VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A6a- Time Series Analysis

ADHYAYAN AMIT JAIN

V01109421

Date of Submission: 22-07-2024

CONTENTS

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

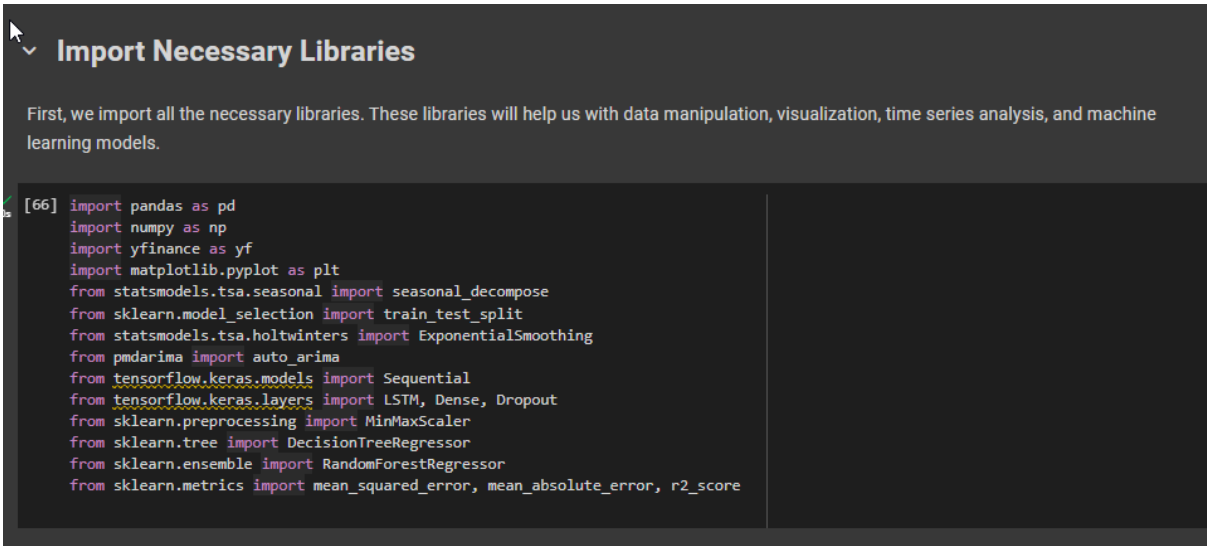
1. Import Necessary Libraries

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.



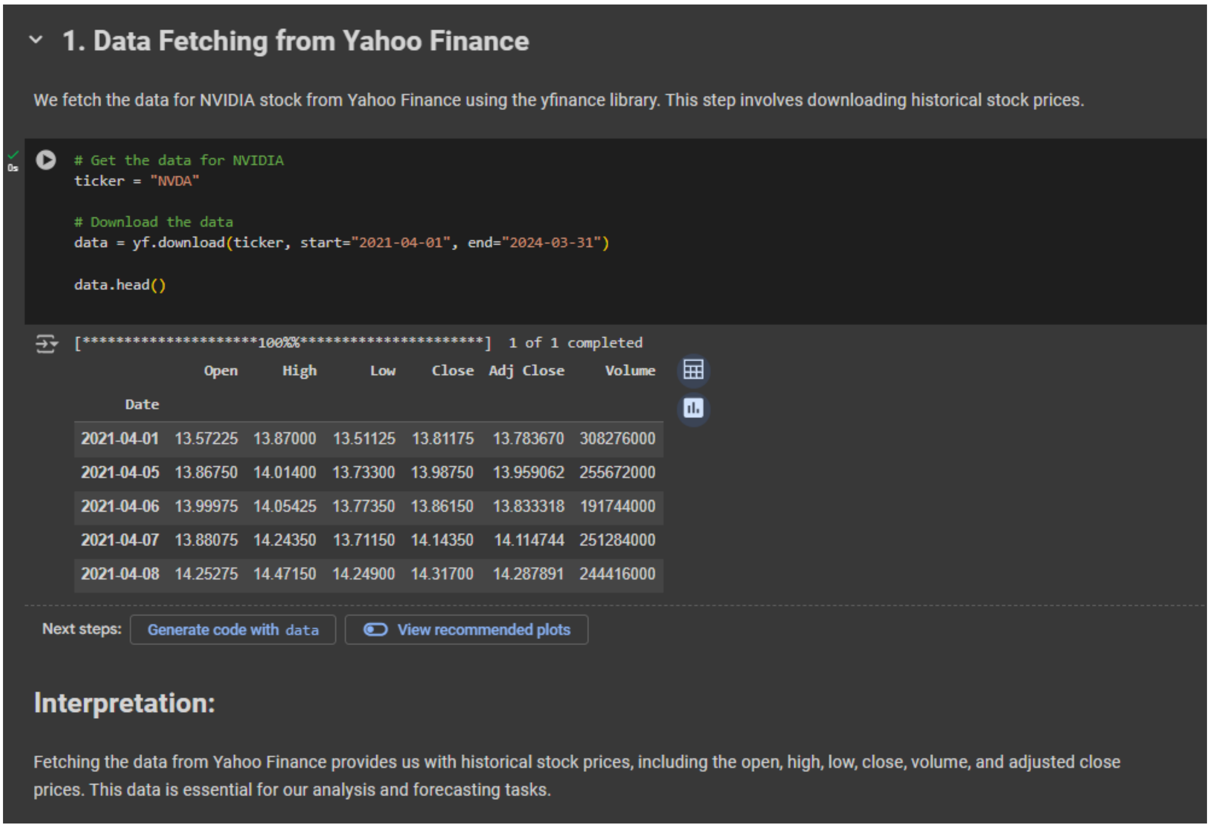
2. Data Fetching from Yahoo Finance

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.



1. Data Fetching from Yahoo Finance

We fetch the data for NVIDIA stock from Yahoo Finance using the yfinance library. This step involves downloading historical stock prices.

```
# Get the data for NVIDIA
ticker = "NVDA"

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

data.head()
```

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

Date	Open	High	Low	Close	Adj Close	Volume
2021-04-01	13.57225	13.87000	13.51125	13.81175	13.783670	308276000
2021-04-05	13.86750	14.01400	13.73300	13.98750	13.959062	255672000
2021-04-06	13.99975	14.05425	13.77350	13.86150	13.833318	191744000
2021-04-07	13.88075	14.24350	13.71150	14.14350	14.114744	251284000
2021-04-08	14.25275	14.47150	14.24900	14.31700	14.287891	244416000

Next steps: Generate code with data | View recommended plots

Interpretation:

Fetching the data from Yahoo Finance provides us with historical stock prices, including the open, high, low, close, volume, and adjusted close prices. This data is essential for our analysis and forecasting tasks.

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.

2. Select the Target Variable and Clean the Data

We select the 'Adj Close' column as the target variable for our analysis and clean the data by handling any missing values.

```
# Select the target variable Adj Close
df = data[['Adj Close']]

# Check for missing values
print("Missing values:")
print(df.isnull().sum())

# Interpolate missing values if any
df.interpolate(method='linear', inplace=True)

# Verify no missing values are left
print(df.isnull().sum())
```

```
Missing values:
Adj Close    0
dtype: int64
Adj Close    0
dtype: int64
<ipython-input-68-148a4509a71c>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.interpolate(method='linear', inplace=True)
```

Interpretation:

Data Selection: We focus on the 'Adj Close' column, which represents the adjusted closing prices, accounting for corporate actions.

Data Cleaning: Handling missing values ensures the dataset's integrity, making it suitable for further analysis and model training.

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()
```

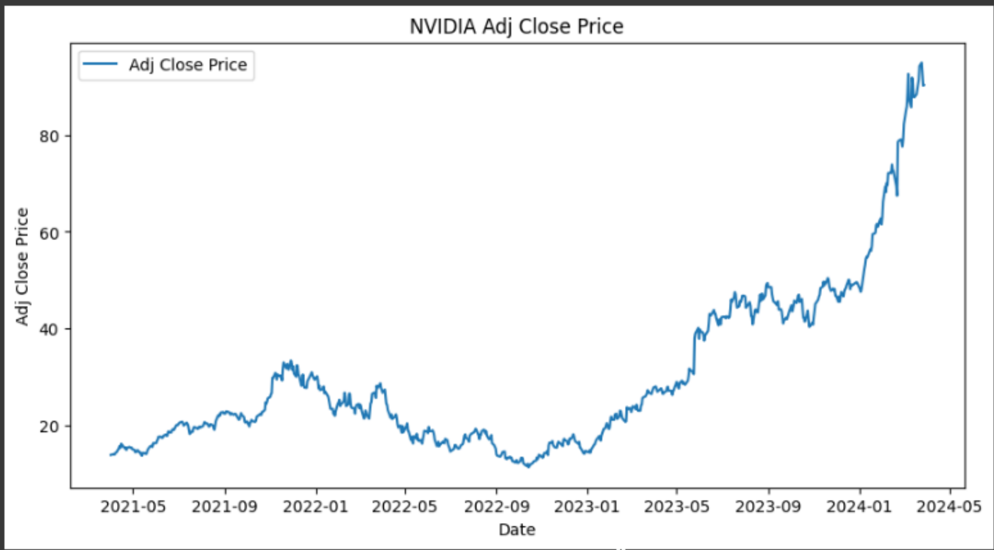
Interpretation:

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

2.1 Plot the Time Series

Plotting the time series helps visualize the stock prices over time.

```
# Plot the data
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:

Visualizing the time series gives us a clear picture of the stock price trends and any visible patterns or anomalies over the selected period.

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:

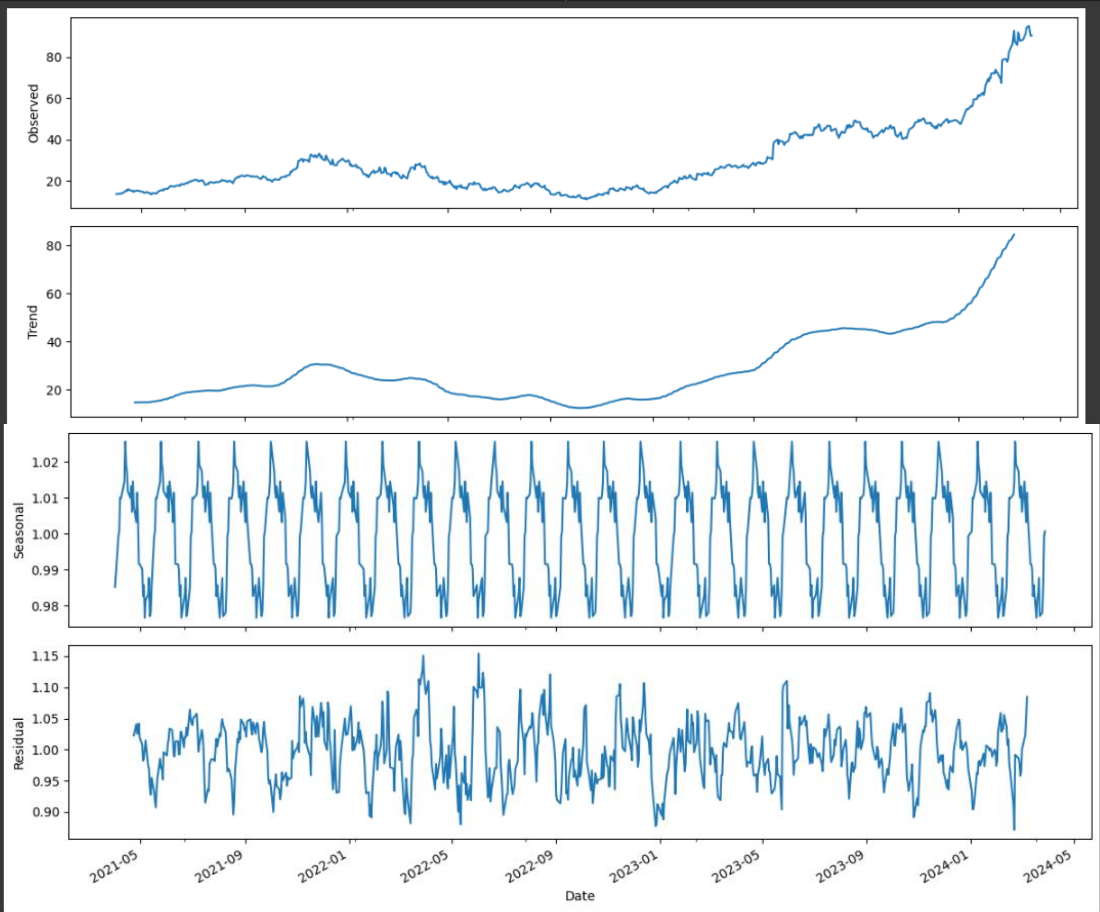
- **Decomposition**: The time series is decomposed into observed, trend, seasonal, and residual components.
    - **Observed**: The original data.
    - **Trend**: The underlying trend in the data.
    - **Seasonal**: Repeating patterns or cycles.
    - **Residual**: Noise or random variation not explained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

2.2 Decomposition of Time Series

Decomposing the time series into its components (trend, seasonal, and residual) helps us understand the underlying patterns.

```
# Decompose the time series
result = seasonal_decompose(df['Adj Close'], model='multiplicative', period=30)

# Plot the decomposed components
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:

Observed: The original time series data.

Trend: Long-term movement in the data, showing the overall direction.

Seasonal: Repeating short-term cycle, indicating regular patterns.

Residual: Random noise, representing the unexplained variability.

6. Univariate Forecasting - Conventional Models/Statistical Models

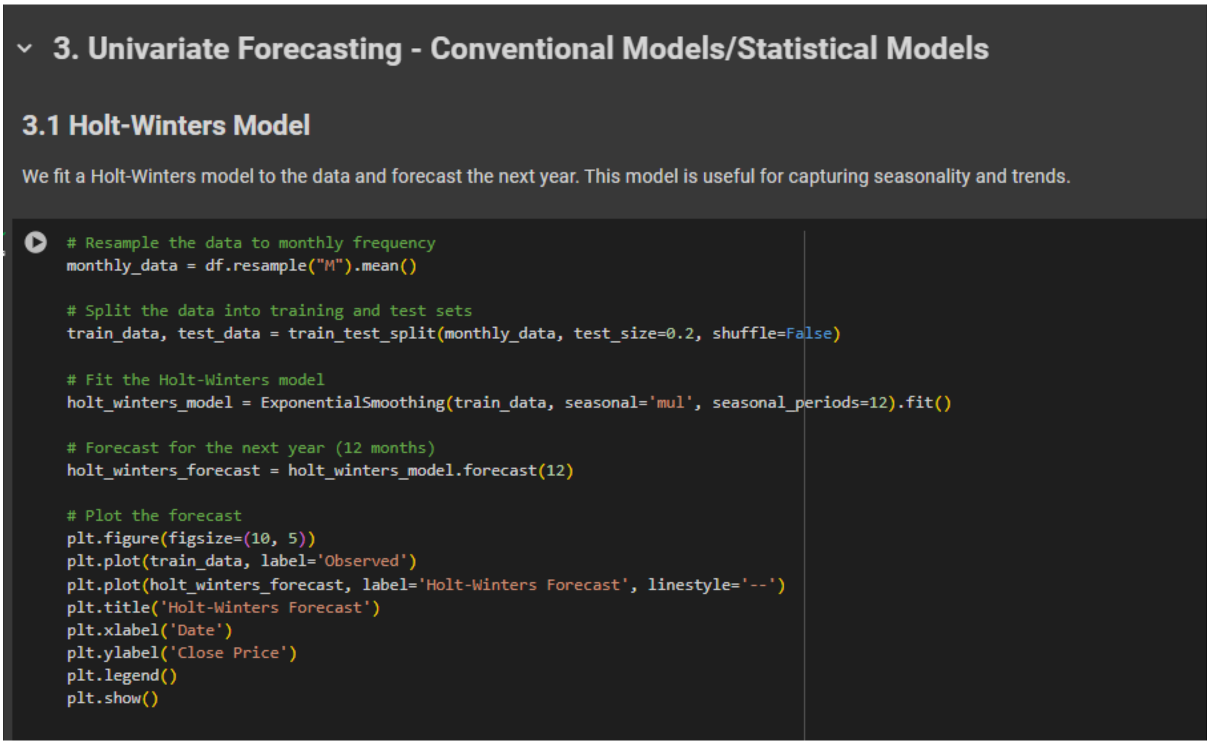
6.1 Holt-Winters Model

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.



3. Univariate Forecasting - Conventional Models/Statistical Models

3.1 Holt-Winters Model

We fit a Holt-Winters model to the data and forecast the next year. This model is useful for capturing seasonality and trends.

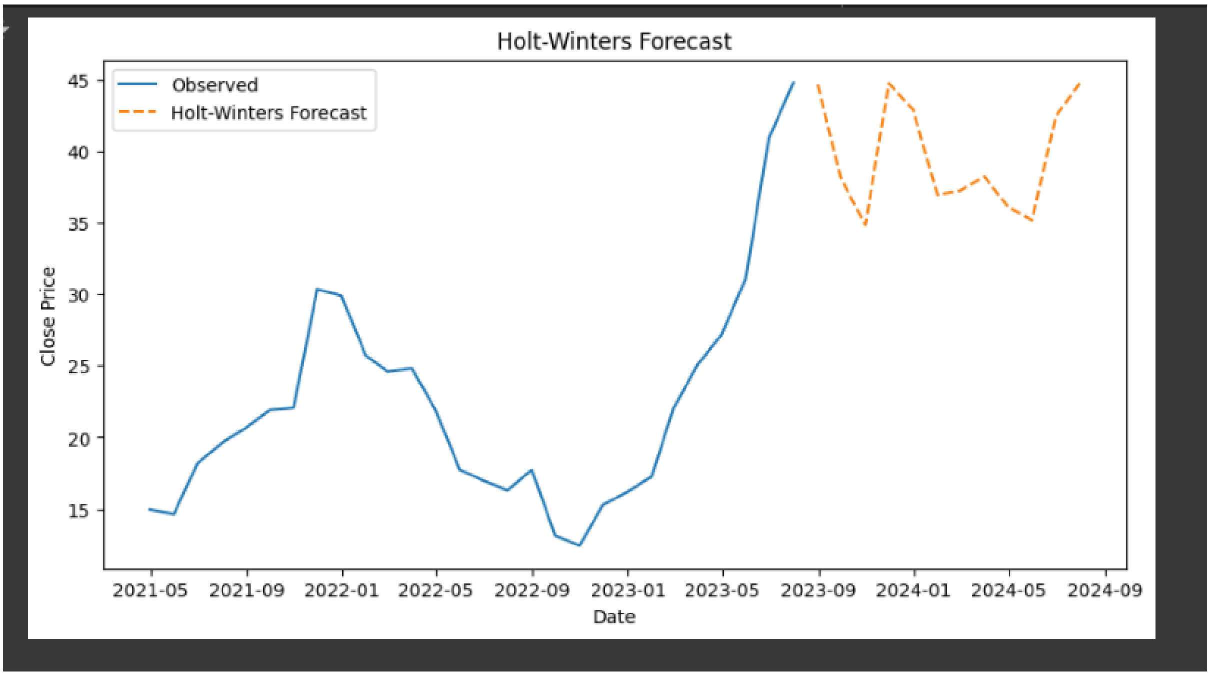
```
# Resample the data to monthly frequency
monthly_data = df.resample("M").mean()

# Split the data into training and test sets
train_data, test_data = train_test_split(monthly_data, test_size=0.2, shuffle=False)

# Fit the Holt-Winters model
holt_winters_model = ExponentialSmoothing(train_data, seasonal='mul', seasonal_periods=12).fit()

# Forecast for the next year (12 months)
holt_winters_forecast = holt_winters_model.forecast(12)

# Plot the forecast
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()
```



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.

Evaluation:

We evaluate the model's performance using metrics like RMSE, MAE, MAPE, and R-squared.

```
[72] # Compute forecast accuracy metrics
     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}')
```

```
RMSE: 23.44291126915836
MAE: 16.190689341118286
MAPE: nan
R-squared: -1.275945838167186
```

Interpretation of Metrics:

RMSE: Standard deviation of the prediction errors.

MAE: Average magnitude of the errors.

MAPE: Percentage measure of the prediction errors.

R-squared: Proportion of the 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')
```

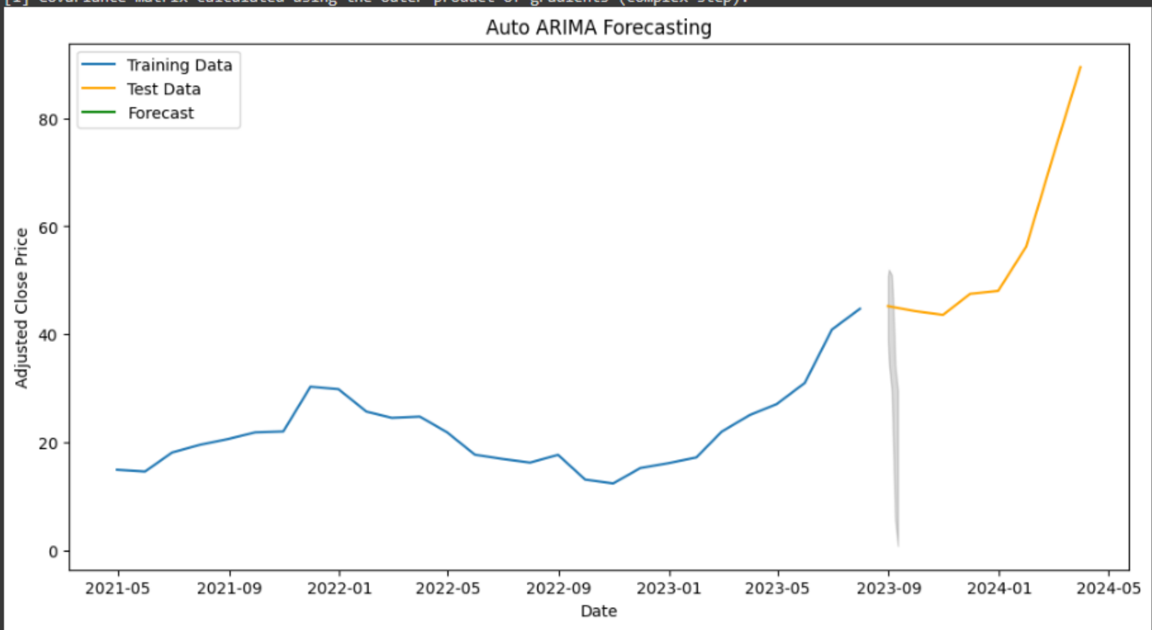
```
plt.title('Auto ARIMA Forecasting')
plt.show()
```

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.

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   28
Model:               SARIMAX(2, 0, 1)   Log Likelihood                 -71.272
Date:                Mon, 22 Jul 2024   AIC                            152.543
Time:                        10:40:05   BIC                            159.204
Sample:                    04-30-2021   HQIC                           154.579
                         - 07-31-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      1.8870      1.682      1.122      0.262      -1.410       5.184
ar.L1          1.8549      0.219      8.467      0.000       1.426       2.284
ar.L2         -0.9370      0.215     -4.351      0.000      -1.359      -0.515
ma.L1         -0.7276      0.427     -1.703      0.089      -1.565       0.110
sigma2         8.3647      2.392      3.497      0.000       3.677      13.052
===================================================================================
Ljung-Box (L1) (Q):                   0.21   Jarque-Bera (JB):                 6.70
Prob(Q):                              0.65   Prob(JB):                         0.04
Heteroskedasticity (H):               1.30   Skew:                             0.91
Prob(H) (two-sided):                  0.71   Kurtosis:                         4.56
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```



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.

```
# Print the computed metrics
print(f'RMSE: {rmse}')
print(f'MAE: {mae}')
print(f'MAPE: {mape}')
print(f'R-squared: {r2}')
```

```
RMSE: 0.23561192872557513
MAE: 0.23561192872557513
MAPE: 0.5207996933371859
R-squared: nan
```

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)
```

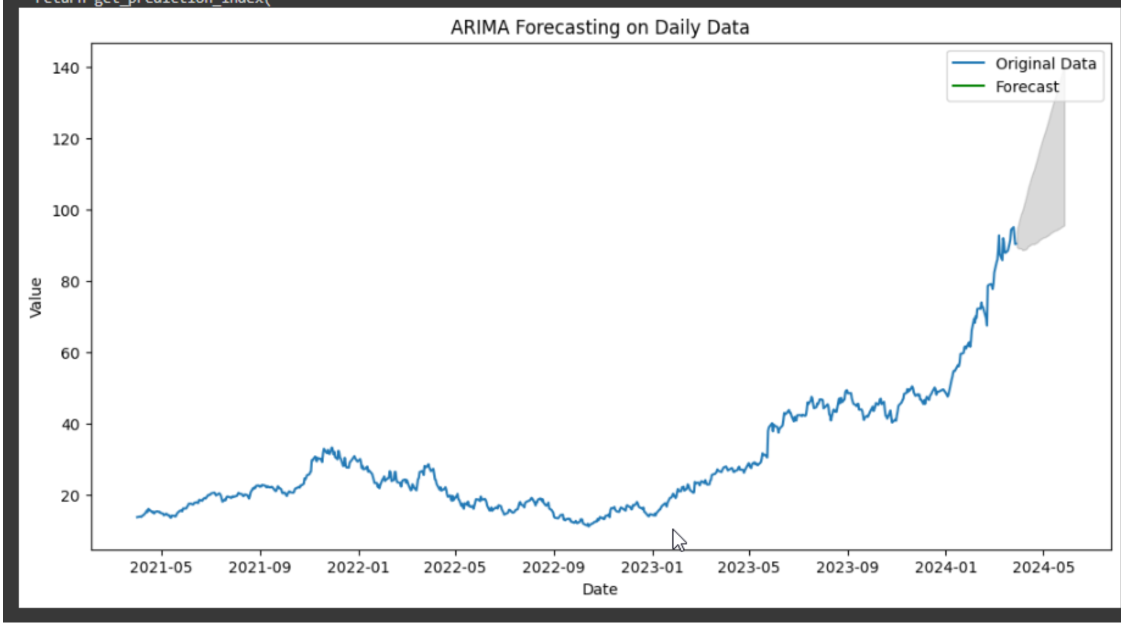
```
plt.legend()
plt.xlabel('Date')
plt.ylabel('Value')
plt.title('ARIMA Forecasting on Daily Data')
plt.show()
```

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.

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                   y   No. Observations:                 753
Model:             SARIMAX(2, 2, 1)x(1, 0, 1, 7)   Log Likelihood               -1096.104
Date:                             Mon, 22 Jul 2024   AIC                           2204.208
Time:                                     10:41:08   BIC                           2231.936
Sample:                                          0   HQIC                          2214.892
                                             - 753
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.0148      0.029     -0.511      0.609      -0.072       0.042
ar.L2         -0.1376      0.025     -5.539      0.000      -0.186      -0.089
ma.L1         -0.9820      0.005   -214.983      0.000      -0.991      -0.973
ar.S.L7       -0.7660      0.110     -6.967      0.000      -0.981      -0.550
ma.S.L7        0.8437      0.098      8.597      0.000       0.651       1.036
sigma2         1.0789      0.026     41.066      0.000       1.027       1.130
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):              9692.08
Prob(Q):                              0.93   Prob(JB):                         0.00
Heteroskedasticity (H):               3.62   Skew:                             1.77
Prob(H) (two-sided):                  0.00   Kurtosis:                        20.24
===================================================================================

  return get_prediction_index(
```



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

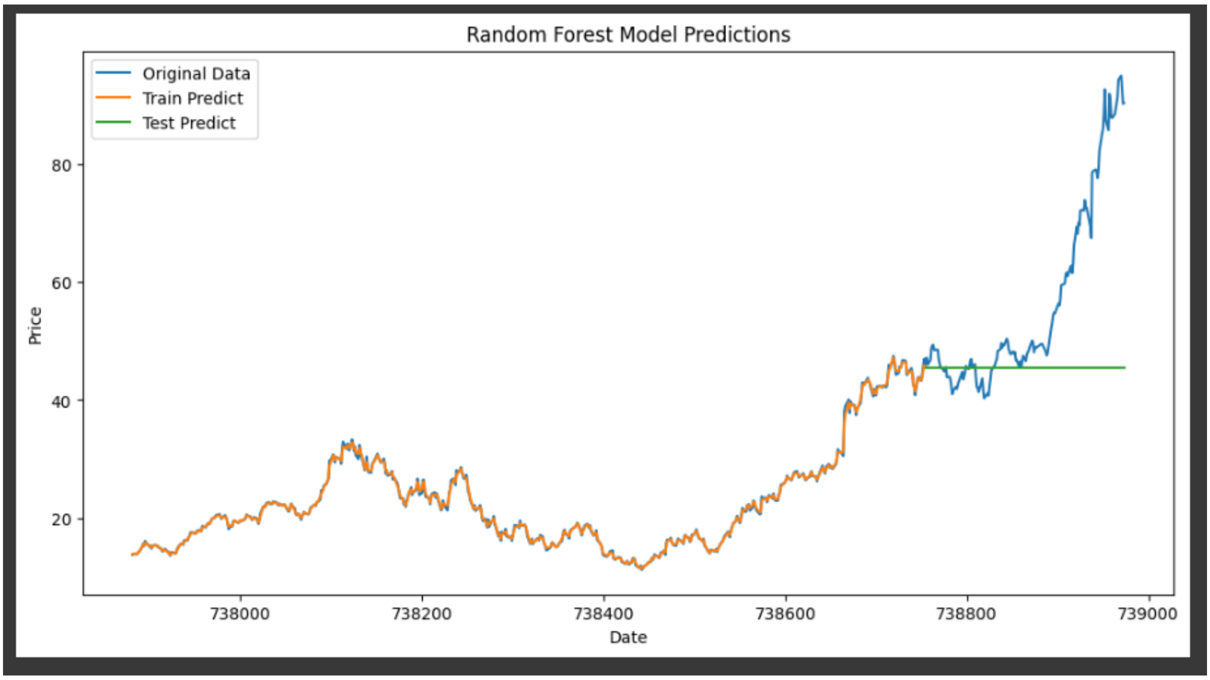
Code:

```
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

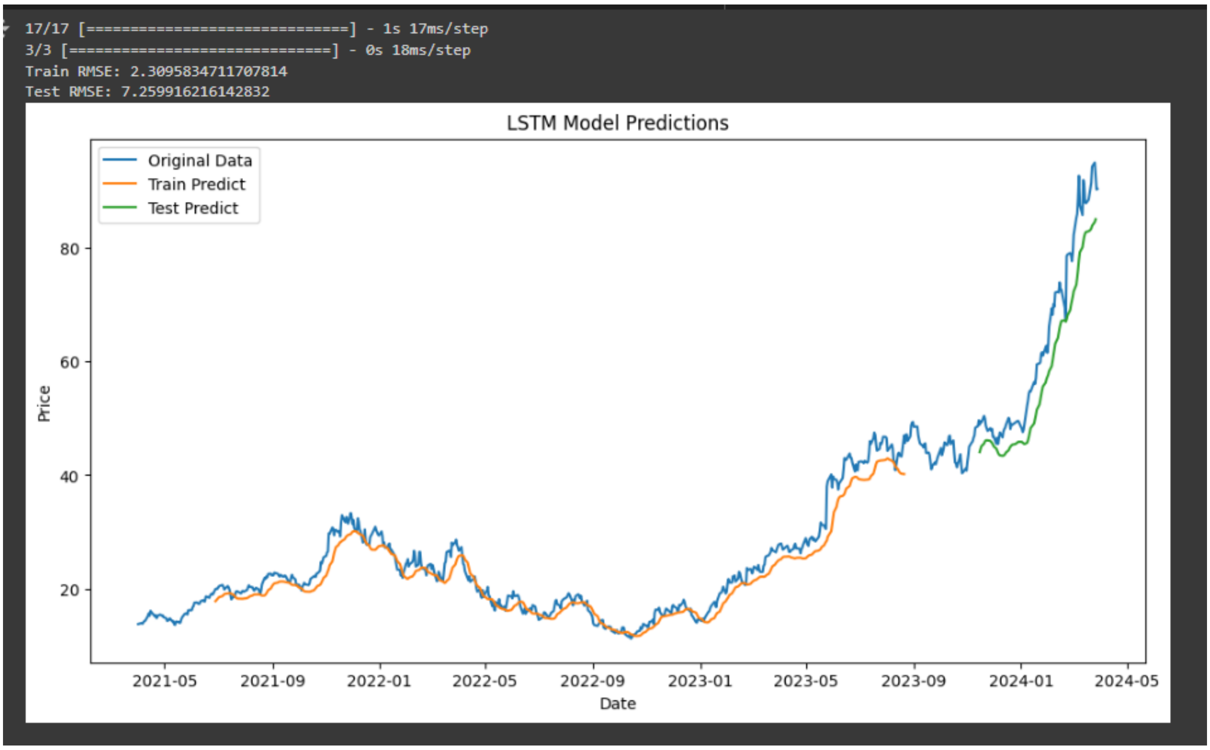
Code:

```
# 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.

```
package ‘tidyverse’ successfully unpacked and MD5 sums checked
package ‘quantmod’ successfully unpacked and MD5 sums checked
package ‘forecast’ successfully unpacked and MD5 sums checked
package ‘fable’ successfully unpacked and MD5 sums checked
package ‘fabletools’ successfully unpacked and MD5 sums checked
package ‘tseries’ successfully unpacked and MD5 sums checked
package ‘keras’ successfully unpacked and MD5 sums checked
package ‘caret’ successfully unpacked and MD5 sums checked
package ‘randomForest’ successfully unpacked and MD5 sums checked
package ‘neuralnet’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\ADHYAYAN\AppData\Local\Temp\Rtmpst13xK\downloaded_packages
```

2. Loading Libraries

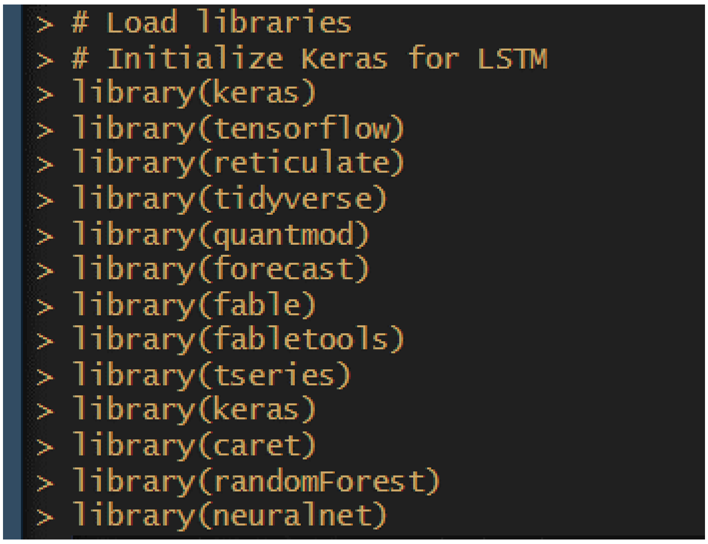
Code:

```
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`).



```
> # 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)
```

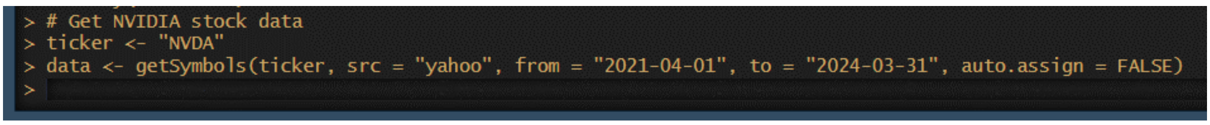
3. Fetching NVIDIA Stock Data

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.



```
> # Get NVIDIA stock data
> ticker <- "NVDA"
> data <- getSymbols(ticker, src = "yahoo", from = "2021-04-01", to = "2024-03-31", auto.assign = FALSE)
>
```

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.

```
> # 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)
> df
          Date Adjusted
1   2021-04-01 13.78367
2   2021-04-05 13.95906
3   2021-04-06 13.83332
4   2021-04-07 14.11475
5   2021-04-08 14.28789
6   2021-04-09 14.37072
7   2021-04-12 15.17808
8   2021-04-13 15.64762
9   2021-04-14 15.24594
10  2021-04-15 16.10444
11  2021-04-16 15.88015
12  2021-04-19 15.33052
13  2021-04-20 15.14040
14  2021-04-21 15.32927
15  2021-04-22 14.82006
```

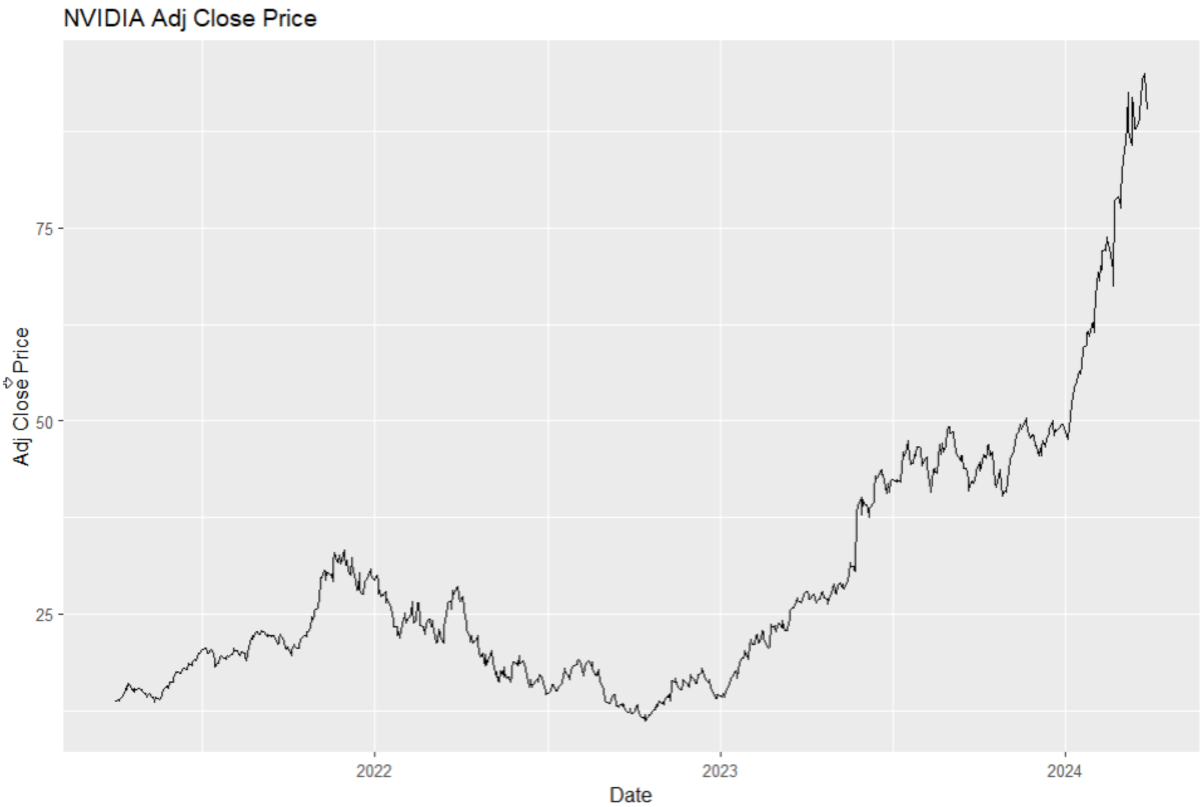
5. Visualizing the Time Series

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.



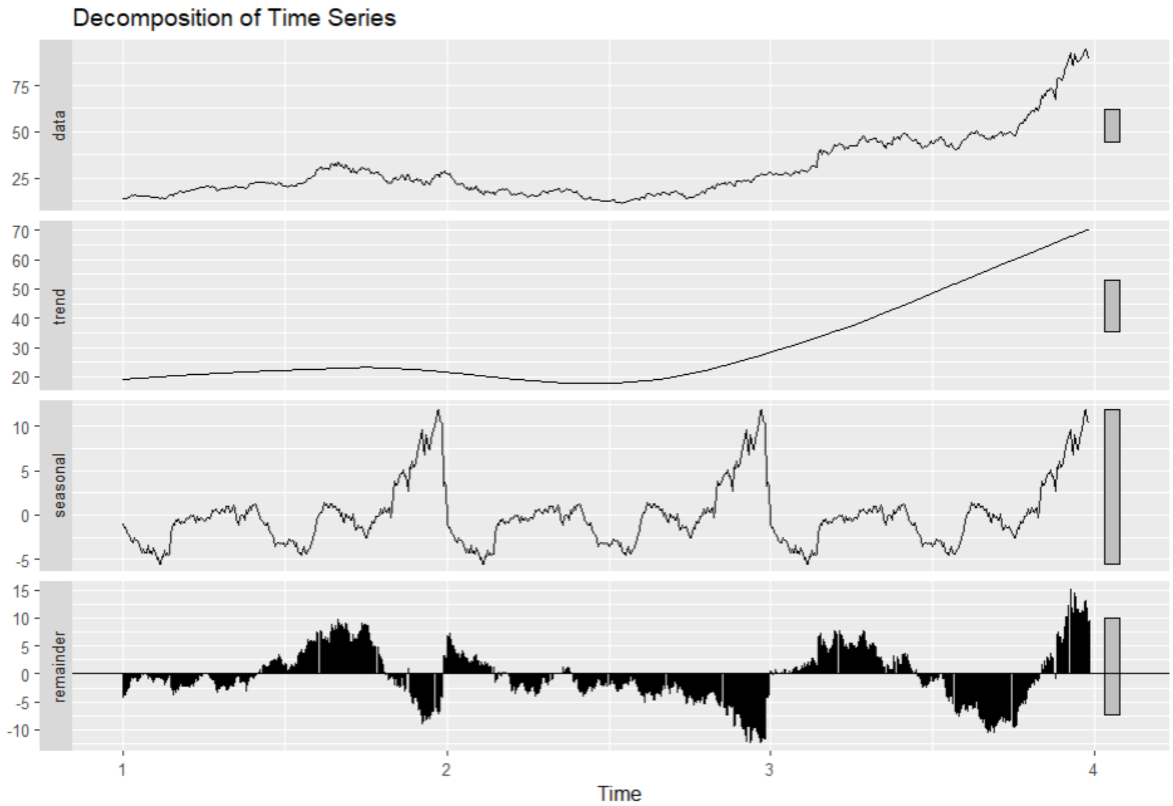
6. Time Series Decomposition

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.

```
> # Aggregate to monthly data
> monthly_data <- df %>%
+   mutate(YearMonth = floor_date(Date, "month")) %>%
+   group_by(YearMonth) %>%
+   summarize(Adjusted = mean(Adjusted)) %>%
+   as.data.frame()
> monthly_data
   YearMonth Adjusted
1  2021-04-01 14.97978
2  2021-05-01 14.65834
3  2021-06-01 18.18473
4  2021-07-01 19.61089
5  2021-08-01 20.67499
6  2021-09-01 21.90085
7  2021-10-01 22.07442
8  2021-11-01 30.33904
```

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.

```
> # 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),]
> train_size
[1] 28
> train_data
    YearMonth Adjusted
1  2021-04-01 14.97978
2  2021-05-01 14.65834
3  2021-06-01 18.18473
4  2021-07-01 19.61089
5  2021-08-01 20.67499
6  2021-09-01 21.90085
7  2021-10-01 22.07442
8  2021-11-01 30.33904
9  2021-12-01 29.88565
10 2022-01-01 25.73144
11 2022-02-01 24.59185
12 2022-03-01 24.81828
13 2022-04-01 21.93136
14 2022-05-01 17.76680
15 2022-06-01 16.99581
16 2022-07-01 16.31203
17 2022-08-01 17.74228
18 2022-09-01 13.16294
19 2022-10-01 12.45948
20 2022-11-01 15.30110
21 2022-12-01 16.20315
22 2023-01-01 17.26341
23 2023-02-01 22.03001
24 2023-03-01 25.09663
25 2023-04-01 27.13838
26 2023-05-01 31.04422
27 2023-06-01 40.90884
28 2023-07-01 44.74920
> test_data
    YearMonth Adjusted
29 2023-08-01 45.24041
30 2023-09-01 44.31428
31 2023-10-01 43.62571
32 2023-11-01 47.52116
33 2023-12-01 48.06389
34 2024-01-01 56.25193
35 2024-02-01 72.54109
36 2024-03-01 89.43481
>
```

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.

```
> # Use Holt-Winters method on monthly data
> holt_winters_model <- HoltWinters(ts(train_data$Adjusted, frequency = 12), seasonal = "multiplicativ
e")
> # Forecasting
> forecast_holt_winters <- forecast(holt_winters_model, h = nrow(test_data))
> # Plot forecast
> autoplot(forecast_holt_winters) +
+   autolayer(test_data$Adjusted, series = "Test Data") +
+   labs(title = "Holt-Winters Forecast", x = "Date", y = "Adjusted Close Price")
```

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.

```
> # Check and handle missing values
> missing_values <- sum(is.na(df))
> print(paste("Missing values:", missing_values))
[1] "Missing values: 0"
> df <- na.omit(df)  # Remove rows with NA values
> missing_values <- sum(is.na(df))
> print(paste("Missing values after interpolation:", missing_values))
[1] "Missing values after interpolation: 0"
```

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.

```
> # ARIMA Model
> arima_model <- auto.arima(train_data$Adjusted)
> summary(arima_model)
Series: train_data$Adjusted
ARIMA(2,0,0) with non-zero mean

Coefficients:
         ar1      ar2     mean
      1.3857  -0.4833  24.6166
s.e.  0.1661   0.1873   5.6307

sigma^2 = 10.77:  log likelihood = -72.71
AIC=153.43   AICc=155.17   BIC=158.76

Training set error measures:
                    ME     RMSE      MAE        MPE     MAPE     MASE         ACF1
Training set 0.2242582 3.100792 2.184223 -0.8812284 9.994621 0.848733 0.0002765124
>
```

12. Random Forest Model

Code:

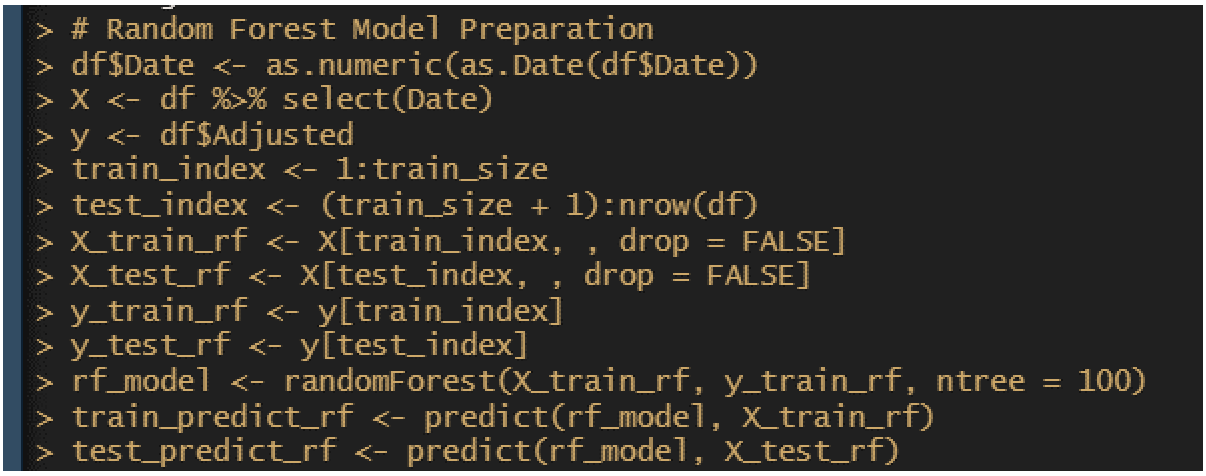
```
df$Date <- as.numeric(as.Date(df$Date))
X <- df %>% select(Date)
y <- df$Adjusted
train_index <- 1:train_size
test_index <- (train_size + 1):nrow(df)

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.



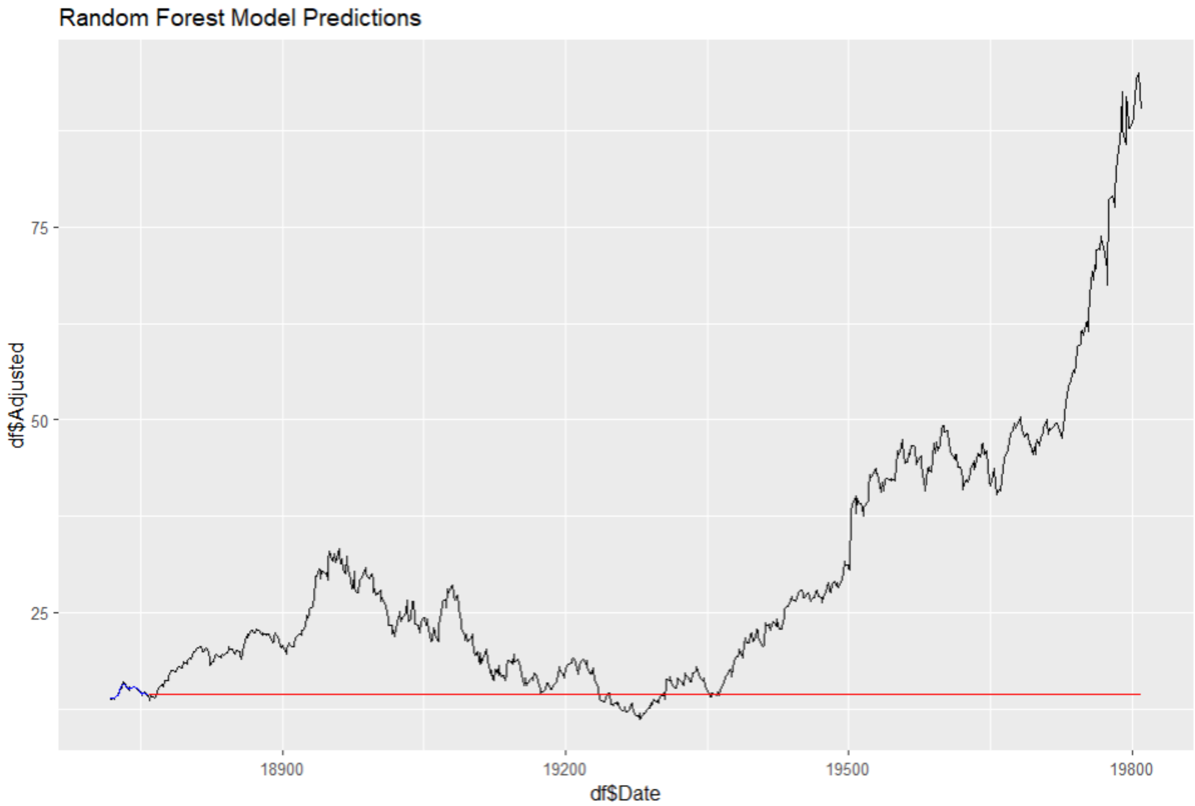
```
> # Random Forest Model Preparation
> df$Date <- as.numeric(as.Date(df$Date))
> X <- df %>% select(Date)
> y <- df$Adjusted
> train_index <- 1:train_size
> test_index <- (train_size + 1):nrow(df)
> X_train_rf <- X[train_index, , drop = FALSE]
> X_test_rf <- X[test_index, , drop = FALSE]
> y_train_rf <- 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)
```

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.

```
> # 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
Warning message:
In cor(y_test_rf, test_predict_rf) : the standard deviation is zero
> cat('Train RMSE (RF):', train_rmse_rf, '\n')
Train RMSE (RF): 0.1607833
> cat('Test RMSE (RF):', test_rmse_rf, '\n')
Test RMSE (RF): 23.36234
> cat('Train MAE (RF):', train_mae_rf, '\n')
Train MAE (RF): 0.1256449
> cat('Test MAE (RF):', test_mae_rf, '\n')
Test MAE (RF): 16.04886
> cat('Train R-squared (RF):', train_r2_rf, '\n')
Train R-squared (RF): 0.9326218
> cat('Test R-squared (RF):', test_r2_rf, '\n')
Test R-squared (RF): NA
```

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.

```
> # 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))
```

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.

```
> # 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')
ANN RMSE: 58.35261
> cat('ANN MAE:', ann_mae, '\n')
ANN MAE: 56.16643
> cat('ANN R-squared:', ann_r2, '\n')
ANN R-squared: 0.9960807
>
```

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
```

```r
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.