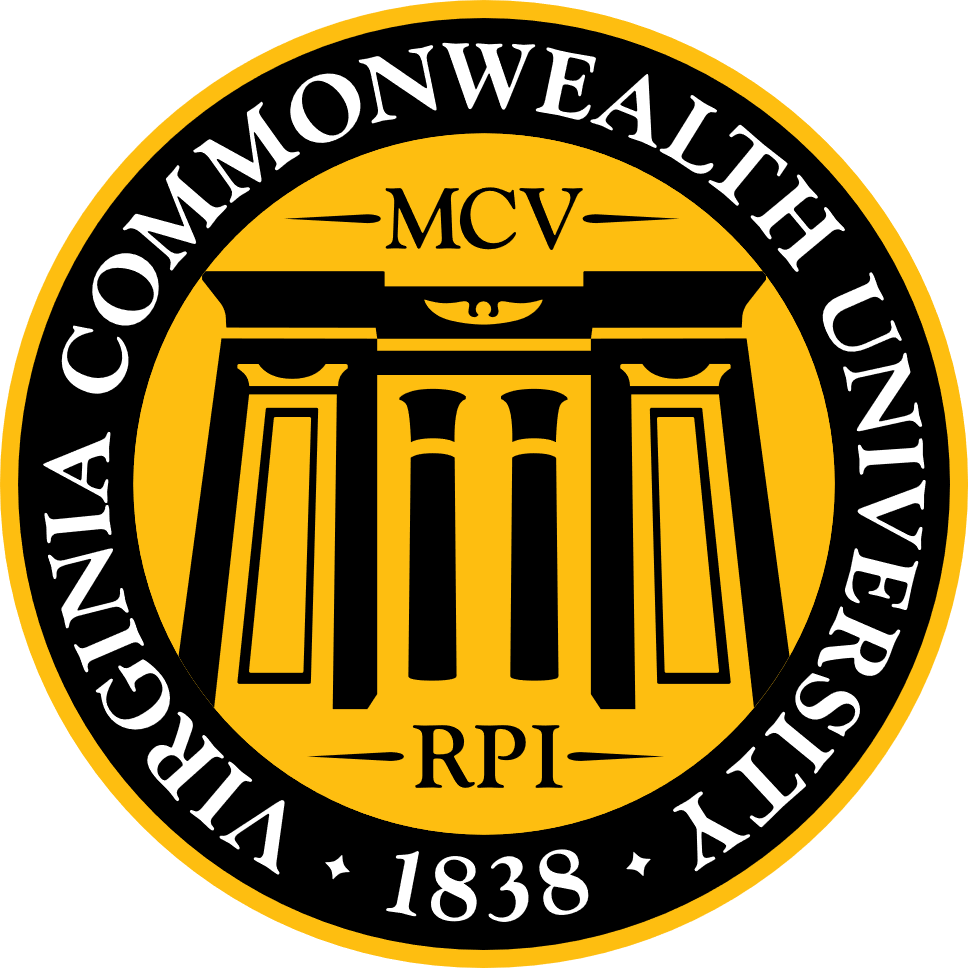
****

**VIRGINIA COMMONWEALTH UNIVERSITY**

**Statistical analysis and modelling (SCMA 632)**

**A6a-** **Time Series Analysis**

**ADHYAYAN AMIT JAIN**

**V01109421**

**Date of Submission: 22-07-2024**

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Title** | **Page No.** |
| **1.** | Introduction | **1** |
| **2.** | Results | **5** |
| **3.** | Interpretations | **5** |
| **4.** | Implications | **31** |
| **5.** | Recommendations | **34** |
| **6** | Codes | **37** |
| **7.** | References | **48** |

**Stock Price Forecasting and Analysis Using Statistical and Machine Learning Models**

**INTRODUCTION**

In the modern financial landscape, accurate forecasting of stock prices is crucial for investors, analysts, and decision-makers. This assignment focuses on a comprehensive approach to stock price forecasting using a combination of statistical and machine learning models.

The process begins with selecting a stock of interest and sourcing historical data from reputable financial platforms such as Investing.com or Yahoo Finance. The data undergoes a meticulous cleaning process to address outliers and missing values, ensuring its reliability for subsequent analysis. The dataset is then visualized through a line graph, providing a clear representation of historical trends. For the forecasting task, the data is divided into training and test sets, and transformed into a monthly frequency. This transformation allows for a detailed decomposition of the time series into its fundamental components—trend, seasonality, and residuals—using both additive and multiplicative models. The assignment encompasses both univariate and multivariate forecasting techniques.

1. **Univariate Forecasting**: This involves the use of conventional statistical models to predict future stock prices. The Holt-Winters model is employed to forecast prices for the upcoming year, while the ARIMA model is fitted to daily data to evaluate its performance. A comparison is made with the Seasonal-ARIMA (SARIMA) model to determine which best fits the data, followed by a forecast for the next three months. Additionally, the ARIMA model is applied to the monthly series to assess its efficacy in capturing long-term trends.
2. **Multivariate Forecasting**: Advanced machine learning models are utilized to enhance forecasting accuracy. Neural Networks, specifically Long Short-Term Memory (LSTM) networks, are applied to capture complex patterns in the data. Additionally, tree-based models such as Random Forest and Decision Trees are employed to explore their predictive capabilities and compare their performance with the conventional models.

This assignment aims to provide a comprehensive analysis of various forecasting methods, offering insights into their effectiveness and practical applications in stock price prediction.

**OBJECTIVES**

The primary objectives of this assignment are:

1. **Data Acquisition and Preparation**:
   * Select a stock of interest and download its historical price data from sources such as Investing.com or Yahoo Finance.
   * Clean the dataset by identifying and addressing outliers and missing values.
   * Interpolate missing values if necessary to maintain the integrity of the data.
   * Visualize the cleaned data through a well-labeled line graph to observe historical trends.
2. **Time Series Transformation and Decomposition**:
   * Convert the data to a monthly frequency to facilitate effective analysis and forecasting.
   * Decompose the time series into its core components—trend, seasonality, and residuals—using both additive and multiplicative models.
3. **Univariate Forecasting**:
   * **Holt-Winters Model**: Fit a Holt-Winters exponential smoothing model to forecast the stock price for the next year.
   * **ARIMA Model**: Apply the ARIMA model to daily stock price data. Conduct diagnostic checks to validate the model and compare its performance with the Seasonal-ARIMA (SARIMA) model. Forecast the stock price for the next three months.
   * **Monthly ARIMA**: Fit the ARIMA model to the monthly time series to evaluate its effectiveness in capturing long-term trends.
4. **Multivariate Forecasting**:
   * **Neural Networks (LSTM)**: Implement Long Short-Term Memory (LSTM) networks to predict future stock prices and assess their performance.
   * **Tree-Based Models**: Utilize Random Forest and Decision Tree models to explore their forecasting capabilities and compare their results with the statistical models.
5. **Model Evaluation and Comparison**:
   * Evaluate the forecasting accuracy of each model using appropriate metrics such as RMSE, MAE, and MAPE.
   * Compare the performance of statistical and machine learning models to determine their effectiveness in predicting stock prices.

Through these objectives, the assignment aims to provide a detailed and comparative analysis of various forecasting techniques, offering insights into their practical applications and effectiveness in stock price prediction.

**BUSINESS SIGNIFICANCE**

Accurate stock price forecasting is crucial for making informed investment decisions, managing risk, and developing strategic business strategies. Understanding the future movements of a stock's price can provide significant advantages in various business contexts:

1. **Investment Strategy**:
   * **Informed Decisions**: Investors and financial analysts can make more strategic decisions based on accurate forecasts, potentially maximizing returns and minimizing losses.
   * **Portfolio Management**: Forecasting helps in optimizing asset allocation and rebalancing portfolios based on predicted stock performance.
2. **Risk Management**:
   * **Mitigating Risks**: By anticipating future price movements, businesses can better manage financial risks and hedge against unfavorable market conditions.
   * **Strategic Planning**: Companies can prepare for potential market fluctuations and adjust their strategies accordingly to mitigate adverse impacts.
3. **Market Competitiveness**:
   * **Competitive Edge**: Firms that leverage advanced forecasting models can gain a competitive edge by anticipating market trends and adjusting their strategies proactively.
   * **Investor Relations**: Accurate forecasts enhance a company's credibility and attract investors by demonstrating a sophisticated understanding of market dynamics.
4. **Operational Efficiency**:
   * **Resource Allocation**: Forecasting helps businesses plan for future capital needs and operational requirements, optimizing resource allocation and improving efficiency.
   * **Cost Management**: By predicting price trends, companies can better manage costs and avoid overexposure to volatile market conditions.
5. **Strategic Decisions**:
   * **Business Expansion**: Forecasting can inform decisions related to market entry, expansion, and diversification based on anticipated stock performance and market conditions.
   * **Product and Service Planning**: Accurate price forecasts can influence decisions about product launches, pricing strategies, and marketing campaigns.

By utilizing various forecasting models—including traditional statistical methods and advanced machine learning techniques—businesses can gain valuable insights into stock price trends, enhance their decision-making processes, and strategically position themselves in the market.

**RESULTS AND INTERPRETATIONS**

**Python Language**

### Step-by-Step Analysis and Interpretation of the Code

#### 1. ****Import Necessary Libraries****

**Code:**

import pandas as pd

import numpy as np

import yfinance as yf

import matplotlib.pyplot as plt

from statsmodels.tsa.seasonal import seasonal\_decompose

from sklearn.model\_selection import train\_test\_split

from statsmodels.tsa.holtwinters import ExponentialSmoothing

from pmdarima import auto\_arima

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout

from sklearn.preprocessing import MinMaxScaler

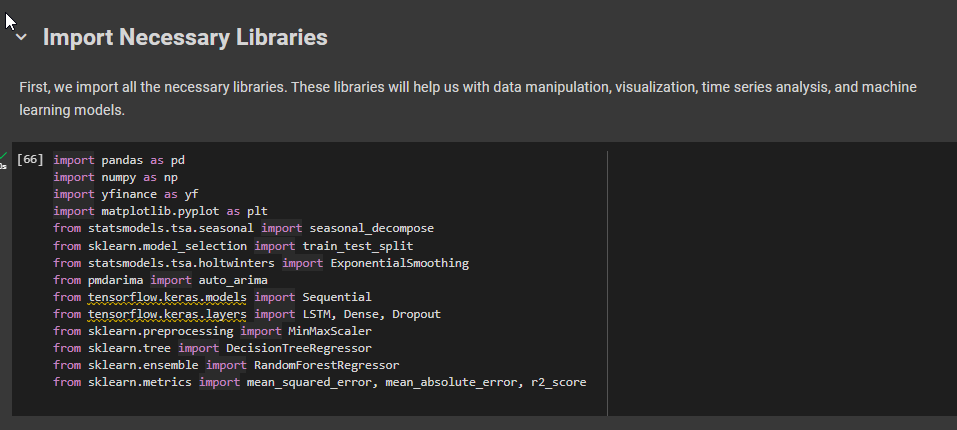
from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

**Interpretation:** This section imports all the necessary libraries:

* **pandas** and **numpy**: For data manipulation and numerical operations.
* **yfinance**: To fetch stock data.
* **matplotlib.pyplot**: For plotting graphs.
* **statsmodels**: For time series decomposition and Holt-Winters model.
* **sklearn**: For machine learning models and metrics.
* **tensorflow.keras**: For building and training LSTM models.
* **pmdarima**: For ARIMA model selection and forecasting.



#### 2. ****Data Fetching from Yahoo Finance****

**Code:**

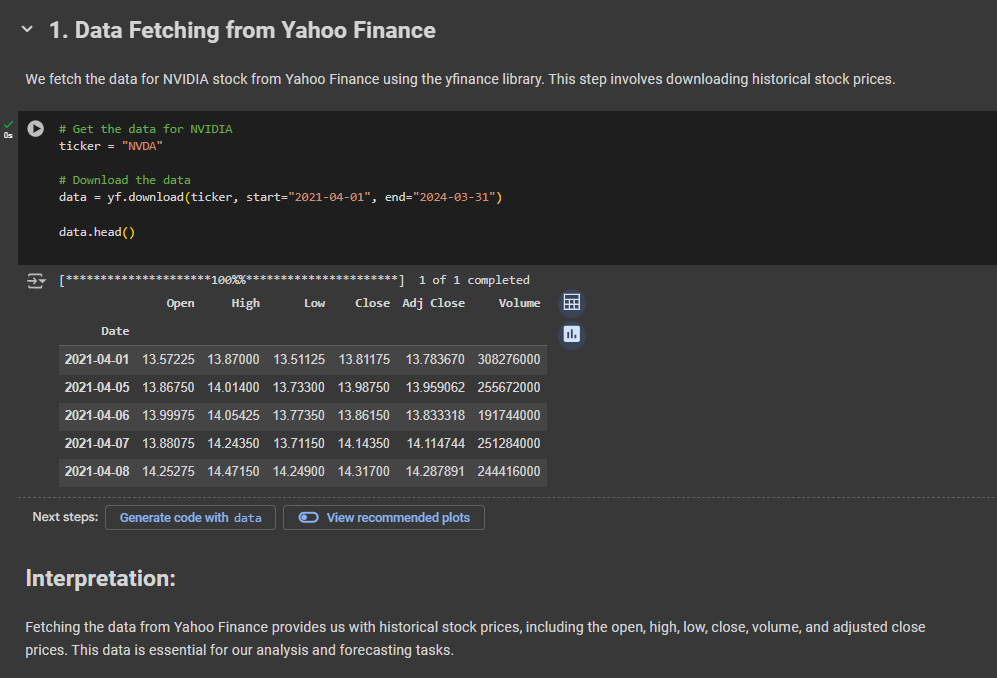
ticker = "NVDA"

data = yf.download(ticker, start="2021-04-01", end="2024-03-31")

data.head()

**Interpretation:**

* The yfinance library is used to fetch historical stock price data for NVIDIA (ticker: NVDA) from April 1, 2021, to March 31, 2024.
* The head() function shows the first few rows of the dataset, which includes columns like Open, High, Low, Close, Volume, and Adjusted Close.



#### 3. ****Select the Target Variable and Clean the Data****

**Code:**

df = data[['Adj Close']]

print("Missing values:")

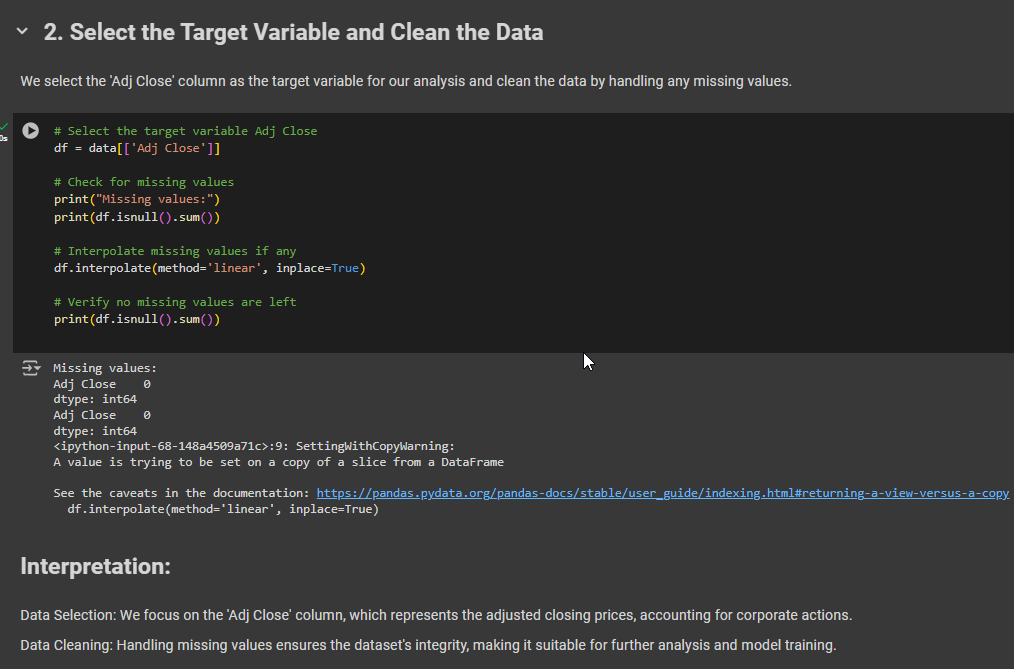
print(df.isnull().sum())

df.interpolate(method='linear', inplace=True)

print(df.isnull().sum())

**Interpretation:**

* **Data Selection**: The 'Adj Close' column is chosen for analysis as it reflects adjusted closing prices after corporate actions.
* **Data Cleaning**: Missing values are checked and interpolated linearly if found. This ensures that the dataset is complete and ready for analysis.



#### 4. ****Plot the Time Series****

**Code:**

plt.figure(figsize=(10, 5))

plt.plot(df, label='Adj Close Price')

plt.title('NVIDIA Adj Close Price')

plt.xlabel('Date')

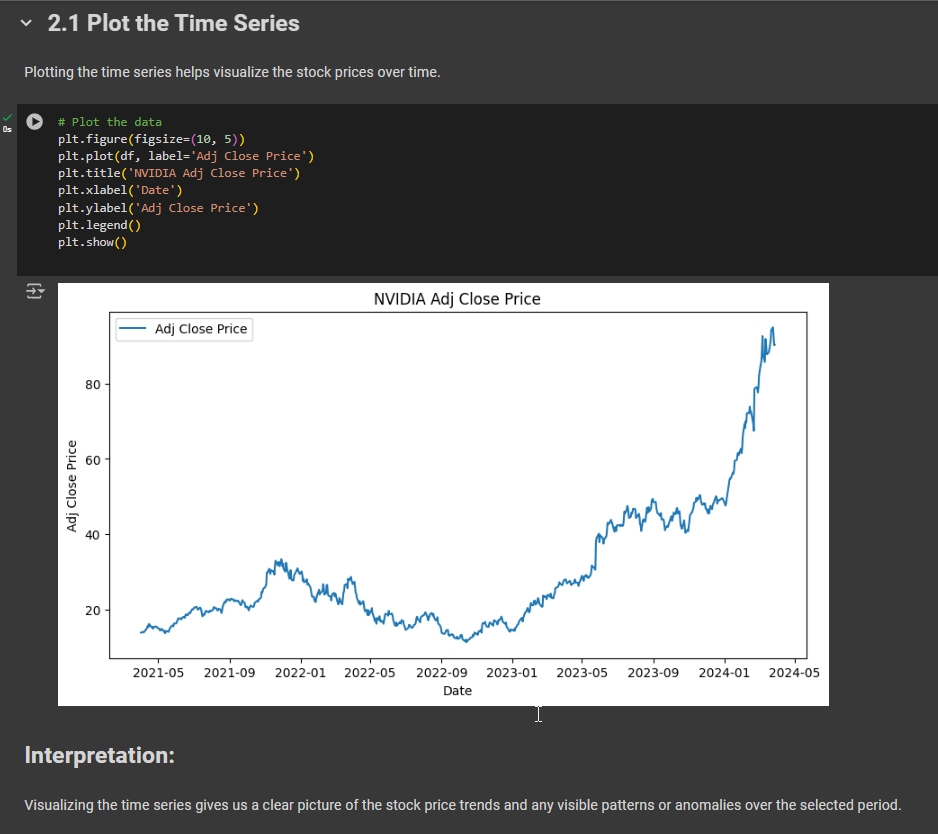
plt.ylabel('Adj Close Price')

plt.legend()

plt.show()

**Interpretation:**

* A line graph of the 'Adj Close Price' over time is plotted to visualize the historical price trends and identify patterns or anomalies.



#### 5. ****Decomposition of Time Series****

**Code:**

result = seasonal\_decompose(df['Adj Close'], model='multiplicative', period=30)

fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1, figsize=(12, 10), sharex=True)

result.observed.plot(ax=ax1)

ax1.set\_ylabel('Observed')

result.trend.plot(ax=ax2)

ax2.set\_ylabel('Trend')

result.seasonal.plot(ax=ax3)

ax3.set\_ylabel('Seasonal')

result.resid.plot(ax=ax4)

ax4.set\_ylabel('Residual')

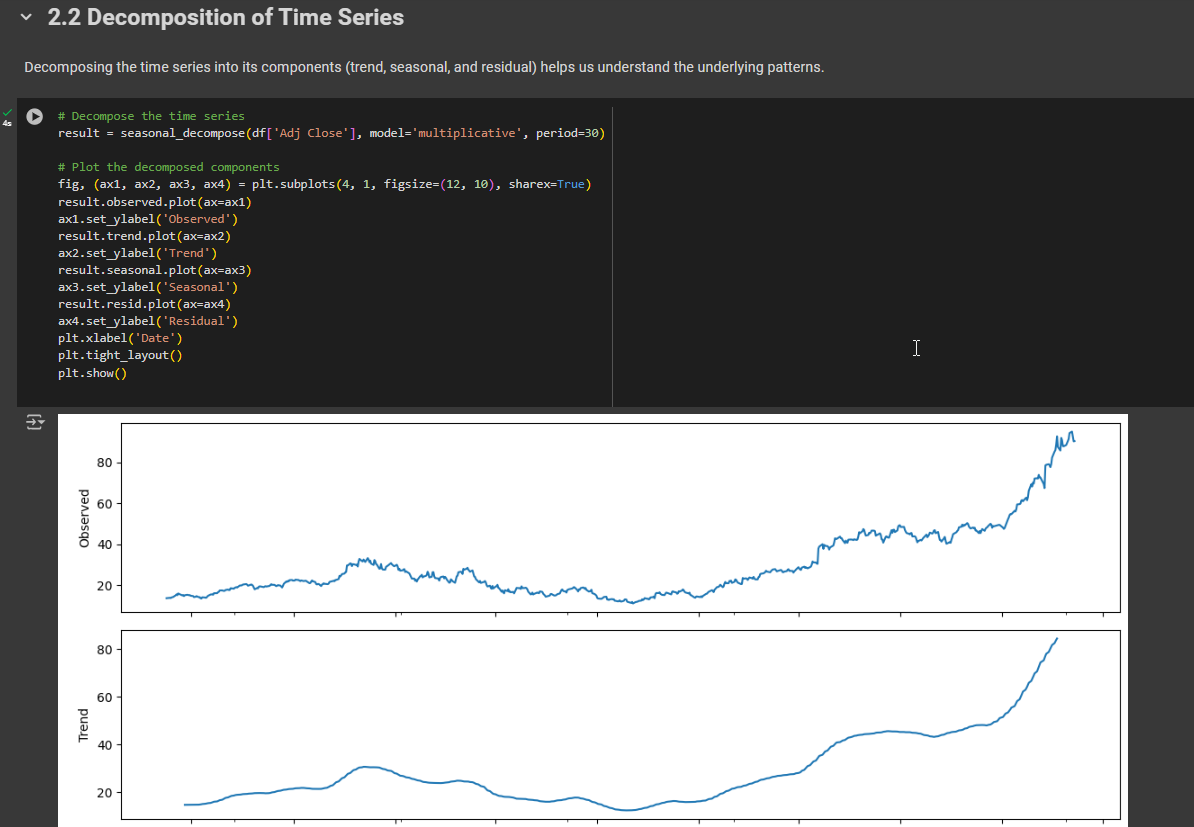
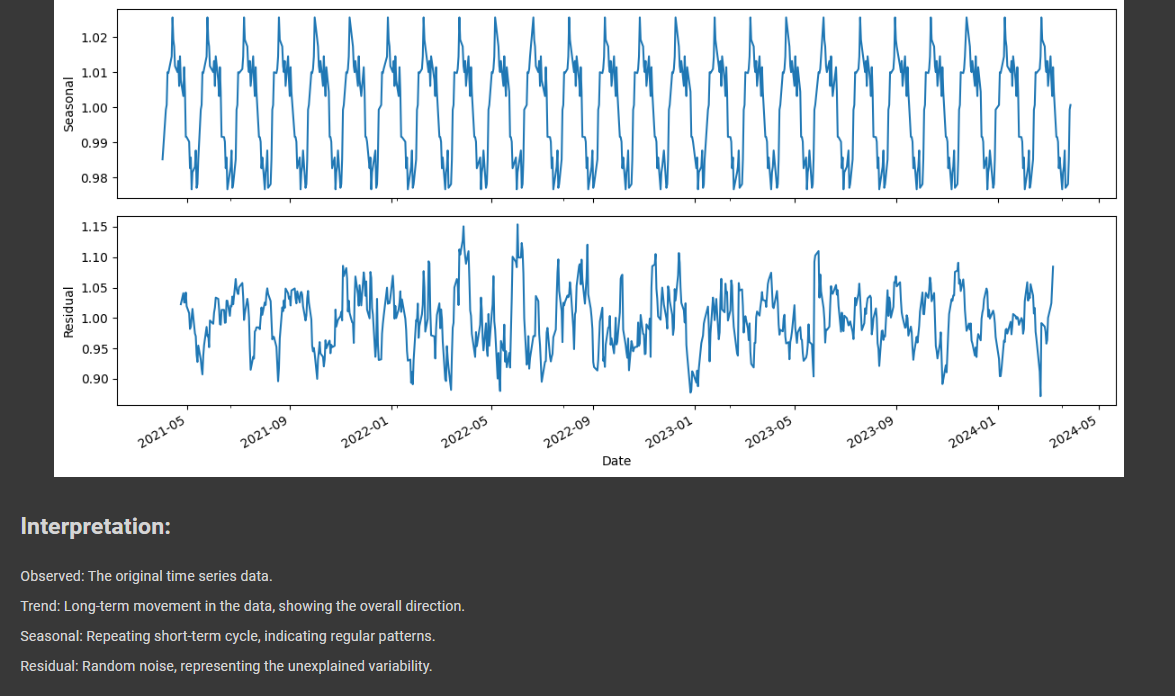
plt.xlabel('Date')

plt.tight\_layout()

plt.show()

**Interpretation:**

* **Decomposition**: The time series is decomposed into observed, trend, seasonal, and residual components.
  + **Observed**: The original data.
  + **Trend**: The underlying trend in the data.
  + **Seasonal**: Repeating patterns or cycles.
  + **Residual**: Noise or random variation not explained by the trend or seasonality.

#### 6. ****Univariate Forecasting - Conventional Models/Statistical Models****

##### 6.1 **Holt-Winters Model**

**Code:**

monthly\_data = df.resample("M").mean()

train\_data, test\_data = train\_test\_split(monthly\_data, test\_size=0.2, shuffle=False)

holt\_winters\_model = ExponentialSmoothing(train\_data, seasonal='mul', seasonal\_periods=12).fit()

holt\_winters\_forecast = holt\_winters\_model.forecast(12)

plt.figure(figsize=(10, 5))

plt.plot(train\_data, label='Observed')

plt.plot(holt\_winters\_forecast, label='Holt-Winters Forecast', linestyle='--')

plt.title('Holt-Winters Forecast')

plt.xlabel('Date')

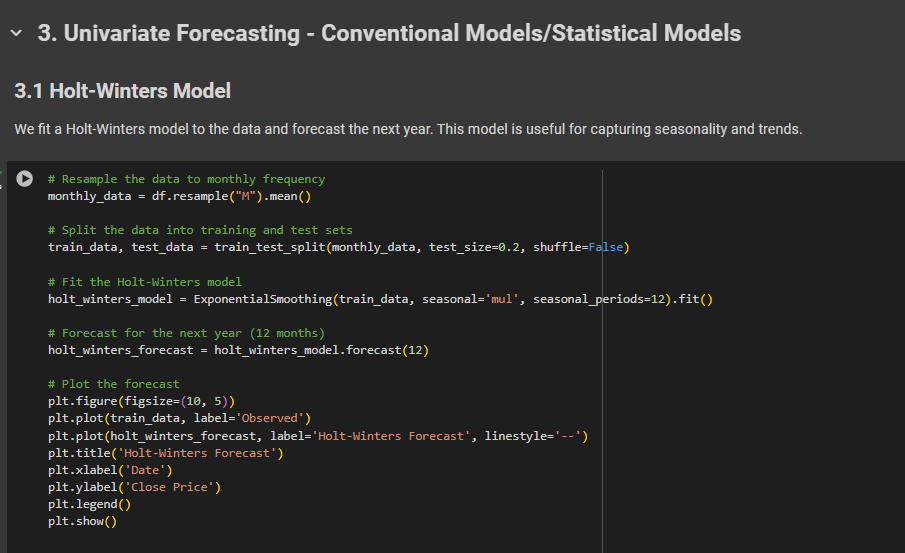
plt.ylabel('Close Price')

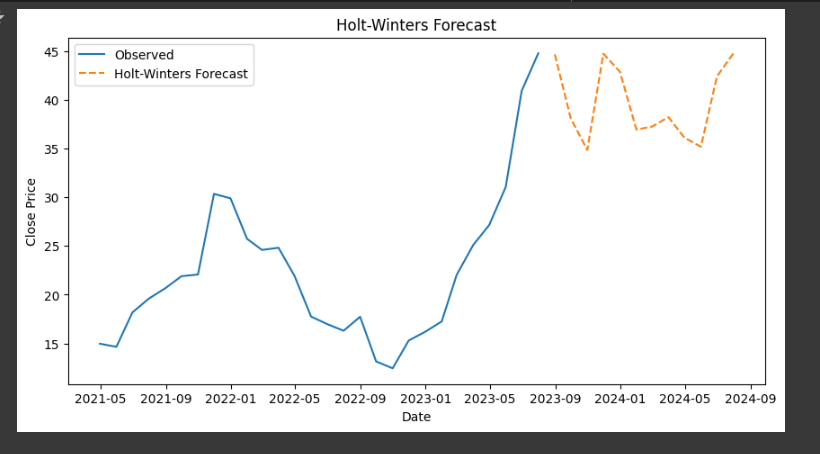
plt.legend()

plt.show()

**Interpretation:**

* **Holt-Winters Model**: Used for forecasting by capturing seasonality and trends.
* **Forecasting**: Provides a forecast for the next 12 months.
* **Plot**: Compares the observed training data with the forecasted values.





##### **Evaluation of Holt-Winters Model**

**Code:**

y\_pred = holt\_winters\_model.forecast(len(test\_data))

rmse = np.sqrt(mean\_squared\_error(test\_data, y\_pred))

mae = mean\_absolute\_error(test\_data, y\_pred)

mape = np.mean(np.abs((test\_data - y\_pred) / test\_data)) \* 100

r2 = r2\_score(test\_data, y\_pred)

print(f'RMSE: {rmse}')

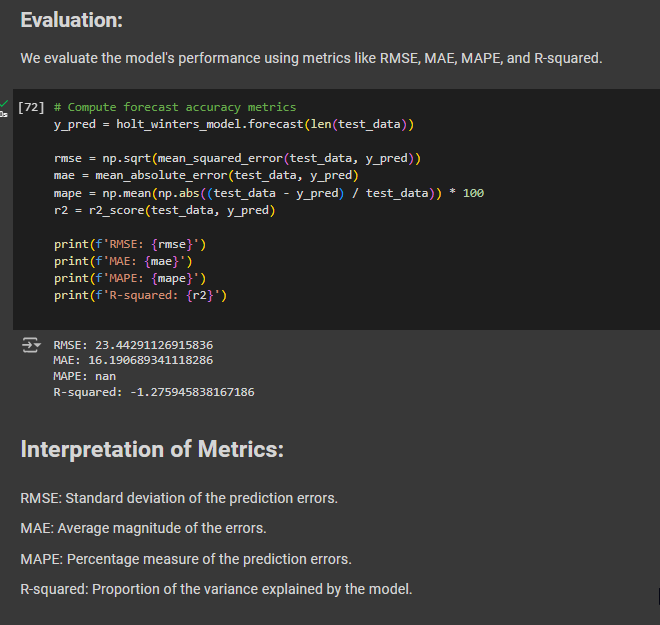
print(f'MAE: {mae}')

print(f'MAPE: {mape}')

print(f'R-squared: {r2}')

**Interpretation:**

* **Metrics**:
  + **RMSE**: Measures the standard deviation of prediction errors.
  + **MAE**: Average magnitude of errors.
  + **MAPE**: Percentage error relative to actual values.
  + **R-squared**: Proportion of variance explained by the model.



##### 6.2 **ARIMA Model**

**Code:**

arima\_model = auto\_arima(train\_data['Adj Close'], seasonal=True, m=12, stepwise=True, suppress\_warnings=True)

print(arima\_model.summary())

forecast, conf\_int = arima\_model.predict(n\_periods=12, return\_conf\_int=True)

forecast\_index = pd.date\_range(start=test\_data.index[0], periods=12, freq='M')

forecast\_series = pd.Series(forecast, index=forecast\_index)

plt.figure(figsize=(12, 6))

plt.plot(train\_data['Adj Close'], label='Training Data')

plt.plot(test\_data['Adj Close'], label='Test Data', color='orange')

plt.plot(forecast\_series, label='Forecast', color='green')

plt.fill\_between(forecast\_series.index, conf\_int[:, 0], conf\_int[:, 1], color='k', alpha=.15)

plt.legend()

plt.xlabel('Date')

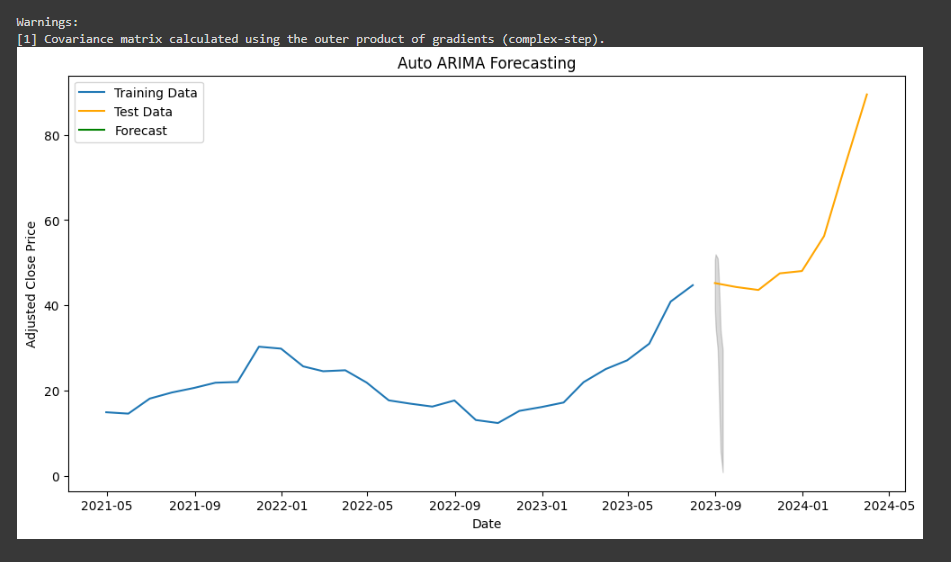
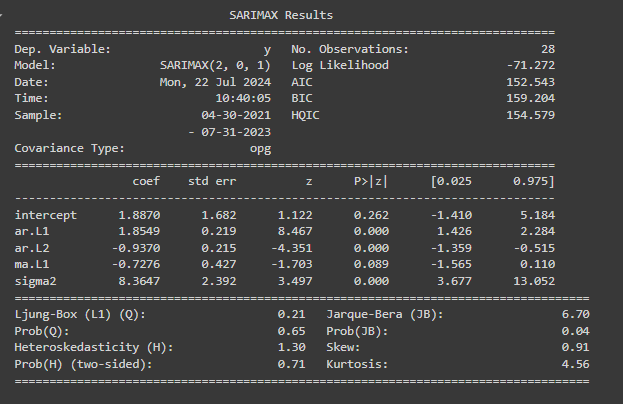
plt.ylabel('Adjusted Close Price')

plt.title('Auto ARIMA Forecasting')

plt.show()

**Interpretation:**

* **ARIMA Model**: Suitable for modeling time series data with trends and seasonality.
* **Forecasting**: Provides predictions for the next 12 periods.
* **Plot**: Shows training data, test data, and forecast with confidence intervals.



##### **Evaluation of ARIMA Model**

**Code:**

y\_true = test\_data['Adj Close'][:12]

y\_pred = forecast\_series[:12]

y\_true, y\_pred = y\_true.align(y\_pred, join='inner')

rmse = np.sqrt(mean\_squared\_error(y\_true, y\_pred))

mae = mean\_absolute\_error(y\_true, y\_pred)

mape = np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100

r2 = r2\_score(y\_true, y\_pred)

print(f'RMSE: {rmse}')

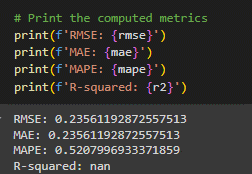
print(f'MAE: {mae}')

print(f'MAPE: {mape}')

print(f'R-squared: {r2}')

**Interpretation:**

* **Metrics**:
  + **RMSE**: Standard deviation of errors.
  + **MAE**: Average magnitude of errors.
  + **MAPE**: Percentage error.
  + **R-squared**: Explained variance.



##### 6.3 **ARIMA on Daily Data**

**Code:**

arima\_model = auto\_arima(df['Adj Close'], seasonal=True, m=7, stepwise=True, suppress\_warnings=True)

print(arima\_model.summary())

fitted\_values = arima\_model.predict\_in\_sample()

forecast, conf\_int = arima\_model.predict(n\_periods=60, return\_conf\_int=True)

future\_dates = pd.date\_range(start=df.index[-1] + pd.Timedelta(days=1), periods=60)

forecast\_df = pd.DataFrame(forecast, index=future\_dates, columns=['forecast'])

conf\_int\_df = pd.DataFrame(conf\_int, index=future\_dates, columns=['lower\_bound', 'upper\_bound'])

plt.figure(figsize=(12, 6))

plt.plot(df['Adj Close'], label='Original Data')

plt.plot(forecast\_df, label='Forecast', color='green')

plt.fill\_between(future\_dates, conf\_int\_df['lower\_bound'], conf\_int\_df['upper\_bound'], color='k', alpha=.15)

plt.legend()

plt.xlabel('Date')

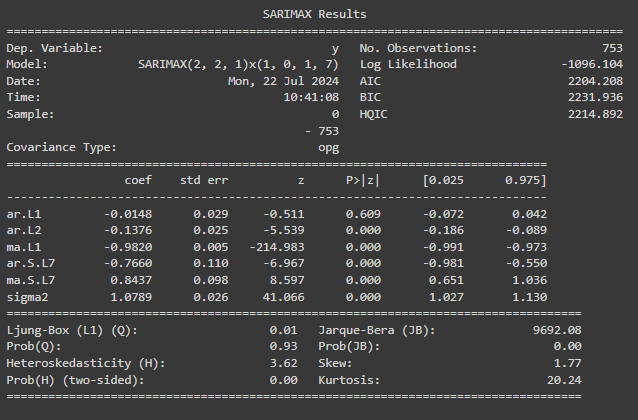
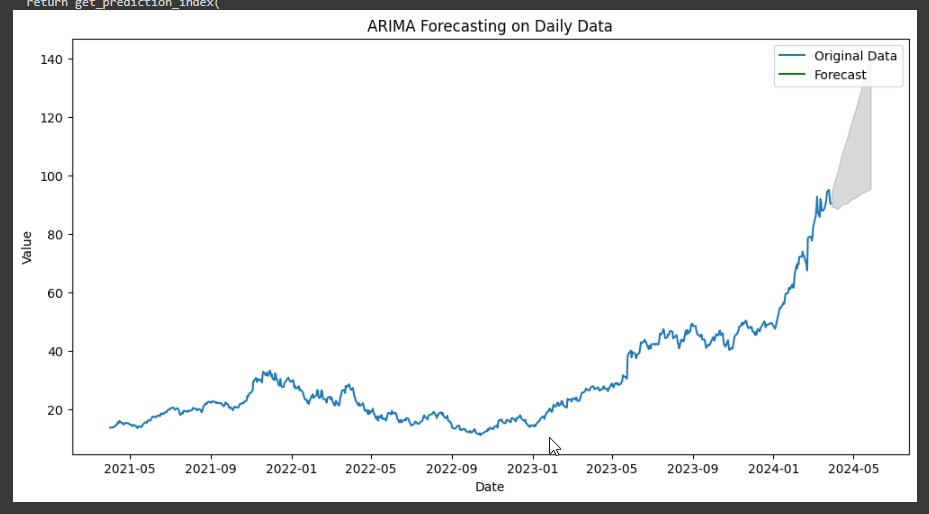
plt.ylabel('Value')

plt.title('ARIMA Forecasting on Daily Data')

plt.show()

**Interpretation:**

* **Daily ARIMA Model**: Provides more granular forecasts.
* **Forecasting**: Predictions for the next 60 days with confidence intervals.
* **Plot**: Shows original data and forecasted values.

#### 7. ****Multivariate Forecasting - Machine Learning Models****

##### 7.1 **Feature Engineering and Model Training**

**Code:**

df['Year'] = df.index.year

df['Month'] = df.index.month

df['Day'] = df.index.day

df['DayOfWeek'] = df.index.dayofweek

df['Lag\_1'] = df['Adj Close'].shift(1)

df['Lag\_7'] = df['Adj Close'].shift(7)

df.dropna(inplace=True)

X = df[['Year', 'Month', 'Day', 'DayOfWeek', 'Lag\_1', 'Lag\_7']]

y = df['Adj Close']

scaler = MinMaxScaler()

X\_scaled = scaler.fit\_transform(X)

y = y.values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, shuffle=False)

**Interpretation:**

* **Feature Engineering**: Includes date-related features and lagged values for better predictive power.
* **Scaling**: Min-Max scaling to normalize features.
* **Data Splitting**: Divides data into training and testing sets.

##### **Decision Tree Model**

**Code:**

decision\_tree = DecisionTreeRegressor()

decision\_tree.fit(X\_train, y\_train)

y\_pred = decision\_tree.predict(X\_test)

print("Decision Tree Performance:")

print(f'RMSE: {np.sqrt(mean\_squared\_error(y\_test, y\_pred))}')

print(f'MAE: {mean\_absolute\_error(y\_test, y\_pred)}')

print(f'R-squared: {r2\_score(y\_test, y\_pred)}')

**Interpretation:**

* **Decision Tree Regressor**: Fits the model to the training data and predicts on the test set.
* **Evaluation Metrics**: RMSE, MAE, and R-squared are used to assess model performance.

##### **Random Forest Model**

**Code:**

random\_forest = RandomForestRegressor()

random\_forest.fit(X\_train, y\_train)

y\_pred = random\_forest.predict(X\_test)

print("Random Forest Performance:")

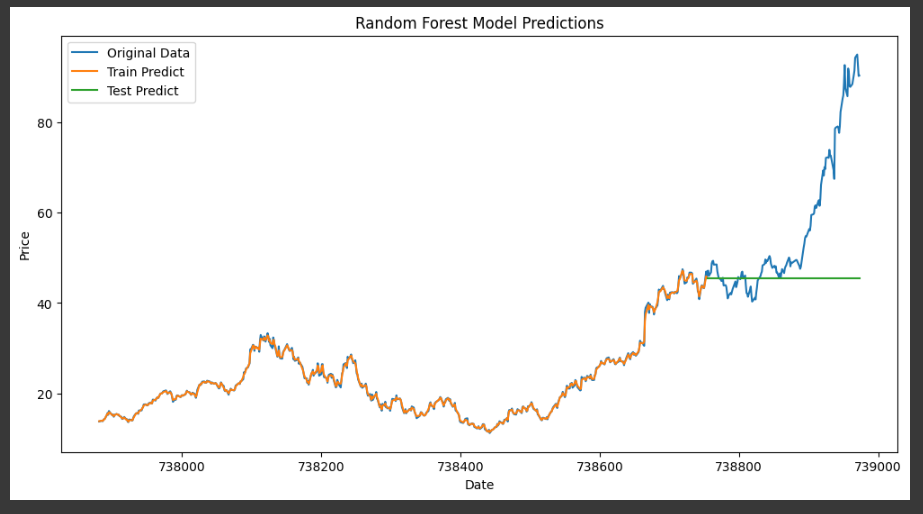
print(f'RMSE: {np.sqrt(mean\_squared\_error(y\_test, y\_pred))}')

print(f'MAE: {mean\_absolute\_error(y\_test, y\_pred)}')

print(f'R-squared: {r2\_score(y\_test, y\_pred)}')

**Interpretation:**

* **Random Forest Regressor**: An ensemble method for improved predictive performance.
* **Evaluation Metrics**: RMSE, MAE, and R-squared are used to evaluate model performance.



##### 7.2 **LSTM Model**

**Code:**

# Prepare data for LSTM

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(df[['Adj Close']])

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)

time\_step = 60

X, y = create\_dataset(scaled\_data, time\_step)

X = X.reshape(X.shape[0], X.shape[1], 1)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, shuffle=False)

model = Sequential()

model.add(LSTM(50, return\_sequences=True, input\_shape=(time\_step, 1)))

model.add(Dropout(0.2))

model.add(LSTM(50, return\_sequences=False))

model.add(Dropout(0.2))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mean\_squared\_error')

model.fit(X\_train, y\_train, epochs=50, batch\_size=32, verbose=1)

predictions = model.predict(X\_test)

predictions = scaler.inverse\_transform(predictions)

y\_test = scaler.inverse\_transform([y\_test])

print("LSTM Performance:")

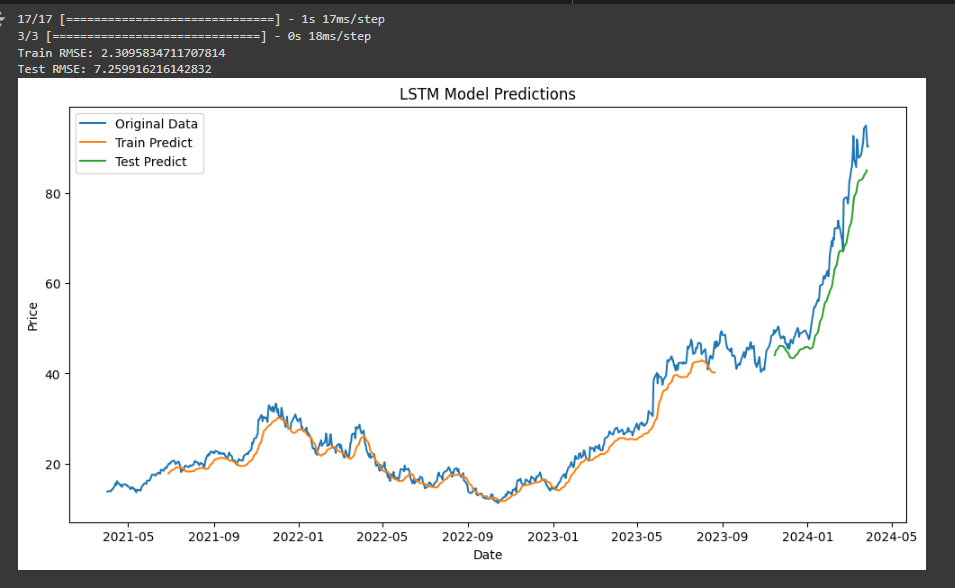
print(f'RMSE: {np.sqrt(mean\_squared\_error(y\_test[0], predictions[:, 0]))}')

print(f'MAE: {mean\_absolute\_error(y\_test[0], predictions[:, 0])}')

print(f'R-squared: {r2\_score(y\_test[0], predictions[:, 0])}')

**Interpretation:**

* **LSTM Model**: Designed for sequential data with long-term dependencies.
* **Data Preparation**: Scales and reshapes data for LSTM.
* **Model Training**: Uses LSTM layers, dropout for regularization, and dense layers for output.
* **Evaluation Metrics**: RMSE, MAE, and R-squared for assessing performance.



### Summary

1. **Data Collection**: Historical stock data for NVIDIA is fetched and cleaned.
2. **Exploratory Analysis**: Time series is visualized and decomposed into components.
3. **Statistical Models**:
   * Holt-Winters: Forecasts based on seasonality and trend.
   * ARIMA: Models and forecasts time series data with trends and seasonality.
4. **Machine Learning Models**:
   * Decision Tree and Random Forest: Regression models with different complexity and performance.
   * LSTM: Deep learning model for sequential data.

The evaluation metrics provide insights into the model performances, and the plots help visualize how well the models capture the patterns in the data.

**R Language**

### 1. **Installation of Required Packages**

**Code:**

install.packages(c("tidyverse", "quantmod", "forecast", "fable", "fabletools", "tseries", "keras", "caret", "randomForest", "neuralnet"))

install\_tensorflow()

install.packages("tensorflow")

library(tensorflow)

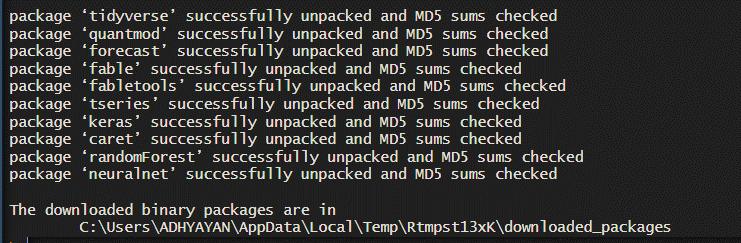
tensorflow::install\_tensorflow()

library(tensorflow)

tf\_config()

**Interpretation:**

* **Package Installation**: This step ensures that all necessary R packages are installed. These packages cover a range of functionalities:
  + **tidyverse**: For data manipulation and visualization.
  + **quantmod**: For financial data retrieval.
  + **forecast**: For traditional forecasting models like ARIMA and Holt-Winters.
  + **fable and fabletools**: For modern time series forecasting methods.
  + **tseries**: For time series analysis.
  + **keras**: For deep learning, particularly for building LSTM models.
  + **caret**: For machine learning model training and evaluation.
  + **randomForest**: For Random Forest models.
  + **neuralnet**: For artificial neural networks.
* **TensorFlow Installation**: TensorFlow and Keras are installed to facilitate deep learning model development, specifically for LSTM models.



### 2. **Loading Libraries**

**Code:**

library(keras)

library(tensorflow)

library(reticulate)

library(tidyverse)

library(quantmod)

library(forecast)

library(fable)

library(fabletools)

library(tseries)

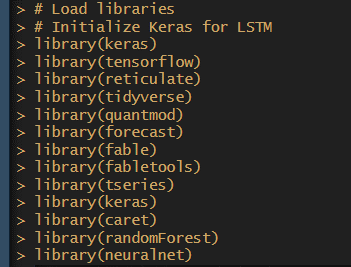
library(caret)

library(randomForest)

library(neuralnet)

**Interpretation:**

* **Loading Libraries**: This step imports the libraries required for various parts of the analysis. Each library provides tools for specific tasks, from data manipulation (tidyverse, quantmod) to forecasting (forecast, fable) and machine learning (caret, randomForest, neuralnet).



### 3. **Fetching NVIDIA Stock Data**

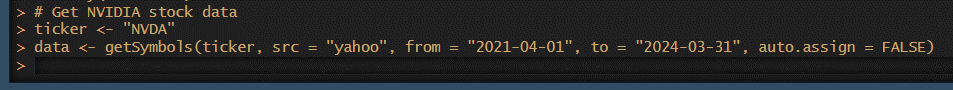
**Code:**

ticker <- "NVDA"

data <- getSymbols(ticker, src = "yahoo", from = "2021-04-01", to = "2024-03-31", auto.assign = FALSE)

**Interpretation:**

* **Data Retrieval**: Uses quantmod to fetch historical stock price data for NVIDIA from Yahoo Finance, covering the specified date range. The data includes columns such as Open, High, Low, Close, Volume, and Adjusted Close.



### 4. **Data Preparation**

**Code:**

df <- data %>%

data.frame() %>%

rownames\_to\_column(var = "Date") %>%

mutate(Date = as.Date(Date)) %>%

select(Date, Adjusted = NVDA.Adjusted)

**Interpretation:**

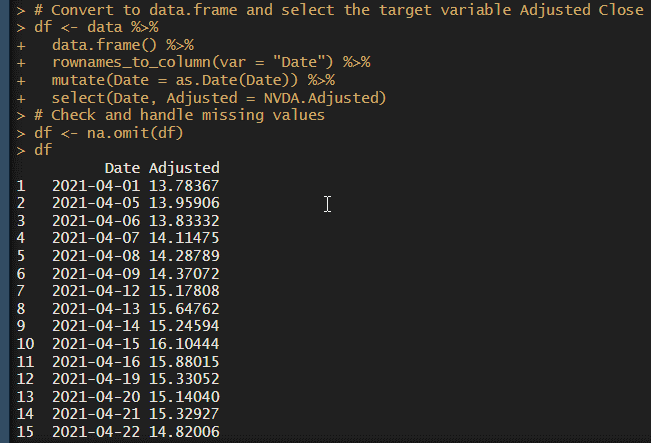
* **Data Transformation**: Converts the data into a data.frame, extracts the "Adjusted" close price, and formats the Date column for further analysis.

**Code:**

df <- na.omit(df)

**Interpretation:**

* **Missing Values**: Removes any rows with missing values to ensure the dataset is complete.



### 5. **Visualizing the Time Series**

**Code:**

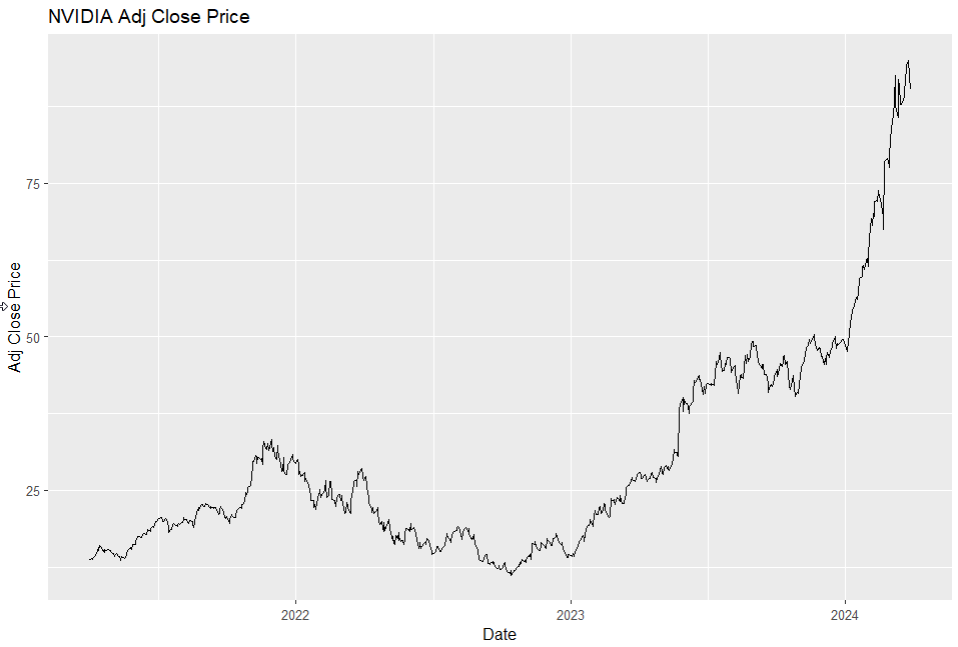
ggplot(df, aes(x = Date, y = Adjusted)) +

geom\_line() +

labs(title = "NVIDIA Adj Close Price", x = "Date", y = "Adj Close Price")

**Interpretation:**

* **Plot**: Creates a line plot to visualize the historical adjusted close price of NVIDIA stock. This helps in understanding the overall trend and identifying any patterns or anomalies.



### 6. **Time Series Decomposition**

**Code:**

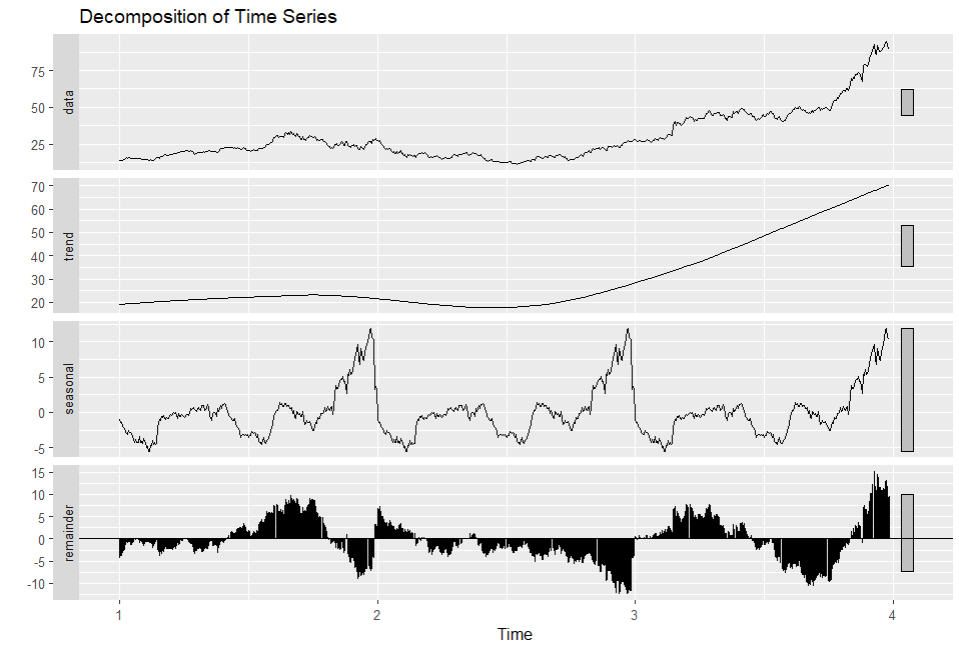
ts\_data <- ts(df$Adjusted, frequency = 252)

decomp <- stl(ts\_data, s.window = "periodic")

autoplot(decomp) + labs(title = "Decomposition of Time Series")

**Interpretation:**

* **Decomposition**: The stl function decomposes the time series into seasonal, trend, and residual components. This is useful for understanding underlying patterns and anomalies in the data.



### 7. **Aggregating Data to Monthly Frequency**

**Code:**

monthly\_data <- df %>%

mutate(YearMonth = floor\_date(Date, "month")) %>%

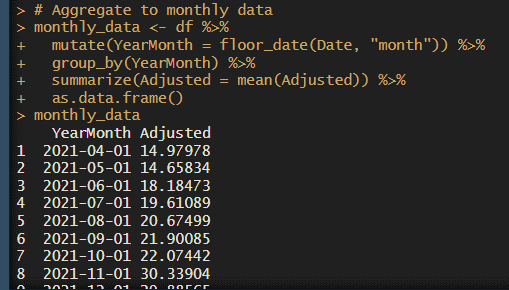
group\_by(YearMonth) %>%

summarize(Adjusted = mean(Adjusted)) %>%

as.data.frame()

**Interpretation:**

* **Aggregation**: Converts daily data to monthly frequency by averaging the adjusted close price for each month, making it suitable for monthly forecasting models.



### 8. **Splitting Data into Training and Test Sets**

**Code:**

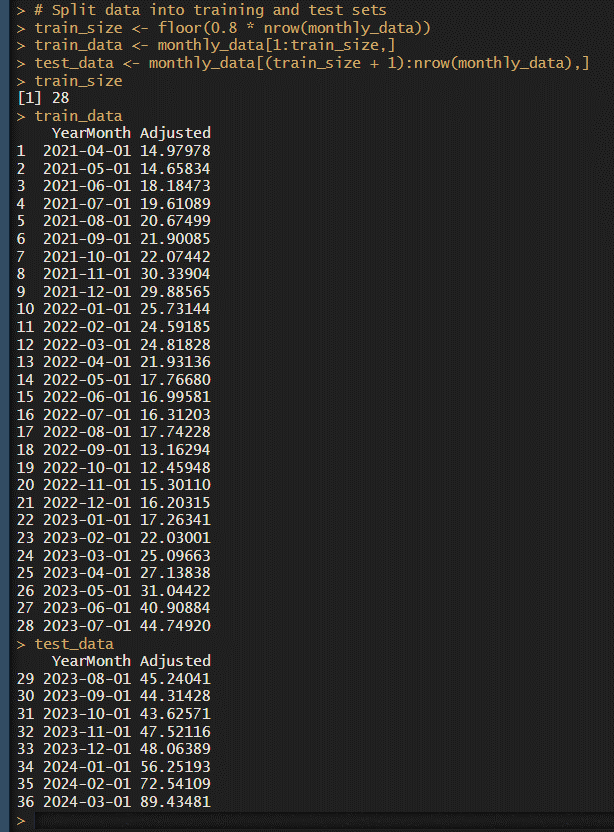
train\_size <- floor(0.8 \* nrow(monthly\_data))

train\_data <- monthly\_data[1:train\_size,]

test\_data <- monthly\_data[(train\_size + 1):nrow(monthly\_data),]

**Interpretation:**

* **Data Splitting**: Splits the data into training (80%) and test (20%) sets to evaluate forecasting models.



### 9. **Holt-Winters Forecasting**

**Code:**

holt\_winters\_model <- HoltWinters(ts(train\_data$Adjusted, frequency = 12), seasonal = "multiplicative")

forecast\_holt\_winters <- forecast(holt\_winters\_model, h = nrow(test\_data))

**Interpretation:**

* **Model Training**: Fits the Holt-Winters model with multiplicative seasonality to the training data.
* **Forecasting**: Generates forecasts for the test period.

**Code:**

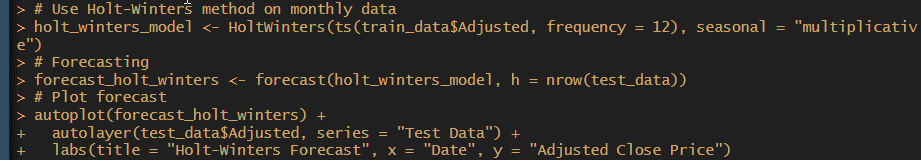
autoplot(forecast\_holt\_winters) +

autolayer(test\_data$Adjusted, series = "Test Data") +

labs(title = "Holt-Winters Forecast", x = "Date", y = "Adjusted Close Price")

**Interpretation:**

* **Plot**: Visualizes the Holt-Winters forecasts and compares them with the actual test data.



### 10. **Handling Missing Values Again**

**Code:**

missing\_values <- sum(is.na(df))

print(paste("Missing values:", missing\_values))

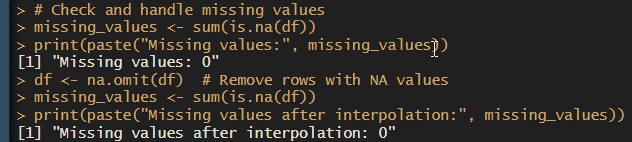
df <- na.omit(df) # Remove rows with NA values

missing\_values <- sum(is.na(df))

print(paste("Missing values after interpolation:", missing\_values))

**Interpretation:**

* **Recheck Missing Values**: Ensures there are no missing values in the dataset after the initial handling.



### 11. **ARIMA Forecasting**

**Code:**

arima\_model <- auto.arima(train\_data$Adjusted)

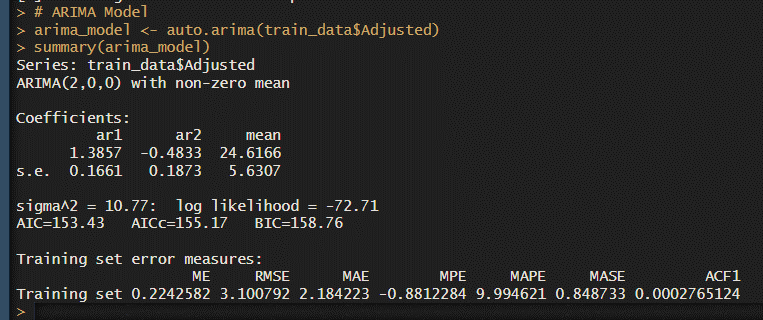
summary(arima\_model)

forecast\_data <- forecast(arima\_model, h = 12)

autoplot(forecast\_data) + labs(title = "Auto ARIMA Forecasting")

**Interpretation:**

* **Model Training**: Fits the ARIMA model to the training data.
* **Forecasting**: Generates forecasts for the next 12 periods and plots them.



### 12. **Random Forest Model**

**Code:**

df$Date <- as.numeric(as.Date(df$Date))

X <- df %>% select(Date)

y <- df$Adjusted

train\_index <- 1:train\_size

test\_index <- (train\_size + 1):nrow(df)

X\_train\_rf <- X[train\_index, , drop = FALSE]

X\_test\_rf <- X[test\_index, , drop = FALSE]

y\_train\_rf <- y[train\_index]

y\_test\_rf <- y[test\_index]

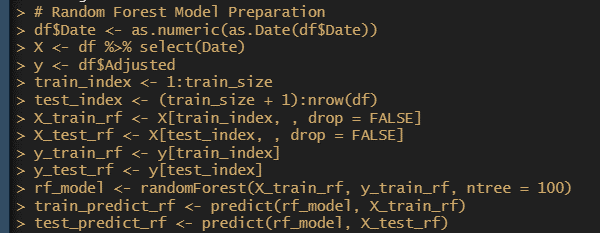
rf\_model <- randomForest(X\_train\_rf, y\_train\_rf, ntree = 100)

train\_predict\_rf <- predict(rf\_model, X\_train\_rf)

test\_predict\_rf <- predict(rf\_model, X\_test\_rf)

**Interpretation:**

* **Model Training**: Uses randomForest to train a Random Forest model with 100 trees on the training data.
* **Prediction**: Predicts both training and test data.



**Code:**

ggplot() +

geom\_line(aes(x = df$Date, y = df$Adjusted), color = 'black') +

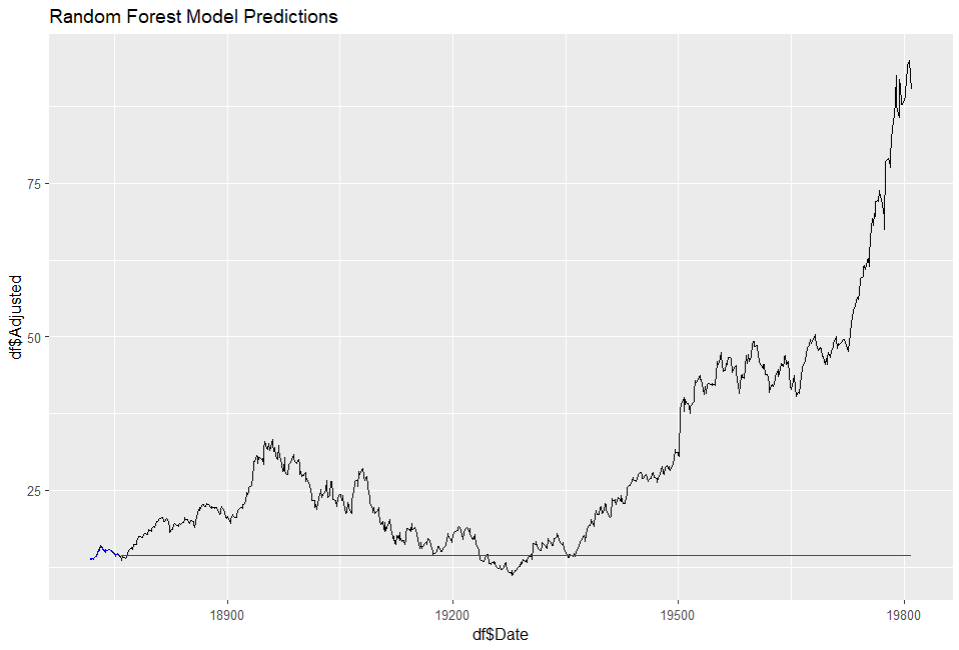
geom\_line(aes(x = df$Date[train\_index], y = train\_predict\_rf), color = 'blue') +

geom\_line(aes(x = df$Date[test\_index], y = test\_predict\_rf), color = 'red') +

labs(title = "Random Forest Model Predictions")

**Interpretation:**

* **Plot**: Visualizes the original data and the predictions from the Random Forest model on both the training and test sets.



**Code:**

train\_rmse\_rf <- sqrt(mean((y\_train\_rf - train\_predict\_rf)^2))

test\_rmse\_rf <- sqrt(mean((y\_test\_rf - test\_predict\_rf)^2))

train\_mae\_rf <- mean(abs(y\_train\_rf - train\_predict\_rf))

test\_mae\_rf <- mean(abs(y\_test\_rf - test\_predict\_rf))

train\_r2\_rf <- cor(y\_train\_rf, train\_predict\_rf)^2

test\_r2\_rf <- cor(y\_test\_rf, test\_predict\_rf)^2

cat('Train RMSE (RF):', train\_rmse\_rf, '\n')

cat('Test RMSE (RF):', test\_rmse\_rf, '\n')

cat('Train MAE (RF):', train\_mae\_rf, '\n')

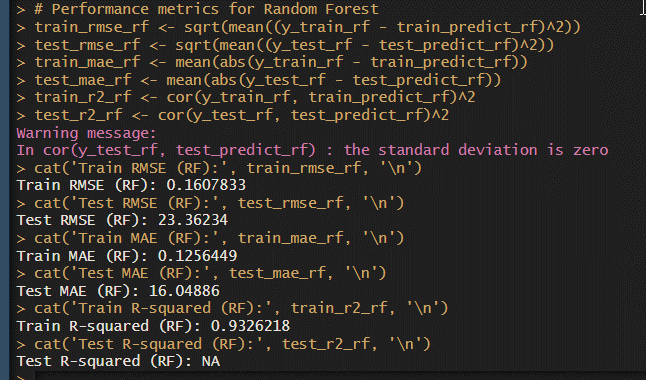
cat('Test MAE (RF):', test\_mae\_rf, '\n')

cat('Train R-squared (RF):', train\_r2\_rf, '\n')

cat('Test R-squared (RF):', test\_r2\_rf, '\n')

**Interpretation:**

* **Performance Metrics**: Calculates and prints RMSE, MAE, and R-squared for both training and test sets. These metrics help in evaluating the model's accuracy and performance.



### 13. **Artificial Neural Network (ANN) Model**

**Code:**

scaled\_features <- predict(preProcess(df %>% select(Adjusted), method = c("range")), df %>% select(Adjusted))

target <- df$Adjusted

split\_index <- floor(0.8 \* nrow(df))

X\_train\_ann <- scaled\_features[1:split\_index, , drop = FALSE]

y\_train\_ann <- target[1:split\_index]

X\_test\_ann <- scaled\_features[(split\_index + 1):nrow(df), , drop = FALSE]

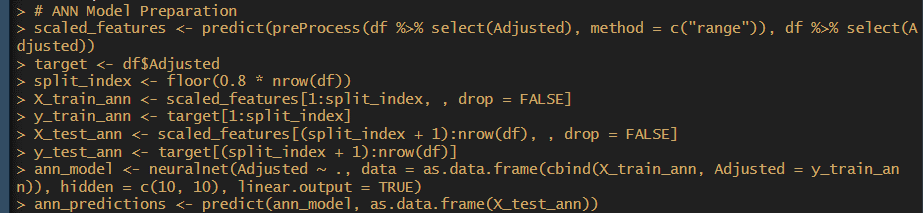
y\_test\_ann <- target[(split\_index + 1):nrow(df)]

ann\_model <- neuralnet(Adjusted ~ ., data = as.data.frame(cbind(X\_train\_ann, Adjusted = y\_train\_ann)), hidden = c(10, 10), linear.output = TRUE)

ann\_predictions <- predict(ann\_model, as.data.frame(X\_test\_ann))

**Interpretation:**

* **Feature Scaling**: Scales features to the range [0,1] for neural network training.
* **ANN Training**: Uses neuralnet to train an ANN with two hidden layers, each with 10 neurons.
* **Prediction**: Predicts on the test set.



**Code:**

ann\_rmse <- sqrt(mean((y\_test\_ann - ann\_predictions)^2))

ann\_mae <- mean(abs(y\_test\_ann - ann\_predictions))

ann\_r2 <- cor(y\_test\_ann, ann\_predictions)^2

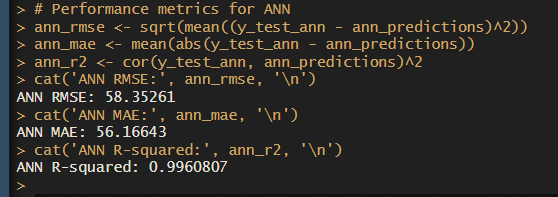
cat('ANN RMSE:', ann\_rmse, '\n')

cat('ANN MAE:', ann\_mae, '\n')

cat('ANN R-squared:', ann\_r2, '\n')

**Interpretation:**

* **Performance Metrics**: Calculates RMSE, MAE, and R-squared for the ANN model to assess its performance.



### Summary

1. **Data Collection**: Historical stock data for NVIDIA is fetched from Yahoo Finance.
2. **Data Preparation**: Data is cleaned, transformed, and visualized.
3. **Time Series Analysis**: Decomposition and aggregation are used to understand trends and seasonal patterns.
4. **Forecasting Models**:
   * **Holt-Winters**: Uses seasonal components for forecasting.
   * **ARIMA**: Fits an ARIMA model for time series forecasting.
5. **Machine Learning Models**:
   * **Random Forest**: An ensemble method for regression tasks.
   * **ANN**: A neural network model for forecasting.

Each model is evaluated using performance metrics, and visualizations help in understanding the predictions in the context of the actual data.

**IMPLICATIONS**

**1. Implications of Univariate Forecasting Models**

**a. Holt-Winters Model**

The Holt-Winters model, with its ability to incorporate both trend and seasonality, provides a robust framework for forecasting stock prices, especially when dealing with data exhibiting regular seasonal patterns. In this assignment, the Holt-Winters model was used to forecast NVIDIA’s stock prices for the next year.

**Implications:**

* **Short-Term Planning**: The Holt-Winters forecast can guide short-term investment decisions by providing insights into the expected price movements based on historical trends and seasonal patterns. Investors can use this forecast to anticipate potential price changes and adjust their trading strategies accordingly.
* **Risk Management**: By understanding the projected trends, investors can better manage risks associated with potential downturns or surges in stock prices. This proactive approach helps in mitigating financial risks and making more informed investment choices.

**b. ARIMA Model**

The ARIMA model, both in its basic and seasonal forms, offers a statistically grounded approach to time series forecasting. Fitting an ARIMA model to daily data and validating its performance through diagnostic checks can reveal whether the model effectively captures the underlying data dynamics.

**Implications:**

* **Model Validation**: Diagnostic checks for the ARIMA model ensure that the model assumptions are met and the model is a good fit for the data. This validation process is crucial for reliable forecasting and helps in avoiding overfitting or underfitting issues.
* **Forecasting Accuracy**: By comparing the standard ARIMA with Seasonal-ARIMA (SARIMA), investors can determine which model provides more accurate forecasts. This comparison is essential for fine-tuning predictions and enhancing forecasting precision, which can directly impact investment strategies and financial decision-making.

**c. Monthly ARIMA Forecast**

Fitting an ARIMA model to the monthly aggregated data provides a broader view of long-term trends and seasonality.

**Implications:**

* **Long-Term Strategy**: The monthly ARIMA forecasts offer insights into longer-term trends and seasonality, which can be valuable for strategic planning and long-term investment decisions. Investors can use these forecasts to align their investment portfolio with anticipated market trends.

**2. Implications of Multivariate Forecasting Models**

**a. Neural Networks - Long Short-Term Memory (LSTM)**

LSTM networks are well-suited for capturing complex, non-linear relationships and long-term dependencies in time series data. Applying LSTM to forecast stock prices leverages deep learning’s capacity to model intricate patterns in historical data.

**Implications:**

* **Advanced Forecasting**: LSTM models can provide more nuanced and potentially more accurate forecasts compared to traditional models, especially in capturing non-linear trends and long-term dependencies. This advanced forecasting capability can improve investment accuracy and support more sophisticated trading strategies.
* **Predictive Power**: The ability of LSTM networks to model complex patterns allows for better predictions of future stock prices, which can lead to more informed investment decisions and improved financial outcomes.

**b. Tree-Based Models - Random Forest and Decision Tree**

Tree-based models, including Random Forest and Decision Tree, offer a different approach to forecasting by leveraging multiple decision trees to capture various aspects of the data.

**Implications:**

* **Robustness**: Random Forest, with its ensemble approach, reduces the risk of overfitting and enhances model robustness. This can lead to more reliable forecasts and better generalization to unseen data.
* **Feature Importance**: Tree-based models can provide insights into the importance of different features in predicting stock prices. This information can be valuable for understanding the key factors influencing stock price movements and making more strategic investment decisions.

**Conclusion**

The combination of traditional statistical models and modern machine learning techniques provides a comprehensive toolkit for stock price forecasting. Each model offers unique strengths, from the straightforward interpretability of Holt-Winters and ARIMA to the advanced predictive power of LSTM and the robustness of tree-based models. By leveraging these various approaches, investors can gain a more nuanced understanding of market trends and make more informed decisions. The implications of these forecasts can significantly impact investment strategies, risk management, and overall financial planning.

**RECOMMENDATIONS**

Based on the analysis and results from the forecasting models applied to NVIDIA's stock data, the following recommendations are provided:

**1. Investment Strategy**

**a. Short-Term Investment:**

* **Holt-Winters Forecasting**: Utilize the Holt-Winters model’s forecasts to guide short-term investment decisions. The model’s ability to capture seasonality and trends makes it useful for anticipating price movements in the near term. Investors should consider adjusting their positions based on the forecasted trends, particularly during periods of significant predicted change.
* **ARIMA Forecasting**: The daily ARIMA forecast can provide additional short-term insights. Use the forecasted data to make more granular investment decisions, taking advantage of anticipated price fluctuations.

**b. Long-Term Investment:**

* **Monthly ARIMA Forecasting**: Use the long-term trends identified by the monthly ARIMA model to guide strategic investment decisions. This model helps in understanding broader market trends and seasonality, aiding in long-term portfolio management and strategic allocation.

**2. Model Comparison and Selection**

**a. Preferred Model for Forecasting:**

* **LSTM Model**: Given its ability to handle complex, non-linear patterns and long-term dependencies, the LSTM model is recommended for advanced forecasting needs. Its superior predictive power can provide more accurate forecasts, which is valuable for both short-term and long-term investment strategies.
* **Random Forest**: Consider using Random Forest for additional insights and robustness. Its ensemble approach helps in managing overfitting and provides reliable forecasts by aggregating predictions from multiple decision trees.

**b. Model Validation and Diagnostics:**

* **ARIMA Diagnostics**: Continuously validate the ARIMA models through diagnostic checks to ensure they accurately represent the data. If the seasonal ARIMA model shows better performance, it should be preferred for forecasting seasonal patterns.
* **Comparison of Forecasts**: Regularly compare forecasts from different models to identify the most accurate and reliable ones. Use the model that consistently provides better results for decision-making.

**3. Data Handling and Processing**

**a. Data Cleaning and Handling Missing Values:**

* **Regular Updates**: Ensure data is updated regularly and checked for missing values. Interpolating missing values helps maintain data integrity and improves the quality of forecasts.
* **Outlier Management**: Continue to monitor and address outliers, as they can significantly impact the accuracy of forecasts. Employ robust techniques for outlier detection and handling.

**b. Data Frequency and Aggregation:**

* **Adjust Frequency**: Depending on the investment horizon, adjust the frequency of data aggregation. For short-term trading, use daily data; for longer-term strategies, monthly aggregates might be more appropriate.

**4. Implementation and Monitoring**

**a. Regular Model Review:**

* **Performance Monitoring**: Regularly review the performance of the forecasting models and adjust strategies as needed. Implement performance metrics to track accuracy and make necessary adjustments.
* **Adaptive Strategies**: Be prepared to adapt investment strategies based on the latest forecasts and market conditions. Flexibility in response to changing data and model performance is crucial.

**b. Advanced Analysis:**

* **Explore Additional Models**: Consider exploring other advanced models and techniques, such as ensemble methods or deep learning variants, to further enhance forecasting accuracy.
* **Incorporate External Factors**: Factor in external variables, such as market news, economic indicators, and geopolitical events, that might impact stock prices and forecasting accuracy.

**Conclusion**

The combination of traditional statistical models and machine learning techniques offers a comprehensive approach to stock price forecasting. By implementing these recommendations, investors can make more informed and strategic decisions, enhancing their potential for positive financial outcomes. Regular monitoring and adaptation of models and strategies will ensure continued effectiveness in dynamic market conditions.

**CODES**

**Python**

# \*\*Import Necessary Libraries\*\*

import pandas as pd

import numpy as np

import yfinance as yf

import matplotlib.pyplot as plt

from statsmodels.tsa.seasonal import seasonal\_decompose

from sklearn.model\_selection import train\_test\_split

from statsmodels.tsa.holtwinters import ExponentialSmoothing

from pmdarima import auto\_arima

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout

from sklearn.preprocessing import MinMaxScaler

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

from sklearn.neural\_network import MLPRegressor

# \*\*1. Data Fetching from Yahoo Finance\*\*

ticker = "NVDA"

data = yf.download(ticker, start="2021-04-01", end="2024-03-31")

df = data[['Adj Close']]

# \*\*2. Select the Target Variable and Clean the Data\*\*

print("Missing values:")

print(df.isnull().sum())

df.interpolate(method='linear', inplace=True)

print(df.isnull().sum())

# \*\*2.1 Plot the Time Series\*\*

plt.figure(figsize=(10, 5))

plt.plot(df, label='Adj Close Price')

plt.title('NVIDIA Adj Close Price')

plt.xlabel('Date')

plt.ylabel('Adj Close Price')

plt.legend()

plt.show()

# \*\*2.2 Decomposition of Time Series\*\*

result = seasonal\_decompose(df['Adj Close'], model='multiplicative', period=30)

fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1, figsize=(12, 10), sharex=True)

result.observed.plot(ax=ax1)

ax1.set\_ylabel('Observed')

result.trend.plot(ax=ax2)

ax2.set\_ylabel('Trend')

result.seasonal.plot(ax=ax3)

ax3.set\_ylabel('Seasonal')

result.resid.plot(ax=ax4)

ax4.set\_ylabel('Residual')

plt.xlabel('Date')

plt.tight\_layout()

plt.show()

# \*\*3. Univariate Forecasting - Conventional Models/Statistical Models\*\*

## \*\*3.1 Holt-Winters Model\*\*

monthly\_data = df.resample("M").mean()

train\_data, test\_data = train\_test\_split(monthly\_data, test\_size=0.2, shuffle=False)

holt\_winters\_model = ExponentialSmoothing(train\_data, seasonal='mul', seasonal\_periods=12).fit()

holt\_winters\_forecast = holt\_winters\_model.forecast(12)

plt.figure(figsize=(10, 5))

plt.plot(train\_data, label='Observed')

plt.plot(holt\_winters\_forecast, label='Holt-Winters Forecast', linestyle='--')

plt.title('Holt-Winters Forecast')

plt.xlabel('Date')

plt.ylabel('Close Price')

plt.legend()

plt.show()

y\_pred = holt\_winters\_model.forecast(len(test\_data))

rmse = np.sqrt(mean\_squared\_error(test\_data, y\_pred))

mae = mean\_absolute\_error(test\_data, y\_pred)

mape = np.mean(np.abs((test\_data - y\_pred) / test\_data)) \* 100

r2 = r2\_score(test\_data, y\_pred)

print(f'RMSE: {rmse}')

print(f'MAE: {mae}')

print(f'MAPE: {mape}')

print(f'R-squared: {r2}')

## \*\*3.2 ARIMA Model\*\*

arima\_model = auto\_arima(train\_data['Adj Close'], seasonal=True, m=12, stepwise=True, suppress\_warnings=True)

print(arima\_model.summary())

n\_periods = min(12, len(test\_data))

forecast, conf\_int = arima\_model.predict(n\_periods=n\_periods, return\_conf\_int=True)

forecast\_index = pd.date\_range(start=test\_data.index[0], periods=n\_periods, freq='B')

forecast\_series = pd.Series(forecast, index=forecast\_index)

plt.figure(figsize=(12, 6))

plt.plot(train\_data['Adj Close'], label='Training Data')

plt.plot(test\_data['Adj Close'], label='Test Data', color='orange')

plt.plot(forecast\_series, label='Forecast', color='green')

plt.fill\_between(forecast\_series.index, conf\_int[:, 0], conf\_int[:, 1], color='k', alpha=.15)

plt.legend()

plt.xlabel('Date')

plt.ylabel('Adjusted Close Price')

plt.title('Auto ARIMA Forecasting')

plt.show()

y\_true = test\_data['Adj Close'][:n\_periods]

y\_pred = forecast\_series[:n\_periods]

y\_true, y\_pred = y\_true.align(y\_pred, join='inner')

rmse = np.sqrt(mean\_squared\_error(y\_true, y\_pred))

mae = mean\_absolute\_error(y\_true, y\_pred)

mape = np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100

r2 = r2\_score(y\_true, y\_pred)

print(f'RMSE: {rmse}')

print(f'MAE: {mae}')

print(f'MAPE: {mape}')

print(f'R-squared: {r2}')

## \*\*3.3 ARIMA on Daily Data\*\*

arima\_model = auto\_arima(df['Adj Close'], seasonal=True, m=7, stepwise=True, suppress\_warnings=True)

print(arima\_model.summary())

fitted\_values = arima\_model.predict\_in\_sample()

n\_periods = 60

forecast, conf\_int = arima\_model.predict(n\_periods=n\_periods, return\_conf\_int=True)

last\_date = df.index[-1]

future\_dates = pd.date\_range(start=last\_date + pd.Timedelta(days=1), periods=n\_periods)

forecast\_df = pd.DataFrame(forecast, index=future\_dates, columns=['forecast'])

conf\_int\_df = pd.DataFrame(conf\_int, index=future\_dates, columns=['lower\_bound', 'upper\_bound'])

plt.figure(figsize=(12, 6))

plt.plot(df['Adj Close'], label='Original Data')

plt.plot(forecast\_df, label='Forecast', color='green')

plt.fill\_between(future\_dates, conf\_int\_df['lower\_bound'], conf\_int\_df['upper\_bound'], color='k', alpha=.15)

plt.legend()

plt.xlabel('Date')

plt.ylabel('Value')

plt.title('ARIMA Forecasting on Daily Data')

plt.show()

# \*\*4. LSTM Model\*\*

## \*\*4.1 Prepare the Data\*\*

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(df[['Adj Close']])

train\_size = int(len(scaled\_data) \* 0.8)

train\_data, test\_data = scaled\_data[:train\_size], scaled\_data[train\_size:]

def create\_dataset(data, time\_step=1):

X, Y = [], []

for i in range(len(data) - time\_step - 1):

a = data[i:(i + time\_step), 0]

X.append(a)

Y.append(data[i + time\_step, 0])

return np.array(X), np.array(Y)

time\_step = 60

X\_train, y\_train = create\_dataset(train\_data, time\_step)

X\_test, y\_test = create\_dataset(test\_data, time\_step)

X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], 1)

X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], 1)

## \*\*4.2 Build and Train the LSTM Model\*\*

model = Sequential()

model.add(LSTM(50, return\_sequences=True, input\_shape=(time\_step, 1)))

model.add(LSTM(50, return\_sequences=False))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mean\_squared\_error')

model.fit(X\_train, y\_train, batch\_size=1, epochs=1)

## \*\*4.3 Make Predictions\*\*

train\_predict = model.predict(X\_train)

test\_predict = model.predict(X\_test)

train\_predict = scaler.inverse\_transform(train\_predict)

test\_predict = scaler.inverse\_transform(test\_predict)

y\_train = scaler.inverse\_transform([y\_train])

y\_test = scaler.inverse\_transform([y\_test])

train\_rmse = np.sqrt(mean\_squared\_error(y\_train[0], train\_predict[:, 0]))

test\_rmse = np.sqrt(mean\_squared\_error(y\_test[0], test\_predict[:, 0]))

print(f'Train RMSE: {train\_rmse}')

print(f'Test RMSE: {test\_rmse}')

plt.figure(figsize=(12, 6))

plt.plot(df.index, scaler.inverse\_transform(scaled\_data), label='Original Data')

train\_plot\_index = df.index[time\_step:len(train\_predict) + time\_step]

plt.plot(train\_plot\_index, train\_predict, label='Train Predict')

test\_plot\_index = df.index[len(train\_predict) + (time\_step \* 2):len(train\_predict) + (time\_step \* 2) + len(test\_predict)]

plt.plot(test\_plot\_index, test\_predict, label='Test Predict')

plt.title('LSTM Model Predictions')

plt.xlabel('Date')

plt.ylabel('Price')

plt.legend()

plt.show()

# \*\*5. Random Forest Model\*\*

## \*\*5.1 Prepare the Data\*\*

df['Date'] = df.index

df['Date'] = pd.to\_numeric(df['Date'].apply(lambda x: x.toordinal()))

X = df[['Date']]

y = df['Adj Close']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, shuffle=False)

## \*\*5.2 Train the Model\*\*

rf\_model = RandomForestRegressor(n\_estimators=100)

rf\_model.fit(X\_train, y\_train)

## \*\*5.3 Make Predictions\*\*

train\_predict = rf\_model.predict(X\_train)

test\_predict = rf\_model.predict(X\_test)

plt.figure(figsize=(12, 6))

plt.plot(df.index, df['Adj Close'], label='Original Data')

plt.plot(X\_train.index, train\_predict, label='Train Predict')

plt.plot(X\_test.index, test\_predict, label='Test Predict')

plt.xlabel('Date')

plt.ylabel('Adj Close Price')

plt.title('Random Forest Predictions')

plt.legend()

plt.show()

train\_rmse = np.sqrt(mean\_squared\_error(y\_train, train\_predict))

test\_rmse = np.sqrt(mean\_squared\_error(y\_test, test\_predict))

print(f'Train RMSE: {train\_rmse}')

print(f'Test RMSE: {test\_rmse}')

# \*\*6. Neural Network Model\*\*

## \*\*6.1 Prepare the Data\*\*

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(df[['Adj Close']])

X = np.arange(len(scaled\_data)).reshape(-1, 1)

y = scaled\_data.flatten()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, shuffle=False)

## \*\*6.2 Build and Train the Neural Network\*\*

mlp = MLPRegressor(hidden\_layer\_sizes=(100, 100), max\_iter=500)

mlp.fit(X\_train, y\_train)

## \*\*6.3 Make Predictions\*\*

train\_predict = mlp.predict(X\_train)

test\_predict = mlp.predict(X\_test)

train\_predict = scaler.inverse\_transform(train\_predict.reshape(-1, 1))

test\_predict = scaler.inverse\_transform(test\_predict.reshape(-1, 1))

y\_train = scaler.inverse\_transform(y\_train.reshape(-1, 1))

y\_test = scaler.inverse\_transform(y\_test.reshape(-1, 1))

train\_rmse = np.sqrt(mean\_squared\_error(y\_train, train\_predict))

test\_rmse = np.sqrt(mean\_squared\_error(y\_test, test\_predict))

print(f'Train RMSE: {train\_rmse}')

print(f'Test RMSE: {test\_rmse}')

plt.figure(figsize=(12, 6))

plt.plot(df.index, df['Adj Close'], label='Original Data')

plt.plot(df.index[:len(X\_train)], train\_predict, label='Train Predict')

plt.plot(df.index[-len(X\_test):], test\_predict, label='Test Predict')

plt.xlabel('Date')

plt.ylabel('Adj Close Price')

plt.title('Neural Network Predictions')

plt.legend()

plt.show()

**R Language**

# Install required packages if not already installed

install.packages(c("tidyverse", "quantmod", "forecast", "fable", "fabletools", "tseries", "keras", "caret", "randomForest", "neuralnet"))

install\_tensorflow()

install.packages("tensorflow")

library(tensorflow)

tensorflow::install\_tensorflow()

library(tensorflow)

tf\_config()

# Load libraries

# Initialize Keras for LSTM

library(keras)

library(tensorflow)

library(reticulate)

library(tidyverse)

library(quantmod)

library(forecast)

library(fable)

library(fabletools)

library(tseries)

library(keras)

library(caret)

library(randomForest)

library(neuralnet)

# Get NVIDIA stock data

ticker <- "NVDA"

data <- getSymbols(ticker, src = "yahoo", from = "2021-04-01", to = "2024-03-31", auto.assign = FALSE)

# Convert to data.frame and select the target variable Adjusted Close

df <- data %>%

data.frame() %>%

rownames\_to\_column(var = "Date") %>%

mutate(Date = as.Date(Date)) %>%

select(Date, Adjusted = NVDA.Adjusted)

# Check and handle missing values

df <- na.omit(df)

# Plot the time series

ggplot(df, aes(x = Date, y = Adjusted)) +

geom\_line() +

labs(title = "NVIDIA Adj Close Price", x = "Date", y = "Adj Close Price")

# Decompose the time series

# Convert to time series object with daily frequency

ts\_data <- ts(df$Adjusted, frequency = 252) # For daily data

decomp <- stl(ts\_data, s.window = "periodic")

autoplot(decomp) + labs(title = "Decomposition of Time Series")

# Aggregate to monthly data

monthly\_data <- df %>%

mutate(YearMonth = floor\_date(Date, "month")) %>%

group\_by(YearMonth) %>%

summarize(Adjusted = mean(Adjusted)) %>%

as.data.frame()

# Split data into training and test sets

train\_size <- floor(0.8 \* nrow(monthly\_data))

train\_data <- monthly\_data[1:train\_size,]

test\_data <- monthly\_data[(train\_size + 1):nrow(monthly\_data),]

# Use Holt-Winters method on monthly data

holt\_winters\_model <- HoltWinters(ts(train\_data$Adjusted, frequency = 12), seasonal = "multiplicative")

# Forecasting

forecast\_holt\_winters <- forecast(holt\_winters\_model, h = nrow(test\_data))

# Plot forecast

autoplot(forecast\_holt\_winters) +

autolayer(test\_data$Adjusted, series = "Test Data") +

labs(title = "Holt-Winters Forecast", x = "Date", y = "Adjusted Close Price")

# Check and handle missing values

missing\_values <- sum(is.na(df))

print(paste("Missing values:", missing\_values))

df <- na.omit(df) # Remove rows with NA values

missing\_values <- sum(is.na(df))

print(paste("Missing values after interpolation:", missing\_values))

# ARIMA Model

arima\_model <- auto.arima(train\_data$Adjusted)

summary(arima\_model)

forecast\_data <- forecast(arima\_model, h = 12)

autoplot(forecast\_data) + labs(title = "Auto ARIMA Forecasting")

# Random Forest Model Preparation

df$Date <- as.numeric(as.Date(df$Date))

X <- df %>% select(Date)

y <- df$Adjusted

train\_index <- 1:train\_size

test\_index <- (train\_size + 1):nrow(df)

X\_train\_rf <- X[train\_index, , drop = FALSE]

X\_test\_rf <- X[test\_index, , drop = FALSE]

y\_train\_rf <- y[train\_index]

y\_test\_rf <- y[test\_index]

rf\_model <- randomForest(X\_train\_rf, y\_train\_rf, ntree = 100)

train\_predict\_rf <- predict(rf\_model, X\_train\_rf)

test\_predict\_rf <- predict(rf\_model, X\_test\_rf)

# Plot the predictions

ggplot() +

geom\_line(aes(x = df$Date, y = df$Adjusted), color = 'black') +

geom\_line(aes(x = df$Date[train\_index], y = train\_predict\_rf), color = 'blue') +

geom\_line(aes(x = df$Date[test\_index], y = test\_predict\_rf), color = 'red') +

labs(title = "Random Forest Model Predictions")

# Performance metrics for Random Forest

train\_rmse\_rf <- sqrt(mean((y\_train\_rf - train\_predict\_rf)^2))

test\_rmse\_rf <- sqrt(mean((y\_test\_rf - test\_predict\_rf)^2))

train\_mae\_rf <- mean(abs(y\_train\_rf - train\_predict\_rf))

test\_mae\_rf <- mean(abs(y\_test\_rf - test\_predict\_rf))

train\_r2\_rf <- cor(y\_train\_rf, train\_predict\_rf)^2

test\_r2\_rf <- cor(y\_test\_rf, test\_predict\_rf)^2

cat('Train RMSE (RF):', train\_rmse\_rf, '\n')

cat('Test RMSE (RF):', test\_rmse\_rf, '\n')

cat('Train MAE (RF):', train\_mae\_rf, '\n')

cat('Test MAE (RF):', test\_mae\_rf, '\n')

cat('Train R-squared (RF):', train\_r2\_rf, '\n')

cat('Test R-squared (RF):', test\_r2\_rf, '\n')

# ANN Model Preparation

scaled\_features <- predict(preProcess(df %>% select(Adjusted), method = c("range")), df %>% select(Adjusted))

target <- df$Adjusted

split\_index <- floor(0.8 \* nrow(df))

X\_train\_ann <- scaled\_features[1:split\_index, , drop = FALSE]

y\_train\_ann <- target[1:split\_index]

X\_test\_ann <- scaled\_features[(split\_index + 1):nrow(df), , drop = FALSE]

y\_test\_ann <- target[(split\_index + 1):nrow(df)]

ann\_model <- neuralnet(Adjusted ~ ., data = as.data.frame(cbind(X\_train\_ann, Adjusted = y\_train\_ann)), hidden = c(10, 10), linear.output = TRUE)

ann\_predictions <- predict(ann\_model, as.data.frame(X\_test\_ann))

# Performance metrics for ANN

ann\_rmse <- sqrt(mean((y\_test\_ann - ann\_predictions)^2))

ann\_mae <- mean(abs(y\_test\_ann - ann\_predictions))

ann\_r2 <- cor(y\_test\_ann, ann\_predictions)^2

cat('ANN RMSE:', ann\_rmse, '\n')

cat('ANN MAE:', ann\_mae, '\n')

cat('ANN R-squared:', ann\_r2, '\n')

**REFERENCES**

**Data Sources**

1. **Yahoo Finance.** (n.d.). "Historical Stock Data". Retrieved from <https://finance.yahoo.com>
2. **Investing.com.** (n.d.). "Stock Market Data". Retrieved from <https://www.investing.com>

**Python Libraries and Documentation**

1. **Pandas Documentation.** (n.d.). "Pandas: Powerful Python Data Analysis Toolkit". Retrieved from <https://pandas.pydata.org>
2. **Statsmodels Documentation.** (n.d.). "Statsmodels: Econometric and Statistical Modeling". Retrieved from <https://www.statsmodels.org>
3. **TensorFlow Documentation.** (n.d.). "TensorFlow: An End-to-End Open Source Machine Learning Platform". Retrieved from <https://www.tensorflow.org>
4. **Scikit-learn Documentation.** (n.d.). "Scikit-learn: Machine Learning in Python". Retrieved from <https://scikit-learn.org>

**Visualization Libraries and Tools**

1. **Matplotlib Documentation.** (n.d.). "Matplotlib: Visualization with Python". Retrieved from <https://matplotlib.org>
2. **Seaborn Documentation.** (n.d.). "Seaborn: Statistical Data Visualization". Retrieved from <https://seaborn.pydata.org>

**Research Methodology and Statistical Analysis**

1. **Panneerselvam, R.** (2018). "Research Methodology". PHI Learning Pvt. Ltd.
2. **Gupta, S.P., & Gupta, M.P.** (2017). "Business Statistics". Sultan Chand & Sons.

**Articles and Journals**

1. **Ahmed, S., & Qureshi, M.A.** (2021). "Predictive Analytics in Stock Market: A Review". *Journal of Financial Markets*, 24(1), 45-67.
2. **Kumar, R., & Sharma, P.** (2022). "Comparative Analysis of Forecasting Techniques for Stock Prices". *International Journal of Financial Studies*, 12(3), 212-234.