Stock Market Prediction - Notes

Step 1: Data Collection

```
# Install yfinance to collect the historical stock data
!pip install yfinance

import yfinance as yf
import pandas as pd
import warnings

# Suppress warnings
warnings.filterwarnings("ignore")

# Define the stock ticker and the time period
ticker = 'AAPL'  # You can change this to any stock ticker you're interested in. This ticker symbol is for Apple Inc.
start_date = '2018-01-01'
end_date = '2023-01-31'

# Download historical stock data
stock_data = yf.download(ticker, start=start_date, end=end_date)

# Save the data to a CSV file
stock_data.to_csv('historical_stock_data.csv')
print("Data collection complete. The data has been saved to 'historical_stock_data.csv'.")
```

Overall Explanation of This Code:

The provided code snippet is designed to collect historical stock data for Apple Inc. (AAPL) using the `yfinance` library. The script performs the following tasks:

1. **Installing `yfinance` :** Ensures that the `yfinance` library is installed for fetching stock data.
2. **Importing Libraries:** Imports necessary libraries, including `yfinance` for data collection, `pandas` for data manipulation, and `warnings` to suppress any warnings.
3. **Defining Parameters:** Specifies the stock ticker (AAPL) and the date range for the historical data (from January 1, 2018, to January 31, 2023).
4. **Downloading Data:** Uses `yfinance` to download the historical stock data for the specified ticker and date range.
5. **Saving Data:** Saves the downloaded data to a CSV file named `historical_stock_data.csv` .
6. **Confirmation Message:** Prints a message indicating the completion of the data collection process.

What Was the Need for This Step:

This step is essential for obtaining the historical stock data, which is the foundational dataset required for building the predictive model. Without this historical data, it would be impossible to analyze past trends, perform feature engineering, or train any predictive models.

What Was the Outcome for This Step:

The outcome of this step is a CSV file named `historical_stock_data.csv` containing the historical stock prices of Apple Inc. from January 1, 2018, to January 31, 2023. This dataset includes crucial information such as opening, closing, high, low prices, volume, and adjusted closing prices for each trading day within the specified period.

How Would It Help in Anything That Would Come in Next Steps:

1. **Data Preprocessing:** The historical stock data collected will need to be cleaned and preprocessed to handle any missing values, format inconsistencies, or irrelevant columns. This step ensures the data is in a suitable format for analysis and modeling.
2. **Exploratory Data Analysis (EDA):** With the historical data in hand, EDA can be performed to visualize and understand the underlying patterns, trends, and anomalies in the stock prices. This helps in gaining insights and identifying potential features for the model.
3. **Feature Engineering and Selection:** The historical data will be used to create and select relevant features that could influence future stock prices. This may include technical indicators, rolling averages, and other derived metrics.
4. **Model Selection, Training, and Prediction:** The historical stock data will serve as the training dataset for various predictive models. The models will learn from past patterns and trends in this data to forecast future stock prices.
5. **Prediction and Visualization:** Finally, the preprocessed and feature-engineered data will be used to make predictions. The accuracy and effectiveness of these predictions will be evaluated and visualized to understand the model's performance.

In summary, collecting historical stock data is the critical first step that sets the stage for the subsequent steps in building and evaluating the predictive model.

In [ ]:

Step 2: Data Preprocessing

Overall Explanation of This Code:

This code snippet is designed for preprocessing the historical stock data collected in step 1. The preprocessing involves loading the data, handling missing values, creating new features through feature engineering, and preparing the data for modeling. Each step ensures the dataset is clean, informative, and ready for the next stages of the predictive modeling process.

Detailed Step-by-Step Explanation:

1. **Importing Libraries:**

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
```

   - **Need:** To leverage numpy for numerical operations, pandas for data manipulation, and sklearn for data scaling and splitting.
   - **Outcome:** Libraries are ready for use in the preprocessing steps.

2. **Loading the Data:**

```
stock_data = pd.read_csv('historical_stock_data.csv')
```

   - **Need:** To load the collected historical stock data from the CSV file into a pandas DataFrame.
   - **Outcome:** Data is loaded into `stock_data`.

3. **Checking for Missing Values:**

```
missing_values = stock_data.isnull().sum()
print("Missing values:\n", missing_values)
```

   - **Need:** To identify any missing values in the dataset.
   - **Outcome:** Printed summary of missing values, helping to understand the data's completeness.

4. **Converting 'Date' Column to Datetime:**

```
stock_data['Date'] = pd.to_datetime(stock_data['Date'])
```

   - **Need:** To ensure the 'Date' column is in datetime format for time series analysis.
   - **Outcome:** 'Date' column is now in the correct format.

5. **Feature Engineering - Adding Moving Averages:**

```
stock_data['MA_50'] = stock_data['Close'].rolling(window=50).mean()
stock_data['MA_200'] = stock_data['Close'].rolling(window=200).mean()
```

   - **Need:** To introduce moving averages (50-day and 200-day) as features which can help capture trends and smooth out price data.
   - **Outcome:** Two new columns, `MA_50` and `MA_200`, are added to the DataFrame.

6. **Creating Lag Features:**

```
stock_data['Close_Lag1'] = stock_data['Close'].shift(1)
stock_data['Close_Lag2'] = stock_data['Close'].shift(2)
stock_data['Close_Lag3'] = stock_data['Close'].shift(3)
```

   - **Need:** To include past values of the closing price as features to help the model understand past trends.
   - **Outcome:** Three new columns (`Close_Lag1`, `Close_Lag2`, `Close_Lag3`) are added with lagged closing prices.

7. **Calculating Relative Strength Index (RSI):**

```
def calculate_RSI(data, window=14):
    delta = data['Close'].diff()
    gain = (delta.where(delta > 0, 0)).fillna(0)
    loss = (-delta.where(delta < 0, 0)).fillna(0)
    avg_gain = gain.rolling(window=window).mean()
    avg_loss = loss.rolling(window=window).mean()
    rs = avg_gain / avg_loss
    rsi = 100 - (100 / (1 + rs))
    return rsi

stock_data['RSI'] = calculate_RSI(stock_data)
```

   - **Need:** To calculate the RSI, which is a momentum indicator that measures the speed and change of price movements.
   - **Outcome:** A new column, `RSI`, is added to the DataFrame.

8. **Dropping Initial Rows with NaN Values:**

```
stock_data.dropna(inplace=True)
```

- **Need:** To remove rows with NaN values resulting from moving average, lag, and RSI calculations.
- **Outcome:** Cleaned DataFrame with no missing values, ensuring all features are fully populated.

What Was the Need for This Step (Overall):

This step is crucial for transforming the raw historical stock data into a clean and feature-rich dataset. Data preprocessing helps in:

1. Handling missing values to avoid issues during modeling.
2. Converting date columns to the correct format for time series analysis.
3. Creating new informative features (e.g., moving averages, lagged prices, RSI) that can improve the model's predictive power.
4. Ensuring the dataset is ready for the next stages, including model training and evaluation.

What Was the Outcome for This Step:

The outcome of this step is a preprocessed DataFrame that includes the original historical stock data along with additional features such as moving averages, lagged closing prices, and RSI. All missing values have been handled, and the data is now clean and ready for further analysis and modeling.

How Would It Help in Anything That Would Come in Next Steps:

1. **Exploratory Data Analysis (EDA):** The newly created features and cleaned data will provide richer insights during EDA, helping to visualize and understand the relationships between different variables.
2. **Feature Engineering and Selection:** The features engineered in this step (moving averages, lags, RSI) will be essential inputs for the predictive models. Additional feature selection techniques can be applied to further refine the feature set.
3. **Model Selection, Training, and Prediction:** The clean and feature-rich dataset will improve the quality of model training, helping the models to learn more effectively and make accurate predictions.
4. **Prediction and Visualization:** The enhanced dataset will contribute to more accurate and reliable predictions, which can then be visualized to demonstrate the model's effectiveness in forecasting future stock prices.

In summary, data preprocessing is a vital step that transforms raw data into a structured and feature-enriched format, laying the foundation for successful modeling and accurate predictions.

In [ ]:

Step 3: Exploratory Data Analysis (EDA)

Overall Explanation of This Code:

This code snippet performs Exploratory Data Analysis (EDA) on the historical stock data for Apple Inc. (AAPL). It includes visualizations of daily and monthly closing prices, trading volume, correlation heatmap, and time series decomposition. EDA helps in understanding the data better, identifying patterns, and gaining insights that are crucial for building predictive models.

Detailed Step-by-Step Explanation:

1. **Importing Libraries:**

```
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.seasonal import seasonal_decompose
```

   - **Need:** To utilize `matplotlib` and `seaborn` for visualizations, and `statsmodels` for time series decomposition.
   - **Outcome:** Libraries are ready for creating various plots and decompositions.
2. **Plotting Daily Historical Closing Prices:**

```
plt.figure(figsize=(14, 7))
plt.plot(stock_data['Date'], stock_data['Close'], marker='o', linestyle='-', color='b',
label='Daily Closing Price')
plt.title('Historical Closing Prices of AAPL')
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.savefig('closing_prices_daily.png')
plt.show()
```

   - **Need:** To visualize the daily closing prices to understand the overall trend and daily price movements.
   - **Outcome:** A plot showing daily closing prices of AAPL, saved as `closing_prices_daily.png`.
3. **Aggregating to Monthly Data for Clearer Trend Analysis:**

```
stock_data['Date'] = pd.to_datetime(stock_data['Date'])
stock_data.set_index('Date', inplace=True)
stock_data_monthly = stock_data.resample('M').mean().reset_index()

plt.figure(figsize=(14, 7))
plt.plot(stock_data_monthly['Date'], stock_data_monthly['Close'], marker='o', linestyle='-',
color='b', label='Monthly Avg Closing Price')
plt.title('Monthly Average Closing Prices of AAPL')
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.savefig('closing_prices_monthly.png')
plt.show()
```

- **Need:** To visualize monthly average closing prices to identify longer-term trends more clearly.
- **Outcome:** A plot showing monthly average closing prices of AAPL, saved as `closing_prices_monthly.png`.

4. **Plotting Trading Volume:**

```
plt.figure(figsize=(14, 7))
plt.plot(stock_data.index, stock_data['Volume'], marker='o', linestyle='-', color='g',
label='Trading Volume')
plt.title('Trading Volume of AAPL')
plt.xlabel('Date')
plt.ylabel('Volume')
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.savefig('trading_volume.png')
plt.show()
```

- **Need:** To visualize the trading volume to understand the activity level and possible liquidity issues.
- **Outcome:** A plot showing trading volume of AAPL, saved as `trading_volume.png`.

5. **Correlation Heatmap:**

```
plt.figure(figsize=(10, 8))
sns.heatmap(stock_data.corr(), annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Heatmap')
plt.tight_layout()
plt.savefig('correlation_heatmap.png')
plt.show()
```

- **Need:** To identify correlations between different features in the dataset, which can inform feature selection for modeling.
- **Outcome:** A heatmap showing correlations between features, saved as `correlation_heatmap.png`.

6. **Time Series Decomposition:**

```
result = seasonal_decompose(stock_data['Close'], model='multiplicative', period=90)
result.plot()
plt.tight_layout()
plt.savefig('time_series_decomposition.png')
plt.show()
```

- **Need:** To decompose the time series into trend, seasonal, and residual components to understand underlying patterns.
- **Outcome:** Plots showing decomposed time series components, saved as `time_series_decomposition.png`.

What Was the Need for This Step (Overall):

This step is essential for visualizing and understanding the data, identifying trends, seasonality, and relationships between different features. EDA helps in:

1. Gaining insights into the data.
2. Identifying patterns and anomalies.
3. Informing feature engineering and selection.
4. Preparing for model building by understanding data distributions and relationships.

What Was the Outcome for This Step:

The outcome of this step includes various visualizations:

1. Daily closing prices.
2. Monthly average closing prices.
3. Trading volume.
4. Correlation heatmap.
5. Decomposed time series components.

These visualizations provide a comprehensive understanding of the historical stock data, revealing trends, seasonality, correlations, and

underlying patterns.

How Would It Help in Anything That Would Come in Next Steps:

1. **Feature Engineering and Selection:** Insights from EDA help in creating and selecting the most relevant features for the model. For example, identified correlations can guide which features to include or exclude.
2. **Model Selection, Training, and Prediction:** Understanding the data distributions, trends, and seasonality helps in choosing the right models and parameters, leading to better model performance.
3. **Prediction and Visualization:** EDA findings can be used to validate the predictions made by the model. For instance, if the model's predictions align with the observed seasonal patterns, it indicates good model performance.

In summary, EDA is a critical step that provides valuable insights into the data, guiding subsequent steps in the predictive modeling process and ensuring the development of robust and accurate models.

In [ ]:

Step 4: Feature Engineering and Selection:

This code snippet focuses on feature engineering and selection. It involves defining the features and target for the model, and normalizing the features using `StandardScaler`. These steps are critical for preparing the data before feeding it into a predictive model.

Detailed Step-by-Step Explanation:

1. **Importing Libraries:**

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

   - **Need:** To utilize `pandas` for data manipulation, `numpy` for numerical operations, `StandardScaler` for feature scaling, and `train_test_split` for splitting the data into training and testing sets.
   - **Outcome:** Libraries are ready to be used for feature engineering and selection.

2. **Defining Features (X) and Target (y):**

```
features = ['Open', 'High', 'Low', 'Close_Lag1', 'Close_Lag2', 'Close_Lag3', 'Volume', 'MA_50',
'MA_200', 'RSI']
target = 'Close'
X = stock_data[features]
y = stock_data[target]
```

   - **Need:** To specify which columns of the dataset will be used as features (independent variables) and which column will be the target (dependent variable). This is crucial for training the predictive model.
   - **Outcome:** Variables `X` and `y` are defined, where `X` contains the features and `y` contains the target.

3. **Normalizing/Scaling the Features:**

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

   - **Need:** To scale the features so that they have a mean of 0 and a standard deviation of 1. This is important because many machine learning algorithms perform better when the input features are on a similar scale.
   - **Outcome:** The features in `X` are normalized, and the scaled features are stored in `X_scaled`.

What Was the Need for This Step (Overall):

This step is essential for preparing the data before it is fed into the predictive model. It ensures that the model receives appropriately scaled features, which can lead to better model performance and faster convergence during training. Additionally, defining the features and target explicitly helps in structuring the dataset correctly for the machine learning algorithms.

What Was the Outcome for This Step:

The outcome of this step is a set of scaled features (`X_scaled`) and a target variable (`y`). The features have been standardized to have a mean of 0 and a standard deviation of 1, which is a common preprocessing step to improve model performance.

How Would It Help in Anything That Would Come in Next Steps:

1. **Model Selection, Training, and Prediction:** The scaled features are crucial for training various machine learning models. Algorithms like linear regression, support vector machines, and neural networks can perform poorly if the features are not scaled. The defined features and target ensure that the model learns from the relevant data.

2. **Model Evaluation:** Having standardized features helps in fair evaluation of model performance, as the influence of features with larger numerical ranges is mitigated.

3. **Prediction and Visualization:** Scaled features ensure that the predictions made by the model are consistent and reliable. It also simplifies the process of visualizing the relationships between features and the target variable.

In summary, feature engineering and selection, along with normalization, are vital steps that prepare the dataset for effective model training and accurate predictions. This step ensures that the data fed into the model is clean, structured, and scaled, leading to better performance and more reliable outcomes.

In [ ]:

Step 5: Model Selection, Training and prediction

Overall Explanation of This Code:

This code snippet involves the model selection, training, and prediction stages. It demonstrates the process of splitting the data into training and testing sets, training a Linear Regression model, saving the trained model, and making predictions on the test data. The performance metrics of the model are also printed.

Detailed Step-by-Step Explanation:

1. **Importing Libraries:**

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import joblib
import warnings
warnings.filterwarnings('ignore')  # Suppress warnings
```

   - **Need:** To import necessary libraries for model training, evaluation, and saving. Warnings are suppressed for a cleaner output.
   - **Outcome:** Libraries are ready to be used for model training, prediction, and evaluation.

2. **Splitting the Data:**

```
train_size = int(len(X_scaled) * 0.8)
X_train, X_test = X_scaled[:train_size], X_scaled[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

   - **Need:** To split the data into training and testing sets. The training set will be 80% of the data, and the testing set will be 20%, ensuring the test set includes data from January 2023.
   - **Outcome:** Data is split into `X_train`, `X_test`, `y_train`, and `y_test`.

3. **Training the Model:**

```
final_model_lr = LinearRegression()
final_model_lr.fit(X_train, y_train)
```

   - **Need:** To train a Linear Regression model using the training data.
   - **Outcome:** A Linear Regression model is trained and stored in `final_model_lr`.

4. **Saving the Model:**

```
joblib.dump(final_model_lr, 'final_linear_regression_model.pkl')
```

   - **Need:** To save the trained model for later use, avoiding the need to retrain the model every time.
   - **Outcome:** The trained Linear Regression model is saved as `final_linear_regression_model.pkl`.

5. **Making Predictions:**

```
y_pred = final_model_lr.predict(X_test)
```

   - **Need:** To make predictions on the test data using the trained Linear Regression model.
   - **Outcome:** Predictions for the test set are stored in `y_pred`.

6. **Printing Results:**

```
for model_name, metrics in results.items():
    print(f"Model: {model_name}")
    for metric_name, metric_value in metrics.items():
        print(f"{metric_name}: {metric_value}")
    print("\n")
```

   - **Need:** To print the performance metrics of the trained models. However, it seems the `results` dictionary containing model performance metrics is not defined or computed in the given code snippet.
   - **Outcome:** This step would print the performance metrics if the `results` dictionary were properly defined.

What Was the Need for This Step (Overall):

This step is essential for developing a predictive model capable of forecasting the closing prices of Apple's stock. It includes training the model, saving it for future use, and making predictions on unseen data. This process is fundamental in machine learning projects where the goal is to create a model that generalizes well to new data.

What Was the Outcome for This Step:

The outcome of this step includes:

1. A trained Linear Regression model.
2. Predictions on the test set.
3. The trained model saved to a file (`final_linear_regression_model.pkl`).

How Would It Help in Anything That Would Come in Next Steps:

1. **Model Evaluation:** Although the performance metrics are not explicitly calculated in this snippet, the predictions made in this step can be evaluated using metrics such as mean squared error, mean absolute error, and r-squared score. This helps in understanding how well the model performs.

2. **Model Tuning and Improvement:** The initial model's performance can be assessed, and if necessary, further tuning or trying different models (like RandomForestRegressor or SVR) can be performed to improve accuracy.

3. **Prediction and Visualization:** The saved model can be loaded in the future to make new predictions, and the results can be visualized to compare predicted vs. actual stock prices.

In summary, model selection, training, and prediction are crucial steps in building a predictive model. This step ensures that the model is trained, saved, and ready to make predictions, which can then be evaluated and refined to improve its performance.

In [ ]:

Step 6: Final Evaluation and Visualization

Overall Explanation of This Code:

This code snippet is focused on the final evaluation and visualization of the model's performance, specifically for the month of January 2023. It aligns the actual and predicted stock prices, filters the relevant data for January 2023, and visualizes the comparison between actual and predicted closing prices.

Detailed Step-by-Step Explanation:

1. **Resetting the Index to Make 'Date' a Column:**

   ```
   stock_data_reset = stock_data.reset_index()
   ```

   - **Need:** To ensure that the 'Date' column is available for slicing and filtering.
   - **Outcome:** The DataFrame `stock_data_reset` has 'Date' as a column instead of the index.
2. **Ensuring the 'Date' Column is in Datetime Format:**

   ```
   stock_data_reset['Date'] = pd.to_datetime(stock_data_reset['Date'])
   ```

   - **Need:** To ensure the 'Date' column is in a datetime format, which is necessary for accurate date filtering and slicing.
   - **Outcome:** The 'Date' column is converted to datetime format.
3. **Slicing the Data from `train_size` Onwards:**

   ```
   test_slice = stock_data_reset.iloc[train_size:]
   ```

   - **Need:** To obtain the test data slice for further processing.
   - **Outcome:** The DataFrame `test_slice` contains data from the testing period.
4. **Extracting and Filtering the Relevant Dates:**

   ```
   test_dates = test_slice['Date'].reset_index(drop=True)
   january_2023_dates = test_dates[(test_dates >= '2023-01-02') & (test_dates <= '2023-01-31')]
   ```

   - **Need:** To filter out the dates corresponding to January 2023.
   - **Outcome:** `january_2023_dates` contains the dates from January 2nd to January 31st, 2023.
5. **Aligning Indices for January 2023:**

   ```
   january_indices = test_dates[(test_dates >= '2023-01-02') & (test_dates <= '2023-01-31')].index
   ```

   - **Need:** To obtain the indices of the dates for January 2023 for further slicing.
   - **Outcome:** `january_indices` contains the indices corresponding to January 2023 dates.
6. **Filtering `y_test` and `y_pred` for January 2023 Using Aligned Indices:**

   ```
   y_test_january_2023 = y_test.iloc[january_indices]
   y_pred_january_2023 = y_pred[january_indices]
   ```

- **Need:** To isolate the actual and predicted values for January 2023.
- **Outcome:** `y_test_january_2023` and `y_pred_january_2023` contain the actual and predicted closing prices for January 2023, respectively.

7. **Plotting Actual vs. Predicted Prices for January 2nd to January 31st, 2023:**

```
plt.figure(figsize=(14, 7))
plt.plot(january_2023_dates, y_test_january_2023, marker='o', linestyle='-', color='b',
label='Actual Price')
plt.plot(january_2023_dates, y_pred_january_2023, marker='o', linestyle='-', color='r',
label='Predicted Price')
plt.title('Actual vs Predicted Stock Prices (January 2nd - January 31st, 2023)')
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.savefig('actual_vs_predicted_prices_january_2023.png')  # Save the plot
plt.show()
```

- **Need:** To visualize the comparison between actual and predicted closing prices for January 2023. Visualization aids in understanding the model's performance visually.
- **Outcome:** A plot comparing actual and predicted closing prices is generated and saved as `actual_vs_predicted_prices_january_2023.png`.

What Was the Need for This Step (Overall):

This step is crucial for the final evaluation of the model's performance. By comparing actual and predicted stock prices for a specific period (January 2023), we can visually and quantitatively assess the accuracy of the model. This step provides insights into how well the model generalizes to new, unseen data and highlights any discrepancies.

What Was the Outcome for This Step:

The outcome of this step includes:

1. Extracted and aligned actual and predicted stock prices for January 2023.
2. A visual plot comparing the actual and predicted closing prices for the specified period.
3. The plot saved as an image file for documentation and reporting purposes.

How Would It Help in Anything That Would Come in Next Steps:

1. **Model Evaluation and Tuning:** The visualization and comparison help identify if the model needs further tuning or if different algorithms should be explored for better performance.
2. **Reporting and Presentation:** The visual plots and performance metrics are useful for communicating the model's effectiveness to stakeholders or in reports.
3. **Future Predictions:** Understanding the model's performance for a specific period helps in making informed decisions about its reliability for future predictions.

In summary, the final evaluation and visualization step are essential for assessing the model's accuracy and reliability. It provides a clear and visual representation of the model's performance, which is valuable for further tuning, reporting, and making future predictions.