Yahoo_Finance_Stock_Market_Analysis

February 5, 2025

1 Overview

Stock market prediction is one of financial analysis's most challenging and exciting tasks. Multiple factors, including market trends, economic conditions, investor sentiment, and company performance influence stock prices. In this notebook, we will analyze the stock market data of four major technology companies:

- Apple Inc. (AAPL)
- Microsoft Corporation (MSFT)
- Amazon.com Inc. (AMZN)
- Tesla Inc. (TSLA)

We will then focus on forecasting Apple's future stock prices using advanced machine-learning techniques. The dataset will be sourced from Yahoo Finance, and the workflow includes exploratory data analysis (EDA), feature extraction, data preprocessing, and model implementation. We will develop Long Short-Term Memory (LSTM) and regression models to predict stock prices effectively.

2 Essential Libraries

For this project, we utilize a range of Python libraries categorized based on their functionalities:

- **Data Handling and Statistical Analysis:** Pandas, NumPy, SciPy, pandas_datareader
- **Data Visualization:** Matplotlib, Seaborn, Plotly (express, graph_objects, subplots)
- **Preprocessing and Scaling:** Scikit-learn (MinMaxScaler, preprocessing, train_test_split)
- **Performance Metrics:** Mean Absolute Error (MAE), Root Mean Squared Error (RMSE)
- **Machine Learning Models:** TensorFlow/Keras (Sequential, Conv1D, Bidirectional LSTM, Dense, Adam, Huber), Scikit-learn (Linear Regression)
- **Financial Analysis and Optimization:** PyPortfolioOpt (Efficient Frontier, Risk Models, Expected Returns, Black-Litterman Model)
- **Stock Market Data:** Yahoo Finance, TA-Lib, QuantStats
- **Time and System Utilities:** Datetime, Pytz, OS, Warnings, Tabulate

These libraries enable efficient data processing, model training, evaluation, visualization, and financial analysis for stock market predictions.Here is the information on how to load libraries and their functions.

2.1 yfinance

`yfinance` is a widely used Python library for retrieving financial market data from Yahoo! Finance. It provides a simple API to access historical stock prices, indices, ETFs, cryptocurrencies, and forex. Additionally, it offers tools for dividend/split adjustments and basic data visualization, making it an essential tool for traders and investors.

[1]:
```
!pip install yfinance
```

2.2 QuantStats

`QuantStats` is a Python library designed for quantitative financial analysis. It provides functions for portfolio performance evaluation, risk analysis, and financial visualization. Its ability to generate comprehensive performance reports makes it a valuable resource for finance professionals.

[2]:
```
!pip install quantstats
```

2.3 PyPortfolioOpt

`PyPortfolioOpt` is a powerful library for portfolio optimization and asset allocation. It supports mean-variance optimization, Black-Litterman models, and custom constraints, allowing users to construct efficient portfolios based on risk-return trade-offs.

[3]:
```
!pip install PyPortfolioOpt
```

2.4 TA (Technical Analysis Library)

`TA` is a Python library for technical analysis, offering a collection of indicators such as Moving Averages, Bollinger Bands, MACD, and RSI. It provides easy-to-use functions to analyze market trends, momentum, and volatility, making it an invaluable tool for traders and analysts.

[4]:
```
!pip install ta
```

2.5 Scikit-learn

`Scikit-learn` is a machine learning library that provides tools for preprocessing, regression, classification, clustering, and model evaluation. In this project, it is primarily used for feature scaling (MinMaxScaler), linear regression, and performance metrics such as Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).

[5]:
```
!pip install scikit-learn
```

2.6 TensorFlow/Keras

`TensorFlow` and `Keras` are deep learning libraries used for building and training neural networks. In this project, they are employed for developing Long Short-Term Memory (LSTM) models for time-series forecasting. Keras provides an easy-to-use API for defining models, while TensorFlow serves as the backend for computation.

[6]:
```
!pip install tensorflow
```

2.7 Plotly

`Plotly` is an interactive visualization library that supports dynamic charting and dashboards. It is used in this project for plotting stock trends, forecasting results, and performance comparisons, providing clear and insightful graphical representations.

[7]:
```
!pip install plotly
```

2.8 Matplotlib and Seaborn

`Matplotlib` and `Seaborn` are widely used libraries for static data visualization. Matplotlib provides low-level plotting functions, while Seaborn enhances the aesthetics and usability of statistical graphs. Together, they enable effective data exploration and result visualization.

[8]:
```
!pip install matplotlib seaborn
```

With an overview of these essential libraries, we can now proceed to the implementation and analysis in the following sections.

3 Getting Started

We will use Yahoo Finance to analyze the stock market performance of four major technology companies: Apple (AAPL), Amazon (AMZN), Tesla (TSLA), and Microsoft (MSFT). Our dataset contains daily returns for these companies from January 1, 2020, to January 1, 2025.

[7]:
```
# List of tickers to download data for
tickers = ['AAPL', 'AMZN', 'TSLA', 'MSFT']

# Download stock data for all tickers
stocks_data = yf.download(tickers, start='2020-01-01', end='2025-01-01',␣
 ↪group_by='ticker',auto_adjust=False)

# Reshape the DataFrame so tickers appear in rows
df = stocks_data.stack(level=0).reset_index()

# Rename columns for clarity
df.columns = ['Date', 'Ticker', 'Open', 'High', 'Low', 'Close','Adj Close',␣
 ↪'Volume']

# Display the reshaped DataFrame
df.tail(10)
```

```
[**********************100%**********************]  4 of 4 completed
```

[7]:

	Date	Ticker	Open	High	Low	Close	\
5022	2024-12-27	MSFT	430.529999	430.529999	435.220001	426.350006	
5023	2024-12-27	TSLA	431.660004	431.660004	450.000000	426.500000	
5024	2024-12-30	AAPL	252.199997	252.199997	253.500000	250.750000	
5025	2024-12-30	AMZN	221.300003	221.300003	223.000000	218.429993	

```
5026 2024-12-30   MSFT  424.829987  424.829987  427.549988  421.899994
5027 2024-12-30   TSLA  417.410004  417.410004  427.000000  415.750000
5028 2024-12-31   AAPL  250.419998  250.419998  253.279999  249.429993
5029 2024-12-31   AMZN  219.389999  219.389999  223.229996  218.940002
5030 2024-12-31   MSFT  421.500000  421.500000  426.730011  420.660004
5031 2024-12-31   TSLA  403.839996  403.839996  427.929993  402.540009
```

```
      Adj Close    Volume
5022  434.600006  18117700
5023  449.519989  82666800
5024  252.229996  35557500
5025  220.059998  28321200
5026  426.059998  13158700
5027  419.399994  64941000
5028  252.440002  39480700
5029  222.970001  24819700
5030  426.100006  13246500
5031  423.790009  76825100
```

Dataset explanation :

Open: Price at market opening.

High: The highest price during the day.

Low: The lowest price during the day.

Close: Price at market close.

Adj Close: The closing price adjusted for dividends, stock splits, etc., making it useful for analysis over time.

Volume: The total number of shares traded during the day.

[8]:
```
df=df.set_index('Date')
```

Examining our dataset, we observe that the data consists of numerical values, with the date serving as the index.

3.1 Descriptive Statistics

`.describe()` provides a summary of descriptive statistics for your dataset, including measures of central tendency (mean, median), dispersion (standard deviation, min, max), and distribution (percentiles). It excludes NaN values and works with both numeric and object data types. The output adapts based on the data type of the columns analyzed.

[9]:
```
df[df['Ticker']=='AAPL'].describe()
```

[9]:
```
              Open         High          Low        Close    Adj Close  \
count  1258.000000  1258.000000  1258.000000  1258.000000  1258.000000
mean    152.344948   154.108800   155.660348   152.379123   153.951948
std      42.119394    41.520877    41.654310    41.305929    41.466618
```

```
min      54.509754    56.092499    57.125000    53.152500    57.020000
25%     127.085897   129.624996   130.755005   127.505001   129.017506
50%     150.926765   152.805000   154.570000   150.825005   152.575005
75%     177.025871   178.850006   180.160000   177.115005   178.500004
max     259.019989   259.019989   260.100006   257.630005   258.190002

              Volume
count   1.258000e+03
mean    9.057384e+07
std     5.325460e+07
min     2.323470e+07
25%     5.546825e+07
50%     7.627980e+07
75%     1.077425e+08
max     4.265100e+08
```

We have only 1,259 records over the five-year period because the data excludes weekends and public holidays when the stock market is closed.

3.2 Information About the Data

The `.info()` method displays details about a DataFrame, such as the index type, column data types, the number of non-null values in each column, and the memory usage.

[10]:
```
df[df['Ticker']=='AAPL'].info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1258 entries, 2020-01-02 to 2024-12-31
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Ticker     1258 non-null   object
 1   Open       1258 non-null   float64
 2   High       1258 non-null   float64
 3   Low        1258 non-null   float64
 4   Close      1258 non-null   float64
 5   Adj Close  1258 non-null   float64
 6   Volume     1258 non-null   int64
dtypes: float64(5), int64(1), object(1)
memory usage: 78.6+ KB
```

3.3 Unique Values

The `df.nunique()` method shows the number of unique values in each column of the DataFrame:

- **Ticker**: 4 unique tickers (AAPL, AMZN, TSLA, MSFT).
- **Open, High, Low, Close, Adj Close**: Thousands of unique price values due to daily fluctuations.

- **Volume**: 5017 unique trading volumes, reflecting the variability in daily stock activity.

[11]:
```
df.nunique()
```

[11]:
```
Ticker          4
Open         4944
High         4724
Low          4645
Close        4702
Adj Close    4683
Volume       5013
dtype: int64
```

4 Exploratory Data Analysis (EDA)

4.1 Distribution of Open Prices

This figure aims to show the distribution of the 'open' prices of the stock. By visualizing the distribution, we can see if it is skewed or symmetric and if there are any outliers. This information can help us understand the range of prices and how the prices are distributed.

`Measuring skewness of each company:`

```
Apple's skewness:  0.11
Tesla's skewness:  0.21
Microsoft's skewness:  -0.017
Amazon's skewness:  0.071
```

4.1.1 Skewness Interpretation for Each Company's Opening Prices

- **Apple's skewness: 0.1**
  Apple's distribution of "Open" prices is slightly positively skewed, indicating that most of the data points are concentrated on the lower end, with a slight tail extending towards higher values.
- **Tesla's skewness: 0.21**
  Tesla shows a mild positive skew, suggesting that the distribution has a small rightward tail, meaning there are a few higher values pulling the average up.
- **Microsoft's skewness: -0.017**
  Microsoft's distribution is very close to being symmetric, with a skewness close to zero. This indicates that the data is almost evenly distributed around the central value.
- **Amazon's skewness: 0.071**
  Amazon's distribution is slightly positively skewed, much like Apple and Tesla, with a small rightward tail suggesting that there are occasional higher values in the data.

4.2 Distribution of Close Prices

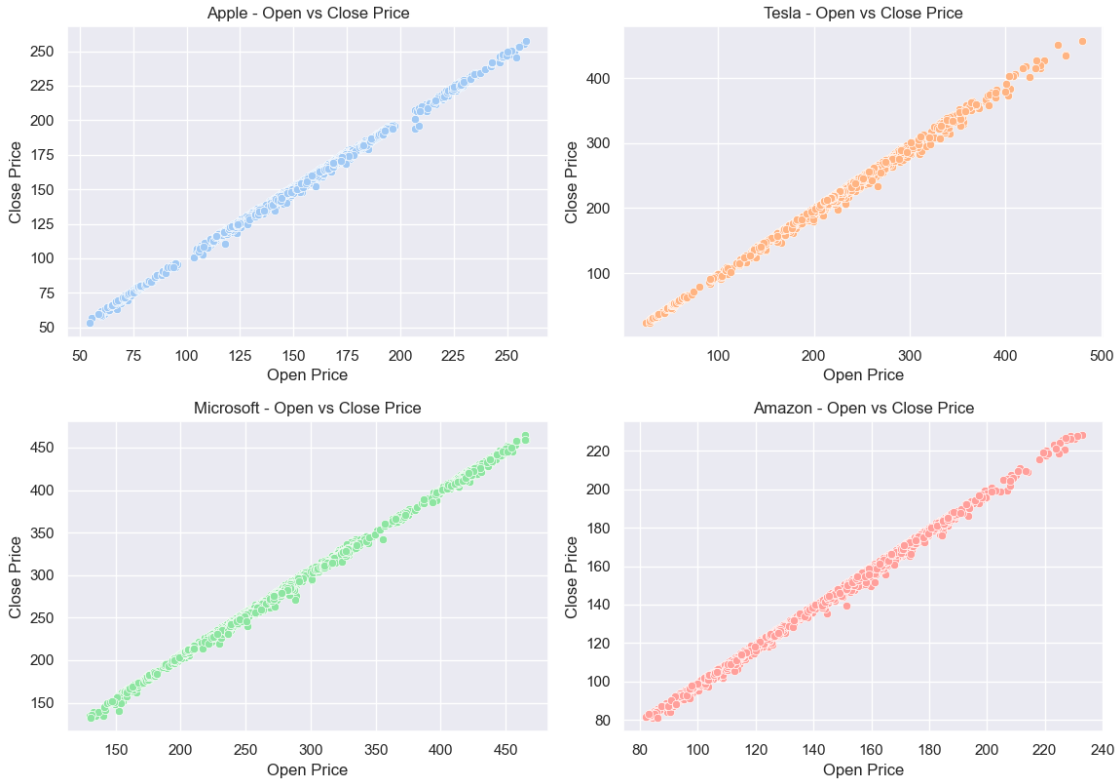
```
Measuring skewness of each company:
Apple's skewness:  -0.19
Tesla's skewness:  0.25
Microsoft's skewness:  -0.302
Amazon's skewness:  -0.147
```

4.2.1 Skewness Interpretation for Each Company's Closing Prices

- **Apple's skewness: -0.19**
  Apple's closing price distribution is slightly negatively skewed, meaning the left tail of the distribution is a bit longer than the right. This suggests that there are occasional lower closing prices, pulling the distribution slightly to the left.
- **Tesla's skewness: 0.25**
  Tesla's closing price distribution is mildly positively skewed. The right tail is slightly longer, indicating that there are occasional higher closing prices that are skewing the distribution to the right.
- **Microsoft's skewness: -0.302**
  Microsoft's distribution shows a more pronounced negative skew. The data has a longer tail on the left side, suggesting that Microsoft has experienced several lower-than-usual closing prices over time, which has shifted the distribution leftward.
- **Amazon's skewness: -0.147**
  Amazon's closing price distribution is also slightly negatively skewed, but less so than Apple's and Microsoft's. This indicates a mild tendency toward lower closing prices, though the effect is not as significant.

In this case, Tesla has a slight positive skew, while Apple, Microsoft, and Amazon all show a negative skew, with Microsoft exhibiting the most pronounced negative skew.

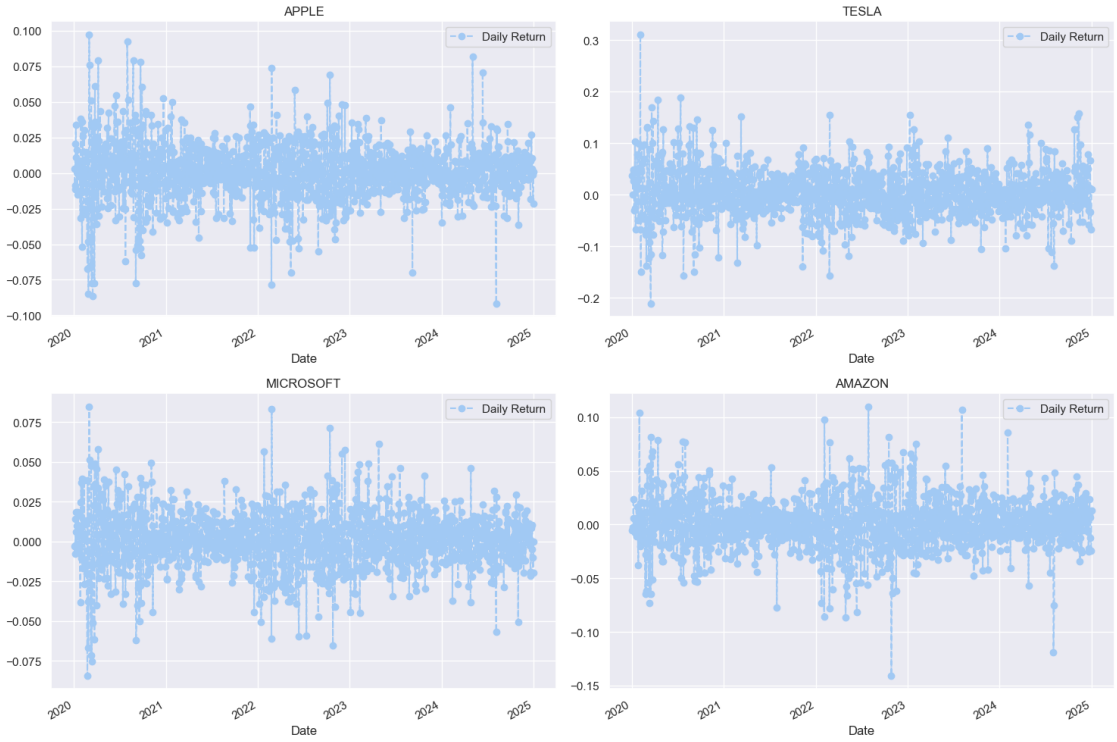
4.3 Open vs Close Price Scatter Plot



- In this figure, we examine the relationship between the stock's opening and closing prices to understand how closely they align throughout the day. By observing whether the prices are correlated or diverge, we can gain insights into how similar or different the stock's value is at the start and end of the trading session. This information helps in assessing the volatility of the stock.

From the plot, we can see that for each company, the open and close prices are close, indicating that there isn't significant fluctuation between the beginning and end of the trading day. This suggests a level of stability in the stock's daily performance for these companies.

4.4 What was the daily return of the stock on average?



4.4.1 Interpretation of the Daily Return Plots

The four histograms show **daily return plots** for **Apple (AAPL), Tesla (TSLA), Microsoft (MSFT), and Amazon (AMZN)** from 2020 to early 2025. Each plot represents the daily percentage change in adjusted closing prices, showing the **volatility** and **return distribution** over time.

Most returns cluster around zero:
- Across all four stocks, the majority of daily returns remain within a small range ($\pm 5\%$), suggesting **relative stability** most of the time.
- However, there are noticeable **outliers** indicating high volatility on certain days.

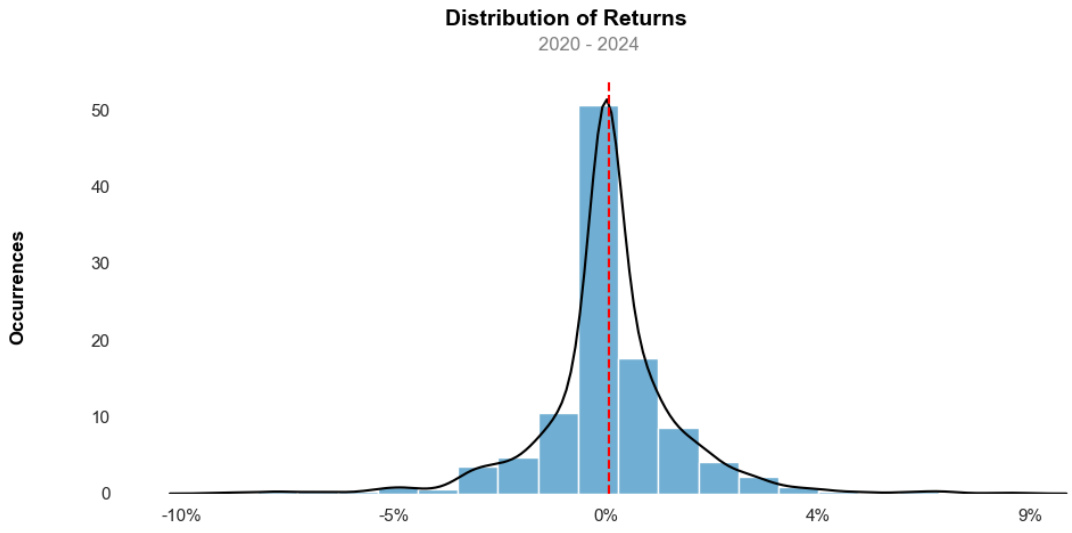
Tesla (TSLA) shows the highest volatility:
- Tesla's plot has extreme daily return spikes, sometimes exceeding $+$**30%** and **-20%**, highlighting **higher price fluctuations**.
- This suggests that Tesla is a **high-risk, high-reward** stock, likely driven by market sentiment, earnings reports, or industry trends (e.g., EV sector growth).

Microsoft (MSFT) has a more balanced and stable return distribution:
- Most of its daily returns stay within $\pm$**10%**, showing less volatility compared to Tesla.
- This aligns with Microsoft's position as a **stable, blue-chip tech company**, less affected by market shocks.

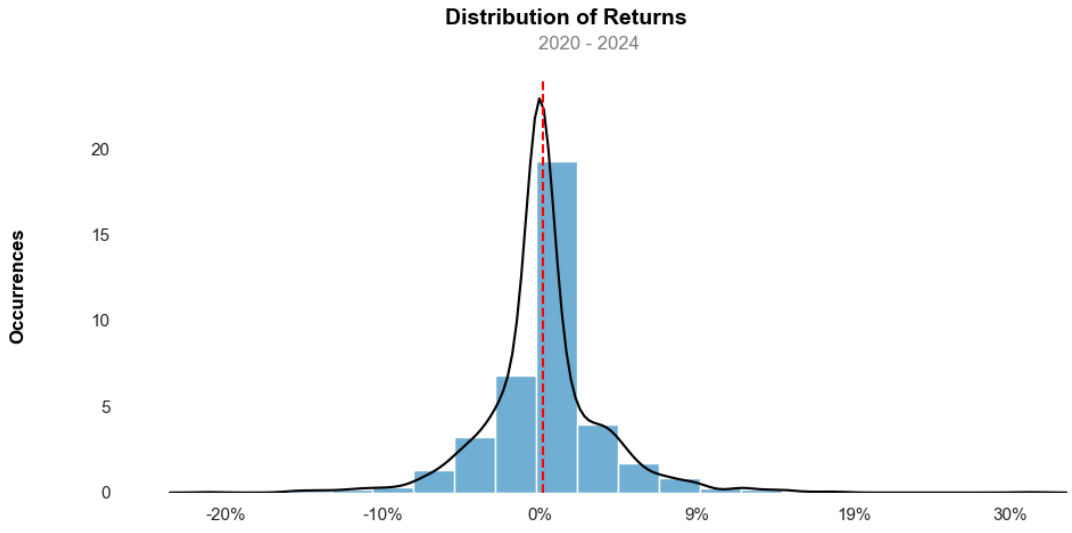
Apple (AAPL) and Amazon (AMZN) show moderate volatility:

- Both exhibit **occasional spikes**, but their daily returns remain mostly between $\pm$**7.5%**.
- Apple, being a major consumer tech company, experiences some volatility around **product launches and earnings reports**.
- Amazon, a major e-commerce and cloud computing player, shows **relatively lower volatility** than Tesla but still has some notable outliers. Now, let's visualize the average daily return using a histogram. We'll leverage Seaborn to generate both a histogram and a KDE plot within the same figure.

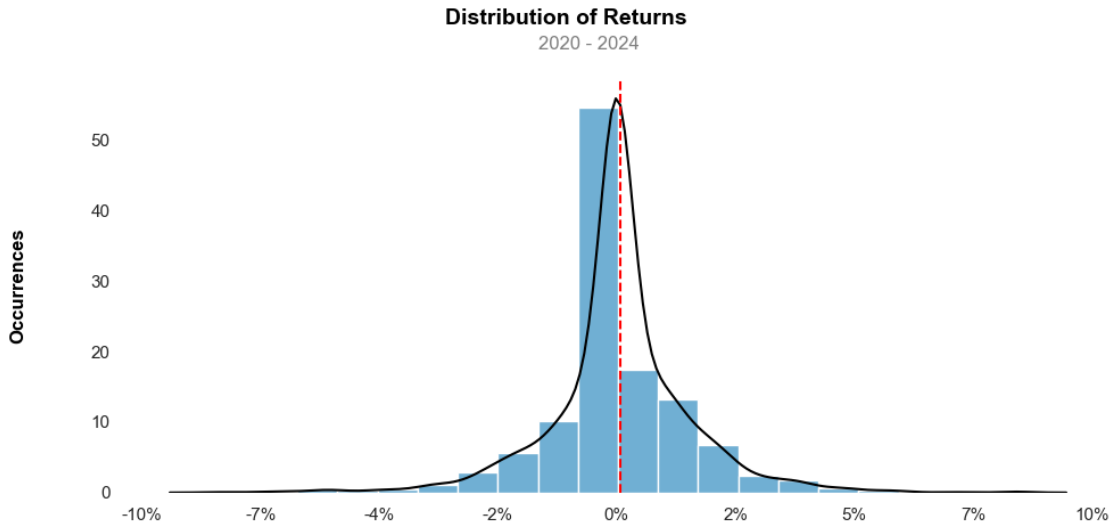
`Apple Daily Returns Histogram`



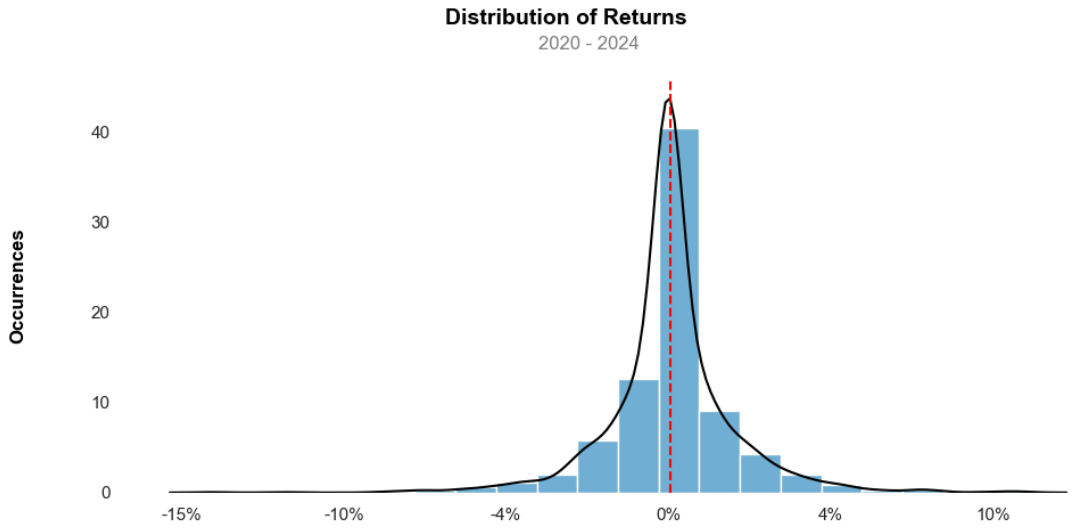
`Tesla  Daily Returns Histogram`



Microsoft Returns Histogram



Amazon Returns Histogram



Histogram visuals provide a clearer representation of daily returns compared to line plots. The analysis highlights the following key points:

- Tesla's returns exhibit high volatility, with frequent extreme values. Positive outliers reach approximately 30%, while negative returns are mostly capped at around -20%, indicating significant price swings.
- Microsoft's returns show a more stable pattern, with values predominantly ranging between -10% and 10%. The majority of daily returns are concentrated near zero, suggesting a more balanced and predictable performance compared to Tesla.

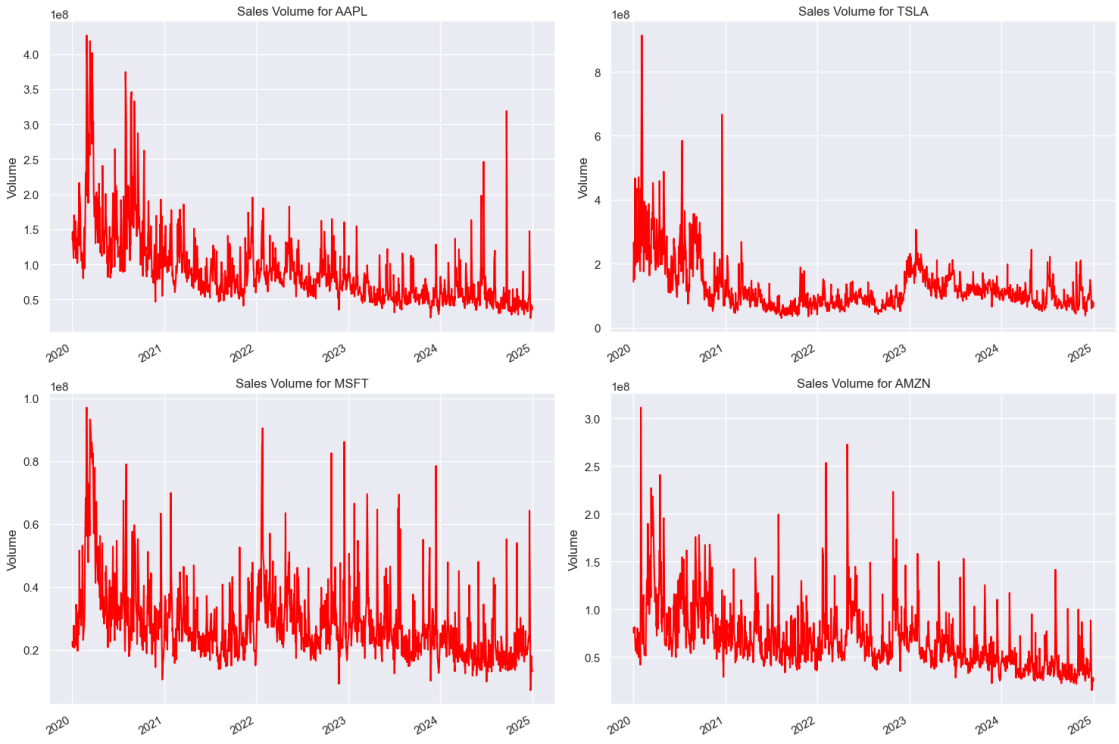
4.5 Adjusted Closing Price

- The **Adjusted Closing Price** is the closing price of a stock adjusted for factors like **dividends**, **stock splits**, and **new stock offerings**. Unlike the regular closing price, it reflects these corporate actions to give a more accurate view of a stock's performance over time. This helps investors assess the true value and growth of a stock, accounting for events that affect the stock price but don't reflect the company's actual market performance.

- Upon analyzing the charts, we can observe that the closing prices for all four companies have generally risen over the years. Microsoft, in particular, reached its highest closing price of approximately $450. However, in 2023, each of the companies experienced a noticeable decline in their closing prices. Among them, Amazon appears to have been the most significantly impacted by this downturn, showing the sharpest decrease.

4.6 Volume of Sales

- The **Volume of Sales** refers to the total number of shares of a stock that are traded during a specific period. It indicates the level of activity or liquidity in the market for that stock. Higher trading volume often suggests increased investor interest, while lower volume can indicate less market activity.



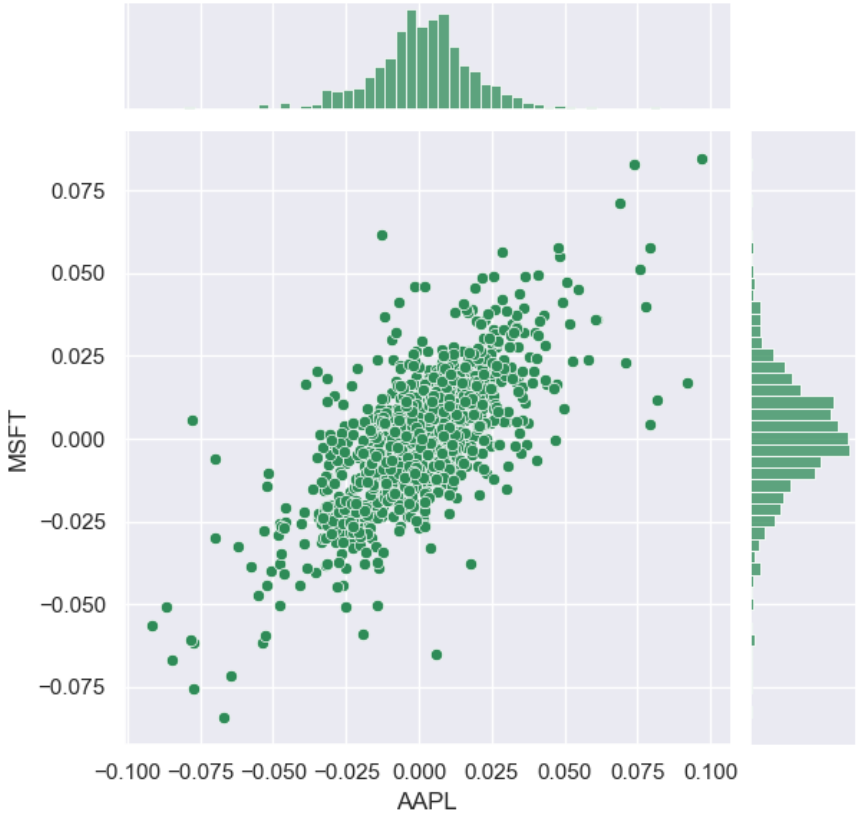
- Initially, Microsoft and Tesla had the highest sales volumes among the companies. However, as we move towards the present day, it's clear that the trading volumes for these companies have declined. This shift may reflect changing market conditions, investor sentiment, or other external factors that have influenced the level of trading activity over time.

4.7 What was the moving average of the various stocks?

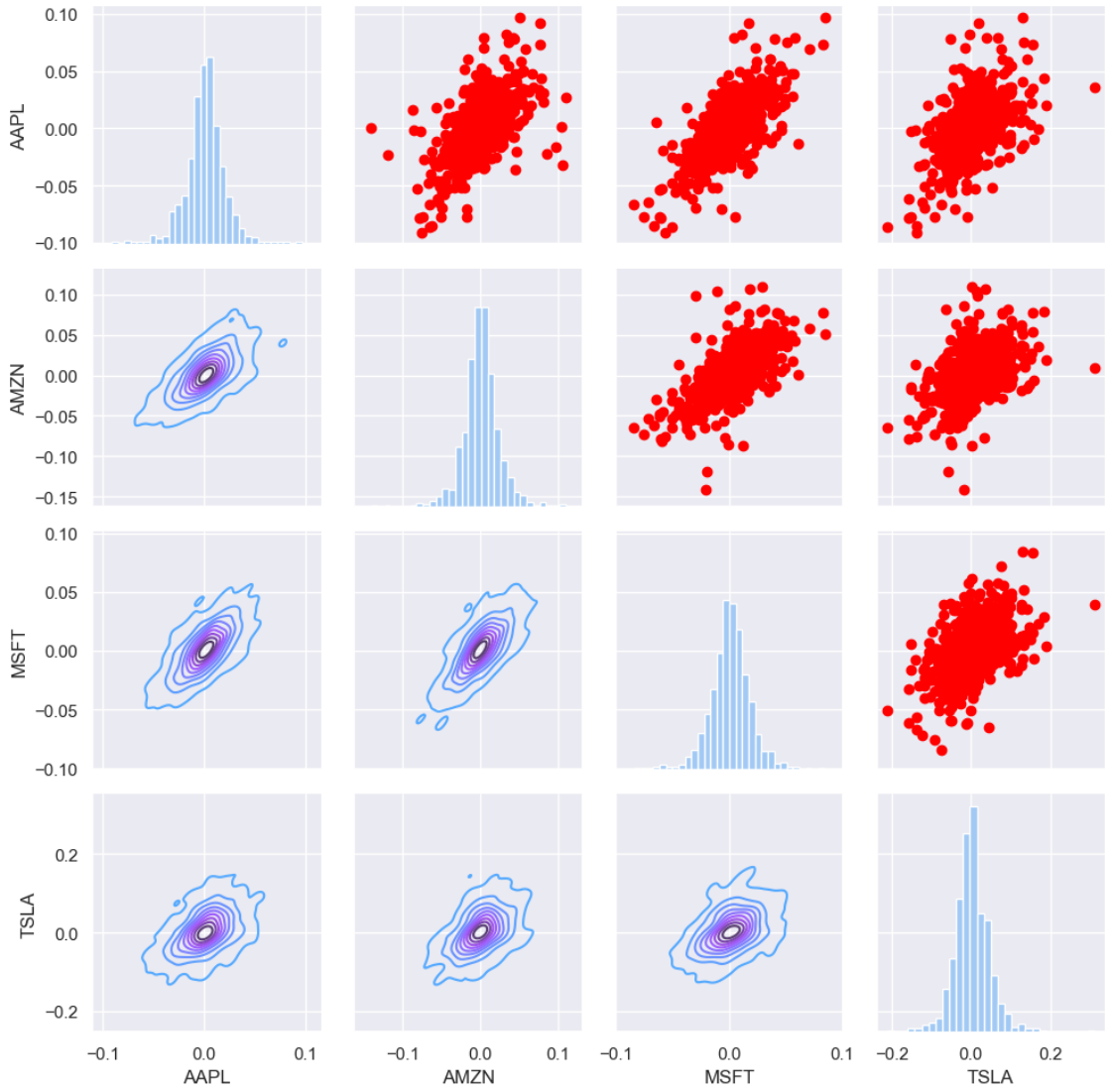
- The **Moving Average (MA)** is a commonly used tool in technical analysis that smooths price data by calculating a continuously updated average. This average is computed over a defined time period, such as 10 days, 20 minutes, 30 weeks, or any other period selected by the trader.

- We have a daily dataset, so we applied moving averages for 10, 20, and 50 days to the data. The graph demonstrates that the 10-day and 20-day moving averages are most effective for capturing trends, as they reflect the underlying patterns while minimizing noise in the data.

4.8 What was the correlation between different stocks closing prices?

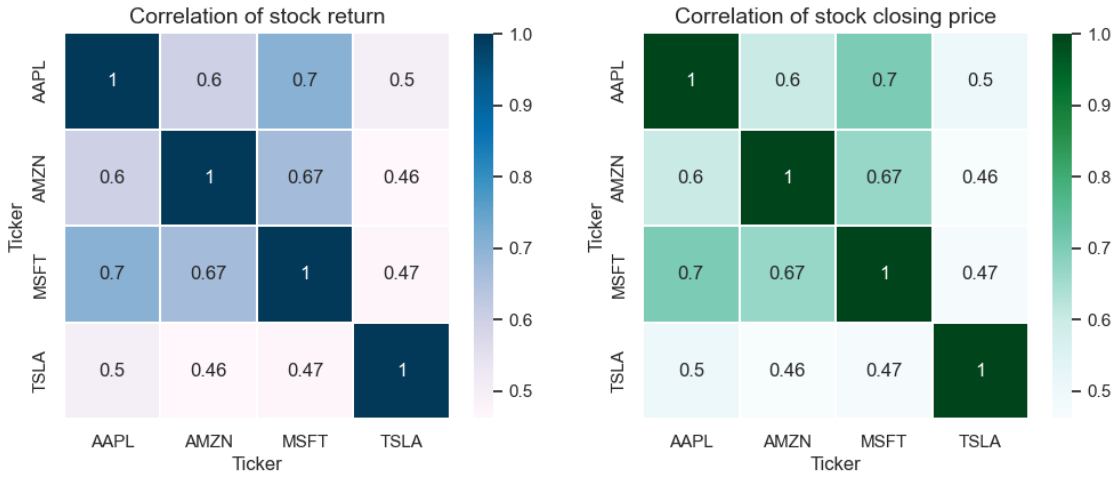


- The joint plot illustrates the relationship between **Apple (AAPL)** and **Microsoft (MSFT)** daily returns.
- The scatter plot at the center indicates a positive correlation, suggesting that when Apple's stock returns increase, Microsoft's stock returns tend to rise as well.
- The histograms on the top and right display the individual return distributions for Apple and Microsoft, which appear approximately normal but may exhibit some skewness.
- The spread of points suggests some degree of volatility in returns, but the positive trend implies these two stocks generally move in sync within the market.



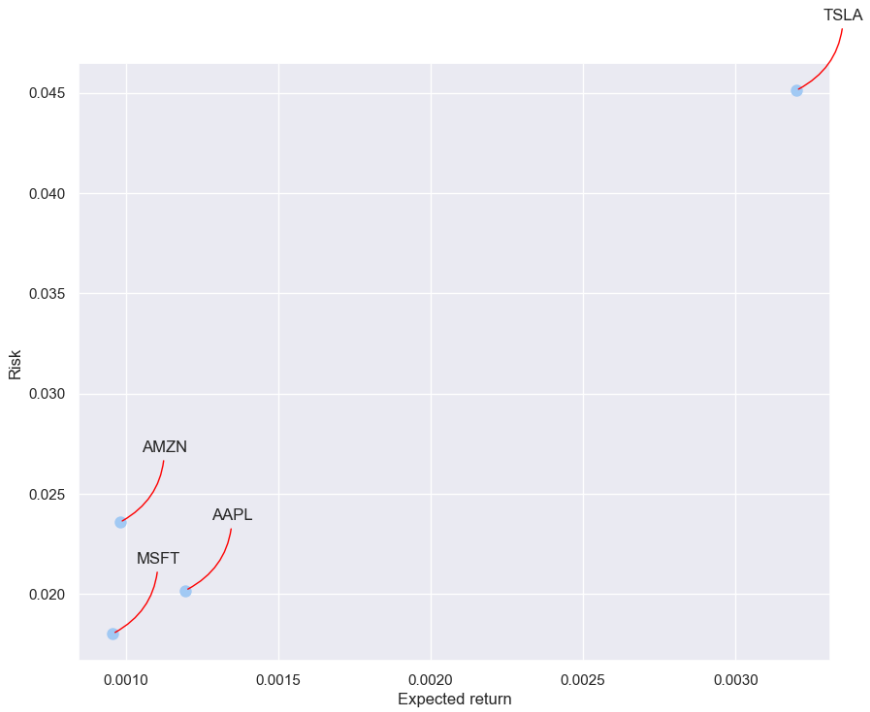
This visualization is a **Seaborn PairPlot (PairGrid)**, commonly used in **exploratory data analysis (EDA)** to examine relationships between multiple numerical variables. The plot show relationships between stock returns of **AAPL (Apple), AMZN (Amazon), MSFT (Microsoft), and TSLA (Tesla)**.

- **AAPL vs AMZN, AAPL vs MSFT, AMZN vs MSFT:** These stocks have strong correlations, as seen in the tight clustering along a diagonal trend.
- **TSLA vs Other Stocks:** Tesla's return distribution has a wider spread, suggesting higher volatility compared to other stocks.
- **Density plots suggest dependencies:** The tighter the density contours, the stronger the correlation.
- **Outliers and extreme returns:** Some scatter plots show individual points far from the main clusters, indicating potential outliers or extreme market movements.



- Two graphics display heatmaps of stock returns and closing prices, respectively. These heatmaps illustrate the relationships between different features. Notably, Apple and Microsoft exhibit a strong positive correlation, as do Amazon and Microsoft. Interestingly, all technology companies show a positive correlation with one another.

4.9 How much value do we put at risk by investing in a particular stock?



- The graph visualizes the risk (likely standard deviation) associated with different companies.

- **Tesla (TSLA)** exhibits the highest risk, positioned at the top, indicating significant price volatility.
- **Microsoft (MSFT)**, **Apple (AAPL)**, and **Amazon (AMZN)** have relatively lower risk levels, clustered closely together.
- The upward position of **Tesla** suggests it experiences the most substantial price fluctuations, making it the most volatile stock in the group.

5 Feature Engineering

- In this section, we will concentrate on predicting the future closing price of **Apple Inc. (AAPL)**. To begin, we will collect historical stock data from **Yahoo Finance**, covering the period from 2010 to the present. After gathering the data, we will proceed with feature extraction, identifying key variables that can enhance the accuracy of our predictions.
- By analyzing historical trends, market patterns, and statistical indicators, we aim to develop a robust forecasting model that provides insights into Apple's potential future performance.

[37]:
```python
def extracting_features(df):
    """Extracts key financial indicators from a Yahoo Finance dataset."""

    df = df.copy()

    #Price Change (Close - Open)
    df['price_change'] = df['Close'] - df['Open']

    #Returns (Daily Percentage Change)
    df['returns'] = df['Close'].pct_change()

    #Create a new feature 'average_price'
    df['average_price'] = (df['Close'] + df['Open']) / 2

    #Price Range (High - Low)
    df['price_range'] = df['High'] - df['Low']

    #Volume Change
    df['volume_change'] = df['Volume'].diff()

    #Moving Average
    df['moving_average_10'] = df['Close'].rolling(window=10).mean()


    #Relative Strength Index (RSI)
    window = 14
    delta = df['Close'].diff()
    gain = (delta.where(delta > 0, 0)).rolling(window=window).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=window).mean()
    rs = gain / loss
    df['RSI'] = 100 - (100 / (1 + rs))
```

```
    # MACD (Moving Average Convergence Divergence)
    short_ema = df['Close'].ewm(span=12, adjust=False).mean()
    long_ema = df['Close'].ewm(span=26, adjust=False).mean()
    df['MACD'] = short_ema - long_ema
    df['MACD_Signal'] = df['MACD'].ewm(span=9, adjust=False).mean()

    # Bollinger Bands (20-Day Moving Average ± 2 Standard Deviations)
    df['20_SMA'] = df['Close'].rolling(window=20).mean()
    df['BB_Upper'] = df['20_SMA'] + (df['Close'].rolling(window=20).std() * 2)
    df['BB_Lower'] = df['20_SMA'] - (df['Close'].rolling(window=20).std() * 2)

    return df

df = extracting_features(df)
```

Below is a detailed explanation of each feature:

- `price_change`: Measures the difference between the closing and opening prices for a given day. This helps determine whether the stock gained or lost value during the trading session.
- `returns`: Represents the percentage change in the closing price from one day to the next, giving insight into the stock's overall performance over time.
- `average_price`: Computes the average of the opening and closing prices for a given day, serving as a general indicator of that day's stock value.
- `price_range`: Captures the difference between the highest and lowest prices of the day, reflecting the stock's intraday volatility.
- `volume_change`: Measures the change in trading volume compared to the previous day, helping to analyze market interest and liquidity trends.
- `moving_average_10`: Calculates the 10-day moving average of the closing price, smoothing short-term fluctuations and indicating the overall trend.
- `RSI` **(Relative Strength Index)**: Measures the momentum of price movements on a scale from 0 to 100. It is calculated using the following formula:

$$RSI = 100 - \frac{100}{1 + RS}$$

where **RS** (Relative Strength) is calculated as:

$$RS = \frac{\text{Average Gain over } n \text{ periods}}{\text{Average Loss over } n \text{ periods}}$$

A high RSI suggests overbought conditions, while a low RSI indicates oversold conditions.

- `MACD` **(Moving Average Convergence Divergence)**: Computes the difference between the 12-day and 26-day exponential moving averages (EMAs) of the closing price. It is used to identify trend direction, strength, and potential reversals.
- The **MACD Signal Line** is an essential technical indicator used in stock analysis. It is calculated as the **9-day moving average of the MACD (Moving Average Convergence Divergence)**. This feature helps smooth out fluctuations in the MACD line and provides clearer signals for potential buy or sell opportunities. Traders often use the MACD Signal Line to identify trend changes and confirm market momentum. The formula for the MACD Signal Line is:

$$\text{MACD Signal} = \text{9-day EMA of MACD}$$

- `20_SMA` **(20-Day Simple Moving Average)**: Represents the simple moving average over 20 days, providing a longer-term trend indication.

- `BB_Upper` **(Upper Bollinger Band)**: The upper bound of Bollinger Bands, calculated using the 20-day SMA plus two standard deviations. It helps identify potential overbought conditions. The formula for the **Upper Bollinger Band** is:

$$\text{Upper Band} = \text{20-day SMA} + (2 \times \sigma)$$

- `BB_Lower` **(Lower Bollinger Band)**: The lower bound of Bollinger Bands, computed as the 20-day SMA minus two standard deviations. It helps identify potential oversold conditions. The formula for the **Lower Bollinger Band** is:

$$\text{Lower Band} = \text{20-day SMA} - (2 \times \sigma)$$

6 Preprocessing

6.1 Normalizing

- In this section, we will perform data normalization using the max-min normalization method. The goal of normalization is to standardize the numerical features in the dataset, ensuring that they all fall within a consistent range or scale.

[13]:
```
# Function to normalize data using MinMax Scaling for all features
def min_max_scaling(data):
    """Applies Min-Max scaling to normalize all features (Open, High, Low,␣
 ↪Close, Adj Close, Volume)."""
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(data)  # Normalize all features
    print("Data normalization using Min-Max Scaling completed.")
    return scaler, scaled_data
```

6.2 Denormalization

- After making predictions with the model, it is necessary to convert the normalized values back to their original scale in order to accurately interpret the results. This step is crucial because

the model typically works with normalized data, but to understand the actual predictions, we need them in their original, unscaled form. Therefore, we perform this denormalization to ensure the predicted values reflect the true price or value.

[42]:
```
def denormalize_predictions(y_pred, df):
    """
    Denormalize the predicted values using the MinMaxScaler fitted on the␣
↪original data.

    Args:
        y_pred (np.array): The normalized predicted stock values.
        df (pd.DataFrame): The original dataframe containing the stock prices.

    Returns:
        np.array: The denormalized predicted values.
    """
    # Extract 'Close' column from the dataframe, which was used for normalization
    close_prices = df['Close'].values.reshape(-1, 1)  # Reshape to (n_samples, 1)

    # Create a MinMaxScaler and fit it using the 'Close' prices
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaler.fit(close_prices)  # Fit the scaler on the 'Close' price column

    # Apply the inverse_transform to denormalize the predicted values
    y_pred_denormalized = scaler.inverse_transform(y_pred.reshape(-1, 1))  #␣
↪Denormalize predictions

    return y_pred_denormalized.flatten()  # Flatten to return a 1D array
```

6.3 Splitting Data

- The below function splits a time series dataset into training and testing sets (70% train, 30% test), creates input features and target labels for a specified number of prediction days (pred_days), and reshapes the data for use in machine learning models such as LSTM. The function returns the reshaped training and testing datasets.

[40]:
```
def split_and_reshape_data(dataframe, pred_days, company):
    prediction_days = pred_days

    # Split the data into 70% training and 30% testing
    train_size = int(np.ceil(len(dataframe) * 0.70))  # 70% for training data
    test_size = len(dataframe) - train_size  # Remaining 30% for testing data
    print(f'The training size for Apple is {train_size} rows')
    print(f'The testing size for {company.title()} is {test_size} rows')

    # Create training and testing datasets
    train_data = dataframe[0: train_size, :]
    test_data = dataframe[train_size - prediction_days:, :]
```

```
    X_train, y_train, X_test, y_test = [], [], [], []

    # Loop to create X_train and y_train for training data
    for i in range(prediction_days, len(train_data)):
        X_train.append(train_data[i - prediction_days: i, 0])  # Features: previous 'pred_days' values
        y_train.append(train_data[i, 0])  # Target: next day's value

    # Loop to create X_test and y_test for testing data
    for i in range(prediction_days, len(test_data)):
        X_test.append(test_data[i - prediction_days: i, 0])  # Features: previous 'pred_days' values
        y_test.append(test_data[i, 0])  # Target: next day's value

    # Convert the lists to numpy arrays
    X_train, y_train, X_test, y_test = np.array(X_train), np.array(y_train), np.array(X_test), np.array(y_test)

    # Reshape the data to be suitable for LSTM model (3D array: samples, time steps, features)
    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

    print('Data for {} split successfully'.format(company.title()))

    return X_train, y_train, X_test, y_test

stock_name = 'Apple'
X_train, y_train, X_test, y_test = split_and_reshape_data(new_df, 30, stock_name)
```

```
The training size for Apple is 894 rows
The testing size for Apple is 383 rows
Data for Apple split successfully
```

7 Modeling

In this section, we will utilize two different forecasting models to evaluate their stock price prediction performance: LSTM and linear regression. We will begin by implementing the Linear Regression.

7.1 Linear Regression

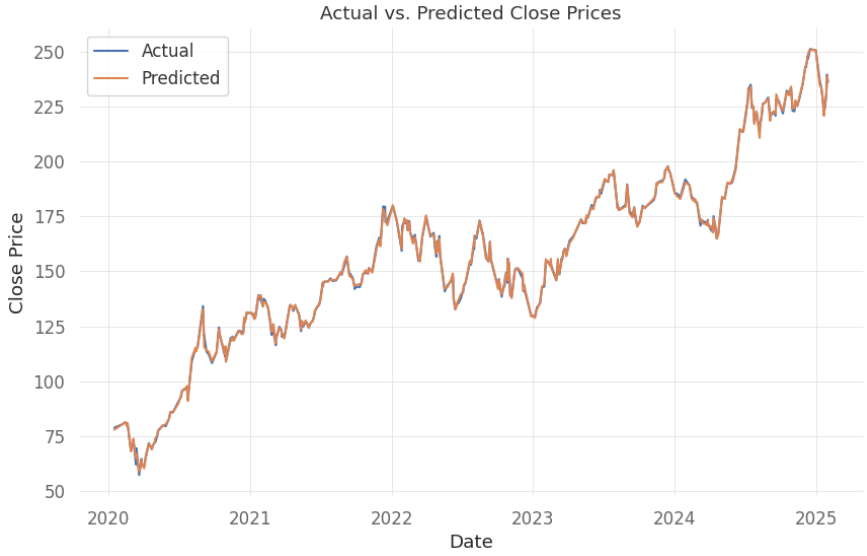
The `linear_prediction` function predicts stock closing prices using linear regression by processing and normalizing historical stock data, then evaluating the model's performance with metrics like MAE, MSE, RMSE, and R^2. It also visualizes the actual vs. predicted values and returns the model, scaler, and evaluation results.

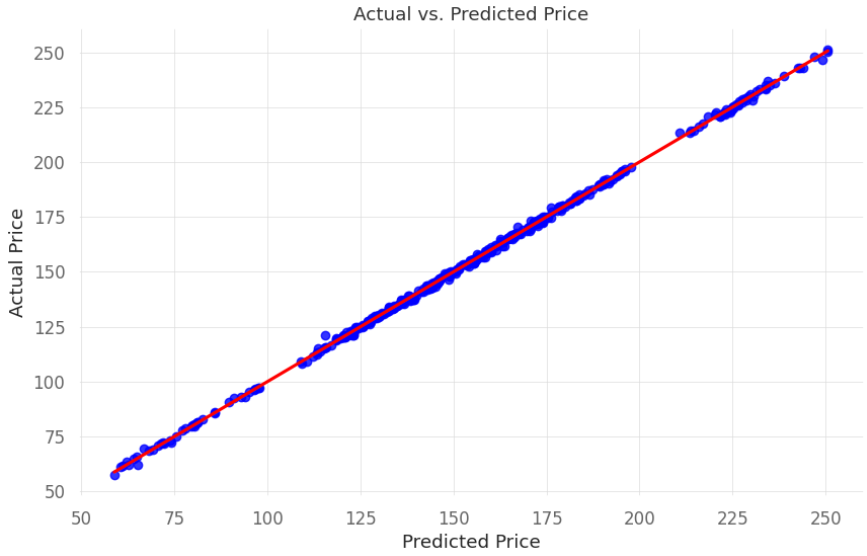
[41]:
```
def linear_prediction(df):
    # Define features and target
    X = df[['Open', 'High', 'Low', 'Volume',
       'moving_average_10', 'RSI', 'MACD', 'MACD_Signal', '20_SMA', 'BB_Upper',
       'BB_Lower']]
    y = df['Close']
    X=X.fillna(0)
    # Normalize features using Min-Max scaling
    scaler = MinMaxScaler()
    X_scaled = scaler.fit_transform(X)

    # Perform train-test split
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.
↪3, random_state=42)
    print("Shapes of X_train, X_test, y_train, y_test:", X_train.shape, X_test.
↪shape, y_train.shape, y_test.shape)

    # Fit the linear regression model
    regressor = LinearRegression()
    regressor.fit(X_train, y_train)

    # Make predictions
    y_pred = regressor.predict(X_test)

    # Calculate metrics using imported functions directly
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)

    # Store intercept and coefficients
    intercept = regressor.intercept_
    coefficients = regressor.coef_

    # Extract the dates corresponding to the test set
    test_dates = y_test.index

    # Create DataFrame to compare actual and predicted values
    compare = pd.DataFrame({
        'Date': test_dates,
        'Actual': y_test.values,
        'Predicted': y_pred
    })

    # Set date as index
    compare.set_index('Date', inplace=True)
```

```
    # Sort DataFrame by date
    compare = compare.sort_index()

    # Display metrics using tabulate
    metrics_data = [
        ['Mean Absolute Error', mae],
        ['Mean Squared Error', mse],
        ['Root Mean Squared Error', rmse],
        ['R^2 Score', r2]
    ]
    print("Metrics:")
    print(tabulate(metrics_data, headers=['Metric', 'Value'], tablefmt='psql'))



    # Display comparison DataFrame
    print("\nComparison DataFrame:")
    print(compare)

    # Plot predicted vs. actual values
    plt.figure(figsize=(10, 6))
    plt.plot(compare.index, compare['Actual'], label='Actual')
    plt.plot(compare.index, compare['Predicted'], label='Predicted')
    plt.xlabel('Date')
    plt.ylabel('Close Price')
    plt.title('Actual vs. Predicted Close Prices')
    plt.legend()
    plt.show()

    # Plot regression plot
    sns.regplot(x=y_pred.flatten(), y=y_test.values.flatten(),␣
 ↪scatter_kws={"color": "blue"}, line_kws={"color": "red"})
    plt.xlabel('Predicted Price')
    plt.ylabel('Actual Price')
    plt.title('Actual vs. Predicted Price')
    plt.show()

    return regressor, scaler, mae, mse, rmse, r2, compare

# Call the function and get the regressor object, scaler, metrics, and␣
 ↪comparison DataFrame
regressor, scaler, mae, mse, rmse, r2, comparison_df = linear_prediction(df)
```

```
Metrics:
+--------------------------+----------+
| Metric                   |    Value |
```

```
|-------------------------+----------|
| Mean Absolute Error     | 0.712869 |
| Mean Squared Error      | 0.897384 |
| Root Mean Squared Error | 0.947304 |
+-------------------------+----------+
```





First Graph: Actual vs. Predicted Close Prices
This line graph compares actual stock prices (blue) with predicted prices (orange) over time.The close alignment of the two lines indicates that the linear regression model effectively captures trends in stock price movements. The model follows both upward and downward trends closely, suggesting a strong predictive ability.
Second Graph: Predicted vs. Actual Price Scatter Plot
This scatter plot visualizes the correlation between predicted and actual prices. The red diagonal

line represents a perfect fit (i.e., predicted price = actual price). The blue data points closely follow this line, indicating high accuracy in predictions. Minimal deviation from the red line suggests that the model has a low error rate.
The linear regression model performs well, as evidenced by the tight alignment of actual and predicted values.The model achieves a strong correlation between predictions and real prices, making it a reliable tool for stock price forecasting.
The provided performance metrics reflect how well the model predicted the stock prices:

- **Mean Absolute Error (0.713)**: On average, the model's predictions are off by **$0.71** from the actual values, indicating relatively small errors.
- **Root Mean Squared Error (0.947)**: The model's error, on average, is approximately **$0.95**, indicating a slightly higher degree of error than the MAE.

7.2 Future Forecasting for 30 days

[43]:
```
def linear_forecasting(df, future_days=30):
    # Define features and target
    X = df[['Open', 'High', 'Low', 'Volume',
       'moving_average_10', 'RSI', 'MACD', 'MACD_Signal', '20_SMA', 'BB_Upper',
       'BB_Lower']]
    y = df['Close']
    X=X.fillna(0)
    # Normalize features using Min-Max scaling
    scaler = MinMaxScaler()
    X_scaled = scaler.fit_transform(X)

    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.
↪3, random_state=42)

    # Fit the linear regression model
    regressor = LinearRegression()
    regressor.fit(X_scaled, y)

    # Predicting for the future dates
    future_dates = pd.date_range(df.index[-1] + pd.DateOffset(1),␣
↪periods=future_days, freq='B')
    future_features = df.iloc[-1*future_days:][['Open', 'High', 'Low', 'Volume',
       'moving_average_10', 'RSI', 'MACD', 'MACD_Signal', '20_SMA', 'BB_Upper',
       'BB_Lower']]
    future_features_scaled = scaler.transform(future_features)
    future_predictions = regressor.predict(future_features_scaled)

    # Creating a DataFrame for future predictions
    future_prediction_df = pd.DataFrame({
        'Date': future_dates,
        'Predicted': future_predictions
    })
    future_prediction_df.set_index('Date', inplace=True)
```

```
    # Create figure with plotly
    fig = go.Figure()

    # Historical data trace
    fig.add_trace(go.Scatter(x=df.index, y=df['Close'], mode='lines+markers',␣
↪name='Historical Close'))

    # Future predictions trace
    fig.add_trace(go.Scatter(x=future_prediction_df.index,␣
↪y=future_prediction_df['Predicted'], mode='lines+markers', name='Predicted␣
↪Close'))

    # Update layout for better interactive controls
    fig.update_layout(
        title='Historical vs Predicted Close Prices',
        xaxis=dict(
            rangeselector=dict(
                buttons=list([
                    dict(count=1, label="1m", step="month", stepmode="backward"),
                    dict(count=6, label="6m", step="month", stepmode="backward"),
                    dict(count=1, label="YTD", step="year", stepmode="todate"),
                    dict(count=1, label="1y", step="year", stepmode="backward"),
                    dict(step="all")
                ])
            ),
            rangeslider=dict(
                visible=True
            ),
            type="date"
        ),
        yaxis=dict(
            title="Close Price",
            autorange=True,
            type="linear"
        )
    )

    fig.show()

    return regressor, scaler, future_prediction_df

regressor, scaler, future_predictions = linear_forecasting(df, future_days=30)

# Print  future_predictions DataFrame
print("Future Predictions:")
print(future_predictions)
```

```
Future Predictions:
             Predicted
Date
2025-01-31  250.672419
2025-02-03  253.071242
2025-02-04  250.444321
2025-02-05  250.844464
2025-02-06  251.787976
2025-02-07  254.465554
2025-02-10  257.569008
2025-02-11  259.323469
2025-02-12  254.758384
2025-02-13  251.978060
2025-02-14  250.645428
2025-02-17  243.386835
2025-02-18  242.796560
2025-02-19  245.863399
2025-02-20  243.729571
2025-02-21  241.843094
2025-02-24  234.631659
2025-02-25  231.462754
2025-02-26  234.004591
2025-02-27  237.975438
2025-02-28  230.532246
2025-03-03  229.378945
2025-03-04  220.718145
2025-03-05  223.374347
2025-03-06  224.791869
2025-03-07  222.978939
2025-03-10  230.733188
2025-03-11  238.523835
2025-03-12  238.786912
2025-03-13  239.273329
```

The forecasting result shows the **predicted stock closing prices** for the next 30 days (from January 31, 2025, to March 13, 2025). These predictions are based on a model that forecasts future values based on historical data and other relevant features. The values represent the estimated closing price of the stock for each corresponding date.

For instance: - On **January 31, 2025**, the predicted closing price is **250.67**. - On **February 3, 2025**, the predicted closing price is **253.07**. - As the dates move forward, the predicted prices fluctuate based on the model's interpretation of trends, seasonal effects, or other factors.

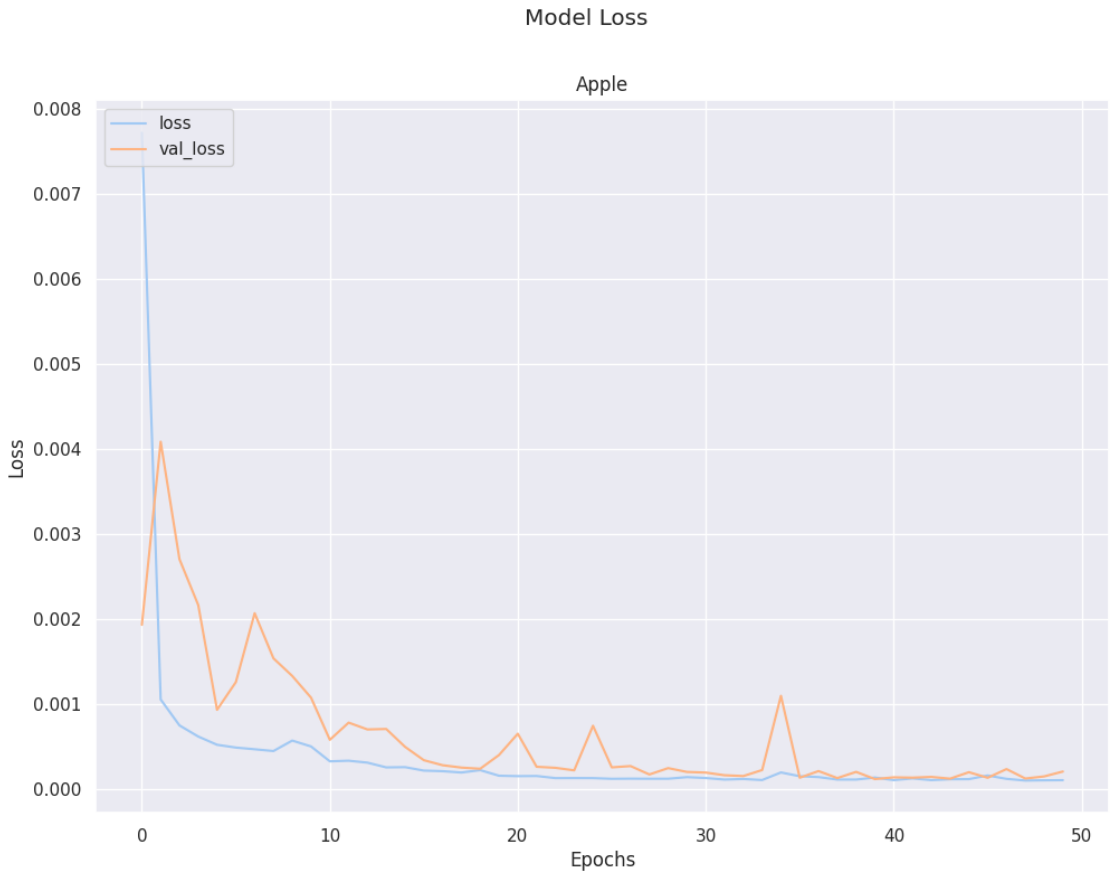
7.3 Long Term Short Memory (LSTM)

LSTM is a type of recurrent neural network (RNN) designed to handle sequential data and time series forecasting. It overcomes the vanishing gradient problem in standard RNNs by using memory cells to retain long-term dependencies.

[18]:
```
def build_model(X_train, y_train, X_test, y_test, company):

    print(f'========= For {company} =========')

    # Building the BiLSTM model
    model = Sequential([
        # 1D Convolutional Layer
        Conv1D(32, 3, strides=1, activation='relu', input_shape=[30, 1]),

        # Bidirectional LSTM Layers
        Bidirectional(LSTM(64, return_sequences=True)),
        Bidirectional(LSTM(64, return_sequences=True)),
        Bidirectional(LSTM(64)),

        # Fully Connected Layer
        Dense(32, activation='relu'),

        # Output Layer for Regression
        Dense(1)
    ])

    # Compile the model with Adam optimizer and Huber loss
    model.compile(optimizer=Adam(), loss=Huber(), metrics=['mse', 'mae'])

    # Train the model
    history = model.fit(X_train, y_train, validation_data=(X_test, y_test),␣
↪epochs=50)

    # Generate predictions on the test data
    y_pred = model.predict(X_test)

    return model, history, y_pred
```

The `build_model` function creates and trains a Bidirectional LSTM (BiLSTM) model to predict stock prices. It begins by adding a Conv1D layer to capture short-term patterns in the stock data, followed by three Bidirectional LSTM layers to capture both past and future dependencies in the time-series data. A Dense layer is used for further feature extraction, and the output layer predicts the stock price. The model is compiled with the Adam optimizer and Huber loss, and trained for 50 epochs with performance metrics MSE and MAE. After training, the function generates stock price predictions on the test data. The function returns the trained model, training history, and predicted prices.

After training the model, let's visualize if there is any reduction in loss.



X-Axis (Epochs) : Represents the number of training iterations (from 0 to 50).
Y-Axis (Loss) : Measures how well the model is performing (lower values indicate better performance).
Blue Line (Loss) : Represents the training loss, which shows how well the model is learning from the training dataset.
Orange Line (Validation Loss): Represents the validation loss, which shows how well the model is performing on unseen validation data.
Initially, both loss and validation loss are high. Over time, both losses decrease, indicating that the model is learning and improving. The training loss (blue line) is consistently lower than the validation loss (orange line), which is expected. Some fluctuations in validation loss suggest some level of overfitting, but overall, the model seems to generalize well.

7.4 Prediction and Metrics

We will evaluate the performance of our model by comparing it with the test data and the training data, scaled accordingly, and analyze the RMSE, MAE, and R-squared values.

[21]:
```
y_pred_denormalized = denormalize_predictions(y_pred, df)
y_test_denormalized = denormalize_predictions(y_test, df)
# Calculate RMSE
rmse = np.sqrt(np.mean((y_pred_denormalized - y_test_denormalized) ** 2))
```

```
    # Calculate MAE (Mean Absolute Error)
mae = mean_absolute_error(y_test, y_pred)

    # Print all metrics
print(f'The RMSE for Apple is {rmse}')
print(f'The MAE for Apple is {mae}')
```

- **RMSE (5.67)**: On average, the model's predictions are **$5.67 off** from the actual Apple stock prices.

- **MAE (4.83)**: The model's predictions deviate from the actual prices by about **$4.83 on average**.

these results show that the model performs well, with **small errors relative to stock price fluctuations** and a **high degree of accuracy**.
Next, we will visualize the predicted values.

- The graphic above illustrates the model's performance across training, validation, and prediction stages. As shown, the predictions closely align with the validation values, indicating the model's strong ability to generalize. Additionally, the test values are accurately predicted, demonstrating the model's effectiveness in forecasting future data.

7.5 Predicting & Forecasting for next 30 Days

- In this section, we use the trained model to predict and forecast stock prices for the next 30 days. The model leverages historical data to generate accurate predictions, helping to estimate future trends and price movements.
- The below code prepares data for time-series prediction by sorting the dataset, splitting it into training and test sets, and extracting the last 50 days of the test data as input (`x_input`) for making future predictions.
- We use the previous 50 days of data because recent trends provide the most relevant information for accurately predicting future values in time-series forecasting.

[38]:
```
def forecast_lstm(model, df, normalized_df, future_days=30, n_steps=50):
    """
    Forecast the next `future_days` using a trained LSTM model.

    Args:
        model: Trained LSTM model.
        df (DataFrame): Original DataFrame with stock prices (should have a␣
↪DateTime index).
        normalized_df (np.array): Normalized dataset used as LSTM input.
        future_days (int): Number of days to predict (default: 30).
        n_steps (int): Number of time steps for LSTM input (default: 50).

    Returns:
        forecast_df (DataFrame): DataFrame with forecasted dates and␣
↪denormalized prices.
    """

    print("\n========= Forecasting Next 30 Days =========")

    # Ensure normalized_df is a NumPy array
    if isinstance(normalized_df, pd.DataFrame):
        normalized_df = normalized_df.values  # Convert to array

    # Validate input shape
    if len(normalized_df.shape) != 2 or normalized_df.shape[1] != 18:
        raise ValueError(f"Expected normalized_df shape (X, 18), but got␣
↪{normalized_df.shape}")

    # Get test data (last 50 days)
    x_input = normalized_df[-n_steps:].reshape((1, n_steps, 18))  # Reshape for␣
↪LSTM input
    temp_input = x_input[0].tolist()  # Convert to a properly structured list of␣
↪lists

    lstm_op = []

    # Ensure df index is datetime format
    df.index = pd.to_datetime(df['Date'])
    last_date = df.index[-1]

    # Generate forecast dates (business days only)
    forecast_dates = pd.date_range(start=last_date + pd.Timedelta(days=1),␣
↪periods=future_days, freq='B')

    # **Fit MinMaxScaler on the original DataFrame**
    scaler = MinMaxScaler()
    df_close = df[['Close']]  # Select the 'Close' price column
```

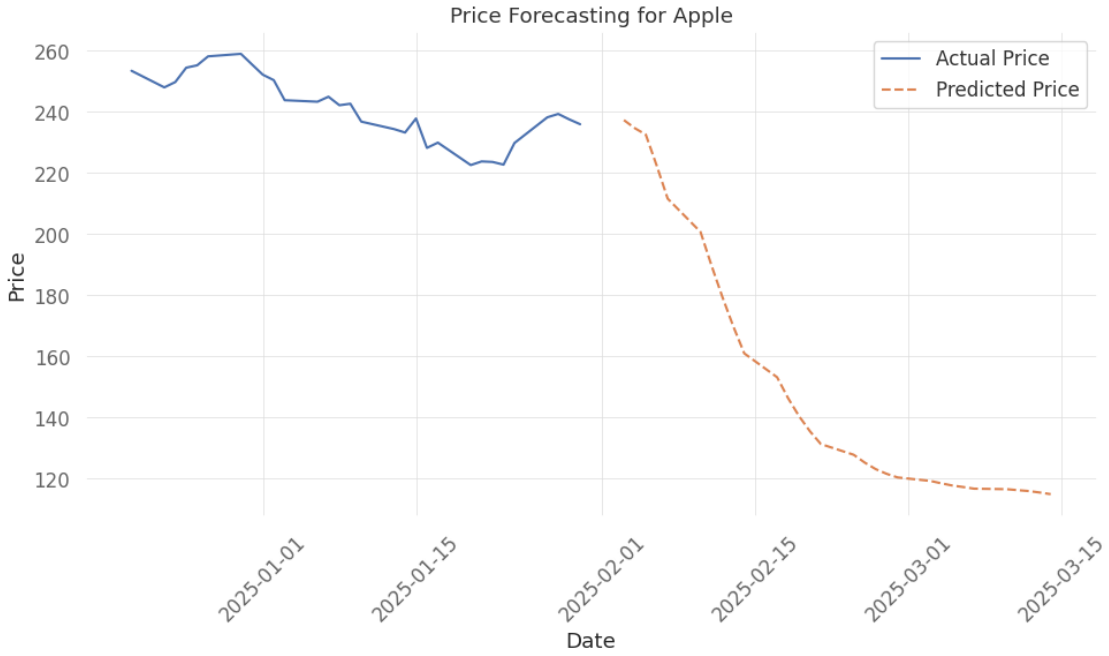
```
    df_scaled = scaler.fit_transform(df_close)  # Fit and transform only the␣
↪Close price column

    # Generate predictions iteratively
    for i in range(future_days):
        # Convert temp_input back to a NumPy array before reshaping
        x_input = np.array(temp_input[-n_steps:])  # Take last 50 time steps
        x_input = x_input.reshape((1, n_steps, 18))  # Reshape for LSTM input

        # Predict next step (normalized output)
        predicted = model.predict(x_input, verbose=0).flatten()  # Ensure 1D␣
↪shape

        # Adjust shape handling based on model output
        if predicted.shape[0] == 1:
            predicted_features = np.zeros(18)  # Create an empty array with 18␣
↪features
            predicted_features[0] = predicted[0]  # Store the predicted Close␣
↪price
        else:
            predicted_features = predicted

        # Append only the first feature (Close price) to output
        lstm_op.append(predicted_features[0])

        # Append full predicted feature set & maintain `n_steps` length
        temp_input.append(predicted_features.tolist())  # Append adjusted␣
↪feature array
        temp_input = temp_input[1:]  # Keep only last `n_steps` elements

    # Convert lstm_op to NumPy array for inverse transformation
    lstm_op = np.array(lstm_op).reshape(-1, 1)  # Reshape to (30, 1) for inverse␣
↪scaling

    # Prepare for denormalization: Create an array with the correct number of␣
↪features (11)
    n_features_in = df_scaled.shape[1]  # The number of features the scaler was␣
↪originally fitted on
    lstm_op_with_features = np.zeros((lstm_op.shape[0], n_features_in))  # (30,␣
↪1)
    lstm_op_with_features[:, 0] = lstm_op.flatten()  # Place predicted Close␣
↪price in the first column

    # Manually denormalize the predicted values (using the min and max of Close)
    X_min = scaler.data_min_[0]  # Minimum value of Close Price
    X_max = scaler.data_max_[0]  # Maximum value of Close Price
```

```
    # Manually denormalize the predictions (flattened lstm_op)
    denormalized_predictions = lstm_op.flatten() * (X_max - X_min) + X_min

    # Debugging: Check the first few denormalized values
    #print(f"Manually Denormalized Predictions: {denormalized_predictions[:5]}")

    # Create DataFrame with predicted dates & prices
    forecast_df = pd.DataFrame({'Date': forecast_dates, 'Predicted Price':␣
↪denormalized_predictions})

    # Ensure that forecast_df has the same length as forecast_dates
    if len(forecast_df) != len(forecast_dates):
        raise ValueError(f"The length of forecast_df ({len(forecast_df)}) does␣
↪not match the length of forecast_dates ({len(forecast_dates)}).")

    forecast_df.set_index('Date', inplace=True)

    print("\nFinal Forecasted Prices:")
    print(forecast_df)

    return forecast_df
```

```
========= Forecasting Next 30 Days =========

Final Forecasted Prices:
            Predicted Price
Date
2025-02-03       214.206466
2025-02-04       212.825532
2025-02-05       210.367815
2025-02-06       206.654896
2025-02-07       202.076864
2025-02-10       196.887045
2025-02-11       191.266472
2025-02-12       185.400750
2025-02-13       179.468563
2025-02-14       173.654463
2025-02-17       167.953298
2025-02-18       162.541444
2025-02-19       157.445528
2025-02-20       152.913761
2025-02-21       148.788183
2025-02-24       145.040797
2025-02-25       141.733982
2025-02-26       138.851994
```

```
2025-02-27        136.444322
2025-02-28        134.404816
2025-03-03        132.441650
2025-03-04        130.840858
2025-03-05        129.697097
2025-03-06        128.763397
2025-03-07        127.775751
2025-03-10        127.178794
2025-03-11        126.446670
2025-03-12        125.749048
2025-03-13        124.993689
2025-03-14        124.161724
```

The forecast predicts a steady decline in Apple's stock price from $214.21 on February 3, 2025, to $124.16 by March 14, 2025, indicating a significant drop of nearly $90 over the next 30 days. The decline is steeper in early February and gradually slows by March, suggesting potential stabilization. We will visualize this result:



This graph visualizes the actual and predicted stock prices for Apple.

- **Blue Line (Actual Price):** Represents the historical stock prices leading up to the forecast period.
- **Orange Dashed Line (Predicted Price):**Shows the predicted stock prices for the future period using a forecasting model.

The actual prices show fluctuations but remain relatively stable before forecasting begins. The predicted prices exhibit a sharp downward trend, indicating a potentially significant decline in Apple's stock price. This trend suggests that the forecasting model expects a bearish movement in the stock.

Possible Reasons for the Decline:

The model might have detected a downward pattern in historical data. External factors such as market trends, economic conditions, or news may have influenced the prediction.

8 Comparing Model Performances

Model	MAE	RMSE
Linear Regression	0.713	0.947
LSTM	5.67	4.83

When we analyze the results closely, we can see that Linear Regression outperforms LSTM in this particular scenario.Let's break it down metric by metric:

- **Mean Absolute Error (MAE)** represents the average absolute difference between the predicted and actual values. Since a lower MAE indicates better accuracy, Linear Regression significantly outperforms LSTM in this dataset.
- **Root Mean Squared Error (RMSE)** penalizes larger errors more than MAE. The higher RMSE for LSTM suggests that it makes larger errors compared to Linear Regression, which provides more stable predictions.

Why Do We Choose These Metrics for Stock Market Analysis?
Both MAE and RMSE are preferred for stock market prediction because they offer distinct advantages that are relevant to the nature of financial forecasting:

- Model Transparency: MAE provides a straightforward interpretation of how well the model is performing in terms of typical prediction errors, which helps stakeholders make informed decisions.
- Emphasis on Large Errors: RMSE penalizes large deviations more heavily, which is critical in stock market forecasting where errors, especially large ones, can lead to severe financial consequences.
- Research Validation: Both MAE and RMSE have been used extensively in research analysis for stock market prediction, reinforcing their validity as reliable metrics in this field. They provide complementary insights into a model's performance. [(1),(2),(3)]

9 Conclusion

In this notebook, we conducted a comprehensive stock market analysis using the Yahoo Finance library. Our approach involved the following key steps:

- **Data Retrieval**: We collected historical stock data from Yahoo Finance for analysis.
- **Exploratory Data Analysis (EDA)**: We visualized trends and patterns using various techniques to gain deeper insights.
- **Data Preprocessing**: We applied essential preprocessing steps, including normalization and data reshaping, to prepare the dataset for modeling.
- **Feature Engineering**: We extracted meaningful indicators such as **Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), Bollinger Bands,** and **Moving Averages** to enhance predictive performance.
- **Train-Test Split**: We divided the dataset into **70% training and 30% testing** to evaluate model performance effectively.
- **Modeling**: We implemented and trained two different machine learning models: **Long**

Short-Term Memory (LSTM) networks and **Linear Regression** for stock price prediction.

- **Performance Evaluation**: We compared the results of both models using key metrics, assessing their predictive accuracy and overall effectiveness.

This analysis provides valuable insights into stock price movements and helps in identifying the most suitable model for future predictions.

10 Future Work & Improvements

- **Hyperparameter Optimization:** We will fine-tune model parameters to enhance predictive accuracy and reduce errors.
- **Model Comparison:** In addition to **LSTM**, we will evaluate and compare performance with **ARIMA**, **XGBoost**, and **CNN** models to identify the most effective approach.
- **Feature Expansion:** We plan to incorporate additional financial indicators and external factors (e.g., market sentiment, economic trends) to improve model robustness.
- **Hybrid Model Approach:** Combining **LSTM** with **ARIMA**, **XGBoost**, and **CNN** to leverage both deep learning and traditional statistical methods for improved performance.

References

[1] Deepak Kumar, Pradeepta Kumar Sarangi, and Rajit Verma. A systematic review of stock market prediction using machine learning and statistical techniques. *Materials Today: Proceedings*, 49:3187–3191, 2022. Elsevier.

[2] Sevgi Sumerli Sarıgül, Ramazan Aaldeimir, and Hayrettin Uzunoğlu. Stock Market Index Prediction Using Machine Learning Techniques: Application of BIST Indices. *JOEEP: Journal of Emerging Economies and Policy*, 9(2):96–106. Seyfettin ERDOĞAN, Publisher.

[3] Nusrat Rouf, Majid Bashir Malik, Tasleem Arif, Sparsh Sharma, Saurabh Singh, Satyabrata Aich, and Hee-Cheol Kim. Stock market prediction using machine learning techniques: A decade survey on methodologies, recent developments, and future directions. *Electronics*, 10(21):2717, 2021. MDPI.