.plot( df_test["ds"], df_test["y"], 'r.')
```

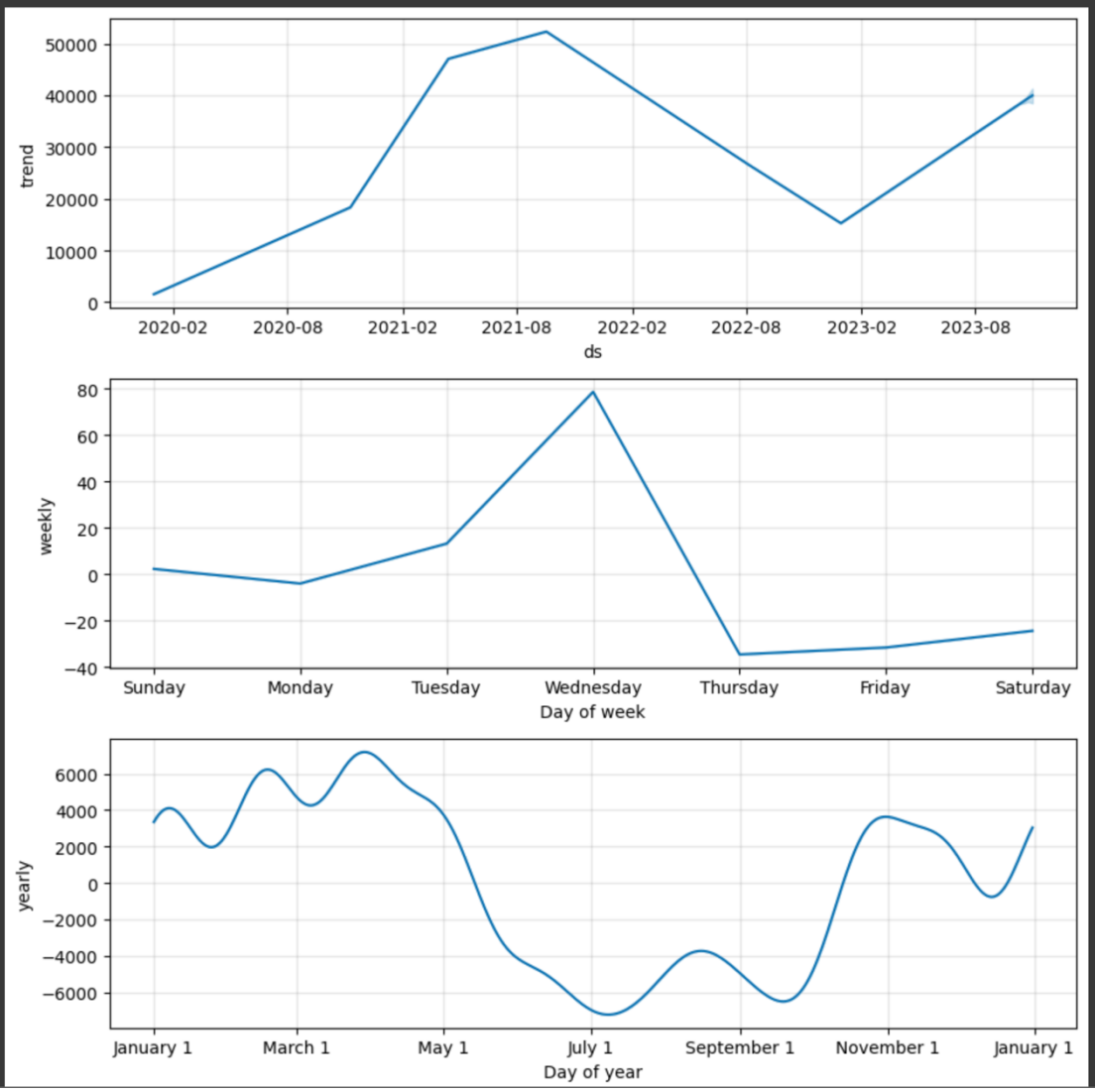
```
[<matplotlib.lines.Line2D at 0x7aae8ebee4a0>]
```



Step 6: Time Series Decomposition

- In step 6, we will decompose the time series forecast.

```
# Visualize the components
m.plot_components(forecast);
```



Step 7: Identify Change Points

Prophet automatically identifies the change points in time series data following the steps below:

1. Specify the percentage of data used for identifying change points. The default is 80%.
2. Identify a large number of uniformly distributed dates with possible trajectories change.
3. Apply a sparse prior on the magnitudes of the change rate, which is similar to L1 regularization.

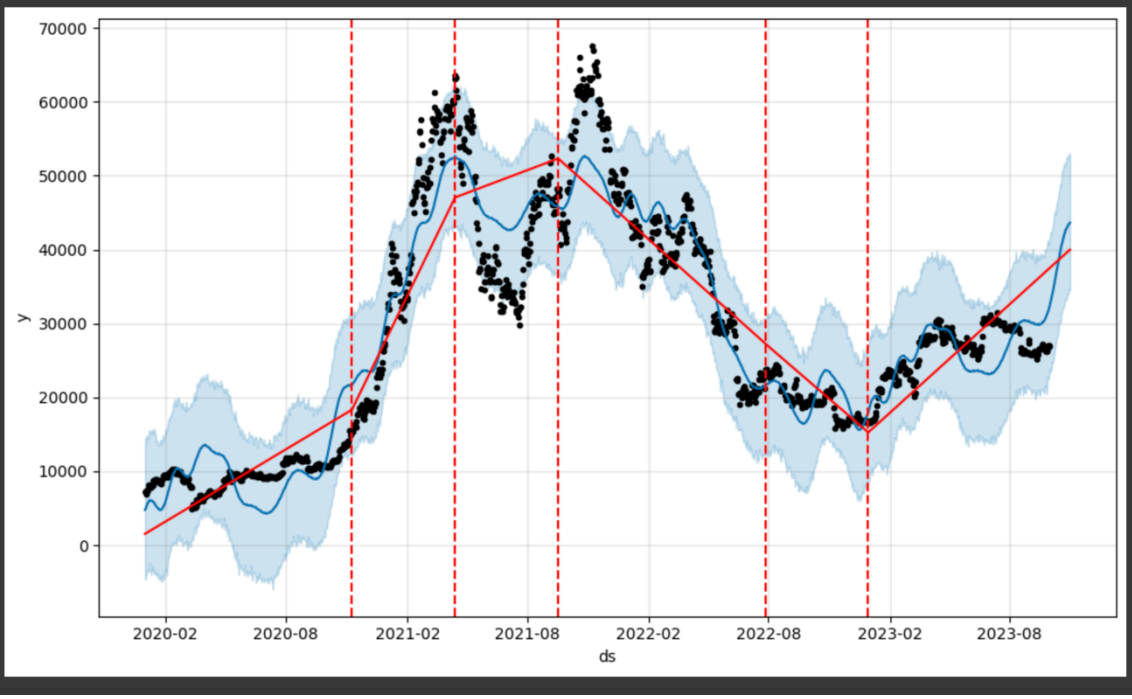
We can list the dates corresponding to the changepoints using .changepoints.

```
# Default change points
print(f'There are {len(m.changepoints)} change points. \nThe change points dates are \n{df.loc[df["ds"].isin(m.changepoints)]}')
```

```
There are 7 change points.
The change points dates are
           ds             y
156  2020-06-05   9665.533203
313  2020-11-09  15332.315430
469  2021-04-14  63109.695312
625  2021-09-17  47267.519531
781  2022-02-20  38431.378906
938  2022-07-27  22930.548828
1094 2022-12-30  16602.585938
```

- In the visualization, the red dotted lines represent the changepoints. It does not include all the seven changepoints in the chart. Only the ones with more changes are included.

```
# Change points to plot
fig = m.plot(forecast)
a = add_changepoints_to_plot(fig.gca(), m, forecast)
```



Step 8: Cross Validation

In step 8, we will do cross-validation for the time series model. Prophet has a cross_validation function to automate the comparison between the actual and the predicted values.

- `m' is the trained model.
- `initial='500 days" means the initial model will be trained on the first 500 days of data.
- `period='60 days" means 60 days will be added to the training dataset for each additional model.
- horizon = '30 days' means that the model forecasts the next 30 days. When only horizon is given, Prophet defaults initial to be triple the horizon, and period to be half of the horizon.

- parallel="processes" enables parallel processing for cross-validation. When the parallel cross-validation can be done on a single machine, "processes" provide the highest performance. For larger problems, "dask" can be used to do cross-validation on multiple machines.

```
# Cross validation
df_cv = cross_validation(m, initial='500 days', period='60 days', horizon = '30 days', parallel="processes")
df_cv.head()
```

	ds	yhat	yhat_lower	yhat_upper	y	cutoff
0	2021-07-13	28970.723732	24441.739139	33471.102451	32702.025391	2021-07-12
1	2021-07-14	28637.359656	24005.945805	32911.206374	32822.347656	2021-07-12
2	2021-07-15	28253.779820	23542.365585	32933.326477	31780.730469	2021-07-12
3	2021-07-16	27881.454292	23235.133778	32699.478480	31421.539062	2021-07-12
4	2021-07-17	27507.684015	22755.281498	32244.428113	31533.068359	2021-07-12

Step 9: Prophet Model Performance Evaluation

Step 9 evaluates the cross-validation model performance.

- MSE (Mean Squared Error) sums up the squared difference between actual and prediction and is divided by the number of predictions.
- RMSE (Root Mean Square Error) takes the square root of MSE.
- MAE (Mean Absolute Error) sums up the absolute difference between actual and prediction and is divided by the number of predictions.
- MAPE (Mean Absolute Percentage Error) sums up the absolute percentage difference between actual and prediction and is divided by the number of predictions. MAPE is independent of the magnitude of data, so it can be used to compare different forecasts. But it's undefined when the actual value is zero.
- MDAPE (Median Absolute Percentage Error) is similar to MAPE. The difference is that it calculates the median instead of taking the average of the absolute percentage difference.
- SMAPE (Symmetric Mean Absolute Percentage Error) is similar to MAPE. The difference is that when calculating absolute percentage error, the denominator is the actual value for MAPE and the average of the actual and predicted value for SMAPE.

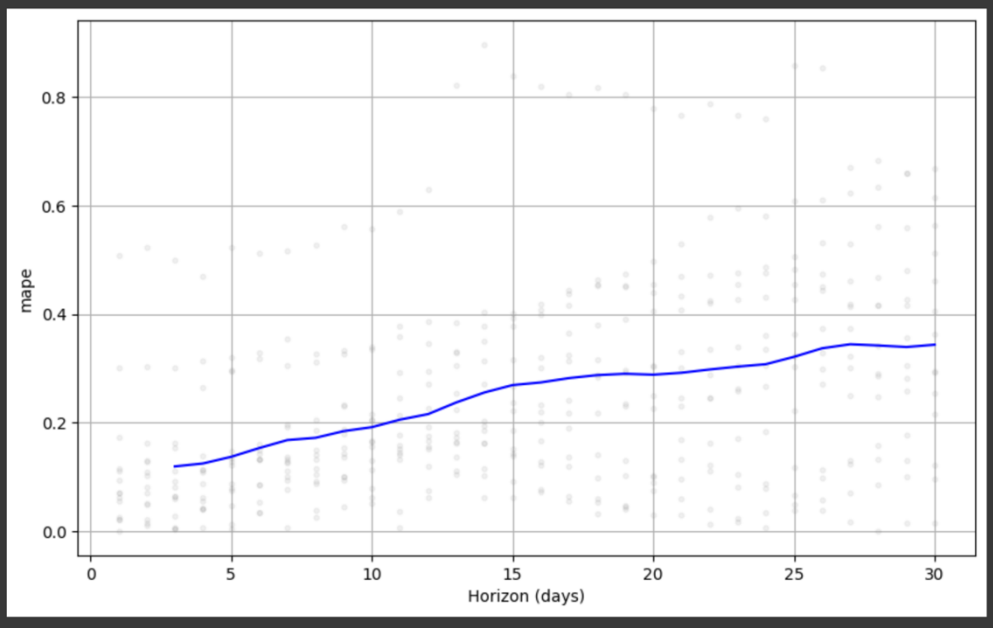
```
# Model performance metrics
df_p = performance_metrics(df_cv)
df_p.head()

```

	horizon	mse	rmse	mae	mape	mdape	smape	coverage
0	3 days	4.279444e+07	6541.746319	3854.192620	0.119112	0.075538	0.111936	0.928571
1	4 days	4.311961e+07	6566.552741	3977.553444	0.124537	0.084168	0.118147	0.928571
2	5 days	4.570097e+07	6760.249244	4247.710260	0.136807	0.089535	0.130239	0.904762
3	6 days	4.951701e+07	7036.832350	4721.718716	0.152993	0.116675	0.147258	0.880952
4	7 days	5.471568e+07	7397.004493	5232.298053	0.167712	0.132234	0.162915	0.809524

plot_cross_validation_metric method from Prophet helps us to plot the cross-validation performance results.

- The x-axis is the horizon. Because we set the horizon to be 30 days, the x-axis has a value up to 30.
- The y-axis is the metric we are interested in. We use mape as an example in this visualization.
- On each day, we can see three dots. This is because there are three models in the cross-validation, and each dot represents the MAPE from one model.
- The line is the aggregated performance across all the models. We can see that MAPE value increases with days, which is expected because time series tend to make better predictions for the near future than the far future.

```
# Visualize the performance metrics
fig = plot_cross_validation_metric(df_cv, metric='mape')
```

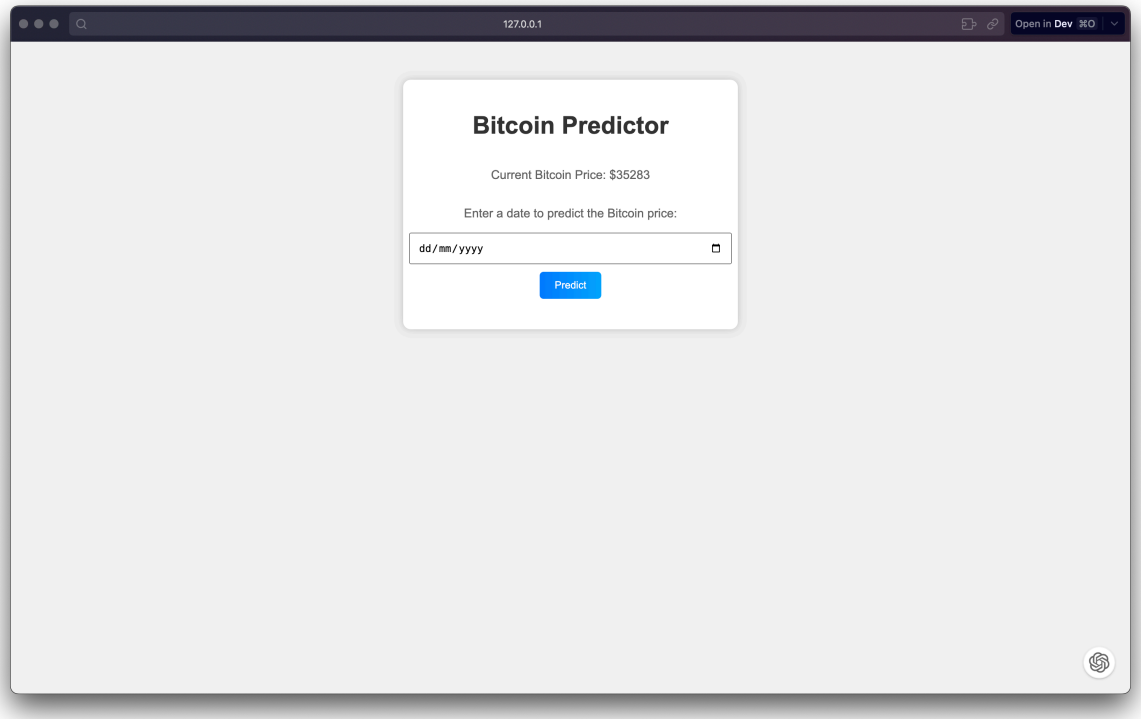


Step-10: Save the Model

```
import pickle
pickle.dump(m,open('fbcrypto.pkl','wb'))
```

Follow the commands to save your model.

Step-11: Create HTML pages



Step-12: Build python code

1) Import required libraries and load the model.
2) Importing the flask module into the project is mandatory. An object of the Flask class is our WSGI application. Flask constructor takes the name of the current module(__name__) as an argument Pickle library to load the model file.
3) The declared constructor is used to route to the HTML page created earlier.
4) In the below code snippet, the '/' URL is bound with the index.html. Hence, when the home page of a web server is opened in the browser, the HTML page (index.html)will be rendered.
5) This step is to prepare our model to make future predictions. This is achieved using the Prophet.make_future_dataframe method and passing the number of days we'd like to predict in the future and storing the data into a forecast variable.
6) Run the application on localhost.

```python
import numpy as np
import pandas as pd
from flask import Flask, request, jsonify, render_template
import pickle
import requests
from datetime import datetime

app = Flask(__name__)

m = pickle.load(open('fbcrypto.pkl','rb'))

def get_current_bitcoin_price():
    url = "https://api.coingecko.com/api/v3/simple/price"
    params = {
        "ids": "bitcoin",
        "vs_currencies": "usd",
    }

    try:
        response = requests.get(url, params=params)
        data = response.json()
        bitcoin_price = data["bitcoin"]["usd"]
        return bitcoin_price
    except Exception as e:
        print(f"Error fetching Bitcoin price: {e}")
        return None


@app.route('/', methods=['GET'])
def index():
    current_bitcoin_price = get_current_bitcoin_price()
    return render_template('index.html', current_bitcoin_price=current_bitcoin_price)
```

```
33
34
35    future = m.make_future_dataframe(periods = 10000)
36    forecast = m.predict(future)
37
38    @app.route('/predict', methods=['POST'])
39    def y_predict():
40        if request.method == "POST":
41            ds = request.form["Date"]
42            ds = str(ds)
43            next_day = datetime.strptime(ds, "%Y-%m-%d").date()
44            forecast_dates = forecast['ds'].dt.date
45            print(forecast['ds'].dt.date)
46
47            # Check if next_day exists in forecast_dates
48            if next_day in forecast_dates.values:
49                # Find the corresponding index in forecast_dates
50                index = forecast_dates[forecast_dates == next_day].index[0]
51                prediction = forecast.loc[index, 'yhat']
52                prediction = round(prediction, 2)
53            else:
54                # Provide a default value or error message if next_day is not found
55                prediction = "Date not found in forecast data."
56
57            return jsonify(prediction)
58
59
60    if __name__ =="__main__":
61        app.run(debug="True")
62
```

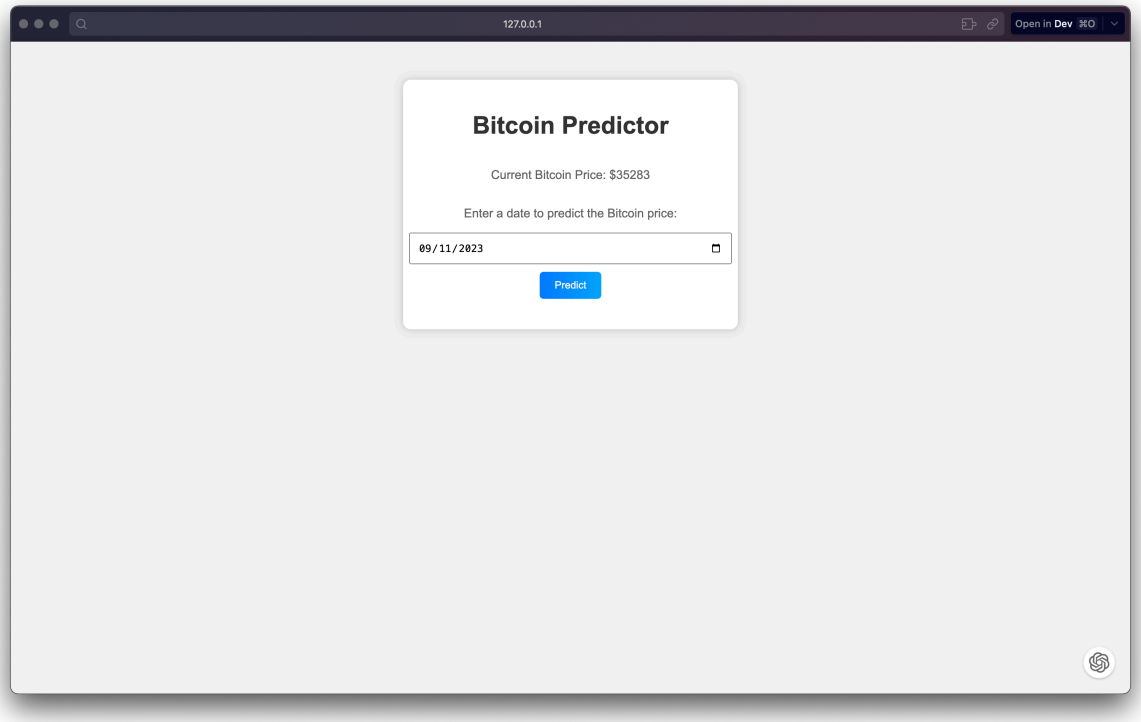
Step-13: Run the Application

- Open the anaconda prompt from the start menu.
- Navigate to the folder where your app.py resides.
- Now type the “python app.py” command.
- It will show the local host where your app is running on **http://127.0.0.1.5000/**
- Copy that local host URL and open that URL in the browser. It does navigate you to where you can view your web page.

```
(base) nivesh@Niveshs-MacBook-Pro flask % python -u "/Users/nivesh/Desktop/SI-GuidedProject-601585-1697638773/bitcoin_prediction project/flask/app.py"
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 850-220-493
127.0.0.1 - - [08/Nov/2023 09:56:36] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [08/Nov/2023 09:56:36] "GET /static/script.js HTTP/1.1" 200 -
127.0.0.1 - - [08/Nov/2023 09:56:36] "GET /static/styles.css HTTP/1.1" 200 -
127.0.0.1 - - [08/Nov/2023 09:56:36] "GET /favicon.ico HTTP/1.1" 404 -
```

Step-14:Predicting the results

To predict the result we need to enter the date on which we want to predict the price.



Click on Predict.

Now this will show the predicted price.