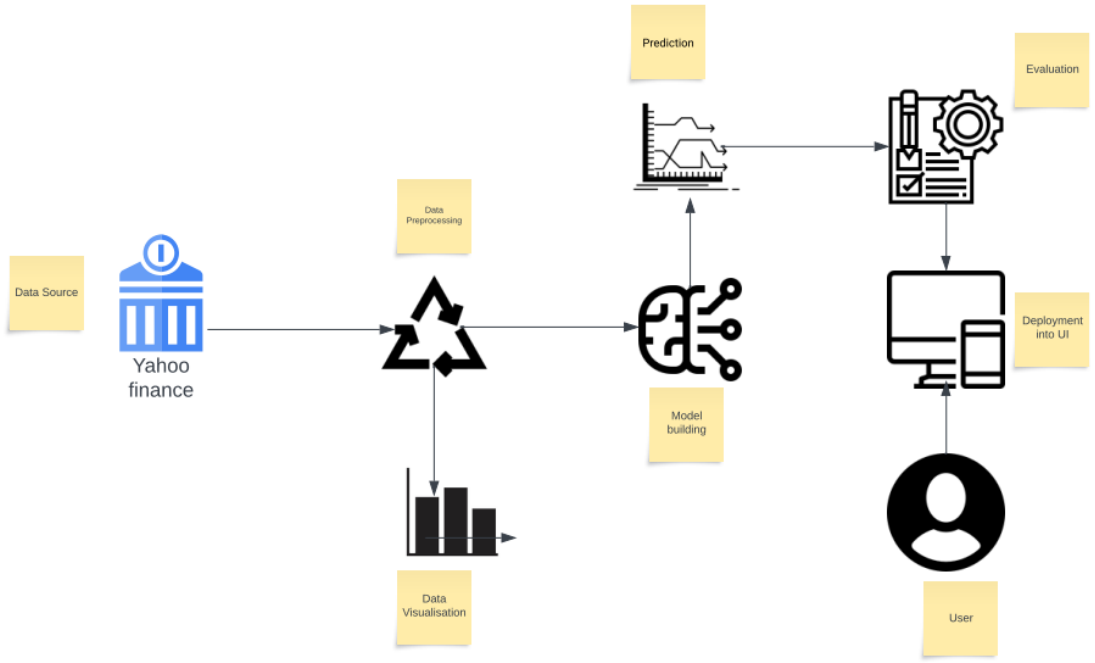
Crypto price prediction using FbProphet

Project idea

The core idea of this project is to be able to predict the price of Bitcoin which is the most valuable cryptocurrency in the world with reasonable accuracy, using time series forecasting libraries like FbProbhet.

This will not only enable us to take trades on the basis of future price but on a larger scale if the model is accurate enough it can also be used to make crypto market more predictable and hence more acceptable in general public who usually wants to avoid high risk environments

Architecture



Learning Outcomes

By working on this project we were able to learn following things-

→Downloading, interpreting and analysing financial data using Python, Yfinnance and Yahoo Finnance.

→Visualization of financial data using Python.

→Time series forecasting using FbProphet.

→Deployment of machine learning algorithms on web using Flask.

Project Flow

→First user interacts with the UI deployed using Flask in order to select the date.

→Then the selected date is sent to the model in the backend which has already been trained on the historical data, the model then uses that historical data as reference to predict the price of Bitcoin on the date selected.

→Then in the final part the out produced by the model is carried to UI where it is showcased as result to the user.

To achieve this flow, we divide our project into **six** major phases which are as follows

→Phase one – Setting up the environment

- In this phase we created an separate anaconda environment for the project, in-order to avoid any clashes in requirements, then we installed the FbProphet, Yfinnance, PlotLy, Flask and other libraries using 'pip install' command.

→Phase two – Data collection

- Once the environment was ready, our first task was to collect the historical data

→Phase three – Data preprocessing and data visualization

- In this phase we visualized the time series data and cleaned the data and made it ready for the algorithm

→Phase four – Model building

- Once the data was ready it was time to feed the data to our FbProphet model in order to train it, this is what was done in this phase

→Phase five – Deployment

- After the model was ready we finally deployed the model using HTML,CSS and Flask

Let's now look at these phases one by one in detail

Setting up the environment

→This phase was simple and straight forward, we started by downloading anaconda navigator, then installed the Jupyter notebook.

→Once this was done we began installing required libraries starting from FbProphet, then PlotLy, Flask and Yfinnance, we accomplished using pip commands.

Data collection

→The data collected in this project comes directly from Yahoo Finance .

→To collect the data we have used Yfinnance library of Python, the data is collected from 01/01/2016 till 02/11/23.

Loading the Dataset

```
[ ]  import yfinance as yf
```

```
[ ]  st_date = "2016-01-01"
     td_date = dt.today().strftime("%Y-%m-%d")

     print("Starting Date:", st_date)
     print("Today's Date :", td_date)
```

```
Starting Date: 2016-01-01
Today's Date : 2023-11-02
```

```
df = yf.download("BTC-USD", st_date, td_date)
df.head()
```

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

Date	Open	High	Low	Close	Adj Close	Volume
2016-01-01	430.721008	436.246002	427.515015	434.334015	434.334015	36278900
2016-01-02	434.622009	436.062012	431.869995	433.437988	433.437988	30096600
2016-01-03	433.578003	433.743011	424.705994	430.010986	430.010986	39633800
2016-01-04	430.061005	434.516998	429.084015	433.091003	433.091003	38477500
2016-01-05	433.069000	434.182007	429.675995	431.959991	431.959991	34522600

Data preprocessing and visualization

→This phase included understanding, visualizing and finally cleaning the data for model to use.

→First we checked weather there were any discrepancies.

```
[ ] df.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2862 entries, 2016-01-01 to 2023-11-01
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Open       2862 non-null   float64
 1   High       2862 non-null   float64
 2   Low        2862 non-null   float64
 3   Close      2862 non-null   float64
 4   Adj Close  2862 non-null   float64
 5   Volume     2862 non-null   int64
dtypes: float64(5), int64(1)
memory usage: 156.5 KB

[ ] df.shape

(2862, 6)

[ ] df.describe()
```

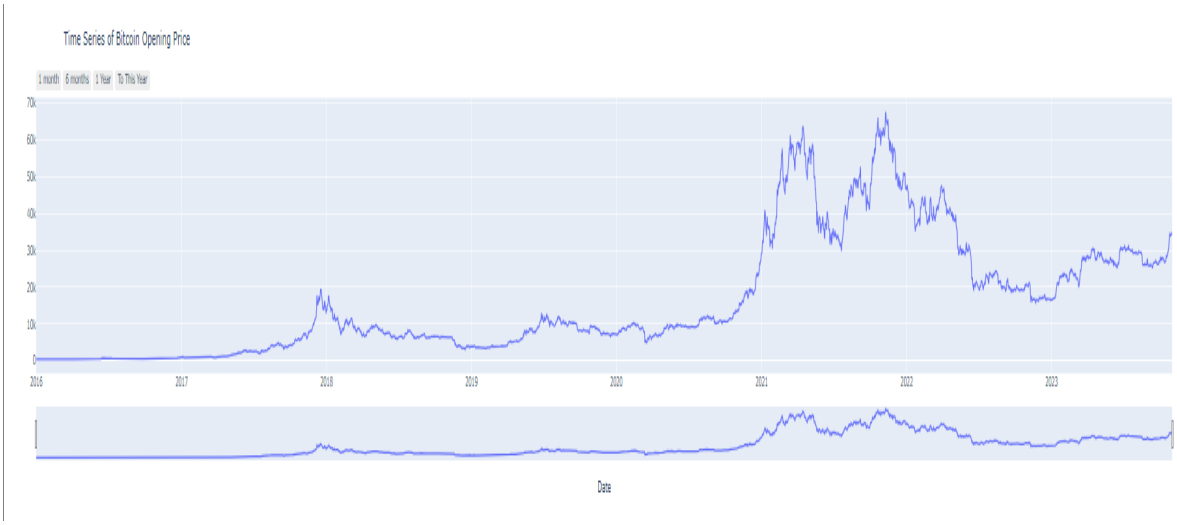
	Open	High	Low	Close	Adj Close	Volume
count	2862.000000	2862.000000	2862.000000	2862.000000	2862.000000	2.862000e+03
mean	16397.794126	16786.357058	15976.882811	16408.447428	16408.447428	1.917040e+10
std	16137.430435	16534.000945	15686.914678	16135.236348	16135.236348	1.945663e+10
min	365.072998	374.950012	354.914001	364.330994	364.330994	2.851400e+07
25%	4034.599426	4110.110962	3968.972778	4039.051270	4039.051270	3.671598e+09
50%	9484.943848	9671.880859	9282.042480	9487.736816	9487.736816	1.564545e+10
75%	26533.612305	26909.723145	26173.731934	26560.643066	26560.643066	2.992057e+10
max	67549.734375	68789.625000	66382.062500	67566.828125	67566.828125	3.509679e+11

```
Checking for NULL Values

[ ] df.isnull().sum()

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

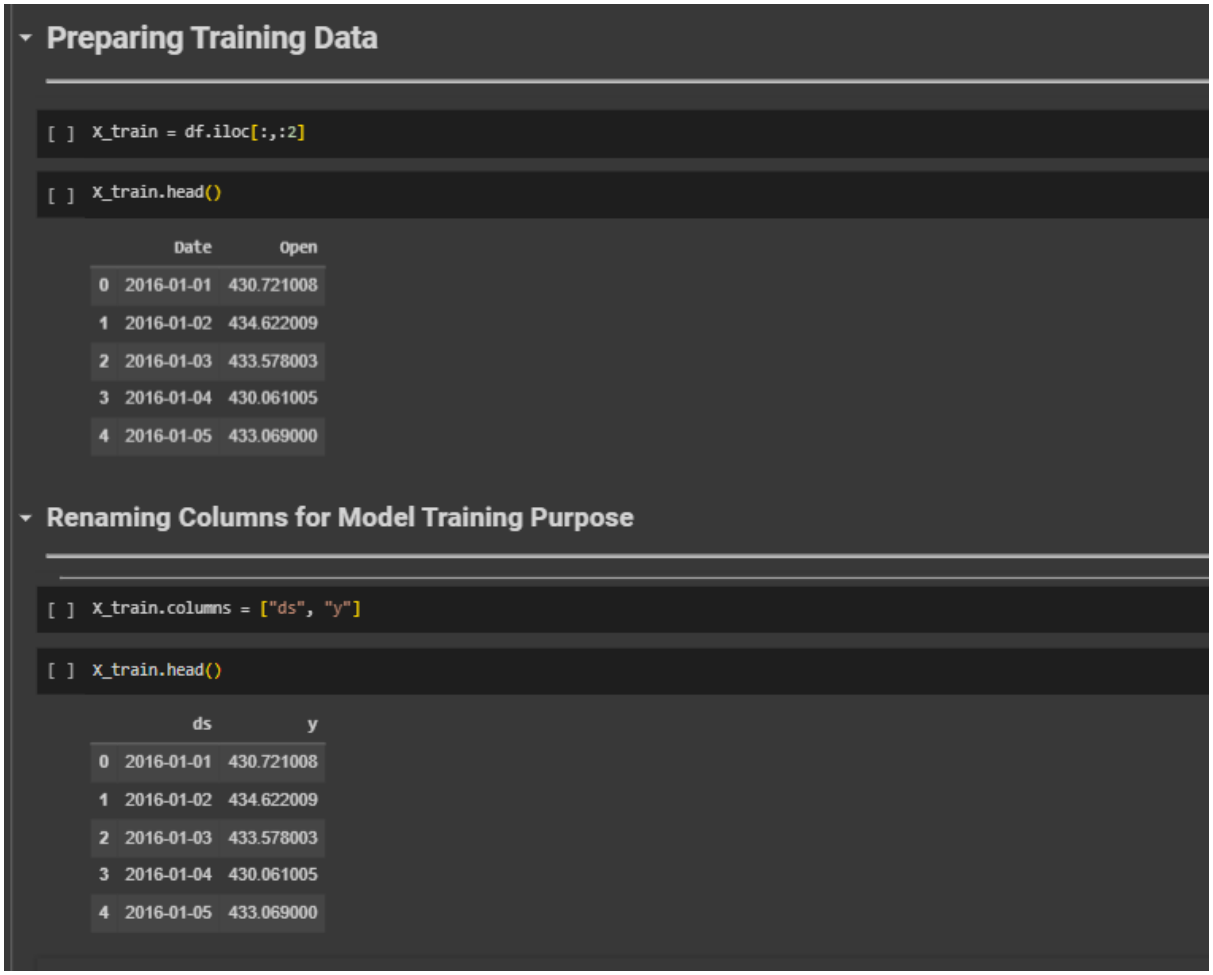
→After checking it was reviled that the there were some discrepancies in indexing of data which we fixed using ‘reset index’ function of ‘pandas’ apart from that there wasn’t anything that could hamper our accuracy of our model, so we started visualizing data to understand it better.

Data Visualisation

```
fig = go.Figure()
fig.add_trace(go.Scatter(x=df["Date"], y=df["Open"]))

fig.update_layout(
    title_text = "Time Series of Bitcoin Opening Price",
    xaxis = dict(

                rangeselector = dict(

                    buttons=list(
                        [
                            dict(count=1, label = "1 month",
                                 step="month", stepmode="backward"),
                            dict(count=6, label = "6 months",
                                 step="month", stepmode = "backward"),
                            dict(count=1, label = "1 Year",
                                step="year", stepmode="backward"),
                            dict(count=1, label = "To This Year",
                                step="year", stepmode="todate")
                        ]
                    )
                ),
                rangeslider = dict(visible = True),
                title = "Date",
    )
)
```



→In the final step of this phase we prepared the data for training and testing, we separated our target variable from rest of the dataset in-order for timeseries prediction to work.



Model building

→Once we had understood the data, cleaned it and separated target variable, it was time for us to actually build the model

→So first we imported the model, then we created an 'instance' of that model.

→After which we called 'fit' object from the model to fit our dataset.

```
[ ] from prophet import Prophet
```

Model Initialisation

```
[ ] prophet_model = Prophet(seasonality_mode = "multiplicative")
```

Training the Model

```
prophet_model.fit(X_train)
```

```
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpck6aktvm/uqb9otjc.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpck6aktvm/i5xj_r_u.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', 's
12:19:20 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
12:19:22 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
<prophet.forecaster.Prophet at 0x7f517fa13640>
```

→Once the model was fitted it was time for us to test it and check if it gave reasonable predictions, we did that using ‘predict’ object of the model.

Future Predictions

```
[ ] next_day = (dt.today() + td(days=1)).strftime("%Y-%m-%d")
    next_day
```

```
'2023-11-03'
```

```
[ ] y_preds[y_preds["ds"] == next_day][view_cols]
```

	ds	yhat	yhat_lower	yhat_upper
2863	2023-11-03	26398.829022	21572.512701	31299.63957

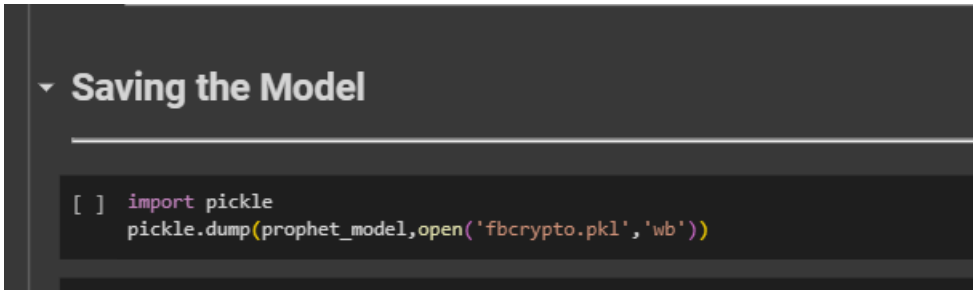
→After which we also plotted the future predictions of the model on the graph to understand it better.

```
[ ] from prophet.plot import plot_plotly, plot_components_plotly
```

```
plot_plotly(prophet_model, y_preds)
```


```
plot_components_plotly(prophet_model, y_preds)
```

→As the last step of this model we saved the it using ‘pickel’



→ After we save the model as a pickle file, we need to deploy the model using the same.

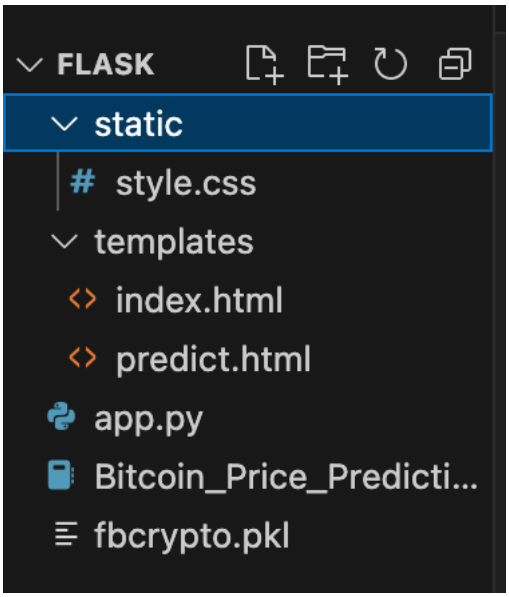
Model deployment

→ We are going to use Flask to deploy our machine learning model.

→ Flask requires a saved version of our ipynb where our model is being developed and we are using pickle to save our model in .pkl format.

→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.

The file structure of our flask deployment looks like this:

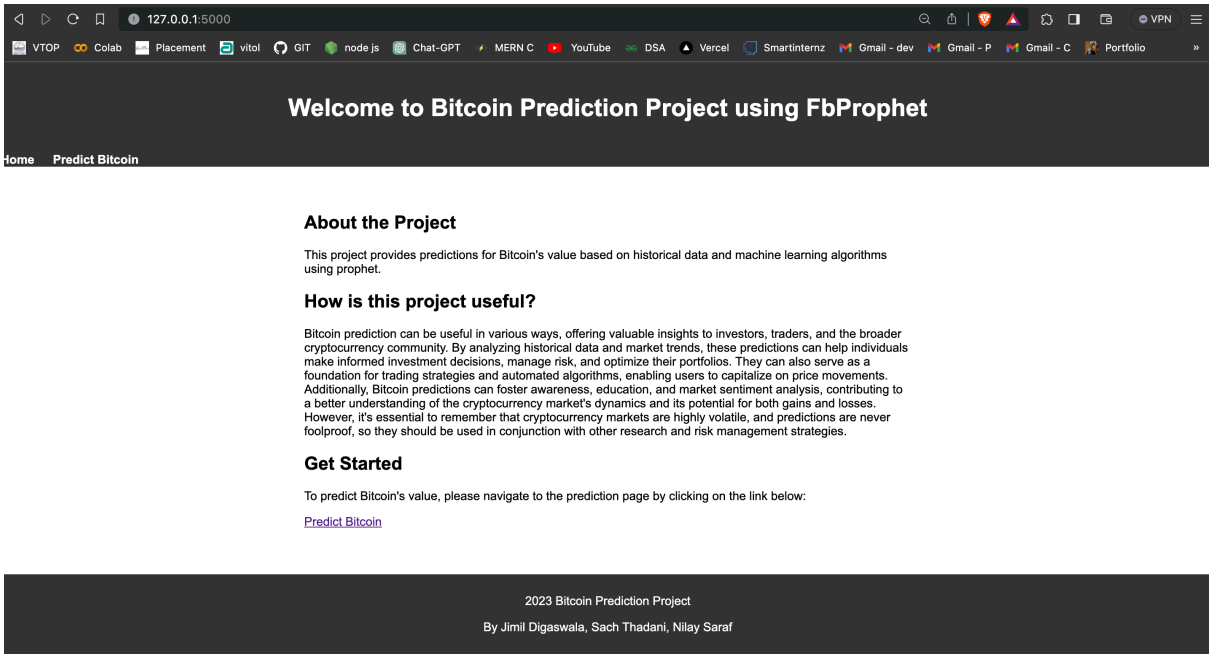


→We have made two html files, index.html for the homepage and predict.html for the prediction page.

→ The code and deployment of the **index.html** looks like this:

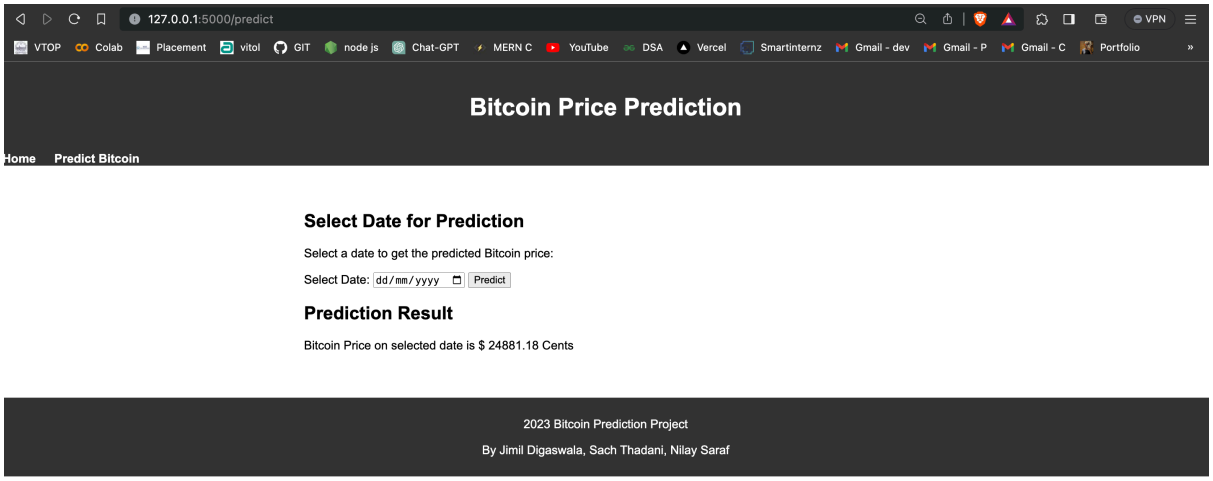
tes > index.html > ...

```
<!DOCTYPE html>
<html>
<head>
    <title>Bitcoin Prediction Project</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <header>
        <h1>Welcome to Bitcoin Prediction Project using FbProphet</h1>
    </header>
    <nav>
        <ul>
            <li><a href="index.html">Home</a></li>
            <li><a href="{{ url_for('prediction') }}">Predict Bitcoin</a></li>
        </ul>
    </nav>
    <main>
        <section>
            <h2>About the Project</h2>
            <p>This project provides predictions for Bitcoin's value based on historical data and machine learning algorithms using prophet.</p>
        </section>
        <section>
            <h2>How is this project useful?</h2>
            <p>Bitcoin prediction can be useful in various ways, offering valuable insights to investors, traders, and the broader cryptocurrency community.
                By analyzing historical data and market trends, these predictions can help individuals make informed investment decisions, manage risk, and
                optimize their portfolios. They can also serve as a foundation for trading strategies and automated algorithms, enabling users to capitalize
                on price movements. Additionally, Bitcoin predictions can foster awareness, education, and market sentiment analysis, contributing to a better
                understanding of the cryptocurrency market's dynamics and its potential for both gains and losses. However, it's essential to remember
                that cryptocurrency markets are highly volatile, and predictions are never foolproof, so they should be used in conjunction with other
                research and risk management strategies.</p>
        </section>
        <section>
            <h2>Get Started</h2>
            <p>To predict Bitcoin's value, please navigate to the prediction page by clicking on the link below:</p>
            <a href="{{ url_for('prediction') }}">Predict Bitcoin</a>
        </section>
    </main>
    <footer>
        <p> 2023 Bitcoin Prediction Project</p>
        <p> By Jimil Digaswala, Sach Thadani, Nilay Saraf </p>
    </footer>
</body>
</html>
```



→ The code and deployment of the **predict.html** looks like this:

```html
<!DOCTYPE html>
<html>
<head>
    <title>Bitcoin Prediction - Select Date</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <header>
        <h1>Bitcoin Price Prediction</h1>
    </header>
    <nav>
        <ul>
            <li><a href="index.html">Home</a></li>
            <li><a href="predict.html">Predict Bitcoin</a></li>
        </ul>
    </nav>
    <main>
        <section>
            <h2>Select Date for Prediction</h2>
            <p>Select a date to get the predicted Bitcoin price:</p>
            <form method="POST" action="/predict">
                <label for="date">Select Date:</label>
                <input type="date" id="date" name="Date">
                <input type="submit" value="Predict">
            </form>
        </section>
        <section>
            <h2>Prediction Result</h2>
            <p><span>{{ prediction_text }}</span></p>
        </section>
    </main>
    <footer>
        <p>2023 Bitcoin Prediction Project</p>
        <p> By Jimil Digaswala, Sach Thadani, Nilay Saraf </p>
    </footer>
</body>
</html>
```



Now, the main file of flask app.py is required to host the model as it contains the code which links the model, and the two html files responsible for ui and calling methods.

App.py code:

Task1: Importing libraries

```
import numpy as np
import pandas as pd
from flask import Flask, request, render_template
import pickle
```

Task2: Creating our model and loading out model using pandas.

```
app = Flask(__name__)
m = pd.read_pickle("fbcrypto.pkl")
```

Task3: Routing to HTML pages and making future predictions:

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

@app.route('/predict', methods=['POST', 'GET'])
def prediction():
    if request.method == 'POST':
        ds = request.form['Date']
        ds = str(ds)
        next_day = ds

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

        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')

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

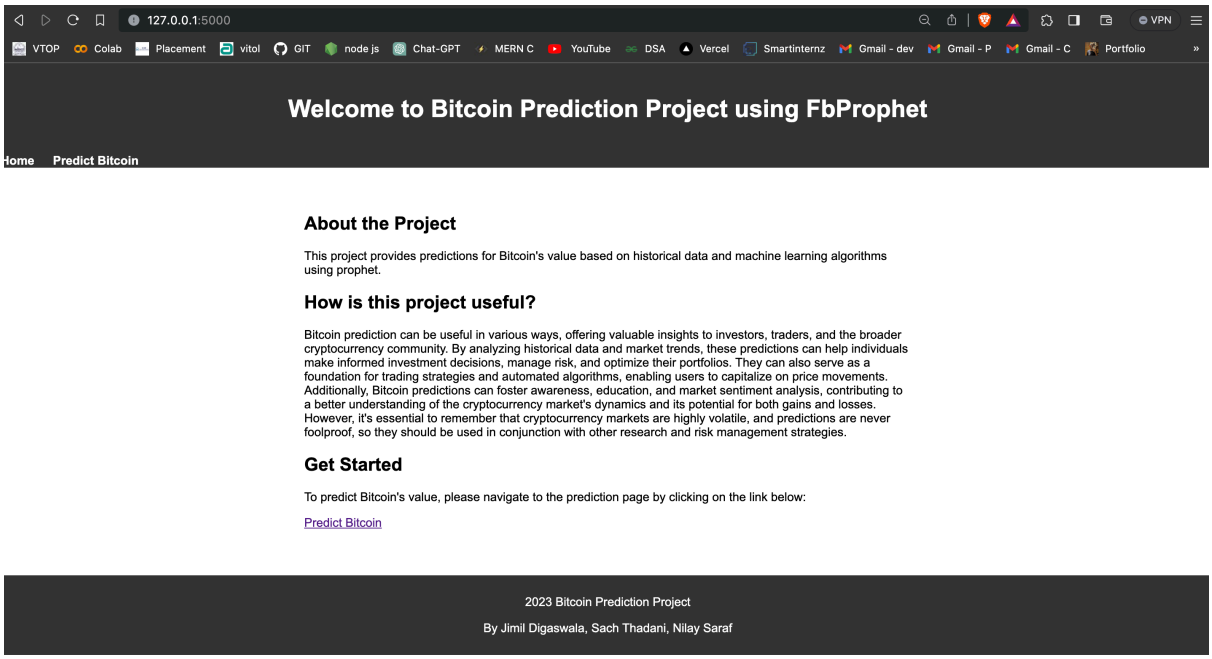
This is done using prophet model and the predictions made by the model is showed on the ui by routing in the html pages and then showing it in the deployment.

Steps to predict the bitcoin price using the ui:

1. Run the app.py file

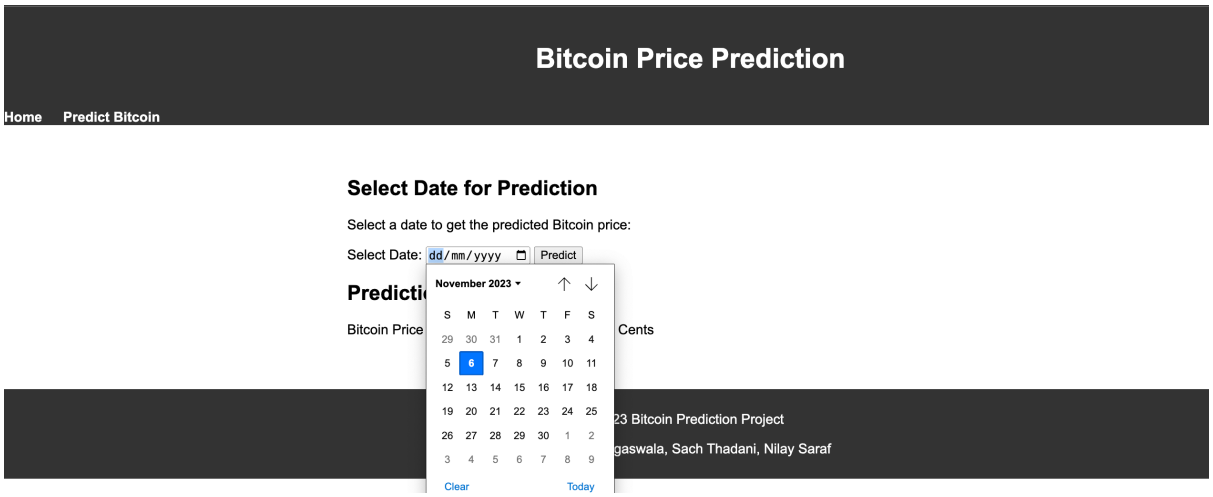
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

/usr/local/bin/python3 /Users/jimildigaswala/Desktop/flask/app.py
(base) jimildigaswala@Jimils-MacBook-Air flask % /usr/local/bin/python3 /Users/jimildigaswala/Desktop/flask/app.py
 * Serving Flask app 'app'
 * Debug mode: off
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
127.0.0.1 - - [06/Nov/2023 15:29:47] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [06/Nov/2023 15:29:47] "GET /static/style.css HTTP/1.1" 200 -
127.0.0.1 - - [06/Nov/2023 15:29:47] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [06/Nov/2023 15:30:43] "GET /predict HTTP/1.1" 200 -
127.0.0.1 - - [06/Nov/2023 15:30:43] "GET /static/style.css HTTP/1.1" 304 -
24881.18
127.0.0.1 - - [06/Nov/2023 15:30:49] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [06/Nov/2023 15:30:49] "GET /static/style.css HTTP/1.1" 304 -
```

2. A localhost link will be provided, navigate to the link and you will see the page given below:



Now, click on the predict button and you will be navigated to the prediction page.

3. In the predict page, you will find a dialog box to select the date you want to predict the bitcoin price for:



4. After selecting the date, click on the predict button and the predicted price should be displayed on the screen.