```
!pip install pandas yfinance scikit-learn matplotlib plotly
```

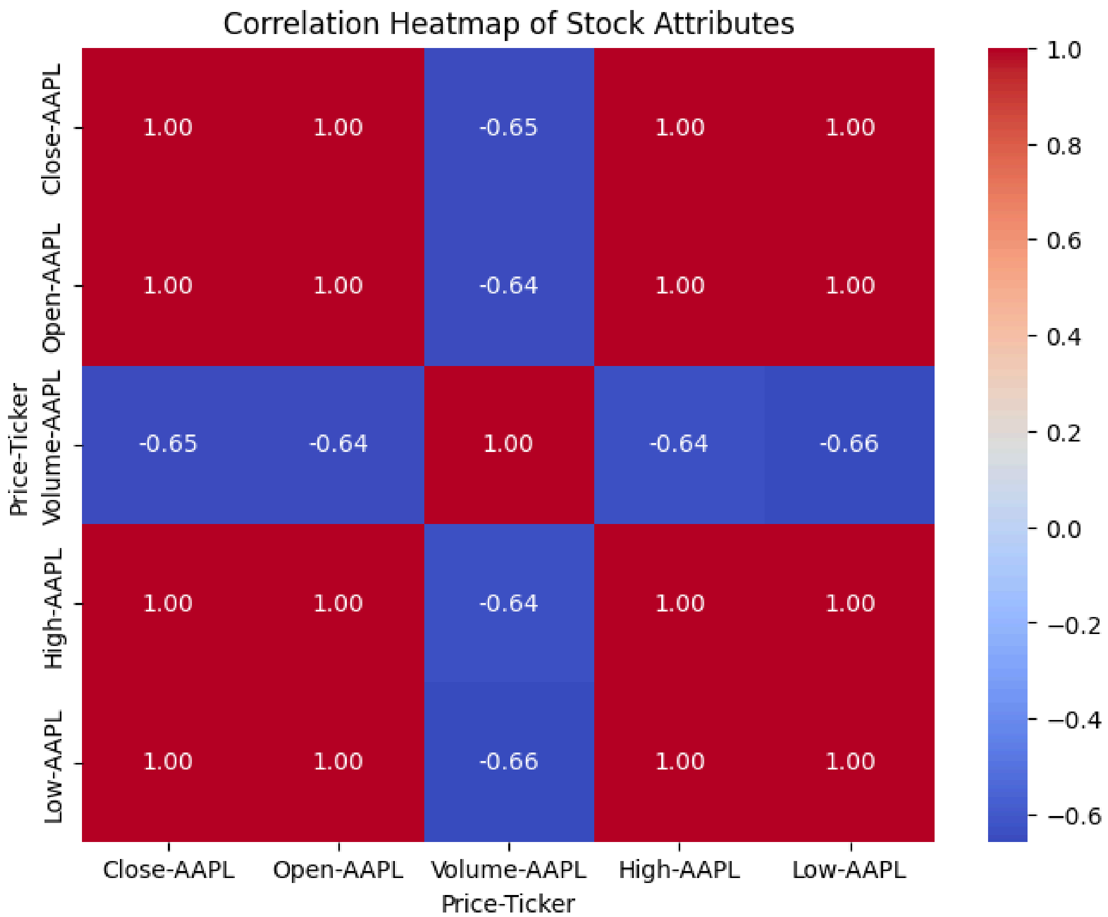
```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.2.2)
Requirement already satisfied: yfinance in /usr/local/lib/python3.10/dist-packages (0.2.50)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.5.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.8.0)
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (5.24.1)
Requirement already satisfied: numpy>=1.22.4 in /usr/local/lib/python3.10/dist-packages (from pandas
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from panda
Requirement already satisfied: requests>=2.31 in /usr/local/lib/python3.10/dist-packages (from yfina
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: lxml>=4.9.1 in /usr/local/lib/python3.10/dist-packages (from yfinance
Requirement already satisfied: platformdirs>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.10/dist-packages (from yf
Requirement already satisfied: peewee>=3.16.2 in /usr/local/lib/python3.10/dist-packages (from yfina
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: html5lib>=1.1 in /usr/local/lib/python3.10/dist-packages (from yfinan
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from mat
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplot
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from ma
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from ma
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matp
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplo
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from mat
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plot
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beauti
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.10/dist-packages (from html5lib>=1
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from html5li
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from request
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from r
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from r
```

```
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.preprocessing import StandardScaler , minmax_scale
```

```
#define a function to fetch stock data

def fetch_stock_data(stock_ticker, start_date="2020-01-01", end_date="2024-01-01"):
    try:
        return yf.download(stock_ticker, start=start_date, end=end_date).reset_index()
    except Exception as e:
        print(f"Error: {e}")
        return None

# Example usage
# df = fetch_stock_data("TSLA")
```

```
# df


def plot_correlation_heatmap(data):
    selected_columns = ['Close', 'Open', 'Volume', 'High', 'Low']
    corr = data[selected_columns].corr()  # Calculate correlation matrix

    # Display correlation values as a heatmap
    plt.figure(figsize=(8, 6))
    sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
    plt.title("Correlation Heatmap of Stock Attributes")
    plt.show()


stock_data = fetch_stock_data("AAPL")
plot_correlation_heatmap(stock_data)
```

```
[*********************100%***********************]  1 of 1 completed
```



Strong Positive Correlation (Red, Value = 1.00):

Close-AAPL, Open-AAPL, High-AAPL, and Low-AAPL all have a perfect positive correlation (1.00) with each other. This means these price-related attributes move in the same direction. If one increases, the others also tend to increase proportionally.

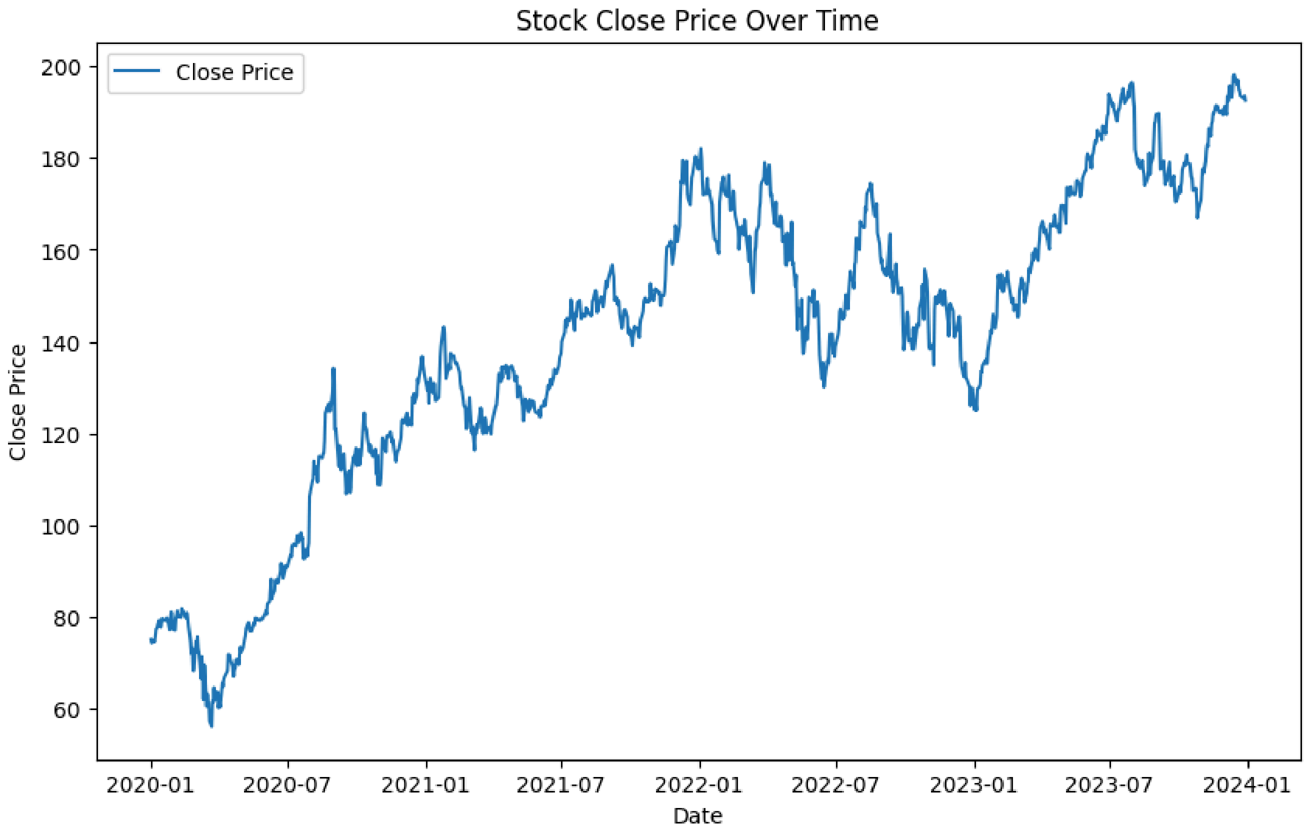
Reason: Since "Close," "Open," "High," and "Low" are directly related to stock prices, their high correlation is natural. They are part of the same data series for a stock.

Moderate Negative Correlation (Blue, ~ -0.64 to -0.66):

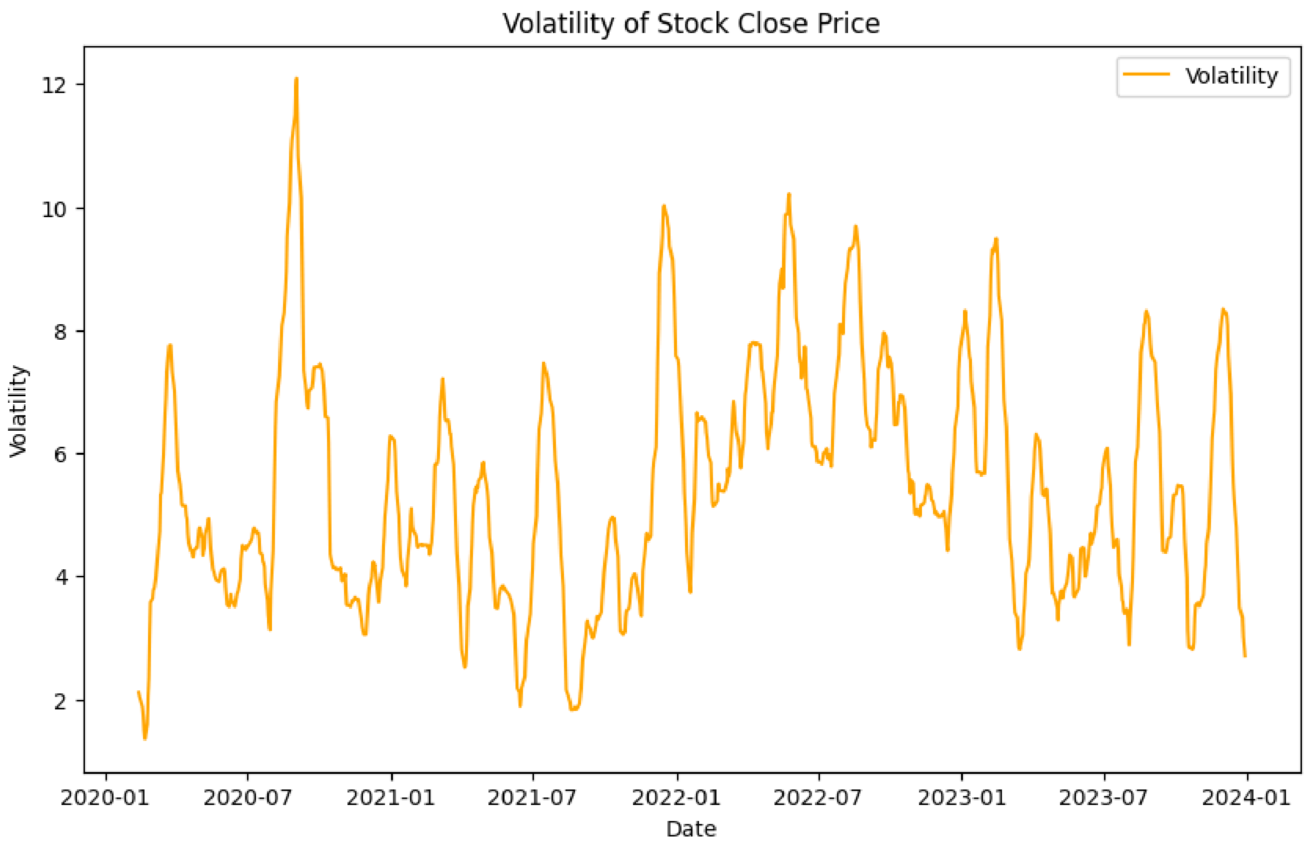
Volume-AAPL has a moderate negative correlation with all price attributes: Volume-AAPL vs Close-AAPL: -0.65 Volume-AAPL vs Open-AAPL: -0.64 Volume-AAPL vs High-AAPL: -0.64 Volume-AAPL vs Low-AAPL: -0.66

Interpretation: This indicates that when Volume (number of shares traded) increases, the price-related metrics (Open, Close, High, Low) tend to decrease slightly. This might suggest: Increased selling activity during high-volume days, which pushes prices down. Volatility in the market.

```
plt.figure(figsize=(10, 6))
plt.plot(stock_data['Date'], stock_data['Close'], label='Close Price')
plt.title('Stock Close Price Over Time')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```




```
# Calculate volatility (30-day rolling standard deviation)
stock_data['Volatility'] = stock_data['Close'].rolling(window=30).std()

# Plot volatility over time
plt.figure(figsize=(10, 6))
plt.plot(stock_data['Date'], stock_data['Volatility'], color='orange', label='Volatility')
plt.title('Volatility of Stock Close Price')
plt.xlabel('Date')
plt.ylabel('Volatility')
plt.legend()
plt.show()
```



Measuring volatility of the stock

High Peaks: Peaks (e.g., mid-2020, early 2022, mid-2022) indicate high volatility. High volatility often occurs during major events like market crashes, earnings reports, economic uncertainty, or external shocks (e.g., COVID-19 impact in 2020). Low Points: Periods where the volatility drops indicate more stable price movements. This could mean reduced uncertainty and consistent trading behavior. Pattern: The chart shows recurring cycles of volatility, suggesting that stock price fluctuations often follow events or trends. Practical Implications:

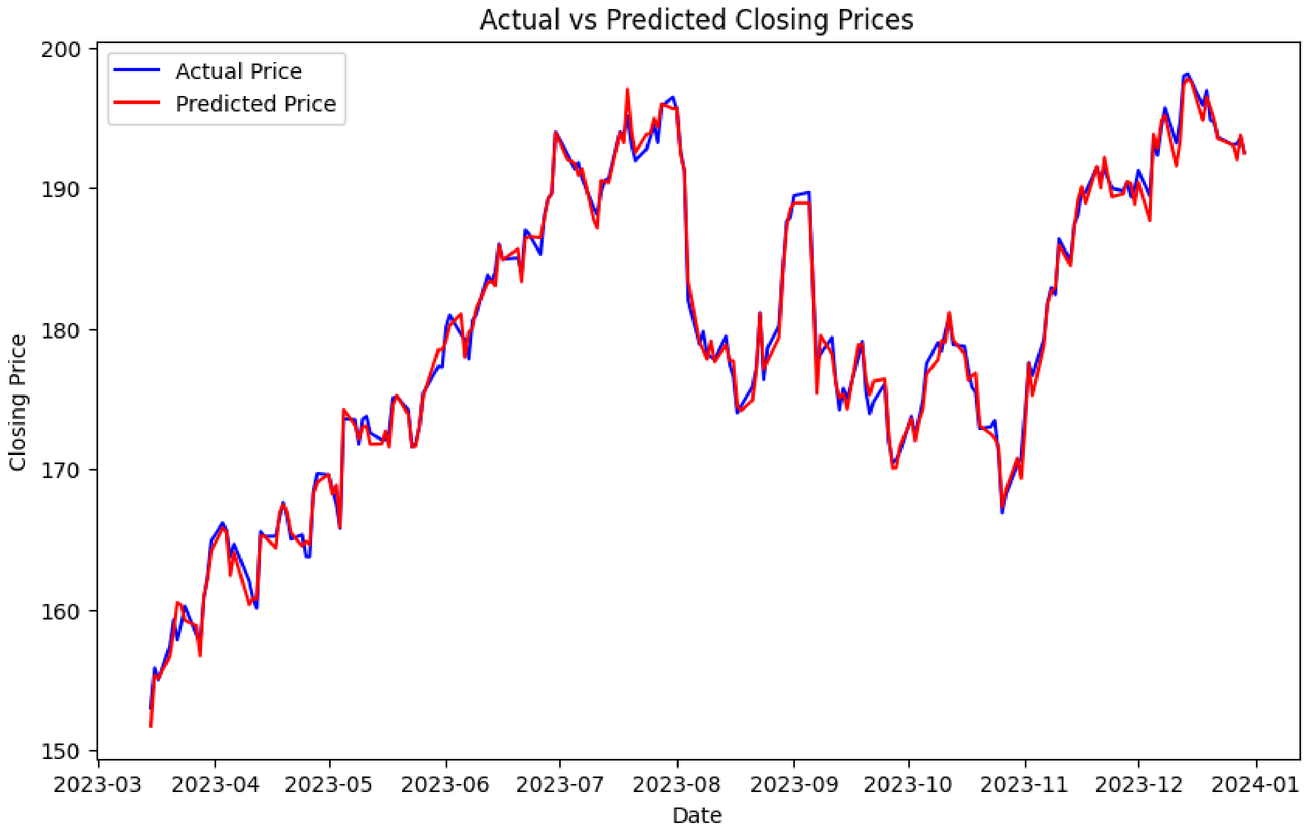
**High volatility* *→ Higher risk and potential for large price swings. Low volatility → More stable price trends, making the stock less risky.

**1. Predict Future Closing Price* *: Train a model to predict the next day's closing price based on historical features like Open, High, Low, Volume, and previous Close prices.

```
def prepare_data(stock_data):
    # Add "Previous Close" feature
    stock_data['Prev_Close'] = stock_data['Close'].shift(1)

    # Remove NaN from the first row
    stock_data = stock_data.dropna()

    # Features and target
    X = stock_data[['Prev_Close', 'Open', 'High', 'Low', 'Volume']]
    y = stock_data['Close']  # Target: next day's Close price

    return X, y, stock_data['Date']


from sklearn.metrics import r2_score
def Linear_Regression_check(X, y, dates):
```

```
    # Split the data into train and test sets
    X_train, X_test, y_train, y_test, dates_train, dates_test = train_test_split(
        X, y, dates, test_size=0.2, random_state=42, shuffle=False
    )

    # Initialize and train the model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Predict on the test set
    y_pred = model.predict(X_test)

    # Evaluate the model
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f"Mean Absolute Error: {mae:.2f}")
    print(f"R-squared: {r2:.2f}")

    # Plot actual vs predicted prices
    plt.figure(figsize=(10, 6))
    plt.plot(dates_test, y_test, label='Actual Price', color='blue')
    plt.plot(dates_test, y_pred, label='Predicted Price', color='red')
    plt.title("Actual vs Predicted Closing Prices")
    plt.xlabel("Date")
    plt.ylabel("Closing Price")
    plt.legend()
    plt.show()


if __name__ == "__main__":
    # Step 1: Fetch the stock data
    ticker = "AAPL"  # Example stock ticker: Apple Inc.
    stock_data = fetch_stock_data(ticker)

    # Step 2: Prepare the data
    X, y, dates = prepare_data(stock_data)

    # Step 3: Train and evaluate the model
    Linear_Regression_check(X, y, dates)
```

```
[*********************100%***********************]  1 of 1 completed
Mean Absolute Error: 0.62
R-squared: 0.99
```



Mean Absolute Error (MAE): 0.62

This is an excellent result since the error is very low. On average, the model's prediction deviates by only 0.62*fromtheactualclosingpriceswhichisminimalforstockpricesinthe*150-$200 range.

R-squared (R^2): *0.99 An R^2 value of 0.99 indicates the model explains 99% of the variance in the data. This is an extremely strong fit, meaning the model is performing very well.

SUCH AN ACCURATE MAE AND R2 ERROR COULD SHOW OVERFITTING , LETS CHECK FOR THAT

```
from sklearn.model_selection import TimeSeriesSplit, cross_val_score

tscv = TimeSeriesSplit(n_splits=5)
model = LinearRegression()
scores = cross_val_score(model, X, y, cv=tscv, scoring='r2')
print("Cross-validation R2 scores:", scores)
print("Mean R2 score:", scores.mean())
```

```
Cross-validation R2 scores: [0.9838707  0.99576748 0.9909305  0.99046169 0.99171093]
Mean R2 score: 0.9905482585553159
```

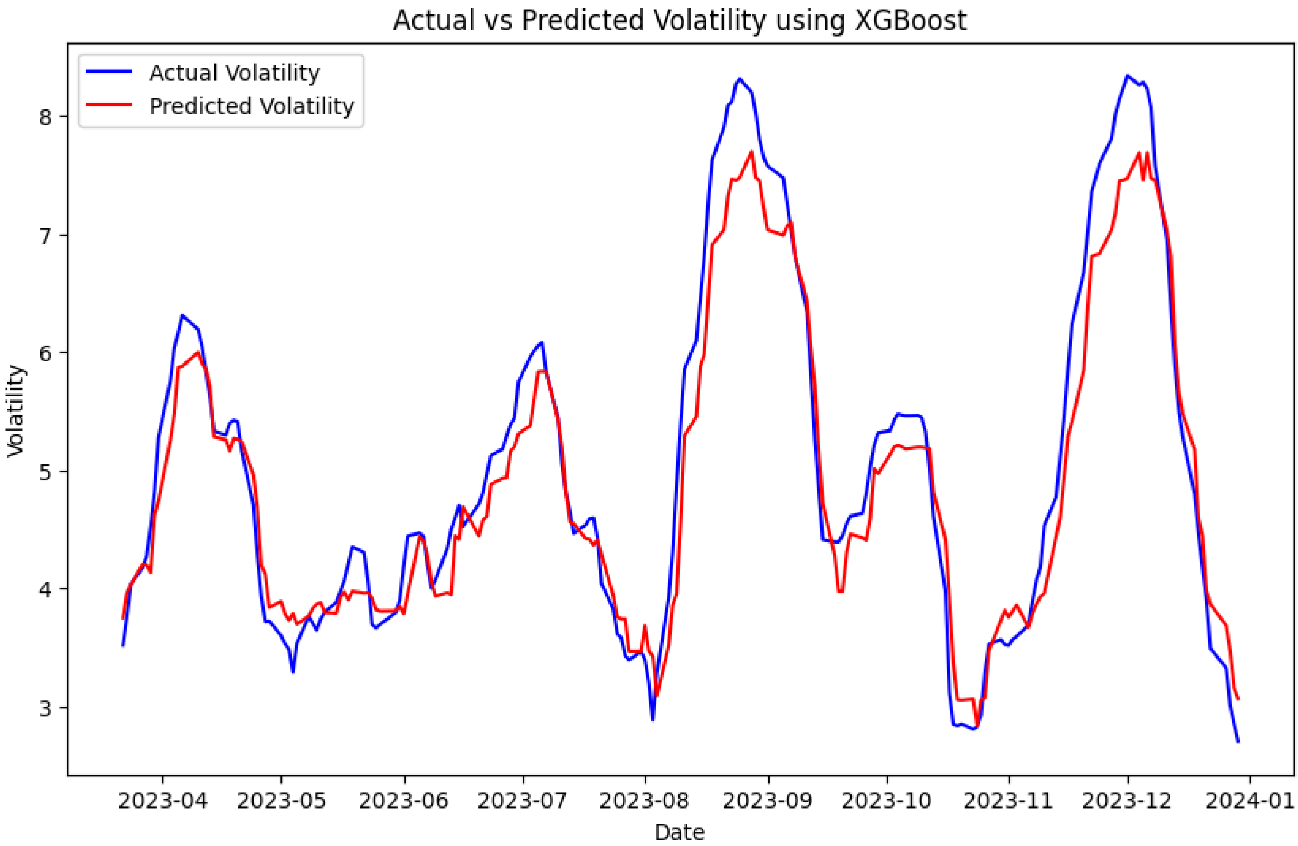
The Cross-Validation R^2 scores and the mean R^2 score (0.99) indicate that your model is performing consistently across different folds of the data. This reduces the likelihood of overfitting.

2. **Predict Stock Volatility** Goal: Predict the stock's future volatility (e.g., 30-day rolling standard deviation) based on past price movements and volume.

```
import xgboost as xgb
def prepare_volatility_data(stock_data):
    # Calculate 30-day rolling standard deviation as the target (Volatility)
    stock_data['Volatility'] = stock_data['Close'].rolling(window=30).std()

    # Add lagged features
    stock_data['Volatility_Lag1'] = stock_data['Volatility'].shift(1)

    # Add rolling statistics for Volume
    stock_data['Volume_SMA_10'] = stock_data['Volume'].rolling(window=10).mean()
    stock_data['Volume_SMA_30'] = stock_data['Volume'].rolling(window=30).mean()

    # Add SMA for prices
    stock_data['SMA_10'] = stock_data['Close'].rolling(window=10).mean()
    stock_data['SMA_20'] = stock_data['Close'].rolling(window=20).mean()

    # Drop rows with NaN
    stock_data = stock_data.dropna()

    # Features and target
    X = stock_data[['Prev_Close', 'Open', 'High', 'Low', 'Volume',
                    'SMA_10', 'SMA_20', 'Volume_SMA_10', 'Volume_SMA_30', 'Volatility_Lag1']]
    y = stock_data['Volatility']
    return X, y, stock_data['Date']

# Train and evaluate XGBoost model
def train_and_evaluate_xgboost(X, y, dates):
    # Sequential Train-Test Split
    X_train, X_test, y_train, y_test, dates_train, dates_test = train_test_split(
        X, y, dates, test_size=0.2, random_state=42, shuffle=False
    )

    # Train XGBoost Regressor
    model = xgb.XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
    model.fit(X_train, y_train)

    # Predictions
    y_pred = model.predict(X_test)

    # Evaluation
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f"Mean Absolute Error: {mae:.4f}")
    print(f"R-squared: {r2:.4f}")

    # Plot Actual vs Predicted Volatility
    plt.figure(figsize=(10, 6))
    plt.plot(dates_test, y_test, label='Actual Volatility', color='blue')
    plt.plot(dates_test, y_pred, label='Predicted Volatility', color='red')
    plt.title("Actual vs Predicted Volatility using XGBoost")
    plt.xlabel("Date")
    plt.ylabel("Volatility")
    plt.legend()
    plt.show()
```

```
if __name__ == "__main__":
    ticker = "AAPL"  # Example stock
    stock_data = fetch_stock_data(ticker)

    # Add previous day's close for consistency with earlier features
    stock_data['Prev_Close'] = stock_data['Close'].shift(1)

    # Prepare features and target
    X, y, dates = prepare_volatility_data(stock_data)

    # Train and evaluate the model
    train_and_evaluate_xgboost(X, y, dates)
```

```
[*********************100%***********************]  1 of 1 completed
Mean Absolute Error: 0.3124
R-squared: 0.9311
```




```
stock_data
```

Price	Date	Adj Close	Close	High	Low	Open	Volume	Prev_Close	Volat
Ticker		AAPL	AAPL	AAPL	AAPL	AAPL	AAPL		
0	2020-	72.796013	75.087502	75.150002	73.797501	74.059998	135480400	NaN	