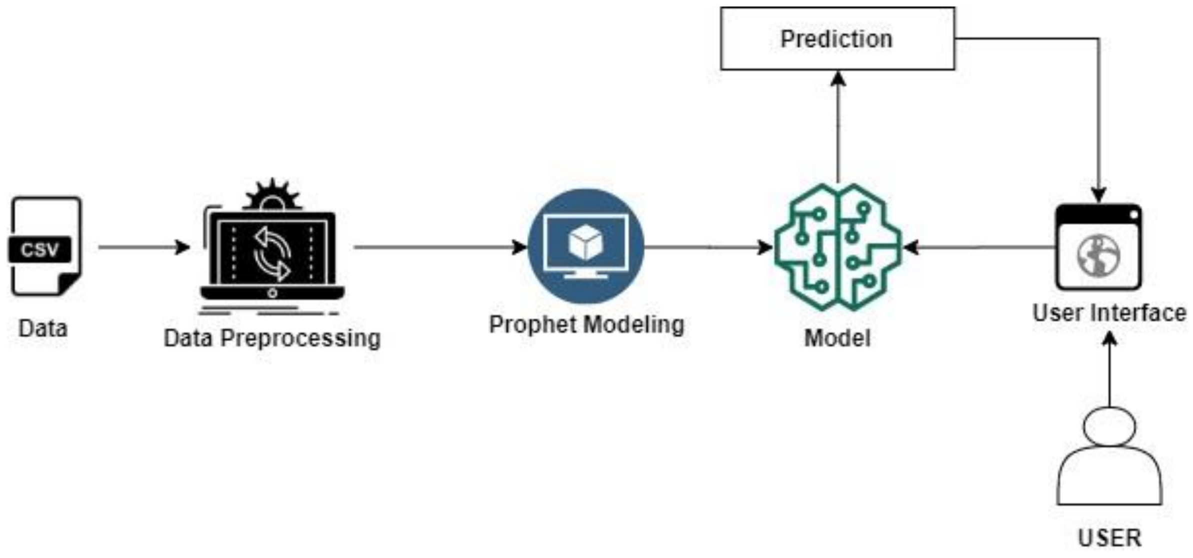
Crypto Price Prediction using FbProphet

Project Idea:

Bitcoin, a well-known cryptocurrency established in January 2009, holds significant value in the global cryptocurrency market. It is actively traded on numerous exchanges, supporting a variety of currencies. What sets Bitcoin apart is its substantial price volatility, far surpassing that of conventional currencies. For instance, in January 2017, its value was at $1,000, but it soared to $16,000 by the end of December 2017. Fast forward to June 2022, and its value stands at $25,711. This extreme price fluctuation characterizes the crypto market, and Bitcoin, with its unique blend of anonymity and system transparency, attracts a large number of investors.

The primary objective of this project is to develop a prediction system for Bitcoin prices using FbProphet, a forecasting model. Given the numerous factors influencing Bitcoin's price, the FbProphet model will be designed to project Bitcoin's future price trends.

Architecture:



Learning Outcomes:

By the end of this project, you will achieve the following learning outcomes:

- Gain a solid understanding of fundamental concepts in time series forecasting.
- Acquire the ability to analyze data and extract insights through data visualization.
- Learn how to develop a web application using the Flask framework.

Project Flow: In this project, we will follow a structured flow to achieve our goals:

1. **User Interaction**:
    - Users will interact with the project's user interface (UI) to input a specific date.

2. **Data Analysis with Integrated Model**:
    - The selected date input will be sent to the integrated model for analysis.
3. **Model Analysis and Prediction**:
    - The model will analyze the input data and make predictions based on the time series analysis.
4. **User Interface Output**:
    - The project will showcase the model's predictions on the UI.

To accomplish these project milestones, we will follow the activities below:

Installation of Pre-requisites:

- Set up your development environment, including:
    - Installing Anaconda IDE or Anaconda Navigator.
    - Installing necessary Python packages.

Data Collection:

- Gather or create the dataset required for this project.

Data Pre-processing:

- Prepare the dataset for analysis, which includes:
    - Importing required libraries.
    - Loading the dataset.
    - Analysing the data.
    - Handling missing values, resetting the index, and renaming columns.
    - Visualizing the time series data.

Model Building:

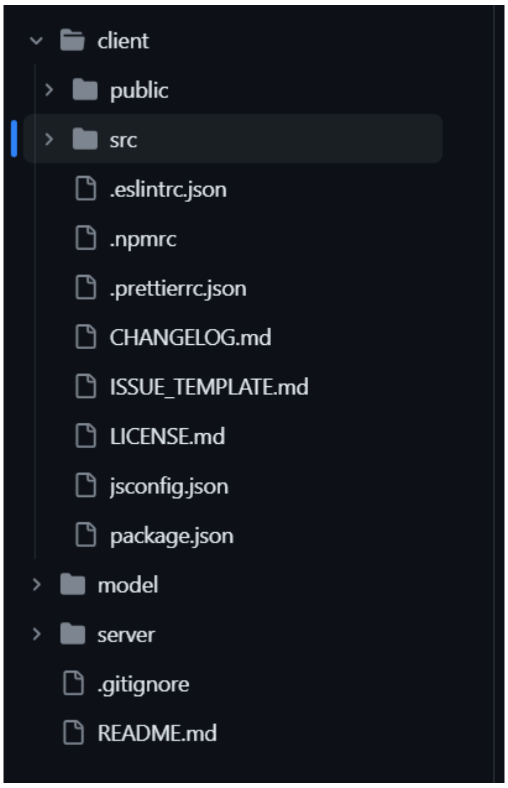
- Build a time series forecasting model using the Prophet library. This involves:
    - Fitting the Prophet model to the data.
    - Making future price predictions.
    - Evaluating the model's performance.
    - Saving the trained model for later use.

Application Building:

- Create the web application using Flask, which includes:
    - Developing an HTML file to design the user interface.
    - Writing Python code to serve as the application's backend, integrating the model for predictions.

Project Structure:

Create a Project folder that contains files as shown below



```
client
  public
  src
  .eslintrc.json
  .npmrc
  .prettierrc.json
  CHANGELOG.md
  ISSUE_TEMPLATE.md
  LICENSE.md
  jsconfig.json
  package.json
model
server
.gitignore
README.md
```

- All the above files will be used to develop a flask application.
- We are building a Flask Application that needs HTML pages stored in the templates folder and a python script app.py for serverside scripting
- The Model training folder contains the training file FB_prophet_Bitcoin_forecasting.ipynb
- The fbcrypto.pkl is the saved model file

Milestone 1: Installation of Pre-requisites To successfully set up your development environment, you need to install the following software and packages:

Activity 1: Install Anaconda IDE / Anaconda Navigator

- Anaconda IDE or Anaconda Navigator is essential for developing and testing your project code. It provides an integrated environment for data science and development.

Steps:

1. Download and install Anaconda Navigator from the following link: Anaconda Navigator Installation Guide

Activity 2: Installation of Python Packages

- To ensure that you have the necessary libraries and packages for your project, follow these steps to install them.

Steps:

1. Open Anaconda Navigator as an administrator.
2. In the Anaconda Navigator terminal, type and execute the following commands:
    - `pip install pystan==2.19.1.1`
    - `pip install yfinance`
    - `conda install -y fbprophet -c conda-forge`

These steps will install the required Python packages and libraries for your project.

Milestone 2: Data Collection For your project, you can collect datasets from various open sources such as Kaggle, data.gov, UCI Machine Learning Repository, etc.

Activity 1: Download the Dataset

- The dataset used in this project has been obtained from Yahoo Finance. Follow the link below to download the dataset covering the last 5 years of Bitcoin price data.

Dataset Link: [BTC-USD](Add link here)

Milestone 3: Data Pre-processing Data preprocessing is a critical step in preparing your dataset for analysis and forecasting.

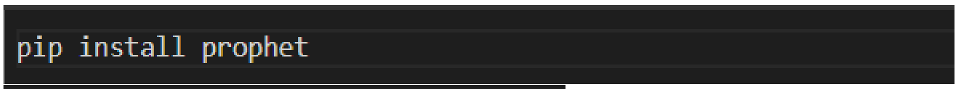
Activity 1: Import Libraries

- Import the necessary libraries for data pre-processing, forecasting using FbProphet, and more.

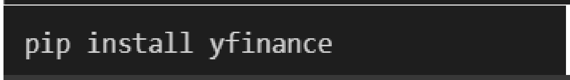
Steps:

- Import the following libraries for your project:
    - Pandas: A powerful data analysis and manipulation tool in Python.
    - Plotly: An interactive, open-source plotting library for data visualization.
    - Yahoo Finance: Used to download market data from the yfinance module.

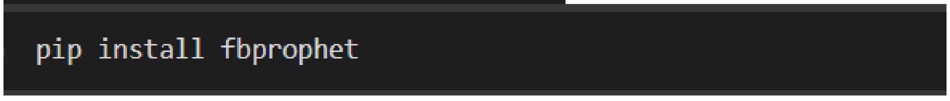
Make sure you have these libraries installed and properly imported to facilitate data pre-processing and analysis in your project.



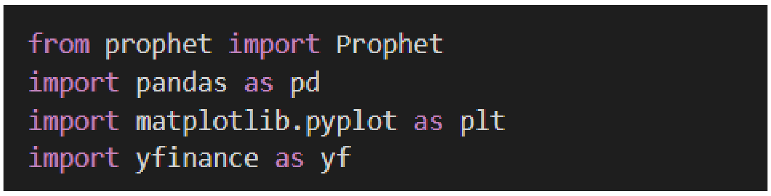
```
pip install prophet
```



```
pip install yfinance
```



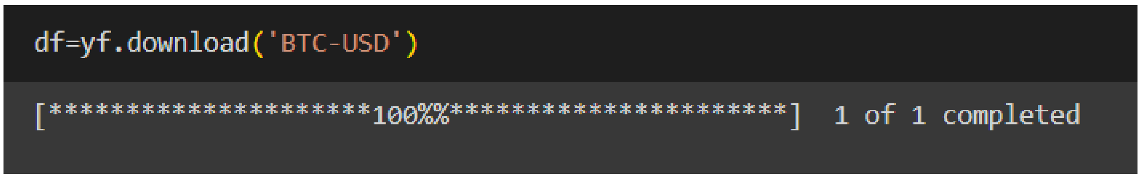
```
pip install fbprophet
```



```python
from prophet import Prophet
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
```

Activity 2: Import Dataset

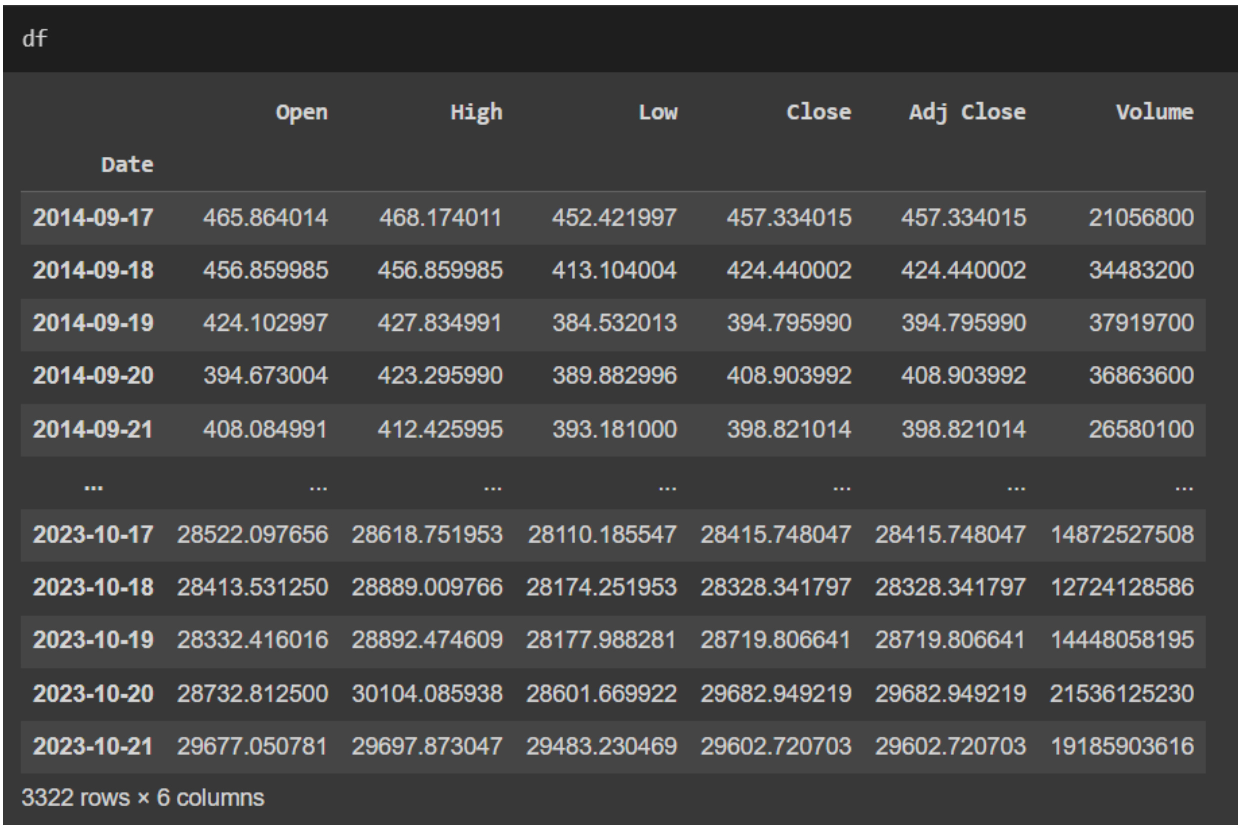
Download the real-time data from the Yahoo Finance library where we need to pass three parameters in the yahoo finance download function i.e. abbreviation name of the cryptocurrency, start date, and today date then we stored it into a variable called df.



```python
df=yf.download('BTC-USD')
```

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

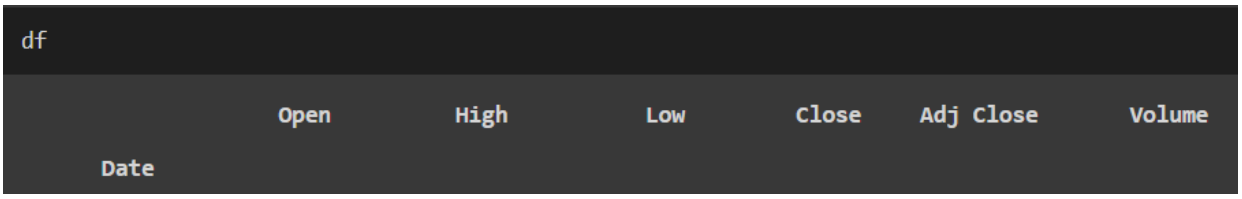
Check the entire dataset.



```python
df
```

```
                   Open          High           Low         Close     Adj Close       Volume
      Date
2014-09-17   465.864014    468.174011    452.421997    457.334015    457.334015     21056800
2014-09-18   456.859985    456.859985    413.104004    424.440002    424.440002     34483200
2014-09-19   424.102997    427.834991    384.532013    394.795990    394.795990     37919700
2014-09-20   394.673004    423.295990    389.882996    408.903992    408.903992     36863600
2014-09-21   408.084991    412.425995    393.181000    398.821014    398.821014     26580100
       ...          ...           ...           ...           ...           ...          ...
2023-10-17 28522.097656  28618.751953  28110.185547  28415.748047  28415.748047  14872527508
2023-10-18 28413.531250  28889.009766  28174.251953  28328.341797  28328.341797  12724128586
2023-10-19 28332.416016  28892.474609  28177.988281  28719.806641  28719.806641  14448058195
2023-10-20 28732.812500  30104.085938  28601.669922  29682.949219  29682.949219  21536125230
2023-10-21 29677.050781  29697.873047  29483.230469  29602.720703  29602.720703  19185903616
3322 rows × 6 columns
```

- Bitcoin Dataset contains the following Columns



```python
df
```

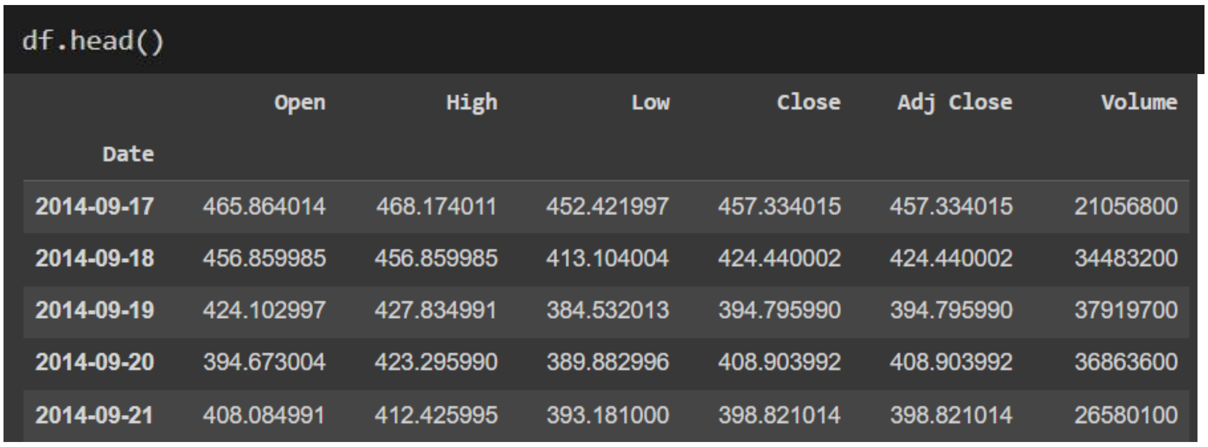
```
           Open      High       Low     Close   Adj Close    Volume
      Date
```

1. **Date:** This attribute represents the specific date or time for which the data is recorded. It provides a chronological order for the time series data.

2. **Open:** The "Open" price is the initial price of the financial asset at the beginning of the time interval. It is typically measured in the quote currency, meaning the currency used for quoting the asset's price. In the case of BTC/USD, this would be the opening price in USD.
3. **High:** The "High" price represents the highest price that the financial asset reached during the given time interval. It is also measured in the quote currency.
4. **Low:** The "Low" price is the lowest price that the financial asset touched during the specified time interval. Like the "High" price, it is also in the quote currency.
5. **Close:** The "Close" price is the final price of the financial asset at the end of the time interval. Similar to "Open," it is expressed in the quote currency.
6. **Adj Close:** "Adj Close" stands for "Adjusted Close." It represents the final price of the financial asset after adjusting for any relevant factors, such as stock splits or dividends. It provides a more accurate representation of the asset's performance.
7. **Volume:** The "Volume" attribute indicates the quantity of the financial asset that was bought or sold during the specified time interval. This volume is typically displayed in the base currency, which is the currency used for the underlying asset. In the case of BTC/USD, it would be the quantity of Bitcoin traded.

Activity 3: Analyse the data

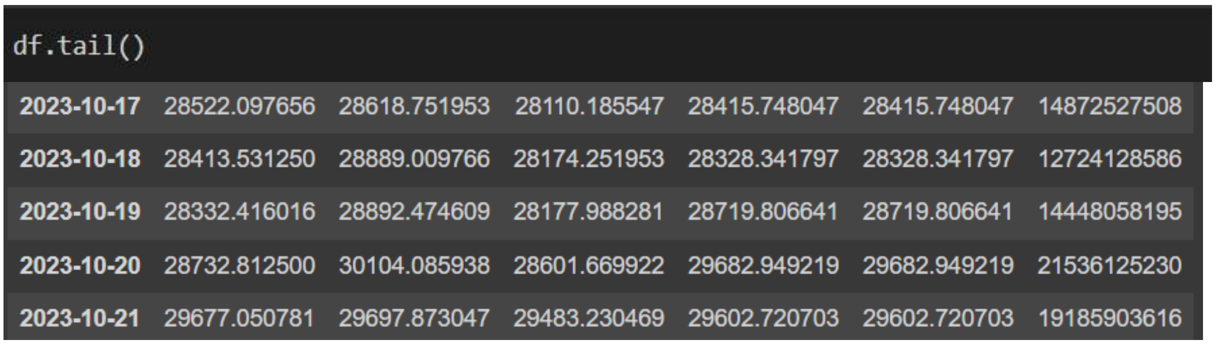
- the head() method is used to return the top n (5 by default) rows of a Data.



```
df.head()
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2014-09-17	465.864014	468.174011	452.421997	457.334015	457.334015	21056800
2014-09-18	456.859985	456.859985	413.104004	424.440002	424.440002	34483200
2014-09-19	424.102997	427.834991	384.532013	394.795990	394.795990	37919700
2014-09-20	394.673004	423.295990	389.882996	408.903992	408.903992	36863600
2014-09-21	408.084991	412.425995	393.181000	398.821014	398.821014	26580100

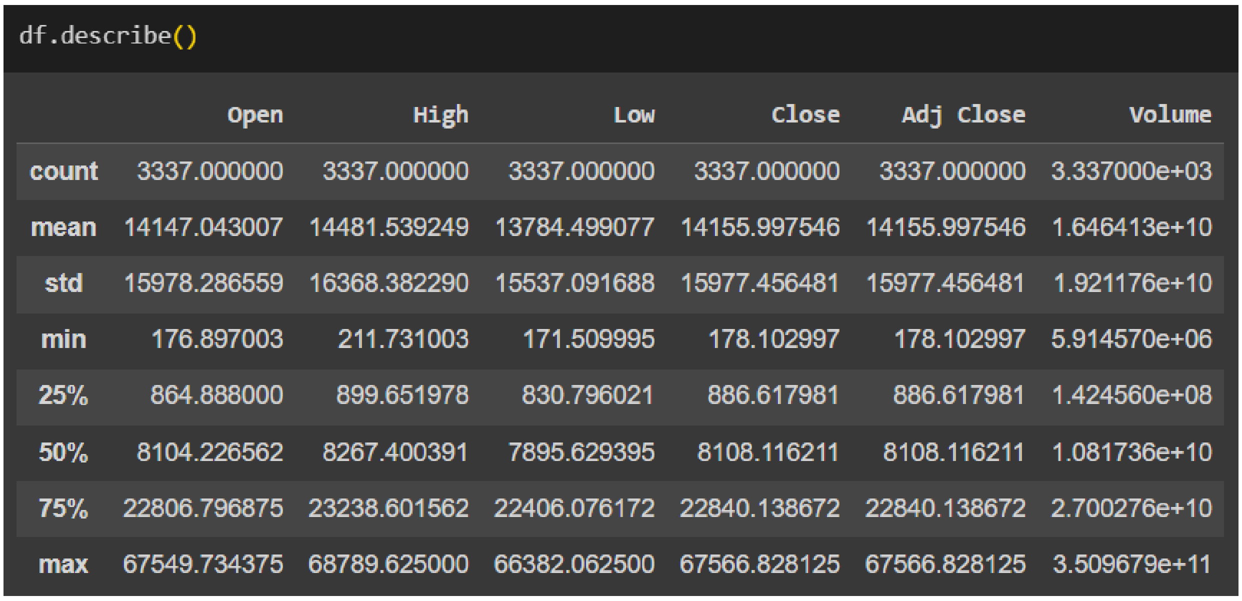
- List the Last five-row of the dataset using the tail function.



```
df.tail()
```

2023-10-17	28522.097656	28618.751953	28110.185547	28415.748047	28415.748047	14872527508
2023-10-18	28413.531250	28889.009766	28174.251953	28328.341797	28328.341797	12724128586
2023-10-19	28332.416016	28892.474609	28177.988281	28719.806641	28719.806641	14448058195
2023-10-20	28732.812500	30104.085938	28601.669922	29682.949219	29682.949219	21536125230
2023-10-21	29677.050781	29697.873047	29483.230469	29602.720703	29602.720703	19185903616

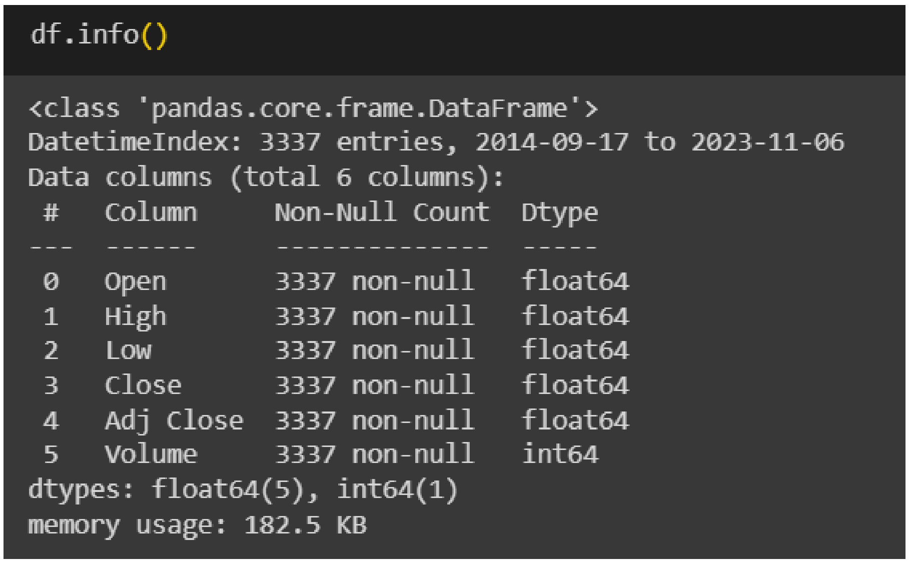
- describe() method computes a summary of statistics like count, mean, standard deviation, min, max, and quartile values.



```
df.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	3337.000000	3337.000000	3337.000000	3337.000000	3337.000000	3.337000e+03
mean	14147.043007	14481.539249	13784.499077	14155.997546	14155.997546	1.646413e+10
std	15978.286559	16368.382290	15537.091688	15977.456481	15977.456481	1.921176e+10
min	176.897003	211.731003	171.509995	178.102997	178.102997	5.914570e+06
25%	864.888000	899.651978	830.796021	886.617981	886.617981	1.424560e+08
50%	8104.226562	8267.400391	7895.629395	8108.116211	8108.116211	1.081736e+10
75%	22806.796875	23238.601562	22406.076172	22840.138672	22840.138672	2.700276e+10
max	67549.734375	68789.625000	66382.062500	67566.828125	67566.828125	3.509679e+11

- info() method prints information about the DataFrame. The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column (non-null values).



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3337 entries, 2014-09-17 to 2023-11-06
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Open       3337 non-null   float64
 1   High       3337 non-null   float64
 2   Low        3337 non-null   float64
 3   Close      3337 non-null   float64
 4   Adj Close  3337 non-null   float64
 5   Volume     3337 non-null   int64  
dtypes: float64(5), int64(1)
memory usage: 182.5 KB
```

Activity 4: Handling Missing Values, reset the index & renaming the column.

1. After loading it is important to check the complete information of data as it can indicate many of the hidden information such as null values in a column or a row

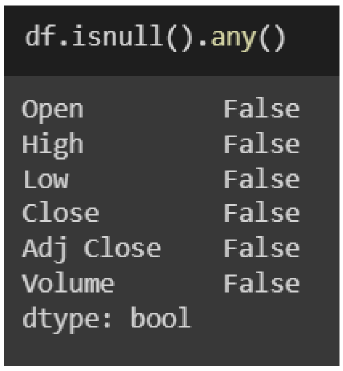
2. Check whether any null values are there or not. if it is present then the following can be done,

   a.Imputing data using the Imputation method in sklearn

   b.Filling NaN values with mean, median, and mode using fillna() method.

3. isnull()- Generate boolean mask indicating missing values.

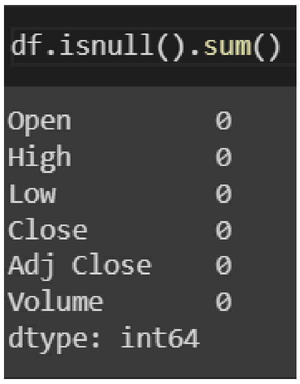
4. We don't have any missing values present in our dataframe.



```
df.isnull().any()
```

```
Open         False
High         False
Low          False
Close        False
Adj Close    False
Volume       False
dtype: bool
```

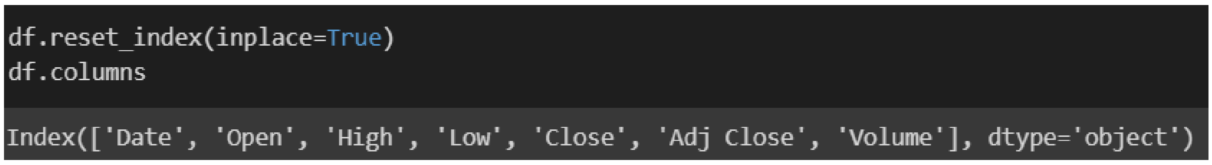
5. Check the total no of missing values presented in the dataset



```
df.isnull().sum()
```

```
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

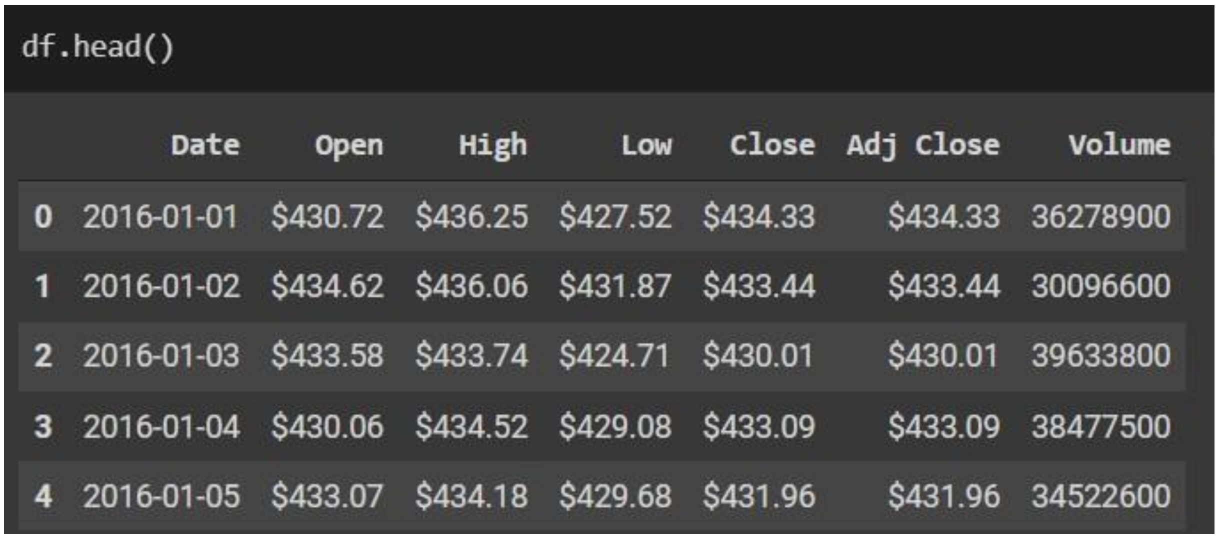
Now use the reset_index() function to generate a new DataFrame or Series with the index reset and it will add a date as a column.



```
df.reset_index(inplace=True)
df.columns
```

```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')
```

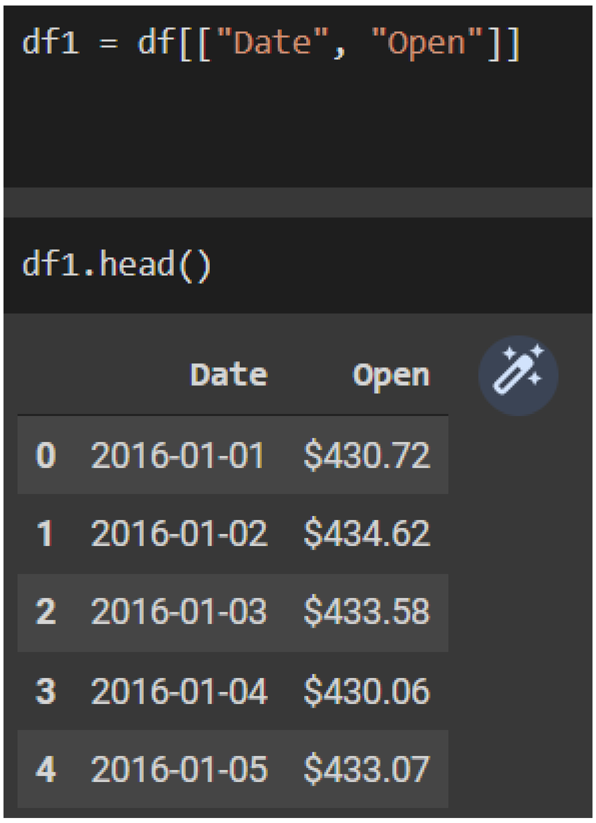
Now check the first five rows of data using the head function.



```
df.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2016-01-01	$430.72	$436.25	$427.52	$434.33	$434.33	36278900
1	2016-01-02	$434.62	$436.06	$431.87	$433.44	$433.44	30096600
2	2016-01-03	$433.58	$433.74	$424.71	$430.01	$430.01	39633800
3	2016-01-04	$430.06	$434.52	$429.08	$433.09	$433.09	38477500
4	2016-01-05	$433.07	$434.18	$429.68	$431.96	$431.96	34522600

Create a new dataframe with the Date and Open column and store it into df1 variable then check the top 5 rows of data using the head function.

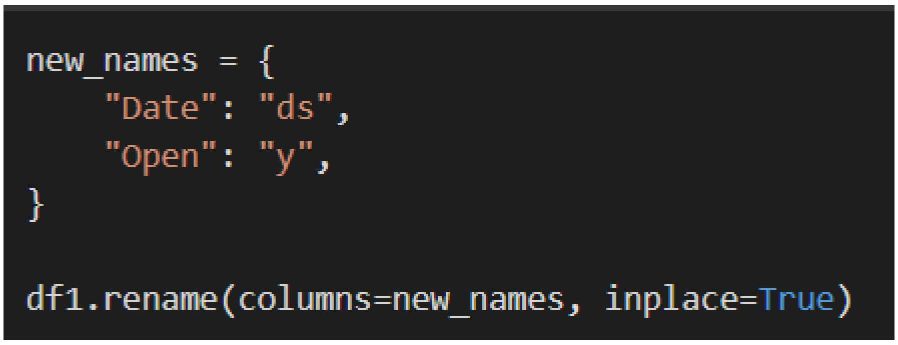


```python
df1 = df[["Date", "Open"]]
```

```python
df1.head()
```

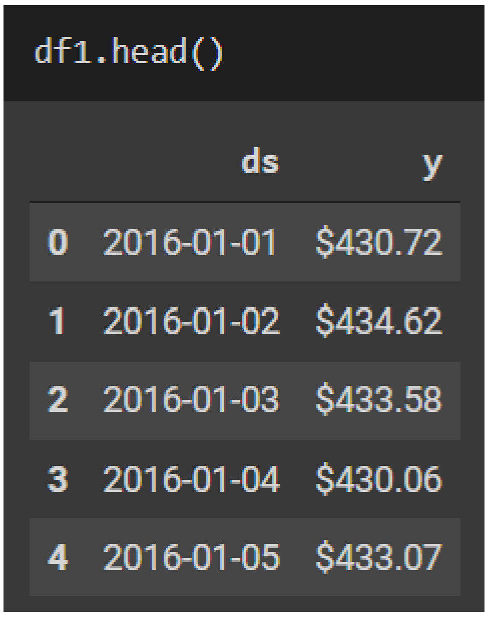
```
         Date     Open
0  2016-01-01  $430.72
1  2016-01-02  $434.62
2  2016-01-03  $433.58
3  2016-01-04  $430.06
4  2016-01-05  $433.07
```

- Renaming all the column names accordingly to the prophet library integration for building the model.



```python
new_names = {
    "Date": "ds",
    "Open": "y",
}

df1.rename(columns=new_names, inplace=True)
```

- List the first five rows of the dataset after the changes.



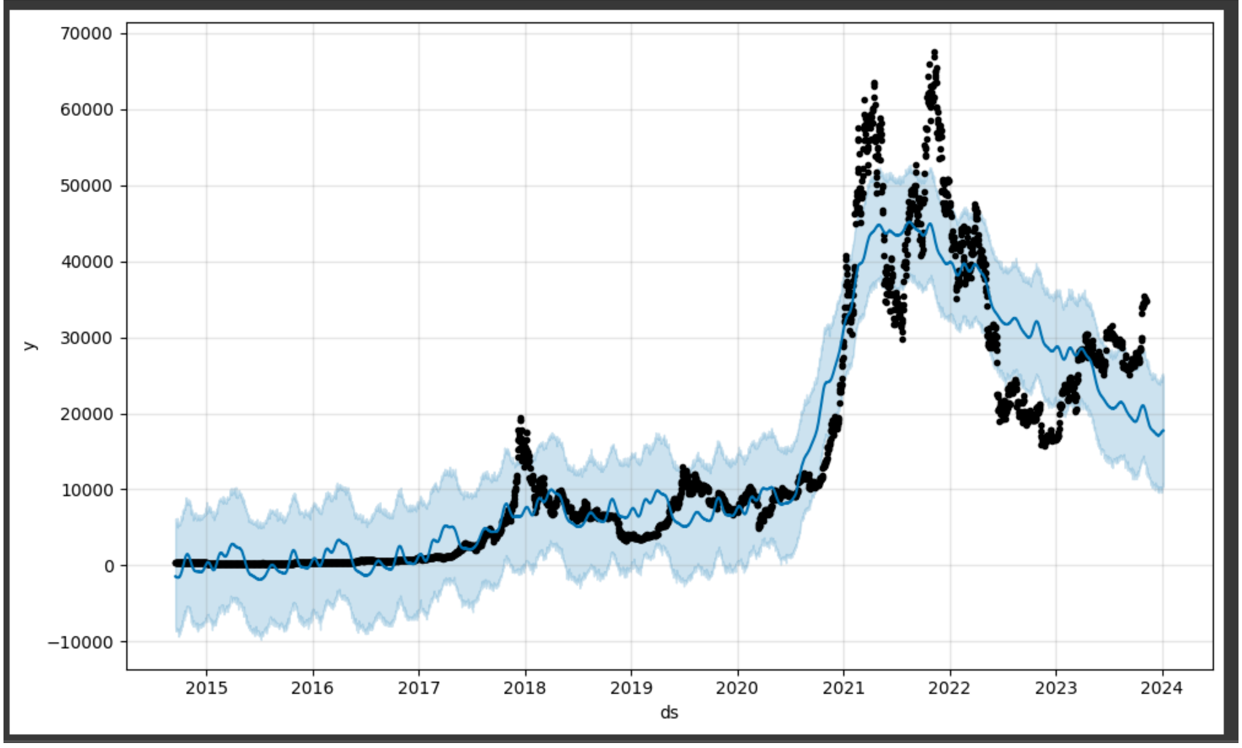
```python
df1.head()
```

```
           ds        y
0  2016-01-01  $430.72
1  2016-01-02  $434.62
2  2016-01-03  $433.58
3  2016-01-04  $430.06
4  2016-01-05  $433.07
```

Activity 5: Visualize Time Series Plot

- Now, let's visualize the data using the Plotly library for Time Series plot of Bitcoin Open Price.

```
fig=model.plot(prediction)
```



Milestone 4: Model Building

In this milestone, you will build the model using the prophet library.

Activity 1: Fitting the prophet library

- Create the instance of the prophet and fit it to the dataset.

By default Prophet fits additive seasonalities, meaning the effect of the seasonality is added to the trend to get the forecast. This time series of the price of Bitcoin where additive seasonality does not work. This time series has a clear yearly cycle, but the seasonality in the forecast is too large at the start of the time series and too small at the end. In this time series, the seasonality is not a constant additive factor as assumed by Prophet, rather it grows with the trend. This is multiplicative seasonality.

the prophet can model multiplicative seasonality by setting seasonality_mode='multiplicative' in the input arguments:

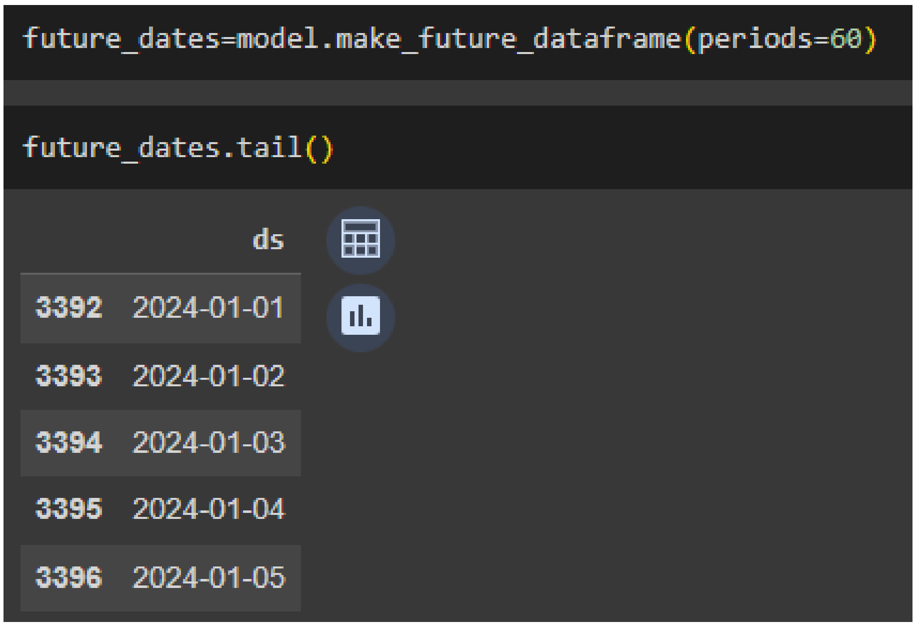
```
m = Prophet(
    seasonality_mode="multiplicative"
)

m.fit(df1)
```

```
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
<fbprophet.forecaster.Prophet at 0x7f7b86e76610>
```

Note: It will take a few minutes to fit the model.

Activity 2: Making Future Predictions

The next 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. We use the periods attribute to specify this. This also includes the historical dates. We'll use these historical dates to compare the predictions with the actual values in the ds column.



```
future_dates=model.make_future_dataframe(periods=60)
```

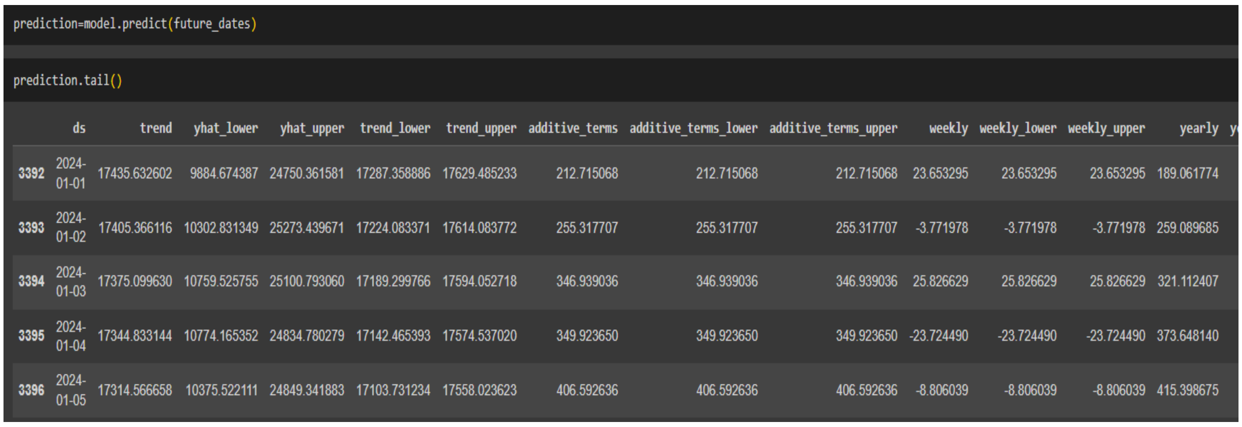
```
future_dates.tail()
```

	ds
3392	2024-01-01
3393	2024-01-02
3394	2024-01-03
3395	2024-01-04
3396	2024-01-05

Activity 3: Evaluate the model

We use the predict method to make future predictions. This will generate a dataframe with an **yhat** column that will contain the predictions.

If we check the head for our forecast dataframe we'll notice that it has very many columns. However, we are mainly interested in **ds, yhat, yhat_lower and yhat_upper**. **yhat** is our predicted forecast, **yhat_lower** is the lower bound for our predictions and **yhat_upper** is the upper bound for our predictions.

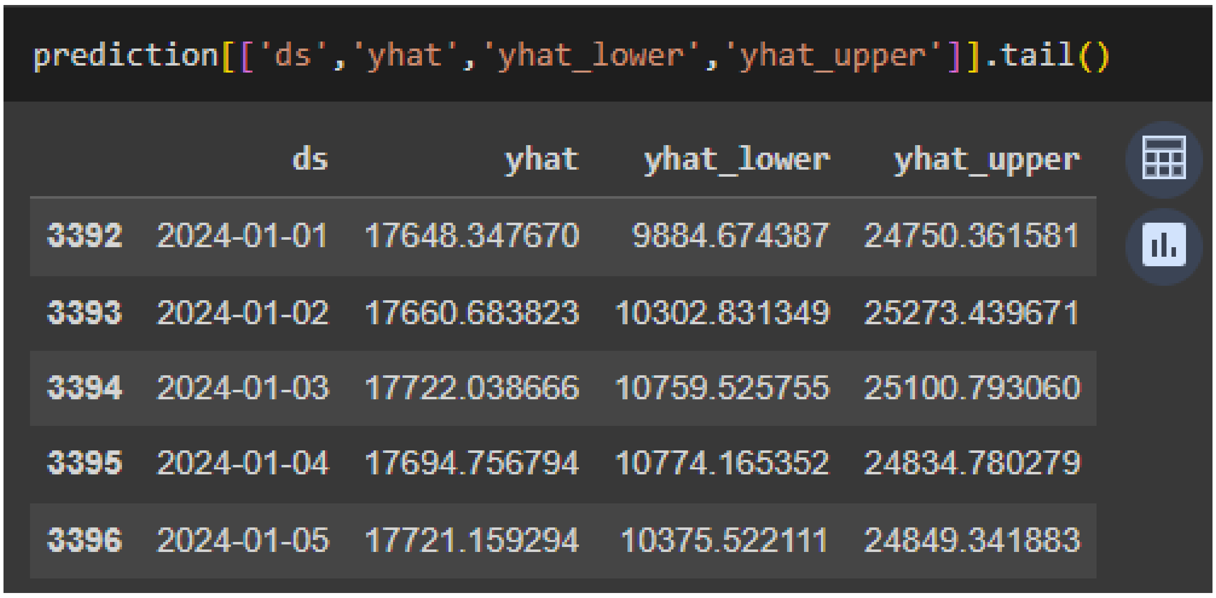


```
prediction=model.predict(future_dates)
```

```
prediction.tail()
```

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_terms_lower	additive_terms_upper	weekly	weekly_lower	weekly_upper	yearly	y
3392	2024-01-01	17435.632602	9884.674387	24750.361581	17287.358886	17629.485233	212.715068	212.715068	212.715068	23.653295	23.653295	23.653295	189.061774	
3393	2024-01-02	17405.366116	10302.831349	25273.439671	17224.083371	17614.083772	255.317707	255.317707	255.317707	-3.771978	-3.771978	-3.771978	259.089685	
3394	2024-01-03	17375.099630	10759.525755	25100.793060	17189.299766	17594.052718	346.939036	346.939036	346.939036	25.826629	25.826629	25.826629	321.112407	
3395	2024-01-04	17344.833144	10774.165352	24834.780279	17142.465393	17574.537020	349.923650	349.923650	349.923650	-23.724490	-23.724490	-23.724490	373.648140	
3396	2024-01-05	17314.566658	10375.522111	24849.341883	17103.731234	17558.023623	406.592636	406.592636	406.592636	-8.806039	-8.806039	-8.806039	415.398675	

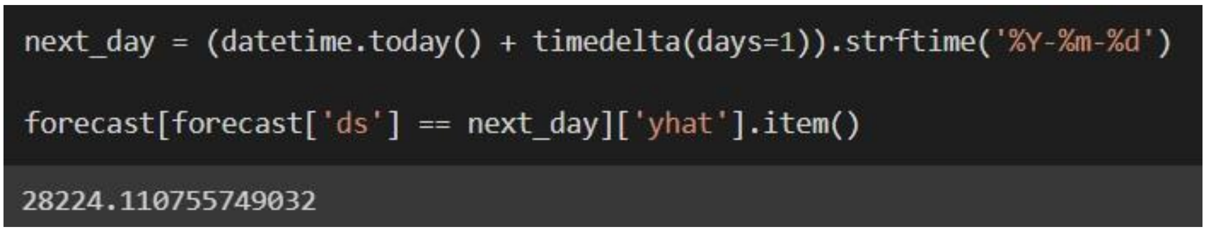
- Get the summary of the forecast.



```
prediction[['ds','yhat','yhat_lower','yhat_upper']].tail()
```

	ds	yhat	yhat_lower	yhat_upper
3392	2024-01-01	17648.347670	9884.674387	24750.361581
3393	2024-01-02	17660.683823	10302.831349	25273.439671
3394	2024-01-03	17722.038666	10759.525755	25100.793060
3395	2024-01-04	17694.756794	10774.165352	24834.780279
3396	2024-01-05	17721.159294	10375.522111	24849.341883

- For predicting the next day's price we calculate the DateTime and stored it into the next_day variable then predict that value.

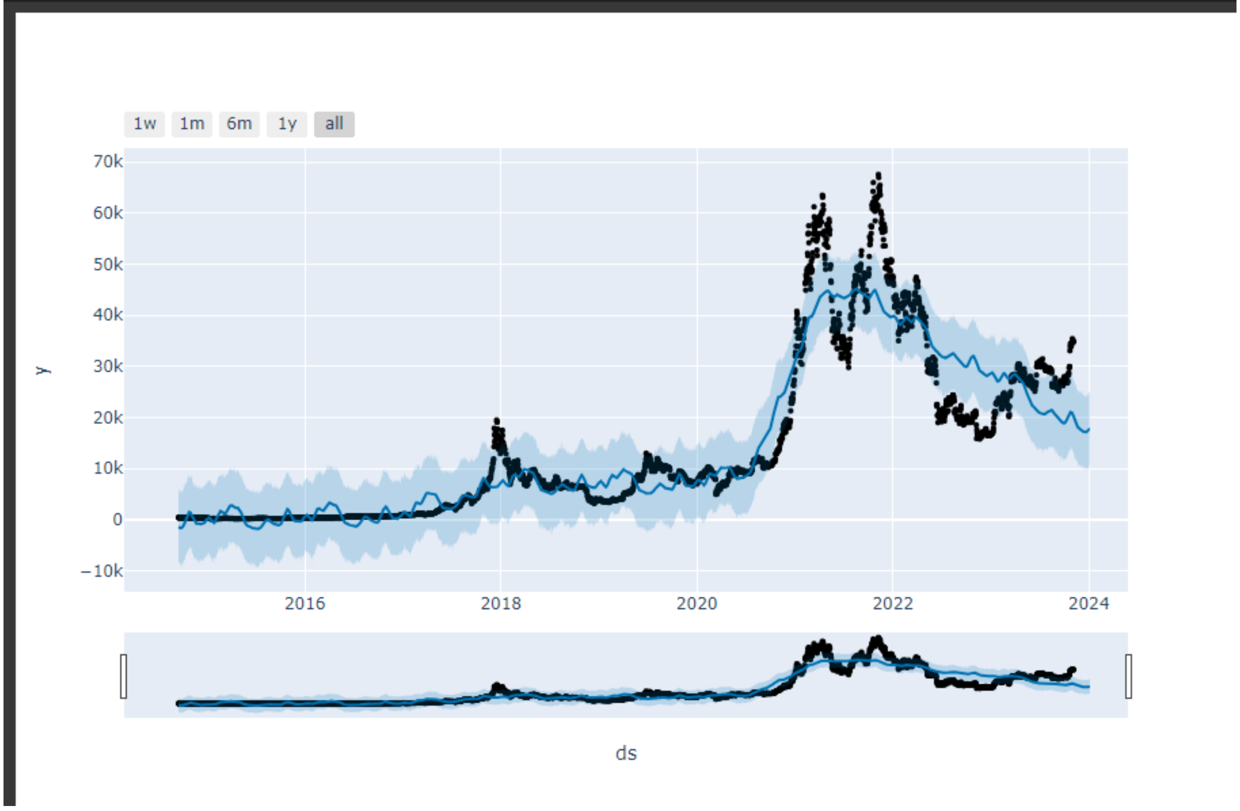


```
next_day = (datetime.today() + timedelta(days=1)).strftime('%Y-%m-%d')

forecast[forecast['ds'] == next_day]['yhat'].item()
```

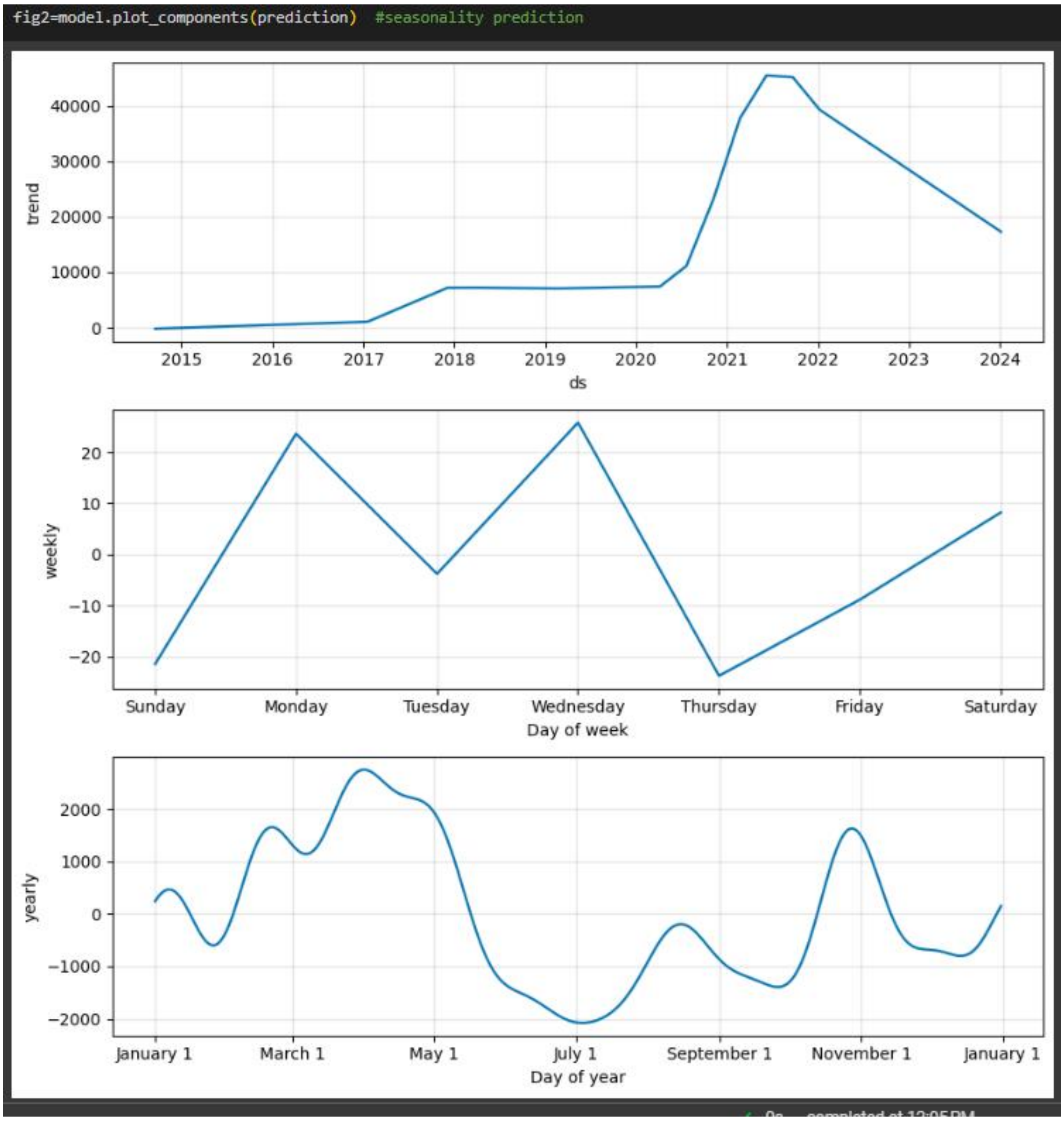
```
28224.110755749032
```

- Now, let’s visualize the forecast value of the Bitcoin price till the next year.

```
from prophet.plot import plot_plotly
plot_plotly(model,prediction)        #for selecting data upto certain time
```



- Visualize the components using plot_components_plotly which showcases the trend of the bitcoin price, Yearly growth in percentage, and weekly growth in percentage.

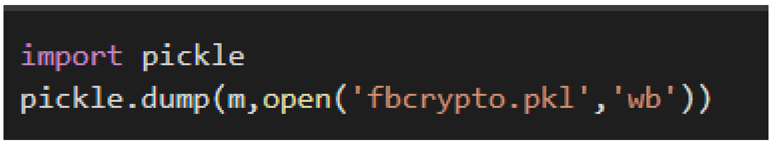


```
fig2=model.plot_components(prediction)  #seasonality prediction
```

Activity 4: Save the model.

This is the final activity of this milestone, here you will be saving the model to integrate to the web application.

- Follow the commands to save your model.



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

Milestone 5: Application Building

Now that we have trained our model, let us build our flask application which will be running in our local browser with a user interface.

In the flask application, the input parameters are taken from the HTML page These factors are then given to the model to predict the price of bitcoin on a selected date and showcased on the HTML page to notify the user. Whenever the user interacts with the UI and selects the "predict" button, the next page is opened where the user selects the date and predicts the output.

Activity 1 : Create HTML Pages

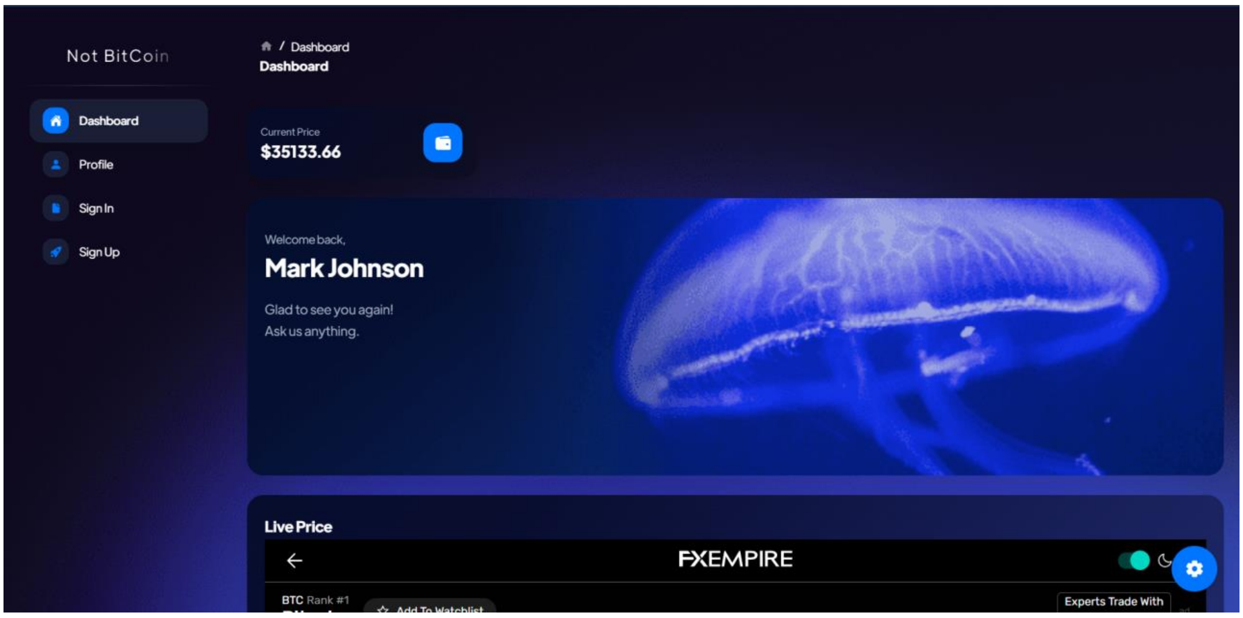
- We use HTML to create the front-end part of the web page.
- Here, we have created 2 HTML pages- predict.html & index.html.
- index.html displays the home page for an introduction to the project
- predict.html gives the prediction of bitcoin based on the selection

date. o For more information regarding HTML & CSS

Reference Links : HTML & CSS

index.html looks like this

predict.html looks like this



Activity 2: Build python code

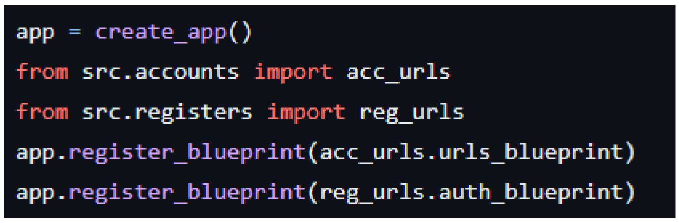
Task 1: Importing Libraries

The first step is usually importing the libraries that will be needed in the program.

```
from flask import request,jsonify
import pickle
import pandas as pd
from src import create_app
```

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.

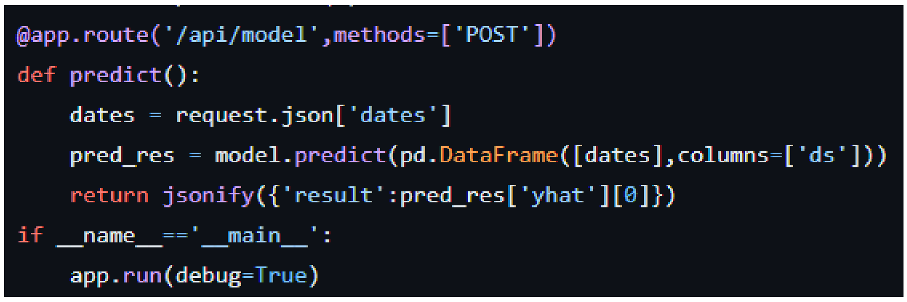
Task 2: Creating our flask application and loading our model by using pickle.load() method



```
app = create_app()
from src.accounts import acc_urls
from src.registers import reg_urls
app.register_blueprint(acc_urls.urls_blueprint)
app.register_blueprint(reg_urls.auth_blueprint)
```

Task 3: Routing to the HTML Pages

Here, the declared constructor is used to route to the HTML page created earlier.

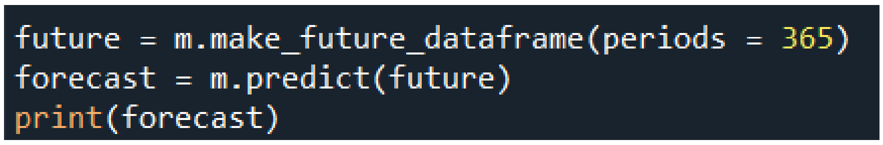


```
@app.route('/api/model',methods=['POST'])
def predict():
    dates = request.json['dates']
    pred_res = model.predict(pd.DataFrame([dates],columns=['ds']))
    return jsonify({'result':pred_res['yhat'][0]})
if __name__=='__main__':
    app.run(debug=True)
```

In the above 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.

Task 4:Making Future Prediction

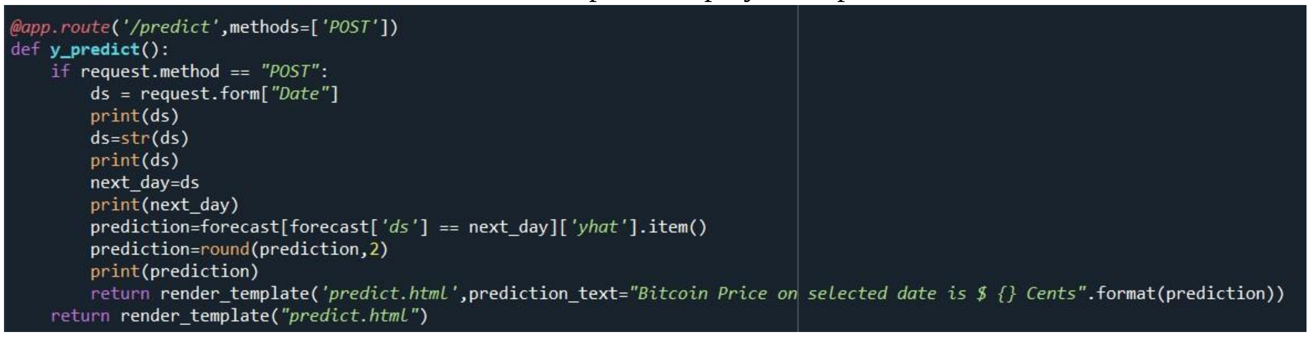
This step is to prepare our model to make future predictions. This is achieved using the Prophet.make_future_dataframe method and passing the number of days we'd like to predict in the future and storing the data into a forecast variable.



```
future = m.make_future_dataframe(periods = 365)
forecast = m.predict(future)
print(forecast)
```

Showcasing prediction on UI:

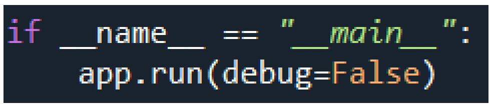
Firstly, we are rendering the index.html template and from there we are navigating to our prediction page that is predict.html. We select the date (year, month & day) these values are sent to the loaded model, and the resultant output is displayed on predict.html.



```
@app.route('/predict',methods=['POST'])
def y_predict():
    if request.method == "POST":
        ds = request.form["Date"]
        print(ds)
        ds=str(ds)
        print(ds)
        next_day=ds
        print(next_day)
        prediction=forecast[forecast['ds'] == next_day]['yhat'].item()
        prediction=round(prediction,2)
        print(prediction)
        return render_template('predict.html',prediction_text="Bitcoin Price on selected date is $ {} Cents".format(prediction))
    return render_template("predict.html")
```

Finally, Run the application

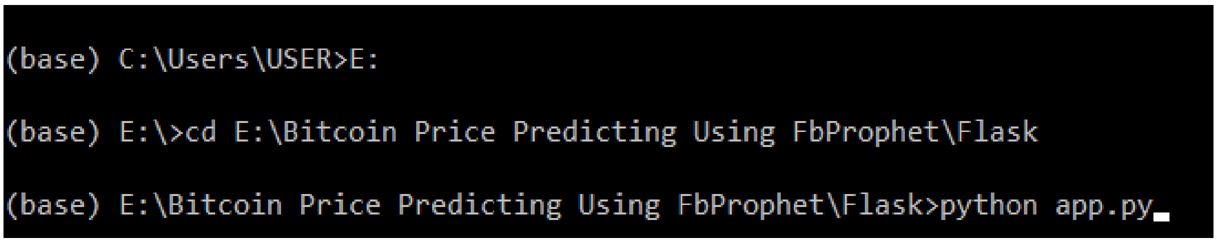
This is used to run the application on localhost.



```python
if __name__ == "__main__":
    app.run(debug=False)
```

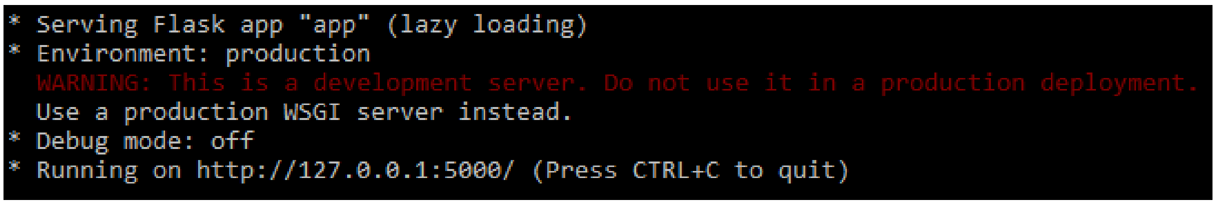
Activity 3: Running of flask Application

- 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) C:\Users\USER>E:

(base) E:\>cd E:\Bitcoin Price Predicting Using FbProphet\Flask

(base) E:\Bitcoin Price Predicting Using FbProphet\Flask>python app.py
```

Then it will run on localhost:5000

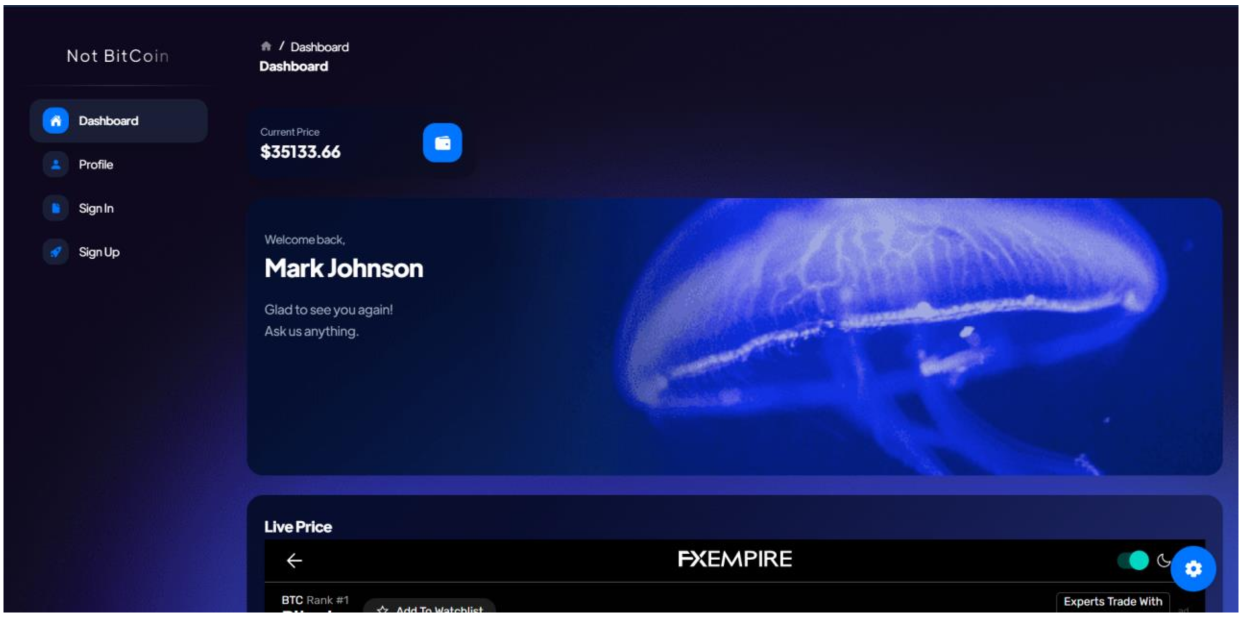


```
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Navigate to the localhost (http://127.0.0.1:5000/) where you can view your web page.

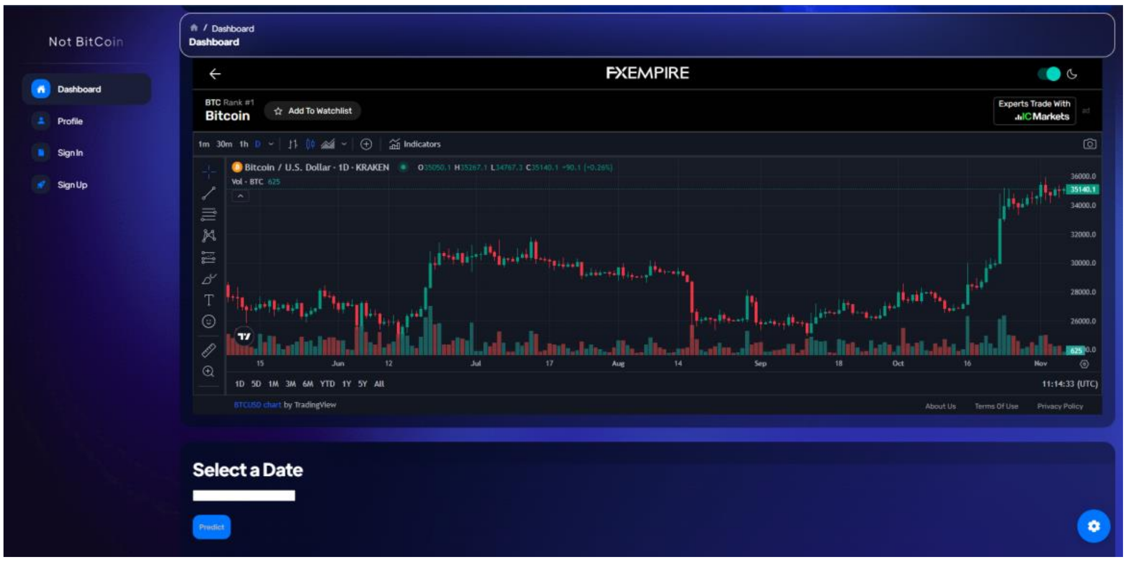
Predicting the results:-

The home page is displayed when we write the localhost URL to the browser.

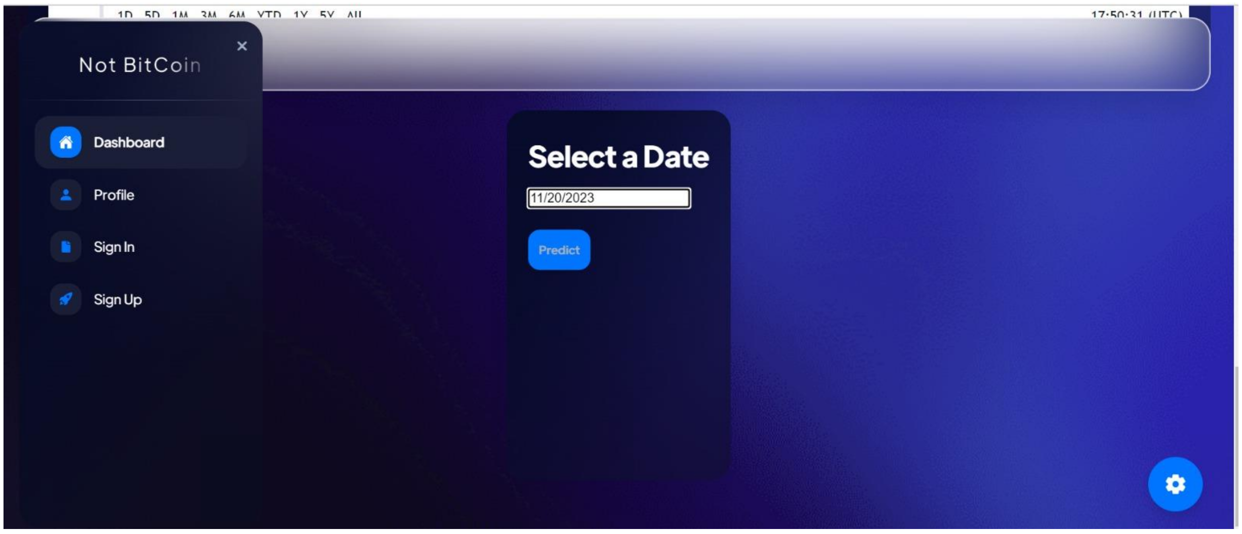


If we select to predict from the navigation bar, we will be redirected to the prediction page.

- Select the date, click on the submit button and the result/prediction will be reflected on the web page.

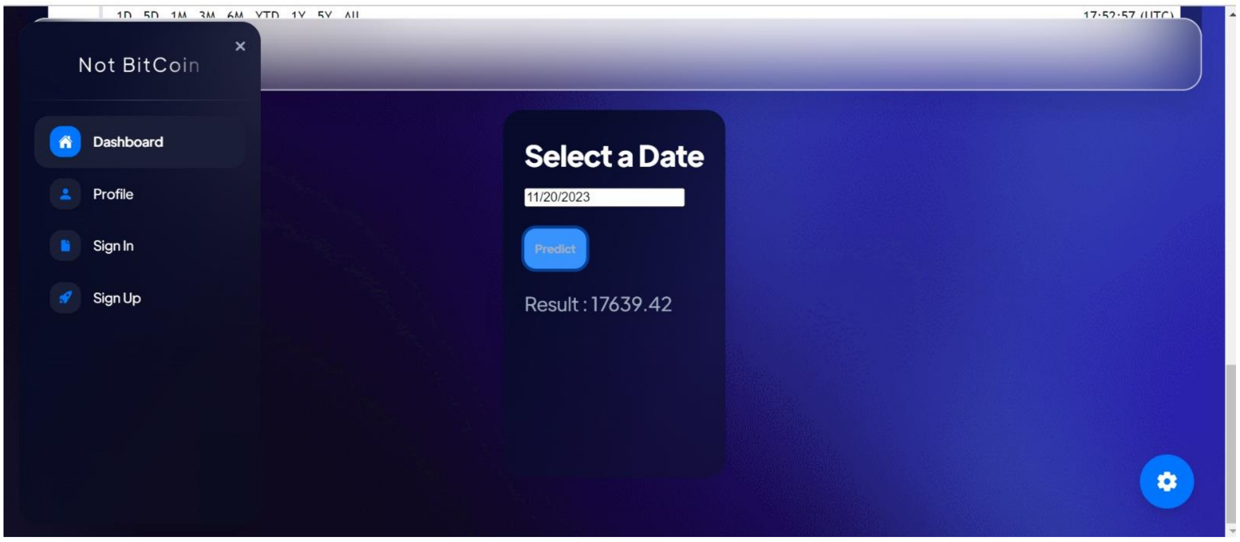


Input 1 :- We will select the date.



11/20/2023

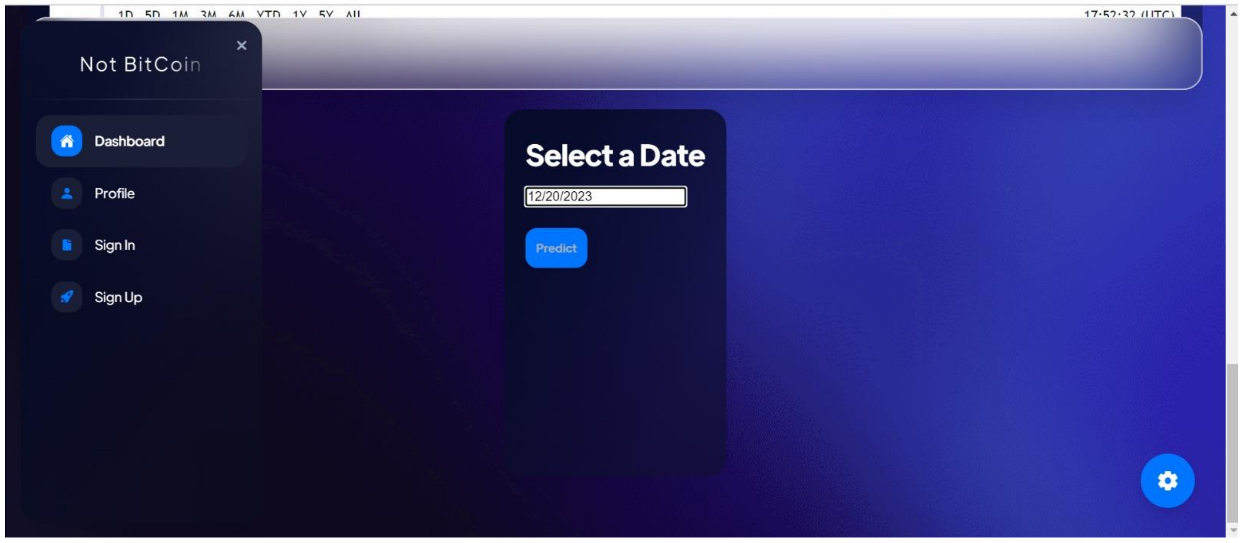
Based on the date selected it predicts the price of Bitcoin.



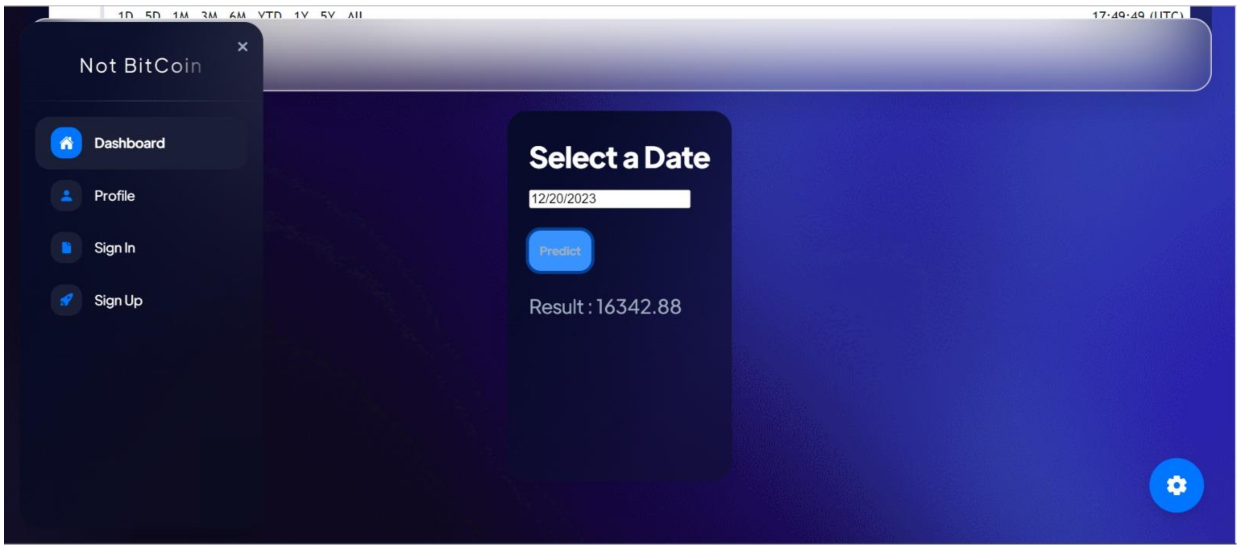
11/20/2023

Result : 17639.42

Input 2:- We will select a different date.



Based on the date selected it predicts the price of Bitcoin.



Result : 16342.88