Stock Price Analysis & Comparison - Code Explanation

# 1. Importing Libraries

```python  
import streamlit as st  
import yfinance as yf  
import pandas as pd  
import matplotlib.pyplot as plt  
import mplfinance as mpf  
from datetime import datetime, timedelta  
from pymongo import MongoClient  
from pymongo.server\_api import ServerApi  
from dotenv import load\_dotenv  
import os  
import numpy as np  
from sklearn.linear\_model import LinearRegression  
```

- \*\*streamlit\*\*: For building the web-based user interface.  
- \*\*yfinance\*\*: To fetch stock market data.  
- \*\*pandas\*\*: For handling and manipulating data frames.  
- \*\*matplotlib & mplfinance\*\*: For plotting stock price charts.  
- \*\*datetime & timedelta\*\*: For date manipulations.  
- \*\*MongoClient & ServerApi\*\*: For connecting to MongoDB Atlas.  
- \*\*dotenv & os\*\*: To load environment variables.  
- \*\*numpy & sklearn\*\*: For numerical operations and stock price prediction.

# 2. Loading Environment Variables

```python  
load\_dotenv()  
```  
- Loads environment variables from a `.env` file, such as MongoDB connection URI.

# 3. MongoDB Connection

```python  
def connect\_to\_mongo():  
 uri = os.getenv("MONGO\_URI")  
 if not uri:  
 st.sidebar.error("⚠️ MongoDB URI not found in environment variables.")  
 return None  
 client = MongoClient(uri, server\_api=ServerApi('1'))  
 try:  
 client.admin.command('ping')  
 st.sidebar.success("✅ Connected to MongoDB Atlas")  
 return client  
 except Exception as e:  
 st.sidebar.error(f"❌ MongoDB connection failed: {e}")  
 return None  
```

- \*\*connect\_to\_mongo()\*\*: Function to connect to MongoDB Atlas using the URI from the `.env` file.  
- \*\*MongoClient\*\*: Creates a MongoDB client instance.  
- \*\*Ping\*\*: Verifies if the connection is successful.  
- \*\*Error handling\*\*: Displays success or failure messages in the sidebar.

# 4. User Registration

```python  
def store\_user\_data(client, name, email, phone, age):  
 if client:  
 db = client["UserDB"]  
 collection = db["users"]  
 user\_info = {  
 "name": name,  
 "email": email,  
 "phone": phone,  
 "age": age,  
 "registered\_at": datetime.now().strftime("%Y-%m-%d %H:%M:%S")  
 }  
 collection.insert\_one(user\_info)  
 st.sidebar.success("✅ User data saved to MongoDB")  
```

- \*\*store\_user\_data()\*\*: Function to save user details in MongoDB.  
- \*\*Database and collection\*\*: Creates a database `UserDB` and a collection `users`.  
- \*\*Insert user data\*\*: Adds the user's name, email, phone, age, and timestamp into the MongoDB collection.

# 5. Fetching Stock Data

```python  
def get\_stock\_data(ticker, start\_date, end\_date):  
 try:  
 stock = yf.Ticker(ticker)  
 data = stock.history(start=start\_date, end=end\_date)  
 return data, stock  
 except Exception as e:  
 st.sidebar.write(f"⚠️ Error fetching {ticker}: {e}")  
 return pd.DataFrame(), None  
```

- \*\*get\_stock\_data()\*\*: Fetches historical stock data using `yfinance`.  
- \*\*Ticker()\*\*: Retrieves stock data for the given symbol.  
- \*\*history()\*\*: Gets stock price data for the specified date range.

# 6. Stock Price Prediction

```python  
def predict\_future\_prices(data, days=30):  
 data['Date'] = data.index.map(lambda x: x.toordinal())  
 X = data['Date'].values.reshape(-1, 1)  
 y = data['Close'].values  
 model = LinearRegression()  
 model.fit(X, y)  
```

- \*\*predict\_future\_prices()\*\*: Predicts future stock prices using linear regression.  
- \*\*Date ordinal conversion\*\*: Converts dates into ordinal numbers for model compatibility.  
- \*\*Linear Regression model\*\*: Fits the model with historical data to predict future prices.

# 7. Plotting Functions

- \*\*plot\_line\_chart()\*\*: Plots historical stock prices.  
- \*\*plot\_moving\_averages()\*\*: Shows 50-day and 200-day moving averages.  
- \*\*plot\_candlestick\_chart()\*\*: Displays stock prices in candlestick format.  
- \*\*plot\_rsi()\*\*: Calculates and displays the Relative Strength Index (RSI).  
- \*\*plot\_macd()\*\*: Shows the Moving Average Convergence Divergence (MACD) indicator.

# 8. Exporting Stock Data to CSV

```python  
def export\_to\_csv(data, ticker):  
 csv = data.to\_csv(index=True)  
 st.download\_button(  
 label=f"Download {ticker} Stock Data as CSV",  
 data=csv,  
 file\_name=f"{ticker}\_stock\_data.csv",  
 mime="text/csv"  
 )  
```

- \*\*export\_to\_csv()\*\*: Exports stock data to a CSV file.  
- \*\*st.download\_button()\*\*: Allows users to download the data directly from the app.