📈 Streamlit Stock Forecast App –Venu Madhuri Yerramsetti

This Streamlit app forecasts stock prices using **Yahoo Finance** 📊, Facebook **Prophet** 🔮, and **Plotly** 📈.

✨ Features

- **Historical Data Retrieval:** Instantly fetch historical stock data from Yahoo Finance. 🗓️
- **Intelligent Forecasting:** Utilize Facebook Prophet for robust future stock price predictions. 🔮
- **Interactive Visualizations:** Explore forecasts with dynamic and interactive plots powered by Plotly. 📊
- **Clean User Interface:** Enjoy a simple, intuitive, and responsive UI built with Streamlit. ✨

📦 Requirements

All necessary Python packages are listed in `requirements.txt`.

▶️📊 Stock Price Forecast App – Code Explanation Report

☑️ Purpose

This Streamlit app allows users to:

- Input a stock ticker (like `AAPL`, `RELIANCE.NS`)
- Fetch historical stock price data using Yahoo Finance
- Forecast future stock prices using Facebook Prophet
- Visualize the forecast and its components
- Inspect the raw forecast data

🧩 Key Components Explained

1. Libraries Used

- `streamlit`: To build the interactive web app UI
- `yfinance`: To fetch historical stock price data
- `pandas`: For data manipulation
- `plotly`: For interactive charts
- `prophet` (or `fbprophet`): For time-series forecasting

🖼️ User Interface

- **Sidebar**:
  - `Enter stock ticker`: Text box to enter stock symbols (default e.g., `AAPL`)
  - `Forecast period`: Dropdown to choose forecast length (7 to 365 days)
- **Main View**:
  - Displays a table of the recent stock price data used for forecasting
  - Shows:
    - Interactive forecast plot
    - Trend, seasonality components
    - Raw forecast table (last few entries)

🗃️ How It Works (Step-by-Step)

1. **User Input**:
   a. The user types a stock ticker and selects a forecast period.
2. **Data Fetching**:
   a. `yfinance.download()` fetches **5 years** of historical price data.
   b. Only the **last 3 years** are retained for modeling.
3. **Data Preparation**:
   a. The app automatically detects the correct "Close" column (if nested or named differently).

    b. Data is formatted to Prophet's required format:
        i. `ds`: Date column
        ii. `y`: Stock close price (numerical)
4. **Forecasting with Prophet**:
    a. Prophet fits the model to historical data.
    b. It generates future dates based on the selected forecast period.
    c. A forecast is made (`yhat`) with upper and lower confidence intervals.
5. **Visualizations**:
    a. **Forecast Plot**:
        i. Shows actual vs. predicted prices.
        ii. Includes uncertainty bands.
    b. **Forecast Components**:
        i. Breaks forecast into:
            1. Overall trend
            2. Weekly/annual seasonal effects
    c. **Forecast Data**:
        i. Displays last rows of predicted values:
            1. `ds`: Date
            2. `yhat`: Predicted price
            3. `yhat_lower` / `yhat_upper`: Confidence intervals

☑ Block-by-Block Code Explanation

📦 Imports

```
import streamlit as st
import yfinance as yf
import pandas as pd
import plotly.graph_objs as go

try:
    from prophet import Prophet
except ImportError:
    from fbprophet import Prophet
```

```
from prophet.plot import plot_plotly
```

- **Purpose**: Import necessary libraries for UI, data, plotting, and forecasting.
- **Special note**: Handles fallback if `prophet` is installed under the old `fbprophet` name.
- **What it enables**: All future blocks depend on these imports to work.

📋 App Setup

```
st.set_page_config(page_title="📈 Stock Price Forecast App")
st.title("📈 Stock Price Forecast App")
```

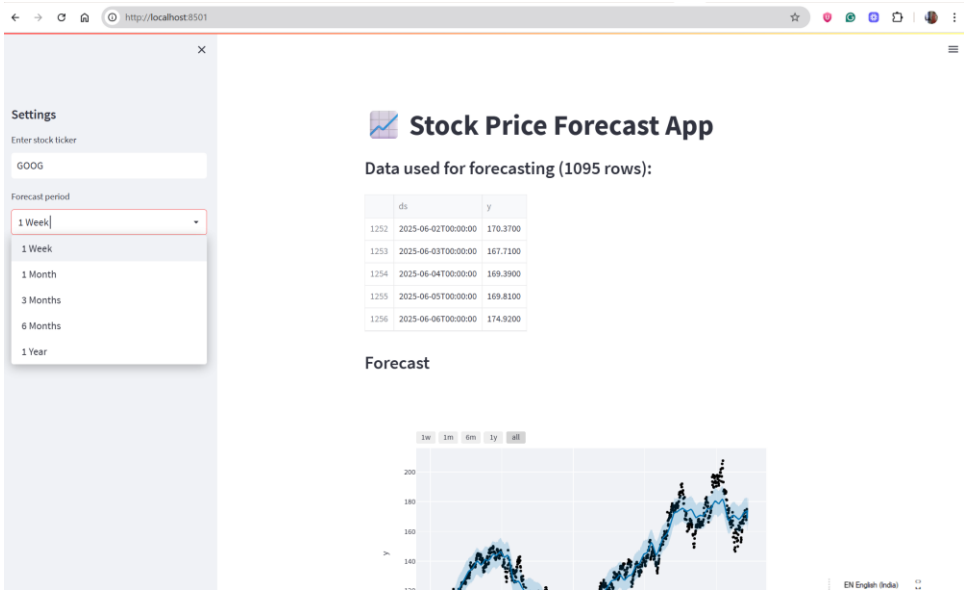
- **Purpose**: Set the page title and main heading in the Streamlit app.
- **User sees**: A clear page title and heading at the top of the web app.

🎛️ Sidebar: User Inputs

```
st.sidebar.header("Settings")
ticker = st.sidebar.text_input("Enter stock ticker", "AAPL").upper()

forecast_options = {
    "1 Week": 7,
    "1 Month": 30,
    "3 Months": 90,
    "6 Months": 180,
    "1 Year": 365
}
forecast_choice = st.sidebar.selectbox("Forecast period",
list(forecast_options.keys()))
period = forecast_options[forecast_choice]
```

- **Purpose**:
    - Allow users to choose the stock ticker and forecast length.

- **User sees**:
  - Input box for ticker (e.g. AAPL)
  - Dropdown for forecast period
- **Variables set**:
  - `ticker`: Stock symbol (converted to uppercase)
  - `period`: Number of days to forecast



"GOOG(Google)" stock - side bar to enter a valid stock and a dropdown box for selecting periods to forecast, and displaying raw data used for forecasting

💾 Data Loading with Caching

```
@st.cache(ttl=3600)
def load_data(ticker_symbol):
    try:
        data = yf.download(ticker_symbol, period="5y", auto_adjust=True)
        data.reset_index(inplace=True)
        return data
    except Exception as e:
        st.error(f"Error loading data: {e}")
        return None
```

```
df = load_data(ticker)
```

- **Purpose**:
  - Fetch 5 years of historical stock data using `yfinance`
  - Cache results for 1 hour to avoid re-downloading
- **Output**:
  - `df`: DataFrame with historical stock prices

📉 Initial Data Checks and Preprocessing

```
if df is not None and not df.empty:
    df = df.tail(365 * 3)  # Keep last 3 years
```

- **Purpose**:
  - Ensure data exists
  - Limit data to the **most recent 3 years** to reduce noise and improve performance

🔎 Finding and Preparing the 'Close' Price

```
close_col = None
 if isinstance(df.columns, pd.MultiIndex):
     df.columns = [' '.join(col).strip() for col in df.columns.values]
 for col in df.columns:
     if 'Close' in col:
         close_col = col
         break
```

- **Purpose**:
  - Dynamically find the column that contains closing prices
  - Ensures robustness even if the DataFrame column is multi-indexed
- **Output**:
  - `close_col`: Column name containing close price

🧠 Format Data for Prophet

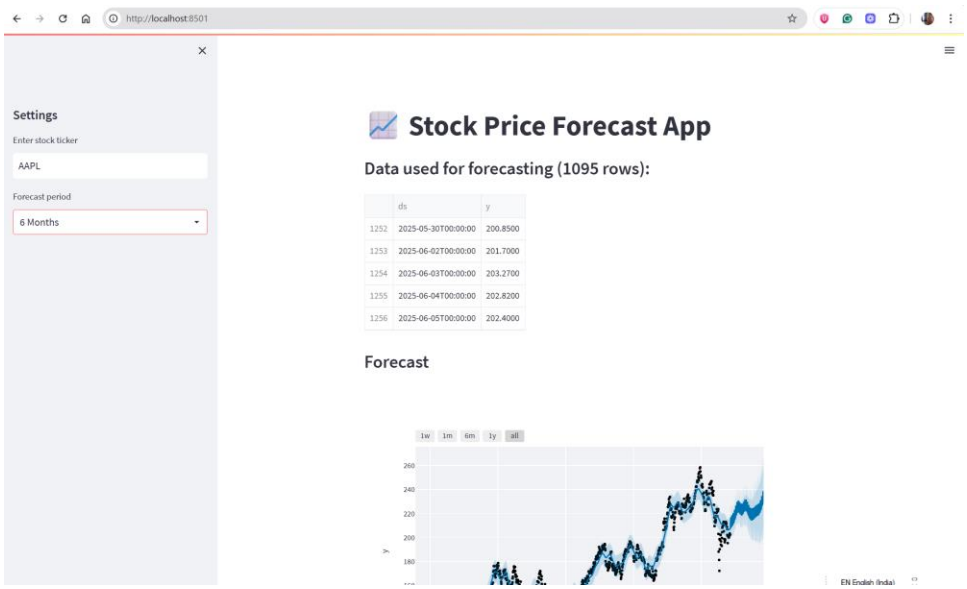
```
   if close_col is None:
        st.error("Close price column not found in data.")
    else:
        df_train = df[['Date', close_col]].rename(columns={'Date': 'ds',
close_col: 'y'})
        df_train['y'] = pd.to_numeric(df_train['y'], errors='coerce')
        df_train = df_train.dropna(subset=['y'])
```

- **Purpose**:
  - Rename columns to Prophet's required format (`ds` for date, `y` for target value)
  - Ensure numerical data and drop any invalid entries

🔍 Show Data Used for Forecasting

```
    st.subheader(f"Data used for forecasting ({len(df_train)} rows):")
     st.write(df_train.tail())
```

- **Purpose**: Let users see the tail (recent rows) of the cleaned data that Prophet will use.
- **Output**: Table with columns `ds` and `y`



“AAPL” Sample display of the Data used for forecasting

Forecasting with Prophet

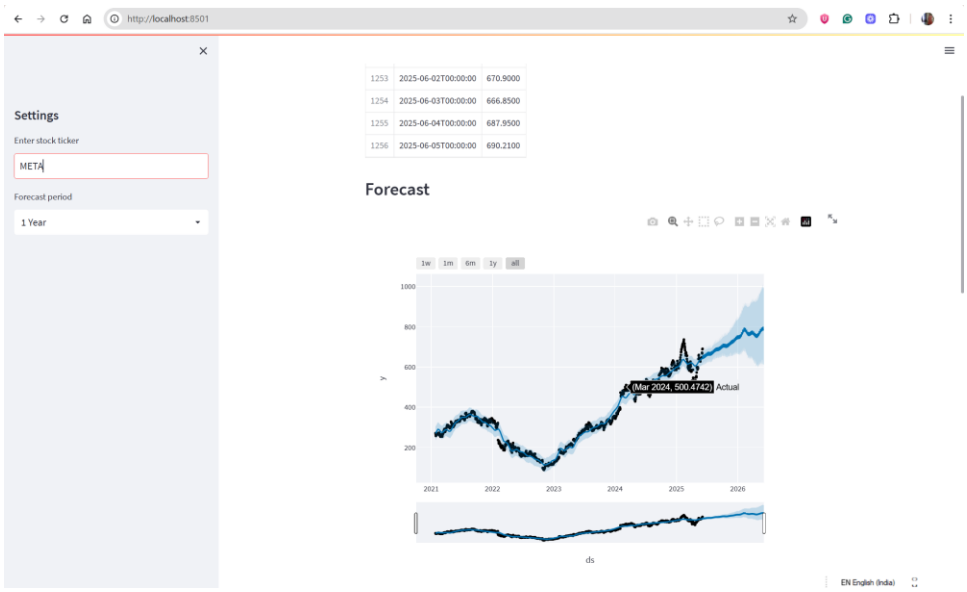
```
try:
    m = Prophet(daily_seasonality=False)
    m.fit(df_train)

    future = m.make_future_dataframe(periods=period)
    forecast = m.predict(future)
```

- **Purpose**:
  - Train Prophet model on historical data
  - Generate future dates
  - Predict future stock prices

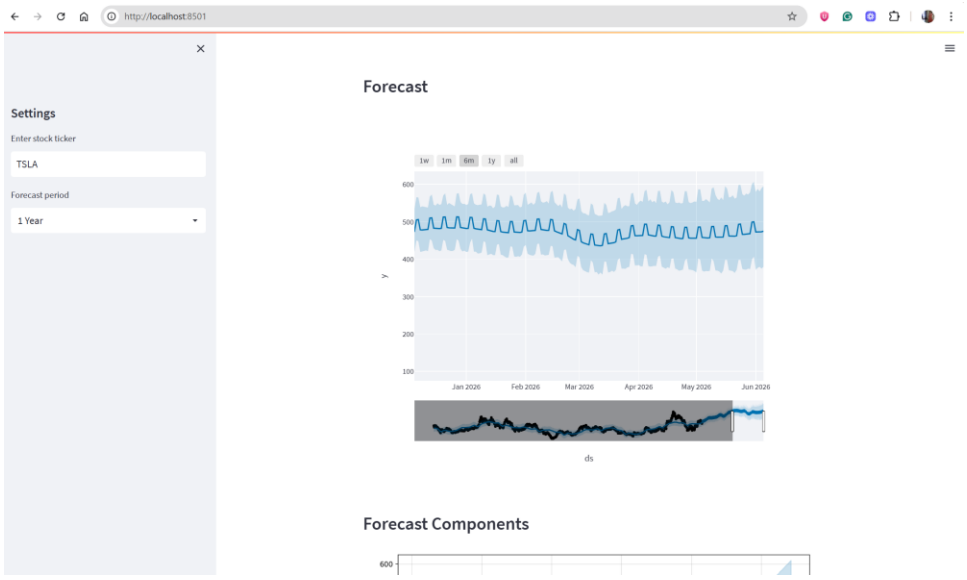
Forecast Plot

```
st.subheader("Forecast")
fig1 = plot_plotly(m, forecast)
st.plotly_chart(fig1, use_container_width=True)
```

- **Purpose**: Display an **interactive plot** with:
  - Historical data
  - Forecasted values
  - Confidence intervals



"META" interactive Forecast Plot showing both Historical and Forecasted values – all periods

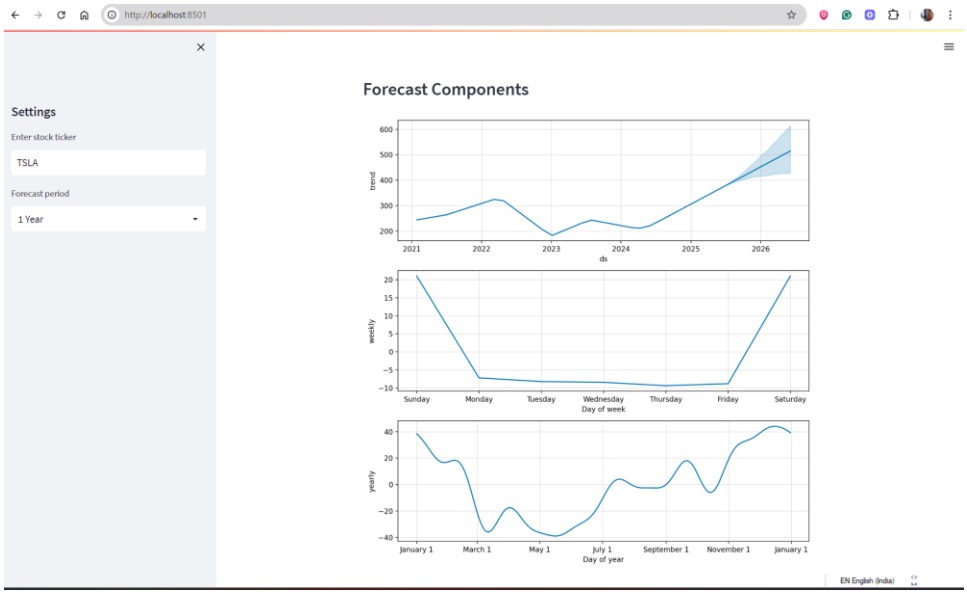


"TSLA" interactive Forecast Plot showing both Historical and Forecasted values – 6 months period

Forecast Components

```
st.subheader("Forecast Components")
 fig2 = m.plot_components(forecast)
 st.pyplot(fig2)
```

- **Purpose**:
  - Break down forecast into:
    - **Trend**
    - **Seasonal patterns** (weekly, yearly)
- **Output**: Static Matplotlib plot



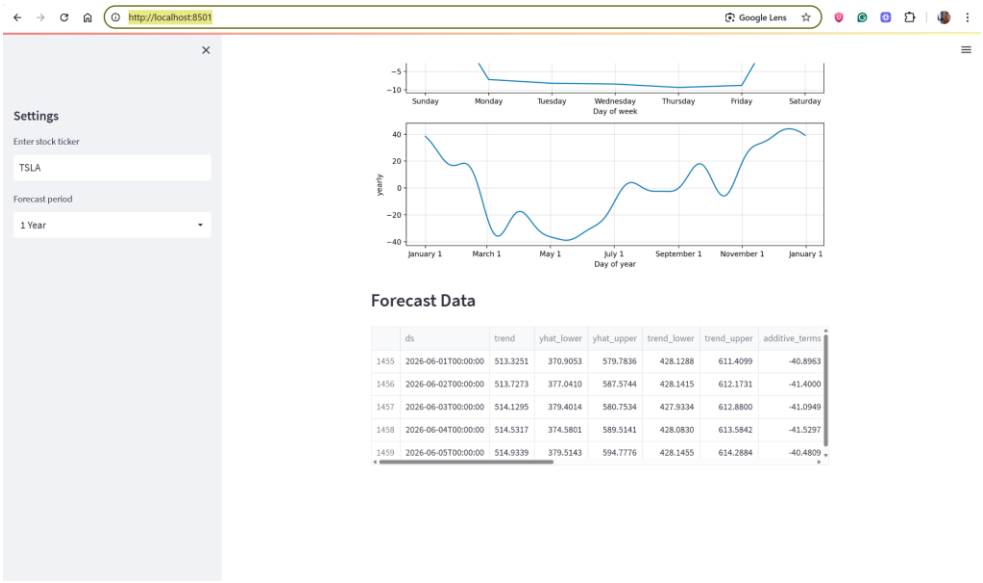
Forecast Components for "TSLA" showing Trend and seasonal patterns (weekly, yearly)

Raw Forecast Data Table

```
st.subheader("Forecast Data")
 st.write(forecast.tail())
```

- **Purpose**: Show the last few rows of the forecast table
- **Columns include**:
  - ds: Date

- `yhat`: Forecasted value
- `yhat_lower`, `yhat_upper`: Uncertainty intervals

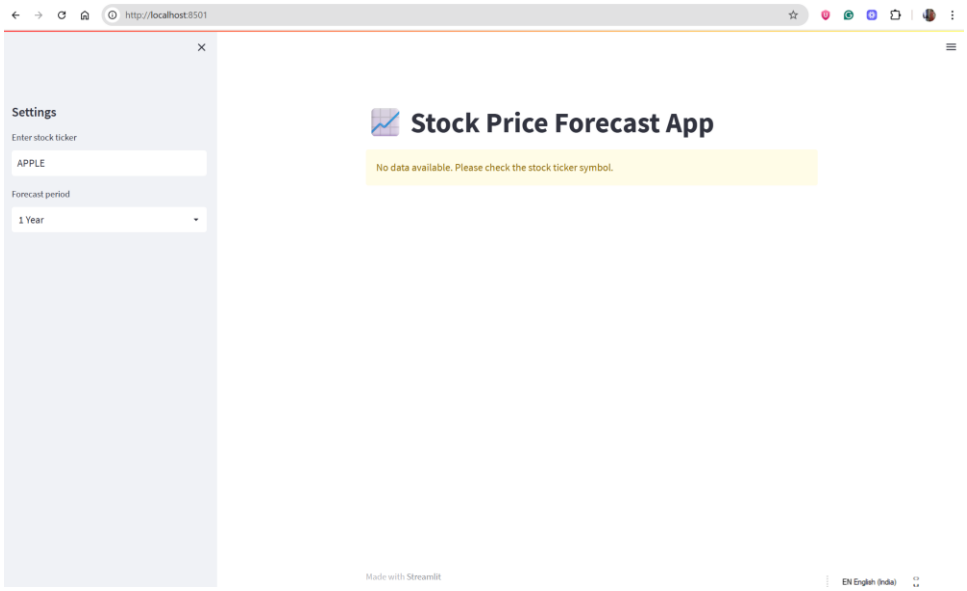


Forecast Data for "TSLA" showing the last few rows from forecast table

❌ Error Handling

```
        except Exception as e:
            st.error(f"Error during forecasting or plotting: {e}")
else:
    st.warning("No data available. Please check the stock ticker symbol.")
```

- **Purpose**:
  - Catch and display any errors from modeling or plotting
  - Warn users if no data was returned from Yahoo Finance



Error for unrecognized Stock ticker Symbol (APPLE instead of AAPL**) - "No data available. Please check the stock ticker symbol"**

▶ How to Run

You have several ways to run and deploy this application:

Follow these steps to get the app running on your own machine:

💻 1. Run Locally

1. **Clone the repository:**

```bash

git clone
[https://github.com/VenuYerramsetti/my_streamlit_project.git](https://github.com/VenuYerramsetti/my_streamlit_project.git)

cd my_streamlit_project
```

2. **Create a Python Virtual Environment (Highly Recommended):** This isolates your project dependencies.

```bash
python -m venv venv
```

3. **Activate the Virtual Environment:**
    a. **On Windows:**

```bash
.\venv\Scripts\activate
```

    b. **On macOS/Linux:**

```bash
source venv/bin/activate
```

4. **Install Required Packages:**

```bash
pip install -r requirements.txt
```

5. **Launch the Streamlit App:**

```bash
streamlit run streamlit_app.py
```

Your default web browser should automatically open to http://localhost:8501/ where you can interact with the app. 🎉

2. Run via Git (After Local Setup):

Once you've cloned the repository and set up your local environment as described in "1. Run Locally", running the app is as simple as:

Make sure your virtual environment is activated

streamlit run streamlit_app.py

🚀 Deploy to Streamlit Cloud

Streamlit Cloud offers a hassle-free way to deploy your app directly from your GitHub repository to a public URL.

Ensure Your Project is on GitHub:

This repository is already hosted on GitHub: https://github.com/VenuYerramsetti/my_streamlit_project.

Navigate to Streamlit Cloud:

Go to streamlit.io/cloud.

Sign In:

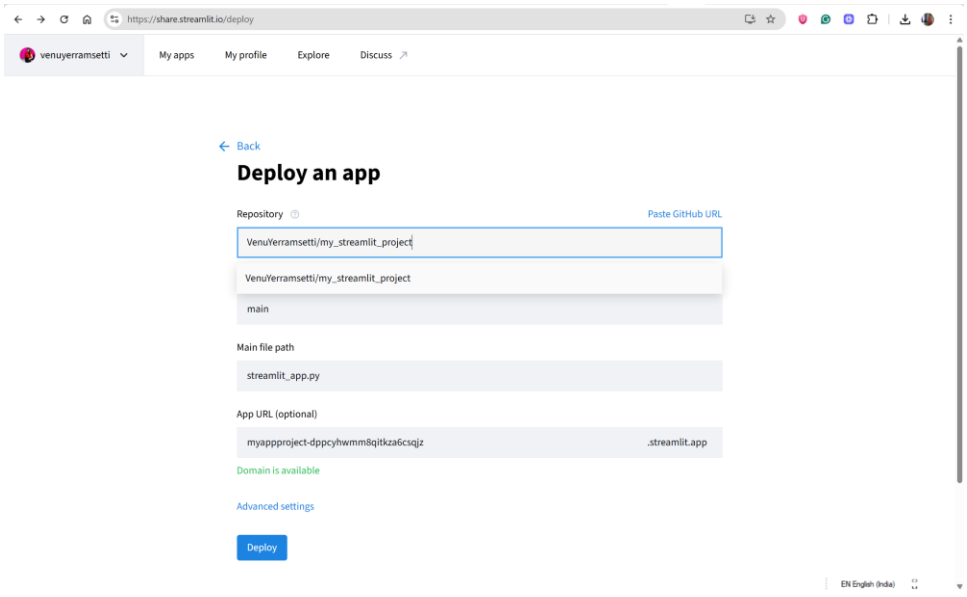
Sign in using your GitHub account.

Click "New app":

On your Streamlit Cloud dashboard, click the "New app" button.

Connect to Repository:

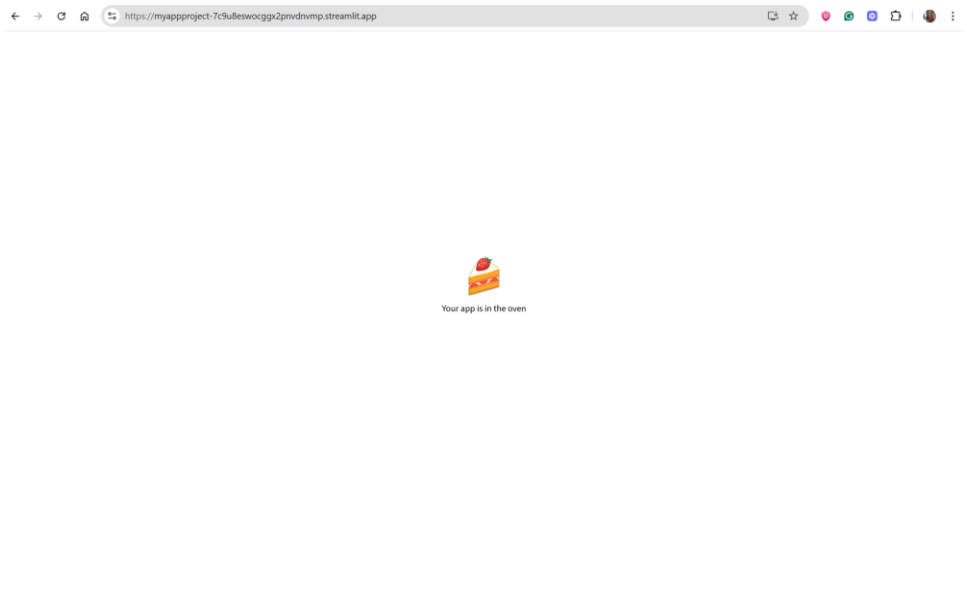
Under "Link to repository", select your GitHub repository: VenuYerramsetti/my_streamlit_project. For "Main file path", enter streamlit_app.py. Leave "Branch" as main (or master if applicable).



Deploying App in streamlit

Click "Deploy!":

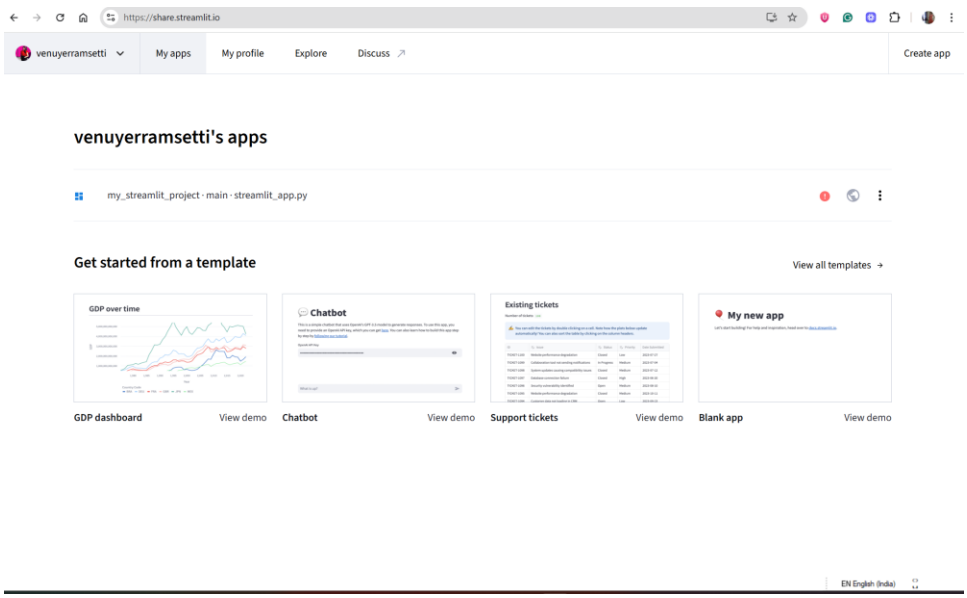
Streamlit Cloud will automatically handle dependency installation (from requirements.txt) and deploy your application. This usually takes just a few minutes. 🚀



Deployment in progress in streamlit

Access Your Live App:

Once deployed, you'll receive a unique public URL (e.g., https://myappproject-7c9u8eswocggx2pnvdnvmp.streamlit.app/) where your app will be live and accessible to anyone! ☑



My_streamlit_project app in streamlit under My apps

📌 What the User Should Expect

Output Element	Meaning
Line Chart (forecast)	Visual comparison between historical and forecasted prices
Trend Component	Long-term price movement (upward/downward)
Seasonality Components	Patterns repeating weekly or yearly
Raw Forecast Table	Dates with predicted prices and uncertainty bounds
Errors	Clear error messages if something goes wrong (e.g., invalid ticker)

⚠ Limitations to Keep in Mind

- Prophet assumes data is **daily and regular**.
- Works best with **smooth, continuous historical prices**.
- Forecasting may be less accurate for:

    - Extremely volatile stocks
    - Penny stocks or delisted companies
- Time zone mismatches or API limits from Yahoo Finance may occasionally cause failures.

☑ Best Used For

- Major stocks with sufficient history (e.g., `AAPL`, `GOOG`, `TSLA`)
- Exploring expected trends and potential future ranges
- Educational or preliminary investment planning (not financial advice)

📌 Summary of What the App Does

Block	Function
Sidebar	Collects ticker and forecast period
Data Load	Downloads 5y historical data
Preprocessing	Filters last 3y, detects close price
Prophet	Fits model, predicts future prices
Plot	Displays forecast + components
Table	Shows raw forecast data
Errors	Captures issues with ticker, data, or modeling