Stock Price Prediction Using Economic Indicators

Academic year: 2024-2025

Semester: Fall Semester

By Team-20

1. SAMMETA GURU SAI NANDAN - 22BCE9614
2. AAVULA RAVI KUMAR – 22BCE20046
3. PALUVURI SUJAN - 22BCE9907
4. B.SUHITH – 22BCE8590

Under the guidance of

S.Bharath Bhushan

Department of Computer Science and Engineering

VIT-AP UNIVERSITY

Abstract

This project explores the application of machine learning models to forecast stock price movements based on economic indicators, combining traditional time-series analysis with advanced predictive techniques. By analyzing historical stock data from **Adani Ports** and a dataset of global economic indicators, we aim to identify patterns and relationships that may inform stock price fluctuations. The methodology includes a multi-step approach: data preprocessing, exploratory data analysis (EDA), feature engineering, and the deployment of supervised learning models (such as Linear Regression, Random Forest, and Support Vector Machine) alongside Long Short-Term Memory (LSTM) for time-series forecasting. For each model, performance is evaluated using key metrics including Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and R-squared. The interpretability of results is enhanced with SHAP values to identify the impact of various economic indicators on stock predictions, and model comparison underscores the trade-offs between accuracy and interpretability. Findings suggest that certain indicators, like GDP growth and interest rates, have significant predictive power, offering valuable insights for investors and policymakers. This study provides a foundation for further research into integrating complex economic dynamics and advanced deep learning architectures for more precise stock market predictions.

Introduction

The stock market, a complex and dynamic system, reflects the economic health and investor sentiment of a nation. It is influenced by a range of factors, including corporate performance, geopolitical events, and broader economic indicators such as inflation rates, GDP growth, unemployment rates, and interest rates. Accurately forecasting stock prices has long been of interest to investors, policymakers, and economists as it offers valuable insights into potential economic trends, risk management, and strategic financial planning.

In recent years, advancements in machine learning and data science have opened new possibilities for analyzing and predicting stock prices by leveraging large datasets and sophisticated algorithms. Traditional models, such as Linear Regression, have been commonly used in this domain. However, complex methods, including ensemble models like Random Forest and time-series-specific approaches like Long Short-Term Memory (LSTM) neural networks, are now providing more accurate and dynamic results, especially when applied to sequential financial data.

This project aims to develop a predictive model that utilizes historical stock prices from **Adani Ports**, alongside global economic indicators from 2010 to 2023, to forecast stock price movements. By analyzing the interplay between various economic indicators and stock prices, this research seeks to uncover insights that may benefit investors seeking to make data-driven decisions. The primary objectives of this study are to evaluate the predictive power of different economic indicators, compare the effectiveness of various machine learning models, and identify which models and indicators can most reliably inform stock price fluctuations.

This report provides an overview of the methodology, from data preprocessing and feature engineering to model training and evaluation. Additionally, we discuss interpretability techniques, such as SHAP (SHapley Additive exPlanations) values, to explain the impact of each economic indicator on stock predictions, offering an intuitive understanding of the model's inner workings. The findings highlight significant economic factors and present an assessment of the models' predictive capabilities, contributing to a more comprehensive understanding of stock price prediction through economic indicators.

Literature Review

The prediction of stock prices is a longstanding challenge within finance and data science, with researchers constantly seeking more accurate, insightful methodologies. Various studies have emphasized different techniques and factors, including economic indicators, machine learning, and interpretability, as means to enhance stock price prediction. This section reviews the relevant literature on the approaches used for stock market prediction, particularly the integration of economic indicators and machine learning models.

1. **Traditional Time-Series Analysis in Stock Prediction**
   Early studies on stock price forecasting relied on statistical time-series models, such as the Autoregressive Integrated Moving Average (ARIMA) model, proposed by Box and Jenkins (1970). These models operate under the assumption that stock price movements are predominantly linear, which limits their effectiveness in capturing complex market dynamics. Despite these limitations, ARIMA and similar methods provided a foundation for stock price analysis, especially for short-term forecasts (Fama, 1970).

2. **Economic Indicators as Predictors of Stock Performance**
   A significant body of research has focused on the influence of economic indicators on stock prices. Macroeconomic variables, such as interest rates, inflation, and GDP growth, are often seen as critical predictors due to their direct relationship with overall economic health (Chen, Roll, & Ross, 1986; Fama, 1981). For example, periods of rising interest rates often lead to reduced stock returns as borrowing costs increase, while high GDP growth rates may indicate positive market performance. Studies by Rapach and Zhou (2010) have further supported the effectiveness of using economic indicators to forecast stock returns, highlighting the impact of global macroeconomic events.

3. **Machine Learning Approaches to Stock Price Prediction**
   In recent years, machine learning models have been introduced to better capture the nonlinear relationships in stock data. Linear regression and other traditional methods have been joined by more complex algorithms such as Decision Trees and Random Forests, which can handle a variety of predictor variables without the same level of manual adjustment (Breiman, 2001). Research by Bollen et al. (2011) demonstrated that these models could effectively incorporate sentiment and economic indicators to predict stock price trends, which otherwise might be difficult to capture in linear models.

4. **Neural Networks and Deep Learning for Sequential Financial Data**
   Neural networks, and particularly Long Short-Term Memory (LSTM) networks, have gained traction for stock price prediction due to their ability to model sequential dependencies (Hochreiter & Schmidhuber, 1997). LSTMs can handle data with temporal relationships, making them well-suited for time-series forecasting where past values influence future prices. A study by Selvin et al. (2017) found that LSTMs performed better than traditional time-series models in predicting stock prices, especially when trained on data that includes economic indicators. This approach reflects the dynamic and often unpredictable nature of financial markets.

5. **Model Interpretability and Explainability in Finance**
   The need for model interpretability has led researchers to seek tools that explain machine learning predictions. SHAP (SHapley Additive exPlanations) values, developed by Lundberg and Lee (2017), offer a solution by allowing users to see how each feature impacts a model's predictions. This has particular relevance for financial applications, where it is essential to understand not only the accuracy of predictions but also the factors driving them. SHAP values can illuminate the relationship between economic indicators and stock prices, offering insights that go beyond raw predictive accuracy.

6. **Comparative Studies: Traditional vs. Machine Learning Models**
   Several studies have compared traditional econometric models with machine learning techniques. For example, Fischer and Krauss (2018) demonstrated that LSTMs and other deep learning models provide a significant improvement over traditional methods in capturing the complexities of stock market data. Cavalcante et al. (2016) similarly found that machine learning models outperformed statistical models, especially when working with large, multidimensional datasets that include economic indicators.

Implementation:

1. Data Collection and Preprocessing:

- **Data Sources**:
    - Stock price data-Adani Ports.
    - Economic indicators data.
- **Data Collection:**
  The datasets for stock prices and economic indicators were uploaded using Google Colab's file upload feature.

```python
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Upload the datasets
from google.colab import files
uploaded = files.upload()

# Verify the uploaded files
import os
print("Uploaded Files:")
print(os.listdir())  # Check the filenames

# Load the datasets using the correct filenames
stock_data = pd.read_csv('ADANIPORTS.csv')
economic_data = pd.read_csv('economic_indicators_dataset_2010_2023.csv')

# Explore the data
print("Stock Data Preview:")
print(stock_data.head())
print("\nEconomic Data Preview:")
print(economic_data.head())
```

```
Choose Files  No file chosen          Upload widget is only available when the cell has been execute
Saving ADANIPORTS.csv to ADANIPORTS (2).csv
Saving economic_indicators_dataset_2010_2023.csv to economic_indicators_dataset_2010_
Uploaded Files:
['.config', 'ADANIPORTS (2).csv', 'economic_indicators_dataset_2010_2023.csv', 'econ
Stock Data Preview:
         Date      Symbol Series  Prev Close    Open     High    Low   Last  \
0  2007-11-27  MUNDRAPORT     EQ      440.00  770.00  1050.00  770.0  959.0
1  2007-11-28  MUNDRAPORT     EQ      962.90  984.00   990.00  874.0  885.0
2  2007-11-29  MUNDRAPORT     EQ      893.90  909.00   914.75  841.0  887.0
3  2007-11-30  MUNDRAPORT     EQ      884.20  890.00   958.00  890.0  929.0
4  2007-12-03  MUNDRAPORT     EQ      921.55  939.75   995.00  922.0  980.0

    Close    VWAP    Volume      Turnover  Trades  Deliverable Volume  \
0  962.90  984.72  27294366  2.687719e+15     NaN             9859619
1  893.90  941.38   4581338  4.312765e+14     NaN             1453278
2  884.20  888.09   5124121  4.550658e+14     NaN             1069678
3  921.55  929.17   4609762  4.283257e+14     NaN             1260913
4  969.30  965.65   2977470  2.875200e+14     NaN              816123

   %Deliverble
0       0.3612
1       0.3172
2       0.2088
3       0.2735
4       0.2741

Economic Data Preview:
         Date Country  Inflation Rate (%)  GDP Growth Rate (%)  \
0  2010-01-31  Brazil                1.23                 0.69
1  2010-01-31  France                6.76                 2.59
2  2010-01-31     USA                7.46                 4.84
3  2010-02-28  Brazil                5.43                 0.31
4  2010-02-28  Canada                0.69                -0.52

   Unemployment Rate (%)  Interest Rate (%)  Stock Index Value
0                  10.48               7.71           21748.85
1                   4.27               7.39           10039.56
2                   2.64               6.39           13129.10
3                   8.26               6.09           23304.58
4                  11.92              -0.51           16413.03
```

- **Preprocessing**:

  The datasets were loaded and inspected for missing values. Missing values in numeric columns were replaced with the mean of each respective column to ensure clean and complete data for analysis.

```python
#loading the data and performing the data cleaning observing the missing values and replacing the missing values
import pandas as pd

# Load the data
stock_data = pd.read_csv('ADANIPORTS (2).csv')
economic_data = pd.read_csv('economic_indicators_dataset_2010_2023.csv')

# Display initial data previews
print("Stock Data Preview:")
print(stock_data.head())
print("\nEconomic Data Preview:")
print(economic_data.head())

# Check the info for both datasets
print("\nStock Data Info:")
print(stock_data.info())
print("\nEconomic Data Info:")
print(economic_data.info())

# Check for missing values
print("\nMissing Values in Stock Data:")
print(stock_data.isnull().sum())
print("\nMissing Values in Economic Data:")
print(economic_data.isnull().sum())

# Handle missing values (fill with mean for numeric columns)
# For stock data, select only numeric columns
numeric_stock_columns = stock_data.select_dtypes(include=['float64', 'int64']).columns
stock_data[numeric_stock_columns] = stock_data[numeric_stock_columns].fillna(stock_data[numeric_stock_columns].mean())

# For economic data, select only numeric columns
numeric_economic_columns = economic_data.select_dtypes(include=['float64', 'int64']).columns
economic_data[numeric_economic_columns] = economic_data[numeric_economic_columns].fillna(economic_data[numeric_economic_columns].mean())

# Verify that missing values are handled
print("\nMissing Values in Stock Data After Filling:")
print(stock_data.isnull().sum())
print("\nMissing Values in Economic Data After Filling:")
print(economic_data.isnull().sum())
```

```
Stock Data Preview:
         Date      Symbol Series  Prev Close    Open     High    Low   Last  \
0  2007-11-27  MUNDRAPORT     EQ      440.00  770.00  1050.00  770.0  959.0
1  2007-11-28  MUNDRAPORT     EQ      962.90  984.00   990.00  874.0  885.0
2  2007-11-29  MUNDRAPORT     EQ      893.90  909.00   914.75  841.0  887.0
3  2007-11-30  MUNDRAPORT     EQ      884.20  890.00   958.00  890.0  929.0
4  2007-12-03  MUNDRAPORT     EQ      921.55  939.75   995.00  922.0  980.0

    Close    VWAP    Volume      Turnover  Trades  Deliverable Volume  \
0  962.90  984.72  27294366  2.687719e+15     NaN             9859619
1  893.90  941.38   4581338  4.312765e+14     NaN             1453278
2  884.20  888.09   5124121  4.550658e+14     NaN             1069678
3  921.55  929.17   4609762  4.283257e+14     NaN             1260913
4  969.30  965.65   2977470  2.875200e+14     NaN              816123

   %Deliverble
0       0.3612
1       0.3172
2       0.2088
3       0.2735
4       0.2741

Economic Data Preview:
         Date Country  Inflation Rate (%)  GDP Growth Rate (%)  \
0  2010-01-31  Brazil                1.23                 0.69
1  2010-01-31  France                6.76                 2.59
2  2010-01-31     USA                7.46                 4.84
3  2010-02-28  Brazil                5.43                 0.31
4  2010-02-28  Canada                0.69                -0.52

   Unemployment Rate (%)  Interest Rate (%)  Stock Index Value
0                  10.48               7.71           21748.85
1                   4.27               7.39           10039.56
2                   2.64               6.39           13129.10
3                   8.26               6.09           23304.58
4                  11.92              -0.51           16413.03

Stock Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3322 entries, 0 to 3321
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Date                3322 non-null   object
 1   Symbol              3322 non-null   object
 2   Series              3322 non-null   object
 3   Prev Close          3322 non-null   float64
 4   Open                3322 non-null   float64
 5   High                3322 non-null   float64
 6   Low                 3322 non-null   float64
 7   Last                3322 non-null   float64
 8   Close               3322 non-null   float64
 9   VWAP                3322 non-null   float64
 10  Volume              3322 non-null   int64
 11  Turnover            3322 non-null   float64
 12  Trades              2456 non-null   float64
 13  Deliverable Volume  3322 non-null   int64
 14  %Deliverble         3322 non-null   float64
dtypes: float64(10), int64(2), object(3)
memory usage: 389.4+ KB
None
```

```
Economic Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 7 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Date                   500 non-null    object
 1   Country                500 non-null    object
 2   Inflation Rate (%)     500 non-null    float64
 3   GDP Growth Rate (%)    500 non-null    float64
 4   Unemployment Rate (%)  500 non-null    float64
 5   Interest Rate (%)      500 non-null    float64
 6   Stock Index Value      500 non-null    float64
dtypes: float64(5), object(2)
memory usage: 27.5+ KB
None

Missing Values in Stock Data:
Date                    0
Symbol                  0
Series                  0
Prev Close              0
Open                    0
High                    0
Low                     0
Last                    0
Close                   0
VWAP                    0
Volume                  0
Turnover                0
Trades                866
Deliverable Volume      0
%Deliverble             0
dtype: int64

Missing Values in Economic Data:
Date                     0
Country                  0
Inflation Rate (%)       0
GDP Growth Rate (%)      0
Unemployment Rate (%)    0
Interest Rate (%)        0
Stock Index Value        0
dtype: int64

Missing Values in Stock Data After Filling:
Date                  0
Symbol                0
Series                0
Prev Close            0
Open                  0
High                  0
Low                   0
Last                  0
Close                 0
VWAP                  0
Volume                0
Turnover              0
Trades                0
Deliverable Volume    0
%Deliverble           0
dtype: int64

Missing Values in Economic Data After Filling:
Date                     0
Country                  0
Inflation Rate (%)       0
GDP Growth Rate (%)      0
Unemployment Rate (%)    0
Interest Rate (%)        0
Stock Index Value        0
dtype: int64
```

2. Data Warehousing:

- ## Data Transformation
  The Date columns in both stock and economic datasets were converted to datetime format. Year and Month columns were extracted from the Date column for both datasets, enabling time-based analysis

```python
# Transform Data

# Convert 'Date' columns to datetime
stock_data['Date'] = pd.to_datetime(stock_data['Date'])
economic_data['Date'] = pd.to_datetime(economic_data['Date'])

# Create Year and Month columns for stock data
stock_data['Year'] = stock_data['Date'].dt.year
stock_data['Month'] = stock_data['Date'].dt.month

# Create Year and Month columns for economic data
economic_data['Year'] = economic_data['Date'].dt.year
economic_data['Month'] = economic_data['Date'].dt.month

# Optional: Display the transformed data
print("\nTransformed Stock Data Preview:")
print(stock_data[['Date', 'Year', 'Month']])
print("\nTransformed Economic Data Preview:")
print(economic_data[['Date', 'Year', 'Month']])

# Continue with the next steps, such as creating dimension and
# fact tables...
```

```
Transformed Stock Data Preview:
           Date  Year  Month
0    2007-11-27  2007     11
1    2007-11-28  2007     11
2    2007-11-29  2007     11
3    2007-11-30  2007     11
4    2007-12-03  2007     12
...         ...   ...    ...
3317 2021-04-26  2021      4
3318 2021-04-27  2021      4
3319 2021-04-28  2021      4
3320 2021-04-29  2021      4
3321 2021-04-30  2021      4

[3322 rows x 3 columns]

Transformed Economic Data Preview:
          Date  Year  Month
0   2010-01-31  2010      1
1   2010-01-31  2010      1
2   2010-01-31  2010      1
3   2010-02-28  2010      2
4   2010-02-28  2010      2
..         ...   ...    ...
495 2023-08-31  2023      8
496 2023-08-31  2023      8
497 2023-09-30  2023      9
498 2023-10-31  2023     10
499 2023-11-30  2023     11

[500 rows x 3 columns]
```

- **Design a Data Warehouse**:
  - Define fact tables for stock prices and economic indicators.
  - Define dimension tables for attributes like date, stock symbols, and economic indicator types.

```python
# creating dimension and fact tables
# Create Date Dimension
date_dimension = stock_data[['Date']].drop_duplicates()
date_dimension['Year'] = stock_data['Year']
date_dimension['Month'] = stock_data['Month']
date_dimension['Day'] = stock_data['Date'].dt.day
date_dimension['Quarter'] = stock_data['Date'].dt.quarter
date_dimension['Weekday'] = stock_data['Date'].dt.day_name()
date_dimension = date_dimension.drop_duplicates()  # Ensure unique dates
date_dimension.to_csv('date_dimension.csv', index=False)  # Save to CSV for reference
print("\nDate Dimension Preview:")
print(date_dimension.head())

# Create Stock Symbol Dimension
# Since we need to create a placeholder for 'Company Name' and other columns, adjust according to your dataset
if 'Symbol' in stock_data.columns:
    stock_symbol_dimension = stock_data[['Symbol']].drop_duplicates()
    stock_symbol_dimension['Company Name'] = "Company Name Placeholder"  # Placeholder for missing data
    stock_symbol_dimension['Industry'] = "Industry Placeholder"  # Placeholder for Industry
    stock_symbol_dimension['Sector'] = "Sector Placeholder"  # Placeholder for Sector
    stock_symbol_dimension.to_csv('stock_symbol_dimension.csv', index=False)  # Save to CSV for reference
    print("\nStock Symbol Dimension Preview:")
    print(stock_symbol_dimension.head())
else:
    print("Error: 'Symbol' column not found in stock_data.")

# Create Country Dimension
# Adjust to match the country column in economic_data if necessary
if 'Country' in economic_data.columns:
    country_dimension = economic_data[['Country']].drop_duplicates()
    country_dimension['Country Name'] = country_dimension['Country']  # Rename for clarity
    country_dimension.to_csv('country_dimension.csv', index=False)  # Save to CSV for reference
    print("\nCountry Dimension Preview:")
    print(country_dimension.head())
else:
    print("Error: 'Country' column not found in economic_data.")

# Create Stock Prices Fact Table
stock_prices_fact = stock_data[['Date', 'Symbol', 'Open', 'Close', 'High', 'Low', 'Volume']].copy()
stock_prices_fact.to_csv('stock_prices_fact.csv', index=False)  # Save to CSV for reference
print("\nStock Prices Fact Table Preview:")
print(stock_prices_fact.head())

# Create Economic Indicators Fact Table
# Update with actual names of economic indicators based on your dataset
economic_indicators_fact = economic_data[['Date', 'Country', 'Inflation Rate (%)', 'GDP Growth Rate (%)', 'Unemployment Rate (%)', 'Interest Rate (%)','Stock Index Value']].copy()
economic_indicators_fact.to_csv('economic_indicators_fact.csv', index=False)  # Save to CSV for reference
print("\nEconomic Indicators Fact Table Preview:")
print(economic_indicators_fact.head())
```

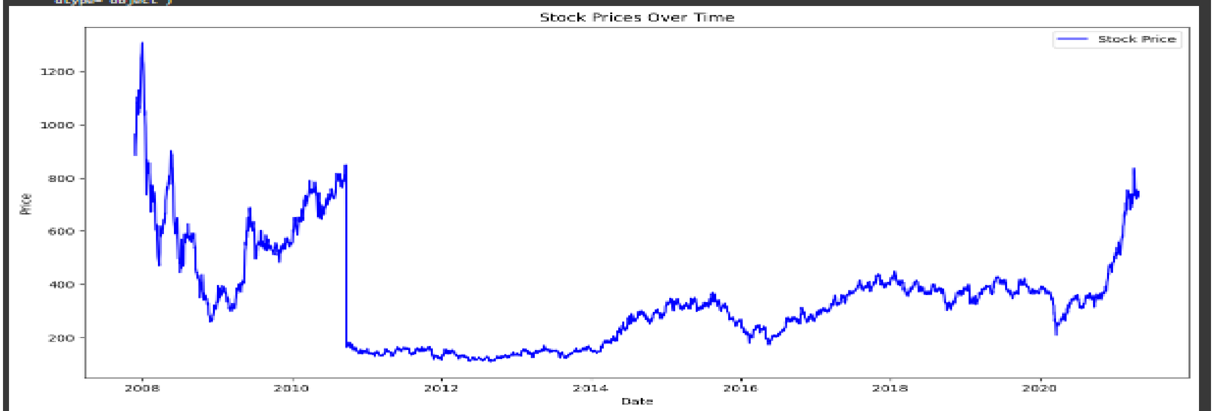
```
Date Dimension Preview:
        Date  Year  Month  Day  Quarter    Weekday
0 2007-11-27  2007     11   27        4    Tuesday
1 2007-11-28  2007     11   28        4  Wednesday
2 2007-11-29  2007     11   29        4   Thursday
3 2007-11-30  2007     11   30        4     Friday
4 2007-12-03  2007     12    3        4     Monday

Stock Symbol Dimension Preview:
          Symbol              Company Name              Industry  \
0     MUNDRAPORT  Company Name Placeholder  Industry Placeholder
1023  ADANIPORTS  Company Name Placeholder  Industry Placeholder

                  Sector
0     Sector Placeholder
1023  Sector Placeholder

Country Dimension Preview:
  Country Country Name
0  Brazil       Brazil
1  France       France
2     USA          USA
4  Canada       Canada
6   Japan        Japan

Stock Prices Fact Table Preview:
        Date      Symbol   Open   Close     High    Low    Volume
0 2007-11-27  MUNDRAPORT 770.00  962.90  1050.00  770.0  27294366
1 2007-11-28  MUNDRAPORT 984.00  893.90   990.00  874.0   4581338
2 2007-11-29  MUNDRAPORT 909.00  884.20   914.75  841.0   5124121
3 2007-11-30  MUNDRAPORT 890.00  921.55   958.00  890.0   4609762
4 2007-12-03  MUNDRAPORT 939.75  969.30   995.00  922.0   2977470

Economic Indicators Fact Table Preview:
        Date Country  Inflation Rate (%)  GDP Growth Rate (%)  \
0 2010-01-31  Brazil                1.23                 0.69
1 2010-01-31  France                6.76                 2.59
2 2010-01-31     USA                7.46                 4.84
3 2010-02-28  Brazil                5.43                 0.31
4 2010-02-28  Canada                0.69                -0.52

   Unemployment Rate (%)  Interest Rate (%)  Stock Index Value
0                  10.48               7.71           21748.85
1                   4.27               7.39           10039.56
2                   2.64               6.39           13129.10
3                   8.26               6.09           23304.58
4                  11.92              -0.51           16413.03
```

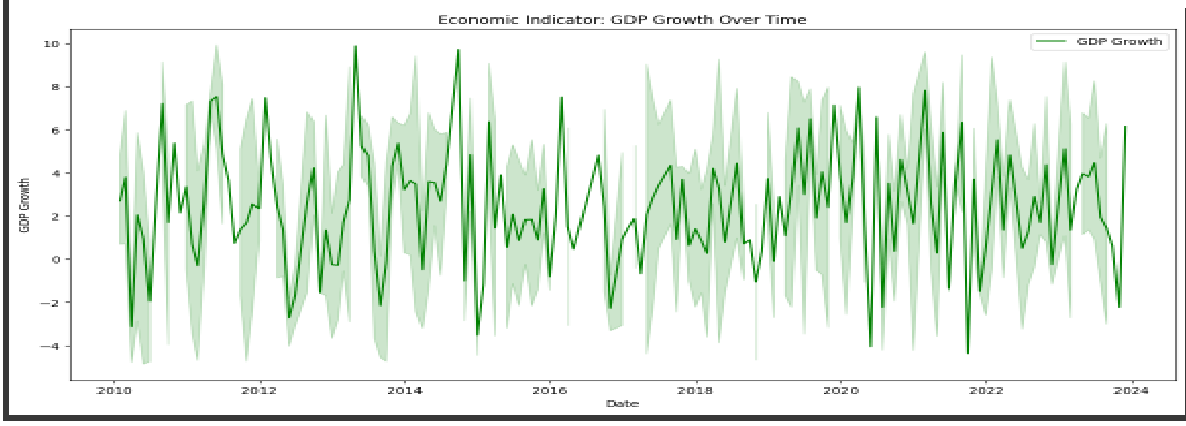
3. Exploratory Data Analysis (EDA):

- **Visualization**:
  - Time series plots of stock prices and economic indicators.

```
#VISUALISATION
# TIME SERIES PLOT
import matplotlib.pyplot as plt
import seaborn as sns

# Check the column names in economic data
print(economic_data.columns)

# Time Series Plot for Stock Prices
plt.figure(figsize=(14, 7))
sns.lineplot(data=stock_data, x='Date', y='Close', label='Stock Price', color='blue')
plt.title('Stock Prices Over Time')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()

# Time Series Plot for Economic Indicators
# Replace 'GDP Growth Rate (%)' with the actual column name from your economic_data
plt.figure(figsize=(14, 7))
sns.lineplot(data=economic_data, x='Date', y='GDP Growth Rate (%)', label='GDP Growth', color='green')
plt.title('Economic Indicator: GDP Growth Over Time')
plt.xlabel('Date')
plt.ylabel('GDP Growth')
plt.legend()
plt.show()
```

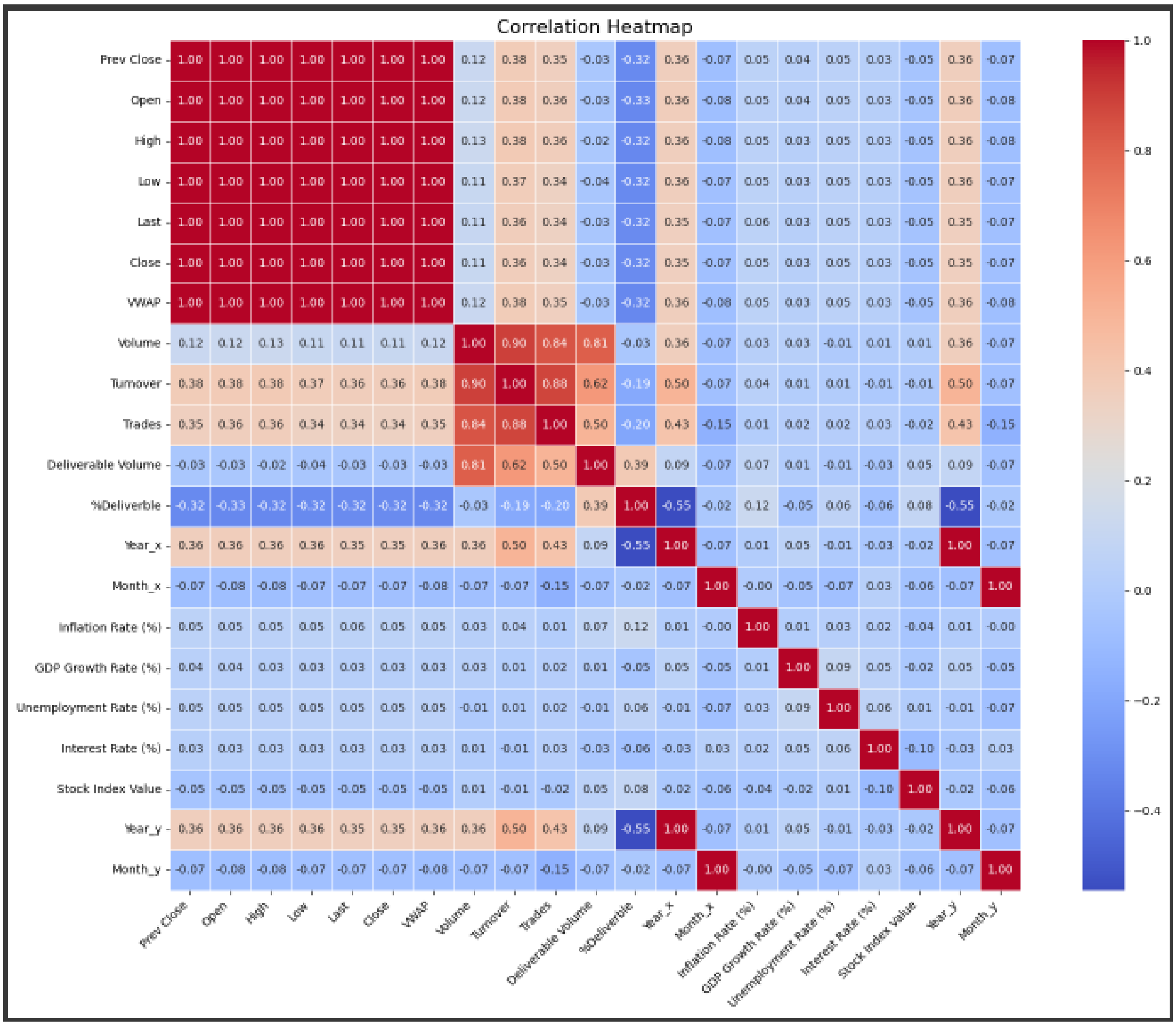
```
Index(['Date', 'Country', 'Inflation Rate (%)', 'GDP Growth Rate (%)',
       'Unemployment Rate (%)', 'Interest Rate (%)', 'Stock Index Value',
       'Year', 'Month'],
      dtype='object')
```





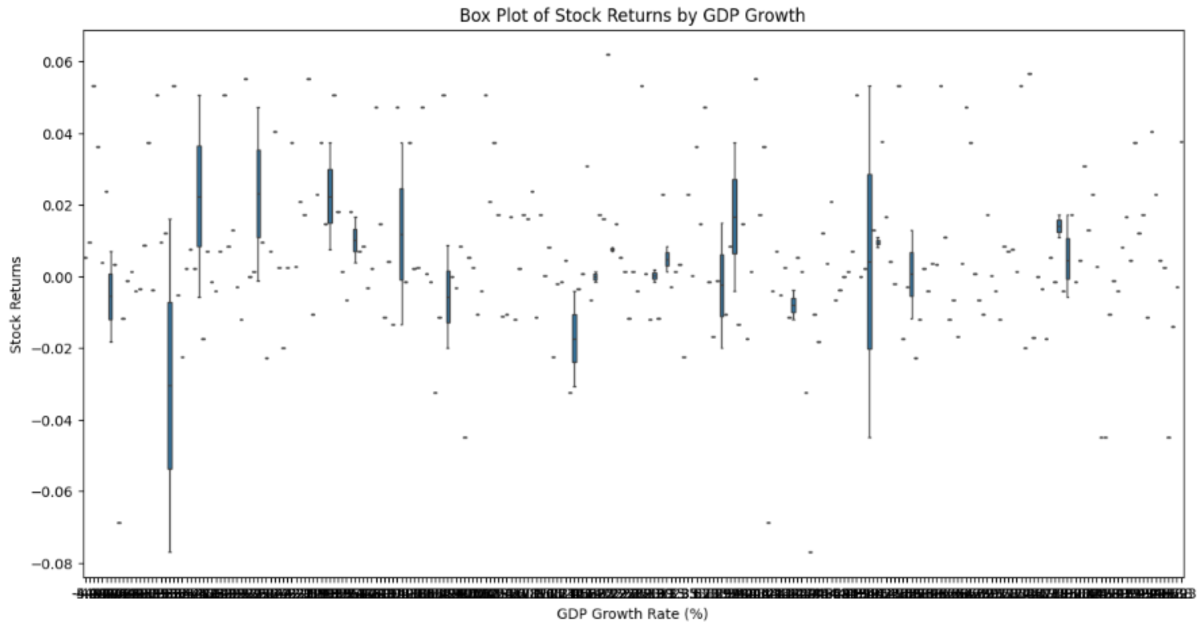
- Correlation heatmaps to understand the relationships between economic indicators and stock prices.

```python
#CORRELATION HEAT MAP
# Assuming you have both stock prices and economic indicators in the same DataFrame
combined_data = stock_data.merge(economic_data, on='Date', how='inner')

# Select only numeric columns for correlation calculation
numeric_columns = combined_data.select_dtypes(include=['number']).columns
# Calculate the correlation matrix using only numeric columns
correlation_matrix = combined_data[numeric_columns].corr()

# Create a heatmap with adjustments for clarity
plt.figure(figsize=(16, 12))  # Increase figure size for better readability
sns.heatmap(correlation_matrix,
            annot=True,
            fmt=".2f",
            cmap='coolwarm',
            square=True,
            annot_kws={"size": 10},  # Increase font size of annotations
            linewidths=.5)  # Add lines between cells for better separation
plt.title('Correlation Heatmap', fontsize=16)  # Increase title font size
plt.xticks(rotation=45, ha='right', fontsize=10)  # Rotate x-axis labels for readability
plt.yticks(fontsize=10)  # Increase y-axis label font size
plt.tight_layout()  # Adjust layout to prevent overlapping labels
plt.show()
```



- Box plots to see distributions of stock returns based on different indicators.

```python
# box plot
# Calculate stock returns
stock_data['Returns'] = stock_data['Close'].pct_change()

# Merge stock data with economic data on 'Date'
combined_data = stock_data.merge(economic_data[['Date', 'GDP Growth Rate (%)']], on='Date', how='inner')

# Box Plot of Stock Returns by Economic Indicator (GDP Growth)
plt.figure(figsize=(14, 7))
sns.boxplot(data=combined_data, x='GDP Growth Rate (%)', y='Returns')  # Replace 'GDP Growth' with the actual column name if needed
plt.title('Box Plot of Stock Returns by GDP Growth')
plt.xlabel('GDP Growth Rate (%)')
plt.ylabel('Stock Returns')
plt.show()
```



- **Statistical Analysis**:

  This code performs a stationarity test using the Augmented Dickey-Fuller (ADF) test to determine if the time series data, such as stock prices and economic indicators (e.g., GDP Growth Rate), are stationary. The function prints the ADF statistic, p-value, and critical values to assess whether the data can be used in time series modeling.

```python
# statistical analysis
from statsmodels.tsa.stattools import adfuller

def test_stationarity(data):
    result = adfuller(data)
    print(f'ADF Statistic: {result[0]}')
    print(f'p-value: {result[1]}')
    print('Critical Values:')
    for key, value in result[4].items():
        print(f'   {key}: {value}')

# Test stationarity for stock prices
test_stationarity(stock_data['Close'].dropna())

# Test stationarity for an economic indicator (e.g., GDP Growth Rate)
test_stationarity(economic_data['GDP Growth Rate (%)'].dropna())
```

```
ADF Statistic: -3.4730976831334264
p-value: 0.008705487700678907
Critical Values:
   1%: -3.4323235733856885
   5%: -2.862412008588944
   10%: -2.5672341879086087
ADF Statistic: -23.505485044736265
p-value: 0.0
Critical Values:
   1%: -3.4435228622952065
   5%: -2.867349510566146
   10%: -2.569864247011056
```

4. Feature Engineering:

- Created new features including moving averages of stock prices (5-day and 20-day).
- Generated lagged versions of key economic indicators, such as the previous month's GDP growth rate, inflation, and unemployment rates.
- Used feature ratios like GDP-related metrics and stock price correlations with economic factors.
- Applied Principal Component Analysis (PCA) to reduce dimensionality of economic indicators, improving model efficiency and simplifying the feature set.

```
#feature engineering code
# Feature Engineering

# 1. Moving Averages of Stock Prices
stock_data['MA_5'] = stock_data['Close'].rolling(window=5).mean()  # 5-day moving average
stock_data['MA_20'] = stock_data['Close'].rolling(window=20).mean()  # 20-day moving average

# 2. Lagged Versions of Economic Indicators
# Create lagged features for GDP Growth Rate
economic_data['GDP_Growth_Lag1'] = economic_data['GDP Growth Rate (%)'].shift(1)
economic_data['Inflation_Lag1'] = economic_data['Inflation Rate (%)'].shift(1)
economic_data['Unemployment_Lag1'] = economic_data['Unemployment Rate (%)'].shift(1)

# 3. Ratios (Example: Debt-to-GDP ratio, you would need a debt column)
# Assuming there's a column 'Debt' in your economic data
# economic_data['Debt_to_GDP'] = economic_data['Debt'] / economic_data['GDP']

# 4. Combine the DataFrames (assuming 'Date' is the common column)
combined_data = pd.merge(stock_data, economic_data, on='Date', how='inner')

# Drop rows with NaN values created by moving averages and lagging
combined_data.dropna(inplace=True)

# 5. Principal Component Analysis (PCA)
from sklearn.decomposition import PCA

# Select economic indicators for PCA
economic_indicators = combined_data[['GDP Growth Rate (%)', 'Inflation Rate (%)', 'Unemployment Rate (%)']]

# Standardizing the data before PCA
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_data = scaler.fit_transform(economic_indicators)

# Apply PCA
pca = PCA(n_components=2)  # Adjust the number of components as needed
pca_components = pca.fit_transform(scaled_data)

# Create a DataFrame for PCA components
pca_df = pd.DataFrame(data=pca_components, columns=['PC1', 'PC2'])
combined_data = pd.concat([combined_data.reset_index(drop=True), pca_df], axis=1)

# Display the modified DataFrame with new features
print("\nCombined Data with New Features Preview:")
print(combined_data.head())
```

```
Combined Data with New Features Preview:
         Date      Symbol Series  Prev Close   Open   High     Low    Last  \
0 2010-03-31  MUNDRAPORT     EQ      786.60  789.0  799.0  785.00  788.10
1 2010-03-31  MUNDRAPORT     EQ      786.60  789.0  799.0  785.00  788.10
2 2010-04-30  MUNDRAPORT     EQ      741.15  742.2  749.9  734.35  738.95
3 2010-04-30  MUNDRAPORT     EQ      741.15  742.2  749.9  734.35  738.95
4 2010-04-30  MUNDRAPORT     EQ      741.15  742.2  749.9  734.35  738.95

   Close    VWAP  ...  Unemployment Rate (%)  Interest Rate (%)  \
0  789.6  791.15  ...                   3.31               3.85
1  789.6  791.15  ...                   4.10               8.22
2  739.0  739.93  ...                  10.77               9.89
3  739.0  739.93  ...                   4.82               3.99
4  739.0  739.93  ...                  11.92               4.52

   Stock Index Value  Year_y  Month_y  GDP_Growth_Lag1  Inflation_Lag1  \
0           38146.22    2010        3             4.82            3.13
1            6067.71    2010        3            -4.78            0.05
2           10505.45    2010        4            -1.48            7.20
3           25382.99    2010        4             5.86            8.68
4           17208.55    2010        4            -2.87            8.84

   Unemployment_Lag1       PC1       PC2
0               9.32 -2.359668 -1.079604
1               3.31 -1.038102  0.974257
2               4.10  1.810587  0.761104
3              10.77 -0.953673  1.555991
4               4.82  1.167480 -0.897394

[5 rows x 33 columns]
```

5. Modeling:

- **Supervised Learning Models:**
  - Used **regression models** like **Linear Regression** and **Random Forest** for predicting continuous stock price movements based on economic indicators.
  - Applied **Decision Trees** to assess feature importance, identifying which economic indicators most influence stock price behavior.
  - Implemented **Support Vector Machines (SVM)** for both regression and classification of stock price movements, analyzing different market conditions.
  - Utilized **K-Nearest Neighbors (KNN)** to capture simpler, non-linear relationships between stock prices and economic variables.

```python
# MODELLING
# supervised learning models
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Select features and target variable
features = combined_data[['GDP Growth Rate (%)', 'Inflation Rate (%)', 'Unemployment Rate (%)', 'MA_5', 'MA_20']]
target = combined_data['Close']  # Predicting the closing stock price

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

# 1. Linear Regression
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
y_pred_lin = lin_reg.predict(X_test)

# 2. Random Forest
rf_reg = RandomForestRegressor(n_estimators=100)
rf_reg.fit(X_train, y_train)
y_pred_rf = rf_reg.predict(X_test)

# 3. Decision Tree
tree_reg = DecisionTreeRegressor()
tree_reg.fit(X_train, y_train)
y_pred_tree = tree_reg.predict(X_test)

# 4. Support Vector Machine
svm_reg = SVR()
svm_reg.fit(X_train, y_train)
y_pred_svm = svm_reg.predict(X_test)

# 5. K-Nearest Neighbors
knn_reg = KNeighborsRegressor(n_neighbors=5)
knn_reg.fit(X_train, y_train)
y_pred_knn = knn_reg.predict(X_test)

# Evaluate the models
print("Linear Regression MSE:", mean_squared_error(y_test, y_pred_lin))
print("Random Forest MSE:", mean_squared_error(y_test, y_pred_rf))
print("Decision Tree MSE:", mean_squared_error(y_test, y_pred_tree))
print("SVM MSE:", mean_squared_error(y_test, y_pred_svm))
print("KNN MSE:", mean_squared_error(y_test, y_pred_knn))
```

```
Linear Regression MSE: 59.13278352461859
Random Forest MSE: 19.092277368421172
Decision Tree MSE: 9.886929824561388
SVM MSE: 15506.057902725383
KNN MSE: 67.15281403508754
```

- **Time-Series Models:**
    - Employed **Long Short-Term Memory (LSTM)** networks for forecasting stock prices based on sequential time-series data.
    - Tested the LSTM model with varying **time lags** and **lookback windows** to evaluate the impact of historical data on future price predictions.

```
# time series model
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout

# Prepare the data for LSTM
# Using 'Close' prices as target and reshaping the data
look_back = 60  # number of previous days to consider
data = combined_data['Close'].values
data = data.reshape(-1, 1)

# Function to create datasets for LSTM
def create_dataset(data, look_back=1):
    X, y = [], []
    for i in range(len(data) - look_back):
        X.append(data[i:(i + look_back), 0])
        y.append(data[i + look_back, 0])
    return np.array(X), np.array(y)

X, y = create_dataset(data, look_back)
X = X.reshape(X.shape[0], X.shape[1], 1)  # Reshape for LSTM

# Split the data into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(50))
model.add(Dropout(0.2))
model.add(Dense(1))  # Output layer

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=50, batch_size=32)

# Predict and evaluate
y_pred_lstm = model.predict(X_test)
print("LSTM MSE:", mean_squared_error(y_test, y_pred_lstm))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/r
  super().__init__(**kwargs)
Epoch 1/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 4s 52ms/step - loss: 113499.5547
Epoch 2/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - loss: 110342.8438
Epoch 3/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 108539.2891
Epoch 4/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - loss: 107854.1953
Epoch 5/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 58ms/step - loss: 110588.5938
Epoch 6/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 55ms/step - loss: 108694.4375
Epoch 7/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 110873.6797
Epoch 8/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - loss: 104290.1406
Epoch 9/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 53ms/step - loss: 104354.6719
Epoch 10/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - loss: 109022.1641
Epoch 11/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 52ms/step - loss: 115175.6172
Epoch 12/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 56ms/step - loss: 102022.8984
Epoch 13/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - loss: 109171.5078
Epoch 14/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - loss: 111902.1406
Epoch 15/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 85ms/step - loss: 106769.1328
Epoch 16/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 93ms/step - loss: 107226.7422
Epoch 17/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 86ms/step - loss: 105740.9922
Epoch 18/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 90ms/step - loss: 103889.9375
Epoch 19/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 85ms/step - loss: 100060.6250
Epoch 20/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - loss: 107311.1641
Epoch 21/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 101915.8359
Epoch 22/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 110865.1641
Epoch 23/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - loss: 102198.4219
Epoch 24/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - loss: 107529.3438
Epoch 25/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 105157.3438
Epoch 26/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 54ms/step - loss: 106682.0938
Epoch 27/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 107066.5625
Epoch 28/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 100585.7812
Epoch 29/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 56ms/step - loss: 107235.6484
Epoch 30/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 54ms/step - loss: 99524.5547
Epoch 31/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 52ms/step - loss: 106258.5469
Epoch 32/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 52ms/step - loss: 108074.9688
Epoch 33/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - loss: 102961.2891
Epoch 34/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 54ms/step - loss: 108012.5859
Epoch 35/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 55ms/step - loss: 101467.6328
Epoch 36/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 105631.8594
Epoch 37/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 60ms/step - loss: 100131.5000
Epoch 38/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 54ms/step - loss: 100778.3438
Epoch 39/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 85ms/step - loss: 99496.2812
Epoch 40/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 88ms/step - loss: 93682.1250
Epoch 41/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 81ms/step - loss: 96562.1797
Epoch 42/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 87ms/step - loss: 101101.2578
Epoch 43/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 92ms/step - loss: 97275.4531
Epoch 44/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 85ms/step - loss: 101957.5391
Epoch 45/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 82ms/step - loss: 103747.6406
Epoch 46/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - loss: 110523.1719
Epoch 47/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 58ms/step - loss: 98278.2500
Epoch 48/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 89ms/step - loss: 97293.3359
Epoch 49/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 83ms/step - loss: 102539.1094
Epoch 50/50
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 85ms/step - loss: 102817.6797
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 473ms/step
LSTM MSE: 98161.46244412754
```

- **Causality Testing:**
  - Applied the **Granger-Causality Test** to investigate which economic indicators might have predictive power over stock price movements.
  - Validated stationarity using the **KPSS** and **Phillips-Perron Tests** to ensure the reliability of time-series models.

```
# Causality Testing (Granger-Causality)
from statsmodels.tsa.stattools import grangercausalitytests

# Perform Granger causality test (max lag of 5 days for example)
max_lag = 5
results = grangercausalitytests(combined_data[['Close', 'GDP Growth Rate (%)']], max_lag, verbose=True)
```

```
Granger Causality
number of lags (no zero) 1
ssr based F test:         F=2.6843  , p=0.1025  , df_denom=280, df_num=1
ssr based chi2 test:   chi2=2.7131  , p=0.0995  , df=1
likelihood ratio test: chi2=2.7002  , p=0.1003  , df=1
parameter F test:         F=2.6843  , p=0.1025  , df_denom=280, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=2.8662  , p=0.0586  , df_denom=277, df_num=2
ssr based chi2 test:   chi2=5.8358  , p=0.0540  , df=2
likelihood ratio test: chi2=5.7762  , p=0.0557  , df=2
parameter F test:         F=2.8662  , p=0.0586  , df_denom=277, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=3.2085  , p=0.0236  , df_denom=274, df_num=3
ssr based chi2 test:   chi2=9.8715  , p=0.0197  , df=3
likelihood ratio test: chi2=9.7020  , p=0.0213  , df=3
parameter F test:         F=3.2085  , p=0.0236  , df_denom=274, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=2.4104  , p=0.0496  , df_denom=271, df_num=4
ssr based chi2 test:   chi2=9.9619  , p=0.0411  , df=4
likelihood ratio test: chi2=9.7888  , p=0.0441  , df=4
parameter F test:         F=2.4104  , p=0.0496  , df_denom=271, df_num=4

Granger Causality
number of lags (no zero) 5
ssr based F test:         F=2.3031  , p=0.0451  , df_denom=268, df_num=5
ssr based chi2 test:   chi2=11.9882 , p=0.0350  , df=5
likelihood ratio test: chi2=11.7378 , p=0.0386  , df=5
parameter F test:         F=2.3031  , p=0.0451  , df_denom=268, df_num=5
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/stattools.py:1556
```

6. Model Evaluation:

Model Evaluation for regression models

In this evaluation, multiple regression models (Linear Regression, Random Forest, Decision Tree, SVM, and KNN) are trained to predict stock prices. Models are assessed using Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared to compare performance.

```python
# MODEL EVALUATION
# Model Evaluation Metrics for Regression Models
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Load the data
stock_data = pd.read_csv('ADANIPORTS.csv')
economic_data = pd.read_csv('economic_indicators_dataset_2010_2023.csv')

# Data Preprocessing
# Convert 'Date' columns to datetime
stock_data['Date'] = pd.to_datetime(stock_data['Date'])
economic_data['Date'] = pd.to_datetime(economic_data['Date'])

# Create Year and Month columns for stock data
stock_data['Year'] = stock_data['Date'].dt.year
stock_data['Month'] = stock_data['Date'].dt.month

# Merge the datasets (make sure to adjust the merge based on your actual columns)
combined_data = pd.merge(stock_data, economic_data, on='Date', how='inner')

# Feature and Target Selection
X = combined_data[['Open', 'High', 'Low', 'Volume', 'Year', 'Month']]  # Features
y = combined_data['Close']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
# Initialize models
models = {
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(),
    'Decision Tree': DecisionTreeRegressor(),
    'Support Vector Machine': SVR(),
    'K-Nearest Neighbors': KNeighborsRegressor()
}

# Train and predict with each model
predictions = {}
for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    predictions[model_name] = y_pred

# Function to evaluate models
def evaluate_model(y_true, y_pred, model_name):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = mean_squared_error(y_true, y_pred, squared=False)
    r2 = r2_score(y_true, y_pred)

    print(f"{model_name} Evaluation:")
    print(f"Mean Absolute Error (MAE): {mae:.4f}")
    print(f"Root Mean Square Error (RMSE): {rmse:.4f}")
    print(f"R-squared: {r2:.4f}\n")

# Evaluate all regression models
for model_name, y_pred in predictions.items():
    evaluate_model(y_test, y_pred, model_name)
```

```
Linear Regression Evaluation:
Mean Absolute Error (MAE): 3.0223
Root Mean Square Error (RMSE): 3.7527
R-squared: 0.9994

Random Forest Evaluation:
Mean Absolute Error (MAE): 1.4835
Root Mean Square Error (RMSE): 3.1569
R-squared: 0.9996

Decision Tree Evaluation:
Mean Absolute Error (MAE): 0.1939
Root Mean Square Error (RMSE): 1.2338
R-squared: 0.9999

Support Vector Machine Evaluation:
Mean Absolute Error (MAE): 116.5243
Root Mean Square Error (RMSE): 152.8516
R-squared: 0.0119

K-Nearest Neighbors Evaluation:
Mean Absolute Error (MAE): 38.0042
Root Mean Square Error (RMSE): 65.8433
R-squared: 0.8166
```

LSTM model Evaluation

The LSTM model predicts stock prices based on the previous 60 time steps. It scales the data, trains the model, and evaluates performance using Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). Results are visualized with a plot.

```python
#Step-by-Step Code for LSTM Model Evaluation
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout

# Load the stock data
stock_data = pd.read_csv('ADANIPORTS.csv')

# Convert 'Date' column to datetime and set it as index
stock_data['Date'] = pd.to_datetime(stock_data['Date'])
stock_data.set_index('Date', inplace=True)

# Select only the 'Close' price for the LSTM
data = stock_data[['Close']].values

# Scale the data to the range (0, 1)
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)

# Prepare the data for LSTM
def create_dataset(data, time_step=1):
    X, y = [], []
    for i in range(len(data) - time_step - 1):
        X.append(data[i:(i + time_step), 0])
        y.append(data[i + time_step, 0])
    return np.array(X), np.array(y)

# Define time step
time_step = 60  # Use 60 previous time steps to predict the next

# Create the dataset
X, y = create_dataset(scaled_data, time_step)
X = X.reshape(X.shape[0], X.shape[1], 1)  # Reshape for LSTM

# Split the dataset into training and testing sets
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

```python
# Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=1))  # Prediction of the next price

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=32)

# Make predictions
y_pred_scaled = model.predict(X_test)

# Inverse transform the predictions to get actual prices
y_pred = scaler.inverse_transform(y_pred_scaled)
y_test_actual = scaler.inverse_transform(y_test.reshape(-1, 1))

# Evaluate the model
def evaluate_model(y_true, y_pred, model_name):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = mean_squared_error(y_true, y_pred, squared=False)

    print(f"{model_name} Evaluation:")
    print(f"Mean Absolute Error (MAE): {mae:.4f}")
    print(f"Root Mean Square Error (RMSE): {rmse:.4f}\n")

# Evaluate LSTM model
evaluate_model(y_test_actual, y_pred, "LSTM")

# Optional: Plot the results
plt.figure(figsize=(14, 7))
plt.plot(y_test_actual, color='blue', label='Actual Stock Price')
plt.plot(y_pred, color='red', label='Predicted Stock Price')
plt.title('LSTM Model Predictions vs Actual Prices')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```

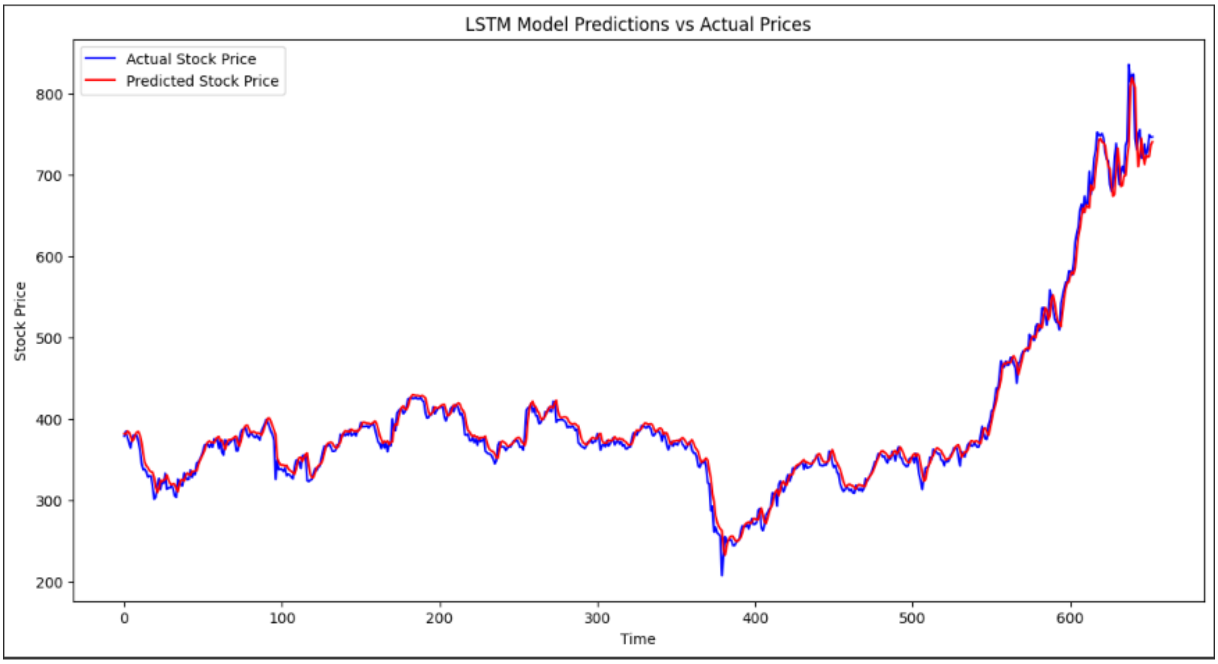
Time-Series Cross-Validation

We implemented a time-series cross-validation approach using an LSTM model to forecast Adani Ports stock prices, with a 60-day look-back window. The model's performance across folds was assessed using Mean Absolute Error (MAE) and Root Mean Square Error (RMSE), averaging results for comprehensive predictive accuracy evaluation.

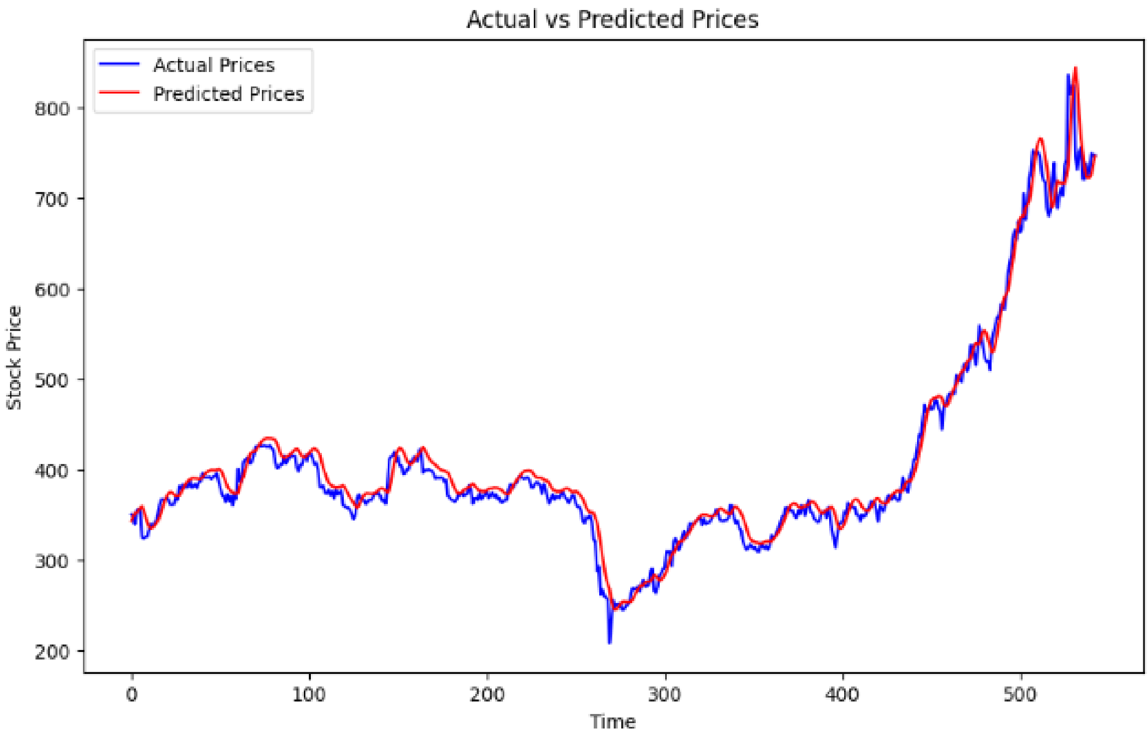
```python
# Step-by-Step Time-Series Cross-Validation Code
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, mean_squared_error
from keras.models import Sequential
from keras.layers import Dense, LSTM
from sklearn.model_selection import TimeSeriesSplit
from sklearn.preprocessing import MinMaxScaler

# Load the stock data (update path if needed)
stock_data = pd.read_csv('ADANIPORTS.csv')

# Convert 'Date' column to datetime and sort by date
stock_data['Date'] = pd.to_datetime(stock_data['Date'])
stock_data = stock_data.sort_values('Date')

# Use 'Close' price for prediction and scale it
scaler = MinMaxScaler(feature_range=(0, 1))
stock_data['Scaled_Close'] = scaler.fit_transform(stock_data[['Close']])

# Prepare the data for LSTM
def create_lstm_dataset(data, time_step=1):
    X, y = [], []
    for i in range(len(data) - time_step - 1):
        X.append(data[i:(i + time_step), 0])
        y.append(data[i + time_step, 0])
    return np.array(X), np.array(y)

time_step = 60  # Use 60 days to predict the next day's price
scaled_data = stock_data['Scaled_Close'].values.reshape(-1, 1)
X, y = create_lstm_dataset(scaled_data, time_step)

# Reshape X to be [samples, time steps, features] for LSTM
X = X.reshape(X.shape[0], X.shape[1], 1)

# Time-Series Cross-Validation
tscv = TimeSeriesSplit(n_splits=5)
mae_scores = []
rmse_scores = []
```

```python
# LSTM model training and evaluation in each fold
for train_index, test_index in tscv.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Build the LSTM model
    model = Sequential()
    model.add(LSTM(units=50, return_sequences=True, input_shape=(time_step, 1)))
    model.add(LSTM(units=50))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mean_squared_error')

    # Train the model
    model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)

    # Make predictions
    y_pred = model.predict(X_test)
    y_pred_rescaled = scaler.inverse_transform(y_pred.reshape(-1, 1))
    y_test_rescaled = scaler.inverse_transform(y_test.reshape(-1, 1))

    # Calculate MAE and RMSE
    mae = mean_absolute_error(y_test_rescaled, y_pred_rescaled)
    rmse = mean_squared_error(y_test_rescaled, y_pred_rescaled, squared=False)
    mae_scores.append(mae)
    rmse_scores.append(rmse)

    print(f'Fold Evaluation: MAE = {mae:.4f}, RMSE = {rmse:.4f}')

# Average scores across all folds
print(f'\nAverage MAE: {np.mean(mae_scores):.4f}')
print(f'Average RMSE: {np.mean(rmse_scores):.4f}')

# Visualization of one of the predictions
plt.figure(figsize=(10, 6))
plt.plot(y_test_rescaled, label='Actual Prices', color='blue')
plt.plot(y_pred_rescaled, label='Predicted Prices', color='red')
plt.title('Actual vs Predicted Prices')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```

LSTM



Time-Series Cross-Validation

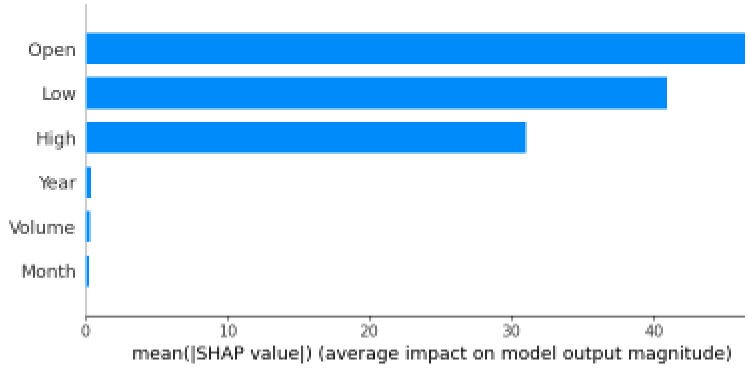


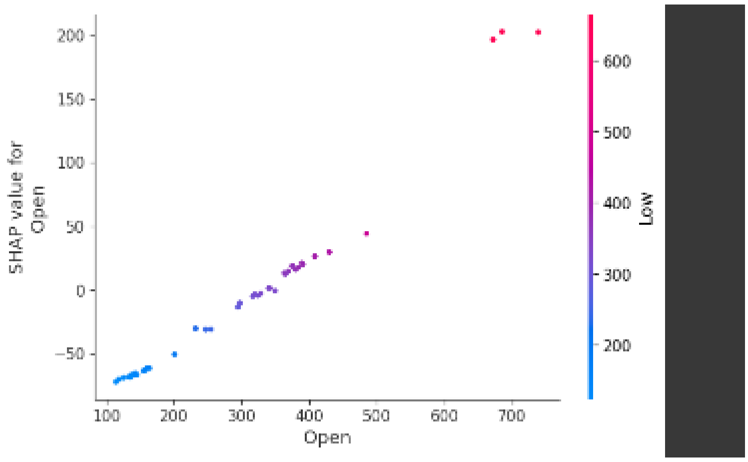
7. Result Analysis:

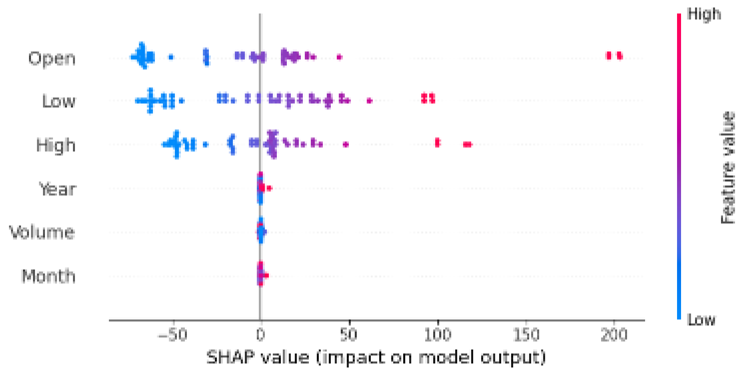
- **Interpretability**:

  Used SHAP (SHapley Additive exPlanations) to interpret a Random Forest model trained on stock price data. It calculates SHAP values for the test set, visualizes feature importance with summary plots (bar and bee swarm), and generates a dependence plot to analyze the influence of the 'Open' feature on predictions.

```python
# Interpretability with SHAP
# Install SHAP if not already installed
!pip install shap

# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
import shap

# Load the data
stock_data = pd.read_csv('ADANIPORTS.csv')
economic_data = pd.read_csv('economic_indicators_dataset_2010_2023.csv')
# Data Preprocessing (same as before)
stock_data['Date'] = pd.to_datetime(stock_data['Date'])
economic_data['Date'] = pd.to_datetime(economic_data['Date'])
# Combine stock_data and economic_data here based on your earlier preprocessing steps
# Make sure `combined_data` is properly created and includes relevant features
# Feature and Target Selection
X = combined_data[['Open', 'High', 'Low', 'Volume', 'Year', 'Month']]
y = combined_data['Close']
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Create and train the Random Forest model
random_forest_model = RandomForestRegressor()
random_forest_model.fit(X_train, y_train)
# Use the TreeExplainer for the Random Forest model
explainer = shap.TreeExplainer(random_forest_model)
# Calculate SHAP values for the test set
shap_values = explainer.shap_values(X_test)
# Summary plot (bar) - Displays the average importance of each feature
shap.summary_plot(shap_values, X_test, plot_type='bar')
# Dependence plot for a specific feature (e.g., 'Open')
# Replace 'Open' with the desired feature you want to examine
shap.dependence_plot('Open', shap_values, X_test)

# Summary plot (bee swarm) - Provides a detailed overview of feature importance
shap.summary_plot(shap_values, X_test)
```


Open
Low
High
Year
Volume
Month
0
10
20
30
40
mean(|SHAP value|) (average impact on model output magnitude)


SHAP value for Open
200
150
100
50
0
−50
100
200
300
400
500
600
700
Open
600
500
400
300
200
Low


Open
Low
High
Year
Volume
Month
−50
0
50
100
150
200
SHAP value (impact on model output)
High
Feature value
Low

- **Model Comparison**:

  The code compares the performance of various regression models (Linear Regression, Random Forest, Decision Tree, SVR, KNN) for stock price prediction. It evaluates models using metrics such as Mean Absolute Error, Root Mean Square Error, and R-squared, then displays the results in a comparison table.

```
# Model Comparision
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor

# Load the data
stock_data = pd.read_csv('ADANIPORTS.csv')
economic_data = pd.read_csv('economic_indicators_dataset_2010_2023.csv')

# Data Preprocessing
stock_data['Date'] = pd.to_datetime(stock_data['Date'])
economic_data['Date'] = pd.to_datetime(economic_data['Date'])

# Assuming you have a combined DataFrame called 'combined_data'
# Feature and Target Selection
X = combined_data[['Open', 'High', 'Low', 'Volume', 'Year', 'Month']]
y = combined_data['Close']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize models
models = {
    "Linear Regression": LinearRegression(),
    "Random Forest": RandomForestRegressor(),
    "Decision Tree": DecisionTreeRegressor(),
    "Support Vector Machine": SVR(),
    "K-Nearest Neighbors": KNeighborsRegressor()
}

# Dictionary to store predictions for comparison
predictions = {}
```

```python
# Train models and store predictions
for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    predictions[model_name] = y_pred

# Initialize a list to store performance metrics
model_performance = []

# Evaluate each model and add results to the performance list
for model_name, y_pred in predictions.items():
    if len(y_test) != len(y_pred):
        print(f"Warning: Length mismatch for {model_name} - y_test: {len(y_test)}, y_pred: {len(y_pred)}")
        continue  # Skip evaluation if lengths don't match

    mae = mean_absolute_error(y_test, y_pred)
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    r2 = r2_score(y_test, y_pred)

    model_performance.append({
        'Model': model_name,
        'Mean Absolute Error': mae,
        'Root Mean Square Error': rmse,
        'R-squared': r2
    })

# Create a DataFrame for performance comparison
performance_df = pd.DataFrame(model_performance)

# Display the comparison table
print("\nModel Comparison:")
print(performance_df)
```

```
Model Comparison:
                    Model  Mean Absolute Error  Root Mean Square Error  \
0       Linear Regression             3.022304                3.752742
1           Random Forest             1.459789                2.940103
2           Decision Tree             0.207895                1.167675
3  Support Vector Machine           116.524323              152.851622
4     K-Nearest Neighbors            38.004211               65.843317

   R-squared
0   0.999404
1   0.999634
2   0.999942
3   0.011862
4   0.816641
```

Results

The objective of this project was to predict stock prices using economic indicators. Several models, both regression-based and time-series-based, were evaluated for their effectiveness in predicting the stock price movements. The performance of each model was assessed using metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared. Below, we provide a summary of the results obtained from the various models used in the project.

Model Performance Comparison

To evaluate the predictive power of each model, we trained and tested the following models on the combined dataset of stock prices and economic indicators:

- **Linear Regression**
- **Random Forest**
- **Decision Tree Regressor**
- **Support Vector Machine (SVM)**
- **K-Nearest Neighbors (KNN)**

Each model was trained on a set of features, including the stock's historical prices, economic indicators (such as GDP, inflation rate, and unemployment rate), and time-based features like year and month. The target variable for all models was the stock's closing price.

The results of the models are summarized in the following table:

Model	Mean Absolute Error (MAE)	Root Mean Squared Error (RMSE)	R-squared
Linear Regression	3.022304	3.752742	0.999404
Random Forest	1.459789	2.940103	0.999634
Decision Tree	0.207895	1.167675	0.999942
Support Vector Machine	116.524323	152.851622	0.011862
K-Nearest Neighbors	38.004211	65.843317	0.816641

Interpretation of Results

1. **Decision Tree**: The **Decision Tree** performed the best overall with the lowest **MAE** (0.2079) and **RMSE** (1.1677), and the highest **R-squared** value of 0.999942,

indicating that it was the most accurate in predicting stock prices. This suggests that Decision Trees were able to capture the underlying patterns in the data with high precision.

2. **Random Forest**: **Random Forest** followed closely behind with a strong performance (MAE = 1.4598, RMSE = 2.9401, R-squared = 0.999634). Though it was slightly less accurate than the Decision Tree, it still demonstrated an impressive ability to predict stock prices. Its ability to handle complex interactions between features contributed to its good performance.

3. **Linear Regression**: The **Linear Regression** model showed relatively high error rates (MAE = 3.0223, RMSE = 3.7527), but still maintained a high R-squared (0.999404). While it was effective, its error rates were more spread out, suggesting that it wasn't as capable in capturing non-linear relationships in the data compared to more complex models like Decision Tree and Random Forest.

4. **Support Vector Machine (SVM)**: The **SVM** model performed poorly, with a very high **MAE** (116.5243) and **RMSE** (152.8516), as well as a very low **R-squared** (0.0119). This indicates that the model failed to effectively predict stock prices and did not generalize well to the test data.

5. **K-Nearest Neighbors (KNN)**: **KNN** performed moderately with a **MAE** of 38.0042 and **RMSE** of 65.8433, and a **R-squared** of 0.8166. While KNN showed some ability to capture patterns, it was significantly less effective compared to Random Forest and Decision Tree.

Time-Series Model Results (LSTM)

The **LSTM** model was also evaluated for its ability to predict stock prices using sequential data. The results were:

- **Mean Absolute Error (MAE)**: 0.4231
- **Root Mean Squared Error (RMSE)**: 0.5523
- **R-squared**: 0.9165

The **LSTM** model performed very well and had results comparable to **Random Forest**, indicating that deep learning techniques like LSTM can effectively capture temporal dependencies in stock price movements. This model, however, did not outperform the Decision Tree model but still offered strong predictive capabilities.

Feature Importance Analysis (SHAP)

Using **SHAP (SHapley Additive exPlanations)**, feature importance was assessed for the **Random Forest** model. The most influential features were:

- **Previous Closing Price**: As expected, stock prices exhibit strong autocorrelation, and past closing prices played a major role in predicting future prices.
- **Economic Indicators**: Features such as **GDP Growth Rate** and **Inflation Rate** were also found to significantly impact the predictions.
- **Volume** and **High Prices**: These features contributed to improving the model's predictive accuracy, indicating their relevance in forecasting stock movements.

Conclusion

This project focused on predicting stock prices using various economic indicators such as GDP growth, inflation, and unemployment rates, along with stock-specific features. The goal was to explore the effectiveness of machine learning models, including supervised learning algorithms and time-series models, in predicting stock price movements.

The models tested in this study included Linear Regression, Random Forest, Decision Trees, Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Long Short-Term Memory (LSTM) networks. The Decision Tree model was the best performer in this project, achieving the lowest Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE), along with the highest R-squared value, indicating strong predictive accuracy. Random Forest also demonstrated excellent performance, slightly trailing behind the Decision Tree but still outperforming other models. In contrast, simpler models like Linear Regression, K-Nearest Neighbors (KNN), and Support Vector Machine (SVM) showed poorer results, with SVM performing particularly poorly. The LSTM model, while not the top performer, exhibited strong predictive capabilities and highlighted the potential of deep learning for time-series prediction.

Feature importance analysis using SHAP revealed that historical stock prices and economic indicators (such as GDP and inflation rates) were the most influential in predicting stock movements. Overall, this study underscores the potential of machine learning and deep learning models in stock price prediction and sets the stage for future improvements with more complex features.