PROJECT: Predict Apple Stock Prices Using Linear Regression

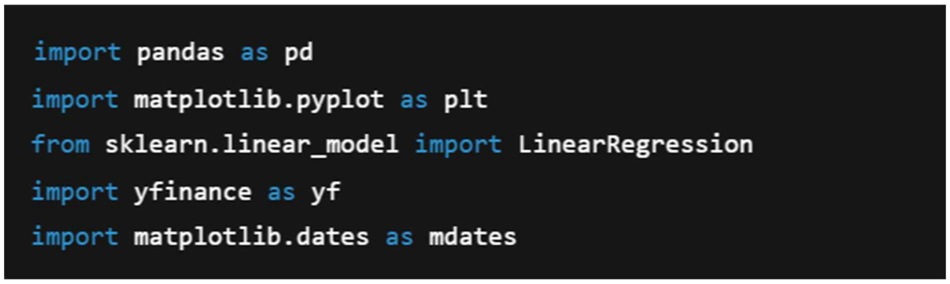
What Are We Trying to Do?

We want to:

- Download Apple (AAPL) stock price data (from 2010 to 2024)
- Use that data to train a **Linear Regression** model
- Predict the **next day's closing price** for any given date
- Compare predicted vs actual price (if available)
- Plot both actual and predicted prices on a graph

Step-by-Step Breakdown

Step 1: Import Required Libraries

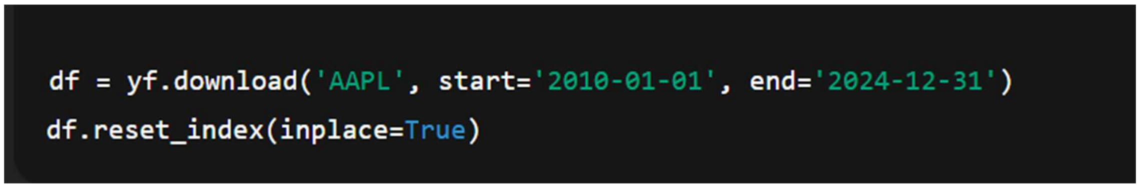


```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import yfinance as yf
import matplotlib.dates as mdates
```

What these do:

- pandas: For handling data tables (like Excel in Python)
- matplotlib.pyplot: For plotting graphs
- LinearRegression: The model that learns from data
- yfinance: Downloads real-time stock data from Yahoo Finance
- mdates: Helps format the x-axis (dates) in plots

Step 2: Download Apple Stock Data from 2010 to 2024

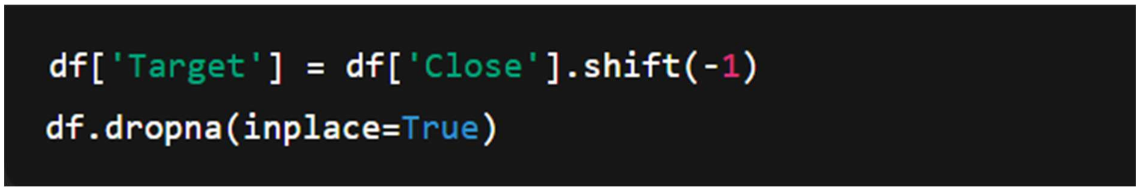


```python
df = yf.download('AAPL', start='2010-01-01', end='2024-12-31')
df.reset_index(inplace=True)
```

Explanation:

- yf.download() fetches daily stock data for Apple (AAPL) for the date range 2010 to 2024.
- The result is a DataFrame with columns like Date, Open, High, Low, Close, Volume, etc.
- reset_index() turns the Date (which was an index) into a normal column so we can access it easily.

🔷 Step 3: Create Target Column for Prediction

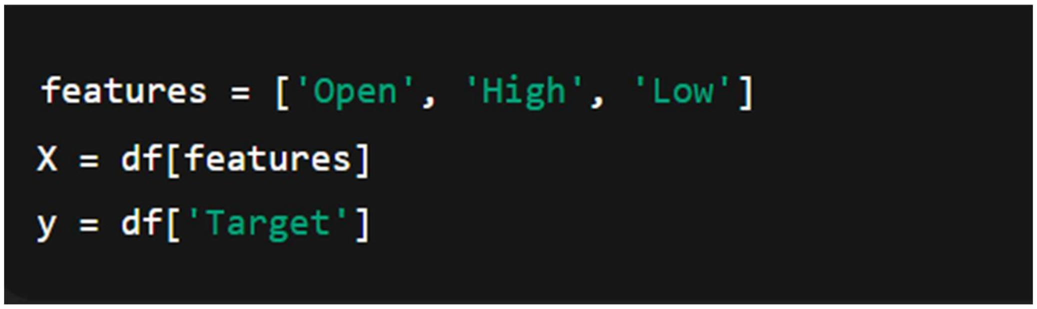


```python
df['Target'] = df['Close'].shift(-1)
df.dropna(inplace=True)
```

Explanation:

- We want to **predict the next day's closing price**, so we shift the Close column **up by 1 row**.
- This way, today's data (Open, High, Low) will be used to predict tomorrow's Close.
- The last row will now have NaN in the target column, so we remove it using dropna().

🔷 Step 4: Select Input Features and Target Variable



```python
features = ['Open', 'High', 'Low']
X = df[features]
y = df['Target']
```

Explanation:

- We define X as our input features: Open, High, and Low prices.
- y is our output — what we want to predict: the next day's Close price (stored in Target).
- We're building a relationship between these inputs and the target using Linear Regression.

🔷 Step 5: Train the Linear Regression Model

```
model = LinearRegression()
model.fit(X, y)
```

Explanation:

- We create a linear regression model.
- .fit(X, y) means we **train** the model: it tries to find the best line that fits the data to minimize prediction error.
- This step is where the machine learns how today's prices relate to tomorrow's closing price.

🔷 Step 6: Predict Closing Prices for All Data

```
df['Predicted_Close'] = model.predict(X)
```

Explanation:

- After training, we use the model to predict the closing price for every row in the dataset (i.e., every date).
- These predictions are stored in a new column called 'Predicted_Close', so we can compare them visually with the real Close prices.

🔷 Step 7: Take User Input

```
print("\nWelcome to the Stock Price Predictor!")
open_price = float(input("Enter the opening price: "))
high_price = float(input("Enter the high price: "))
low_price = float(input("Enter the low price: "))
input_date = input("Enter the date for prediction (YYYY-MM-DD): ")
input_date = pd.to_datetime(input_date)
```

Explanation:

- This section asks the user to enter:
    - That day's opening price
    - High price
    - Low price
    - Date for prediction
- The model will use these 3 prices to predict the **next day's close price**.
- The date is converted into datetime format so we can later find actual prices and plot them.

🔷 **Step 8: Make Prediction Based on User Input**

```
user_input = pd.DataFrame([[open_price, high_price, low_price]], columns=features)
user_predicted_price = model.predict(user_input)[0]
```

Explanation:

- We create a DataFrame from the user's input, matching the same structure as our training data.
- We then predict the next day's closing price using the trained model.
- .predict() returns an array, so we use [0] to get the single value.

🔷 **Step 9: Try to Find the Actual Price from Dataset**

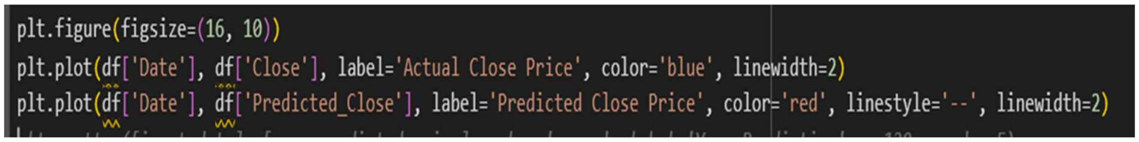
```
actual_price = None
date_match = df[df['Date'] == input_date]

if not date_match.empty:
 actual_price = float(date_match.iloc[0]['Close'])
```

Explanation:

- We try to find the **actual closing price** from the dataset for the date the user gave.
- If it exists, we store it in actual_price. Otherwise, we keep it as None.

🔷 Step 10: Plot the Data

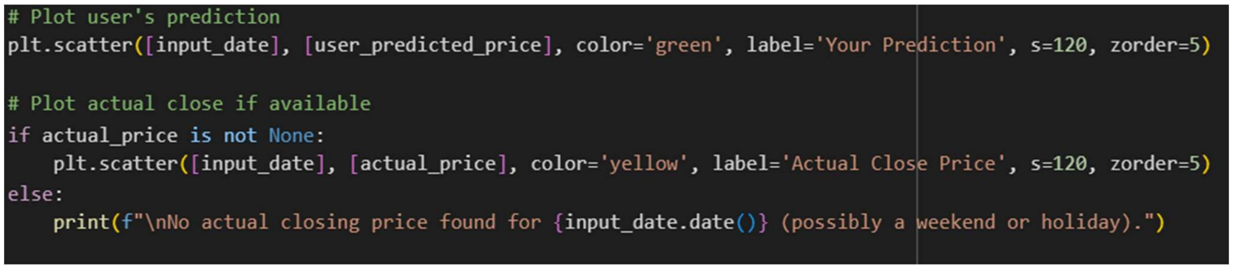


```
plt.figure(figsize=(16, 10))
plt.plot(df['Date'], df['Close'], label='Actual Close Price', color='blue', linewidth=2)
plt.plot(df['Date'], df['Predicted_Close'], label='Predicted Close Price', color='red', linestyle='--', linewidth=2)
```

Explanation:

- We create a large figure.
- First line (blue) shows the **real stock prices**.
- Second line (red, dashed) shows the **model's predicted prices** over time.

🔷 Add User Prediction and Actual to Plot



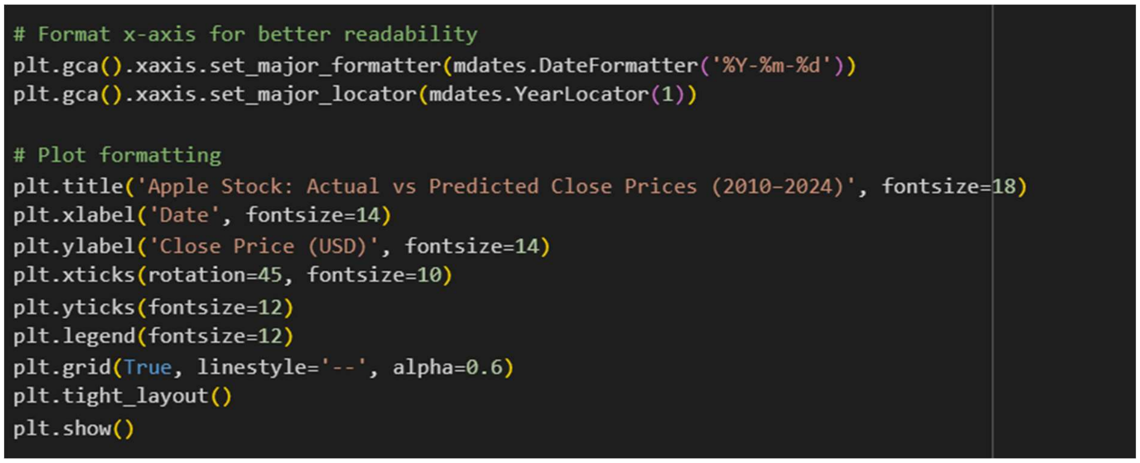
```
# Plot user's prediction
plt.scatter([input_date], [user_predicted_price], color='green', label='Your Prediction', s=120, zorder=5)

# Plot actual close if available
if actual_price is not None:
    plt.scatter([input_date], [actual_price], color='yellow', label='Actual Close Price', s=120, zorder=5)
else:
    print(f"\nNo actual closing price found for {input_date.date()} (possibly a weekend or holiday).")
```

Explanation:

- A green dot is placed at the predicted value for the date user entered.
- If the actual closing price is available, it shows a yellow dot too.
- If it's a weekend or holiday, there's no market data, so it skips it.

🔷 Final Plot Formatting



```
# Format x-axis for better readability
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.gca().xaxis.set_major_locator(mdates.YearLocator(1))

# Plot formatting
plt.title('Apple Stock: Actual vs Predicted Close Prices (2010-2024)', fontsize=18)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Close Price (USD)', fontsize=14)
plt.xticks(rotation=45, fontsize=10)
plt.yticks(fontsize=12)
plt.legend(fontsize=12)
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

Explanation:

- Formats the x-axis with date labels
- Adds title, labels, gridlines, and shows the final graph

🔷 Step 11: Display Prediction Results

```
print(f"\nPredicted closing price for {input_date.date()} is: ${round(user_predicted_price, 2)}")

if actual_price is not None:
 print(f"Actual closing price for that date: ${round(actual_price, 2)}")
else:
 print("Actual closing price not available for that date.")
```

Explanation:

- Prints the model’s predicted price
- If actual price is found, it prints that too
- Otherwise, it tells you it's not available (weekend or market closed)

✅ What I Did:

- I used **Linear Regression** to predict Apple’s **next-day closing stock price**.
- I collected data from **2010 to 2024** using yfinance.
- I trained the model using Open, High, and Low prices.
- I predicted prices and visualized both **actual vs predicted**.
- I also took user input to predict and compare for any specific date.

What I Learned:

- I learned how to apply **machine learning** on real financial data.
- I understood how to use **pandas**, **scikit-learn**, and **matplotlib**.
- I saw that Linear Regression works, but it has its limits.

⚠ Challenges Faced During the Project:

1. Limited Features Used: Only Open, High, and Low prices were included, ignoring other market influencers.
2. Overfitting Risk: Model was trained and tested on the same dataset without validation.
3. Model Simplicity: Linear Regression is too basic for the complex, non-linear nature of stock movements.

The End….