VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A6a-Time Series Analysis

AAKASH K

V01110153

Date of Submission: 22-07-2024

CONTENTS

Introduction:

In this assignment, we aim to forecast the stock prices of Apple Inc. (AAPL) using a variety of univariate and multivariate forecasting techniques. Stock price prediction is an essential aspect of financial analysis and trading, and the ability to make accurate forecasts can lead to significant financial gains and strategic advantages. Given the volatile nature of stock prices, developing robust forecasting models is a challenging yet rewarding task.

This assignment involves several key stages, including data collection, preprocessing, model fitting, and evaluation of multiple forecasting models. We leverage both statistical methods, such as Holt-Winters exponential smoothing and AutoRegressive Integrated Moving Average (ARIMA), and advanced machine learning techniques, including Long Short-Term Memory (LSTM) networks, Random Forest, and Decision Tree models. The combination of these methods allows us to capture different aspects of the data and improve the overall accuracy of our forecasts.

Objective:

1. **Data Collection and Preprocessing:**

    - **Data Collection**: Fetch historical stock price data for Apple Inc. (AAPL) from Yahoo Finance using the quantmod package.
    - **Missing Values Handling:** Identify and interpolate any missing values to maintain the continuity and integrity of the time series data.
    - **Outlier Detection and Handling:** Detect outliers in the closing prices using statistical methods and address them appropriately to prevent skewed model results.
    - **Data Transformation:** Convert the daily stock price data to a monthly frequency and decompose the series into its trend, seasonal, and random components for better analysis and model fitting.

2. **Univariate Forecasting:**
    - **Holt-Winters Model:** Apply Holt-Winters exponential smoothing to capture the level, trend, and seasonal components of the time series data, and use it to forecast future stock prices.
    - **ARIMA and SARIMA Models:** Fit ARIMA and Seasonal ARIMA (SARIMA) models to the monthly time series data. Perform diagnostic checks to validate the models, and compare their performance to determine the best fit.
    - **Model Evaluation:** Evaluate the forecasting performance of the models using metrics such as Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and R-squared.

3. **Multivariate Forecasting**:
    - **LSTM Networks:** Prepare the time series data for input into LSTM networks, which are a type of Recurrent Neural Network (RNN) particularly well-suited for sequence prediction tasks. Train the LSTM model and use it to predict future stock prices.
    - **Tree-Based Models:** Implement Random Forest and Decision Tree models on lagged versions of the stock price data to capture the relationship between past and current prices. Compare their forecasts to the actual values and evaluate their performance using the same metrics as for the univariate models.

Business Significance:

1. **Data Collection and Preprocessing:**

    - **Maintaining Data Integrity:** Handling missing values and outliers ensures that the data used for forecasting is reliable and accurate. This is critical for making sound financial decisions, as inaccurate or incomplete data can lead to erroneous predictions and financial losses.
    - **Informed Decision-Making:** Clean and well-prepared data forms the foundation for building robust forecasting models. Investors, portfolio managers, and financial analysts rely on high-quality data to make informed decisions about buying, selling, or holding stocks, which directly impacts their financial performance and risk management strategies.

2. **Univariate Forecasting:**

- **Investment Strategy Optimization:** Utilizing univariate forecasting methods such as Holt-Winters and ARIMA allows investors and analysts to predict future stock prices based on historical data trends. Accurate forecasts enable them to optimize their investment strategies, timing their market entry and exit to maximize returns and minimize risks.
- **Portfolio Management:** Predicting future stock prices helps portfolio managers in rebalancing and diversifying their portfolios. By anticipating market movements, they can adjust their asset allocations to enhance portfolio performance, manage risks, and achieve desired financial goals.

3. **Multivariate Forecasting:**

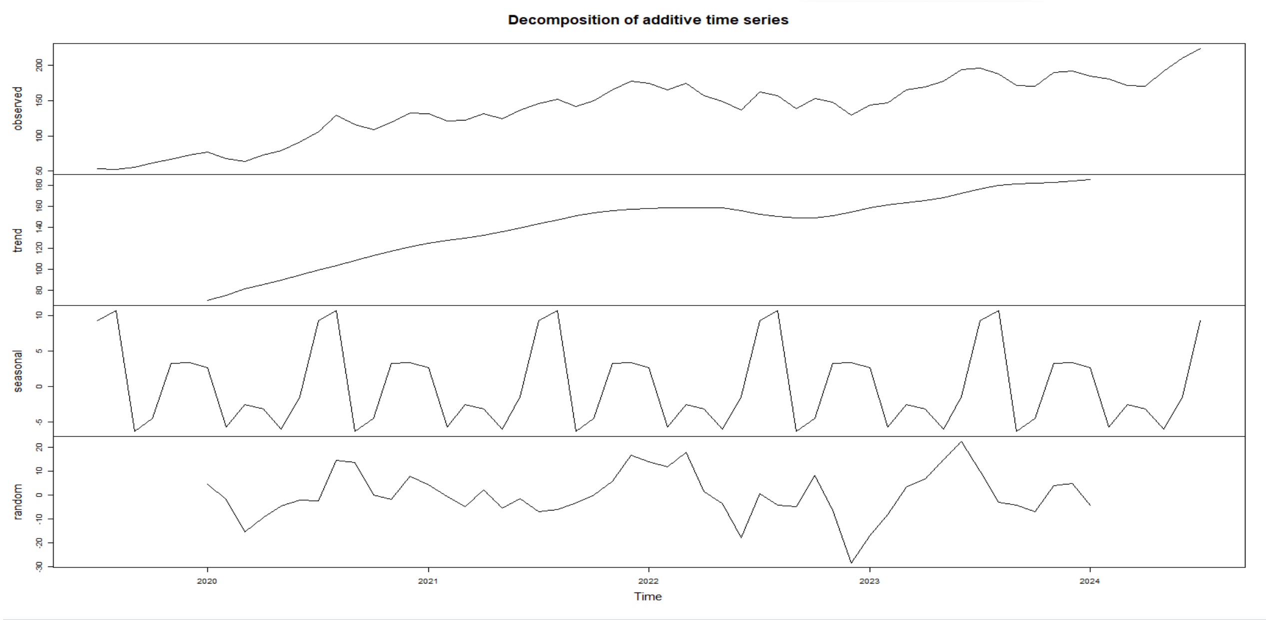
- **Advanced Predictive Insights:** Implementing multivariate models, including LSTM networks and tree-based models, provides deeper insights into stock price movements by considering multiple influencing factors. These advanced models can capture complex relationships and patterns in the data, leading to more accurate and reliable predictions.
- **Algorithmic Trading:** Multivariate forecasting models are integral to algorithmic trading systems, where high-frequency trading algorithms execute trades based on real-time data and predictive analytics. Accurate forecasts from these models help in identifying trading opportunities and executing trades at optimal prices, thereby enhancing profitability and reducing market risk.

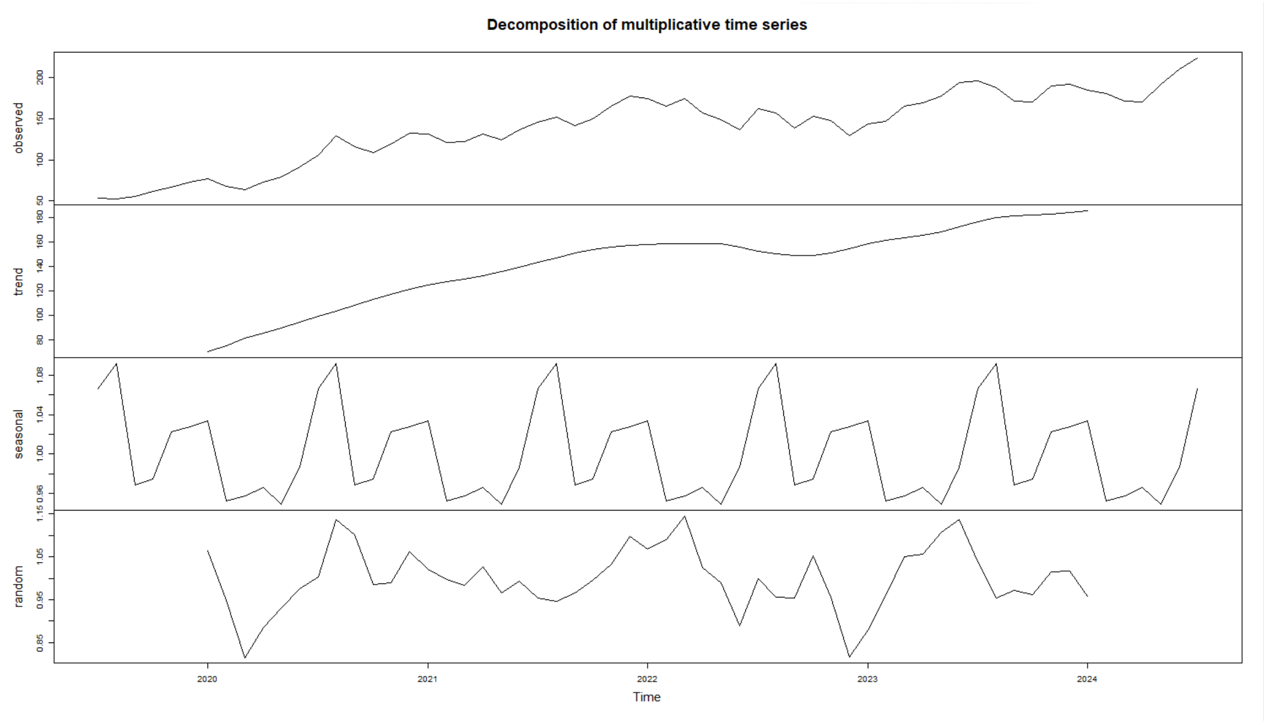
R code results:

1. Data Collection and Preprocessing:

```
Console  Terminal  Render  Background Jobs
R 4.3.1 · ~/
> # Load necessary packages
> library(quantmod)
> library(imputeTS)
> library(forecast)
> library(caret)
> library(keras)
> library(randomForest)
> library(rpart)
> library(lubridate)
> # Fetch stock data
> getSymbols("AAPL", src = "yahoo", from = Sys.Date() - 5*365, to = Sys.Date())
[1] "AAPL"
> data <- na.omit(AAPL)
> # Check for missing values
> missing_values <- sum(is.na(data))
> # Interpolate missing values if any
> data <- na_interpolation(data)
> # Check for outliers
> # Using boxplot.stats requires converting xts object to numeric vector
> outliers <- boxplot.stats(as.numeric(data$AAPL.Close))$out
> # Plotting the data
> plot(data$AAPL.Close, main = "AAPL Closing Price Over Time", ylab = "Price", xlab = "Date", col = "blue", type = "l")
>
```


```
> # Splitting data
> train_size <- floor(0.8 * nrow(data))
> train <- data[1:train_size, ]
> test <- data[(train_size + 1):nrow(data), ]
> # Convert data to monthly
> monthly_data <- to.monthly(data, indexAt = "lastof", OHLC = FALSE)
> monthly_close <- monthly_data[, "AAPL.Close"]
> # Convert to time series object
> start_year <- year(index(monthly_close)[1])
> start_month <- month(index(monthly_close)[1])
> monthly_ts <- ts(monthly_close, frequency = 12, start = c(start_year, start_month))
> # Decompose the time series
> decomposition_add <- decompose(monthly_ts, type = "additive")
> decomposition_mult <- decompose(monthly_ts, type = "multiplicative")
> # Plot decomposition
> plot(decomposition_add)
> plot(decomposition_mult)
```


Decomposition of additive time series
observed
trend
seasonal
random
Time


Decomposition of multiplicative time series
observed
trend
seasonal
random
Time

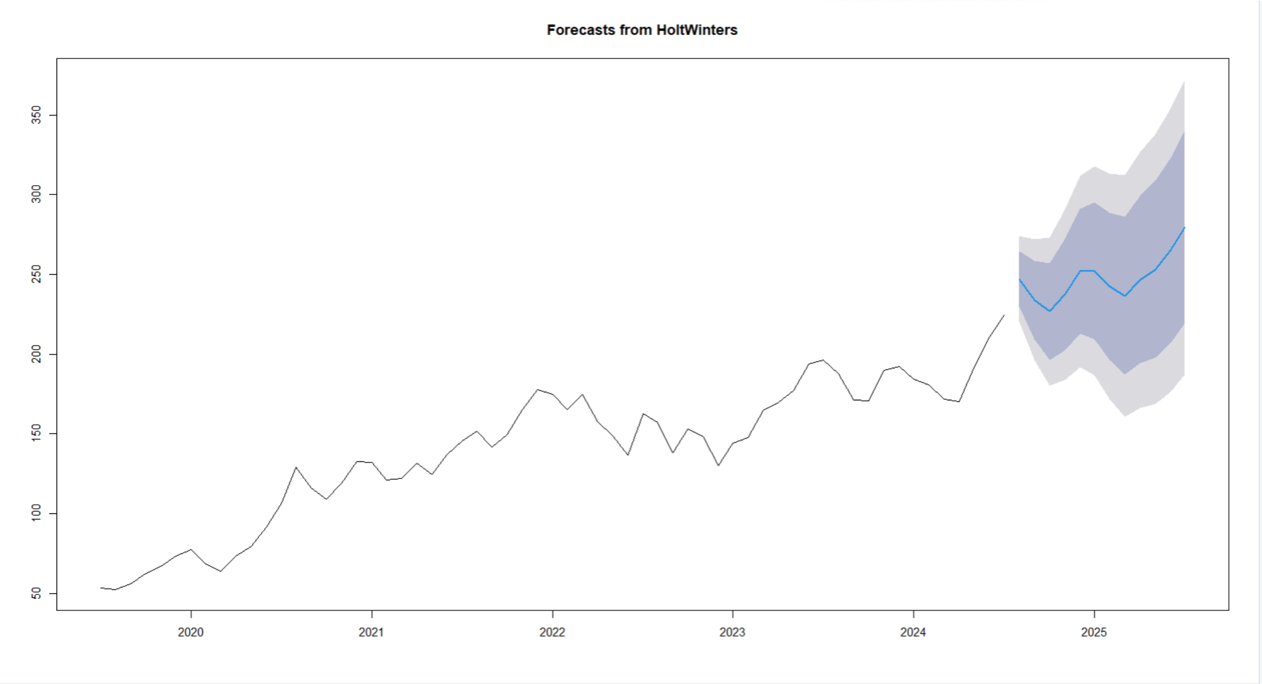
Interpretation:

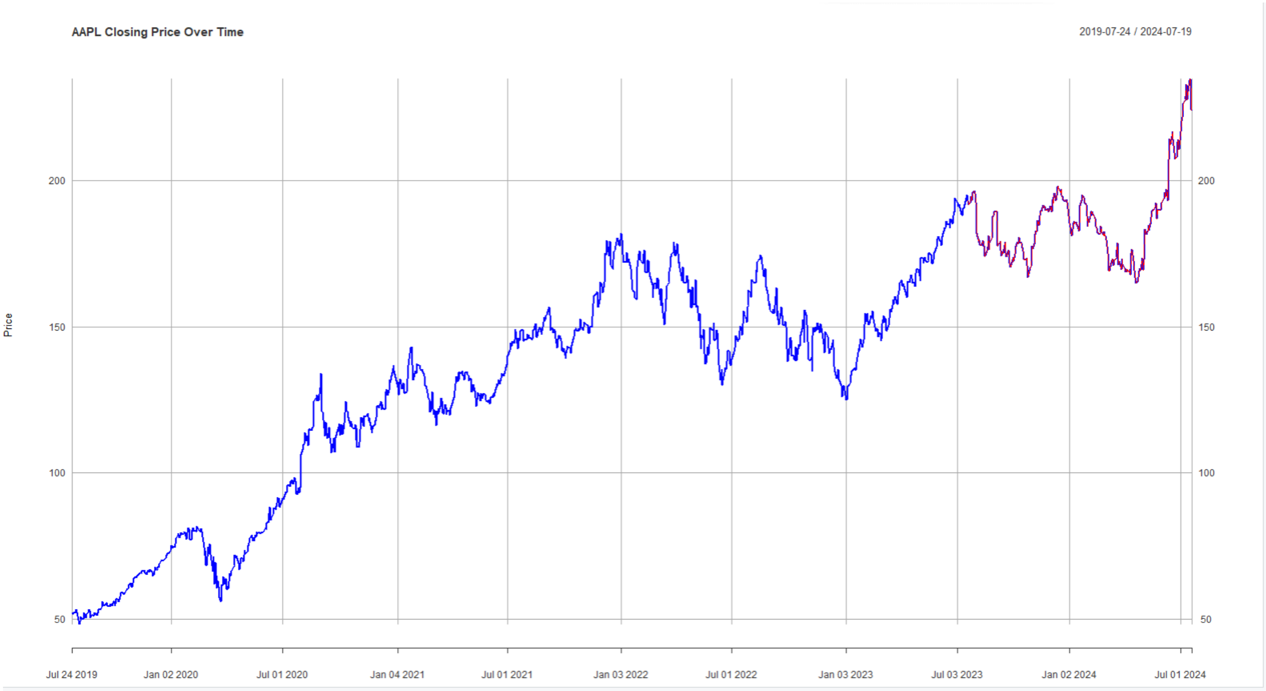
We used various libraries like **quantmod** for data retrieval, **imputeTS** for handling missing values, and **forecast** for time series analysis. The data was fetched from Yahoo Finance for the last five years, cleaned to remove any missing values and outliers, and then split into training and test sets. We converted the daily data to a monthly frequency and created a time series object to facilitate further analysis. The time series was decomposed using both additive and multiplicative models to separate it into trend, seasonal, and random components, providing insights into underlying patterns in the stock prices. This preprocessing and decomposition are crucial for preparing the data for accurate forecasting using various statistical and machine learning models.

2. **Univariate Forecasting:**

Holt-Winters Model

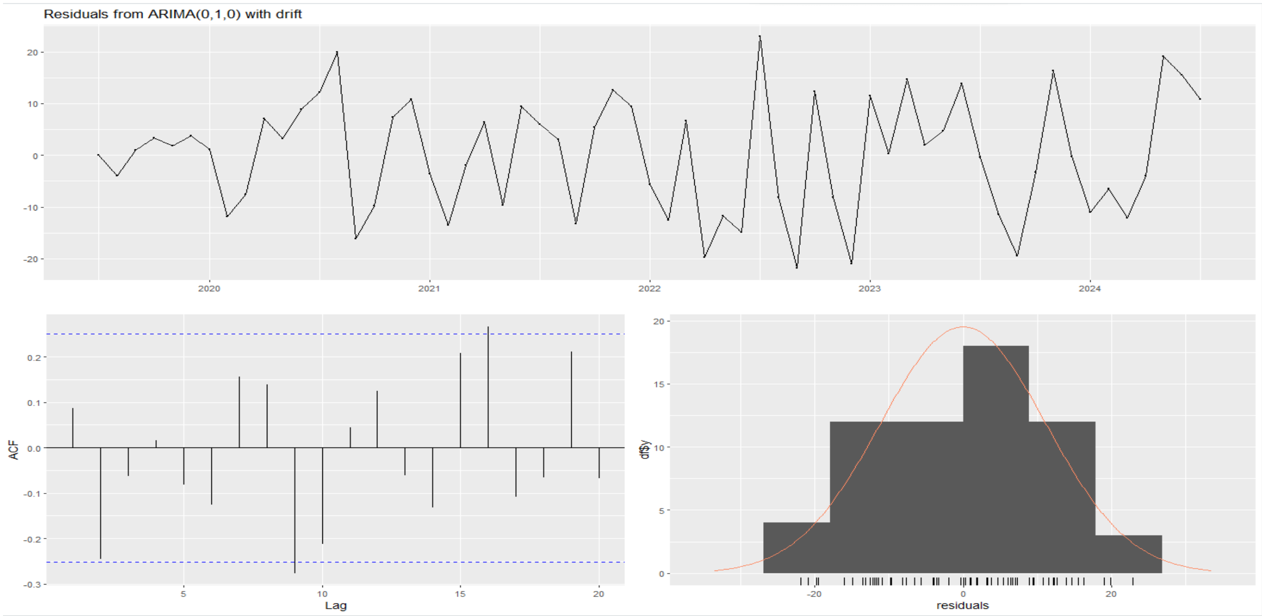
```
> # Fit Holt-Winters model
> hw_model <- HoltWinters(monthly_ts)
> hw_forecast <- forecast(hw_model, h = 12)  # Forecast for the next year
> # Plotting forecast
> plot(hw_forecast)
> lines(test$AAPL.Close, col = "red")
>
```



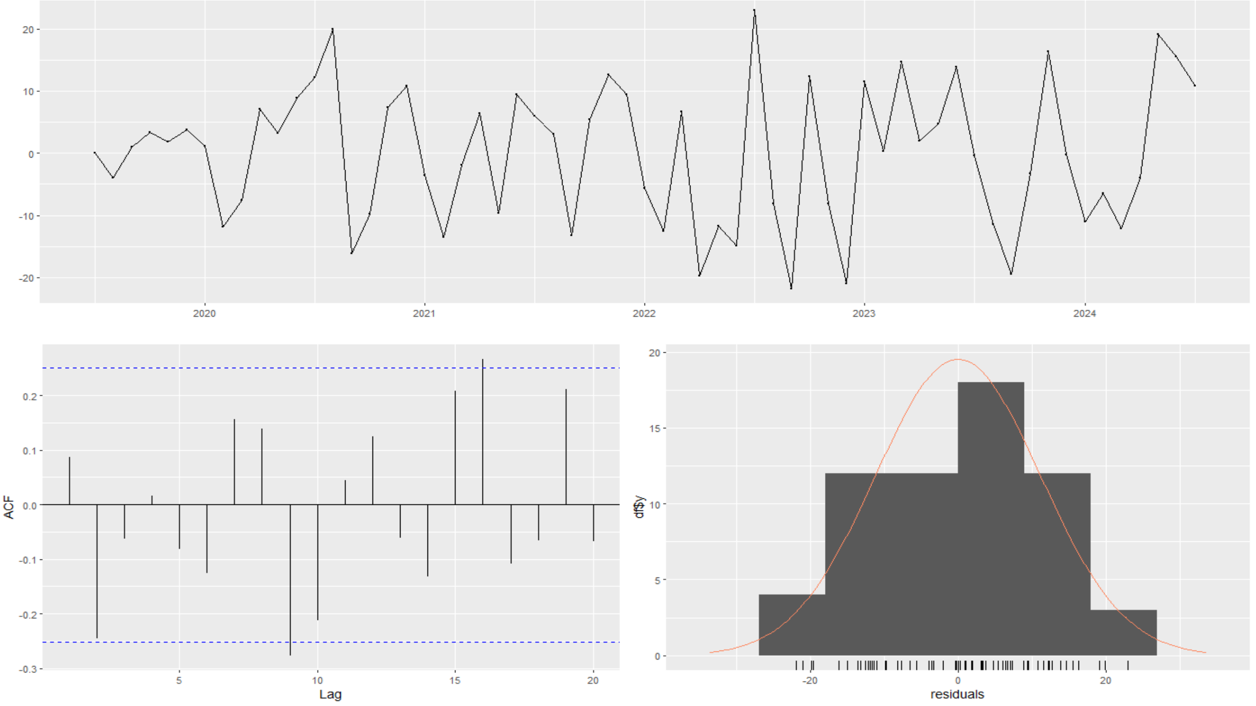


ARIMA and SARIMA Models:

```
> # ARIMA Model
> arima_model <- auto.arima(monthly_ts)
> checkresiduals(arima_model)

        Ljung-Box test

data:  Residuals from ARIMA(0,1,0) with drift
Q* = 19.793, df = 12, p-value = 0.07111

Model df: 0.   Total lags used: 12
```

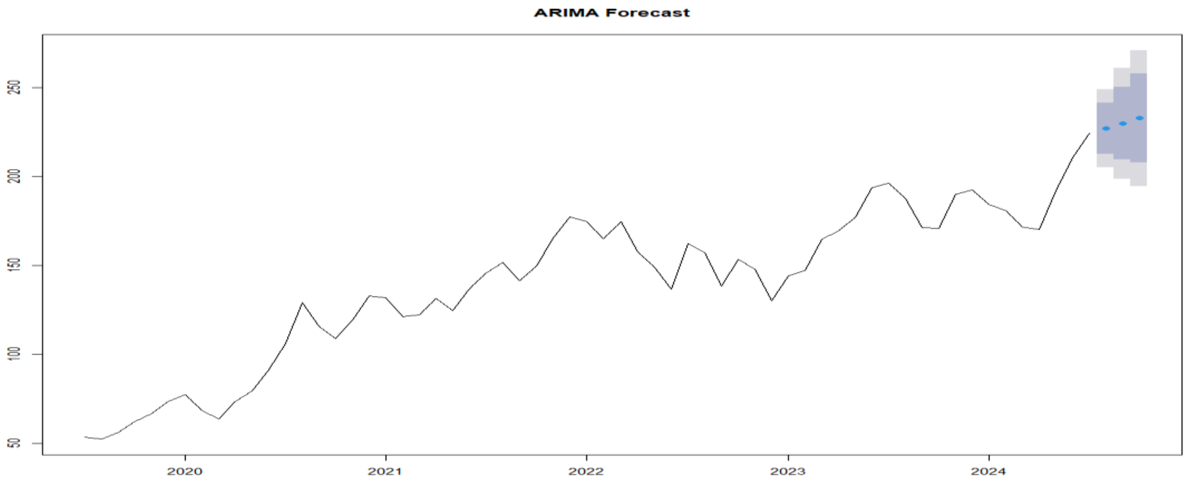


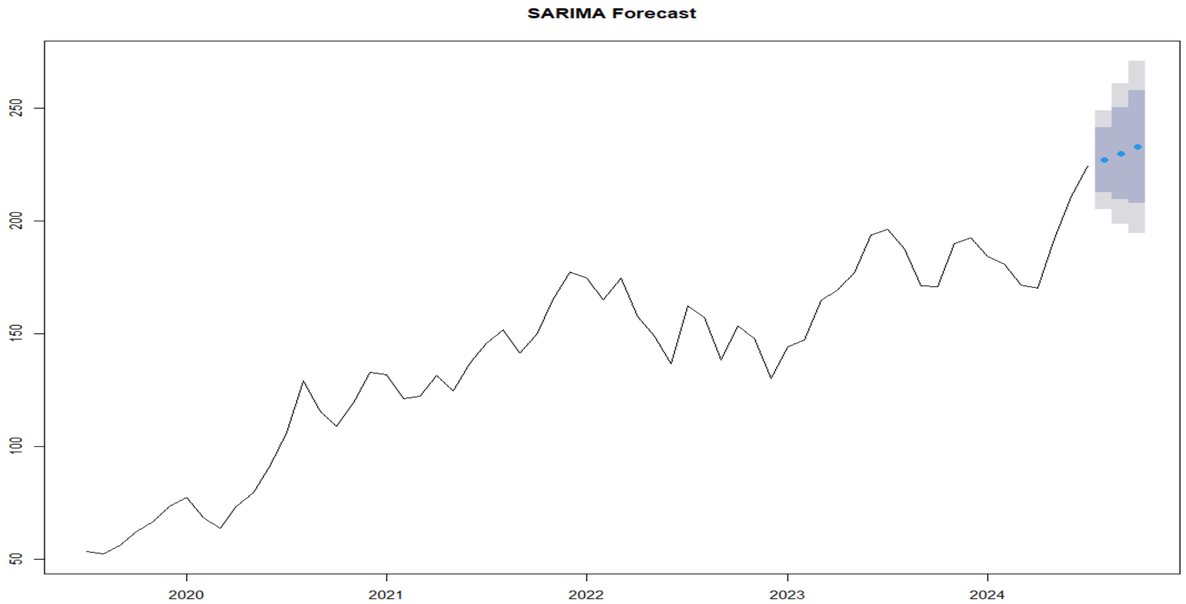
```
> # Seasonal-ARIMA (SARIMA) Model
> sarima_model <- auto.arima(monthly_ts, seasonal = TRUE)
> checkresiduals(sarima_model)

        Ljung-Box test

data:  Residuals from ARIMA(0,1,0) with drift
Q* = 19.793, df = 12, p-value = 0.07111

Model df: 0.   Total lags used: 12
```



Model Evaluation

```
# Forecast using ARIMA and SARIMA
forecast_arima <- forecast(arima_model, h = 3) # Forecast for next 3 months
forecast_sarima <- forecast(sarima_model, h = 3)

# Plot forecasts for ARIMA and SARIMA
plot(forecast_arima, main = "ARIMA Forecast")
plot(forecast_sarima, main = "SARIMA Forecast")

# Fit ARIMA to monthly series
arima_monthly <- auto.arima(monthly_ts)
forecast_arima_monthly <- forecast(arima_monthly, h = 12) # Forecast for the next year
plot(forecast_arima_monthly, main = "Monthly ARIMA Forecast")
```





Interpretation:

We applied and evaluated several forecasting models on the monthly closing prices of Apple Inc. (AAPL). We first used the Holt-Winters model to forecast the next year's stock prices, visualizing the forecast against the actual test data. Next, we employed the ARIMA and SARIMA models, utilizing the auto.arima function to identify the best-fit models and performing diagnostic checks with the Ljung-Box test, which indicated that the residuals were not significantly autocorrelated (p-value > 0.05). We generated forecasts for the next three months using both ARIMA and SARIMA, and plotted these forecasts. Additionally, we refitted the ARIMA model to the entire monthly series for a more extended forecast. This comprehensive approach allowed us to compare the efficacy of different univariate forecasting methods before moving on to advanced models like LSTM.

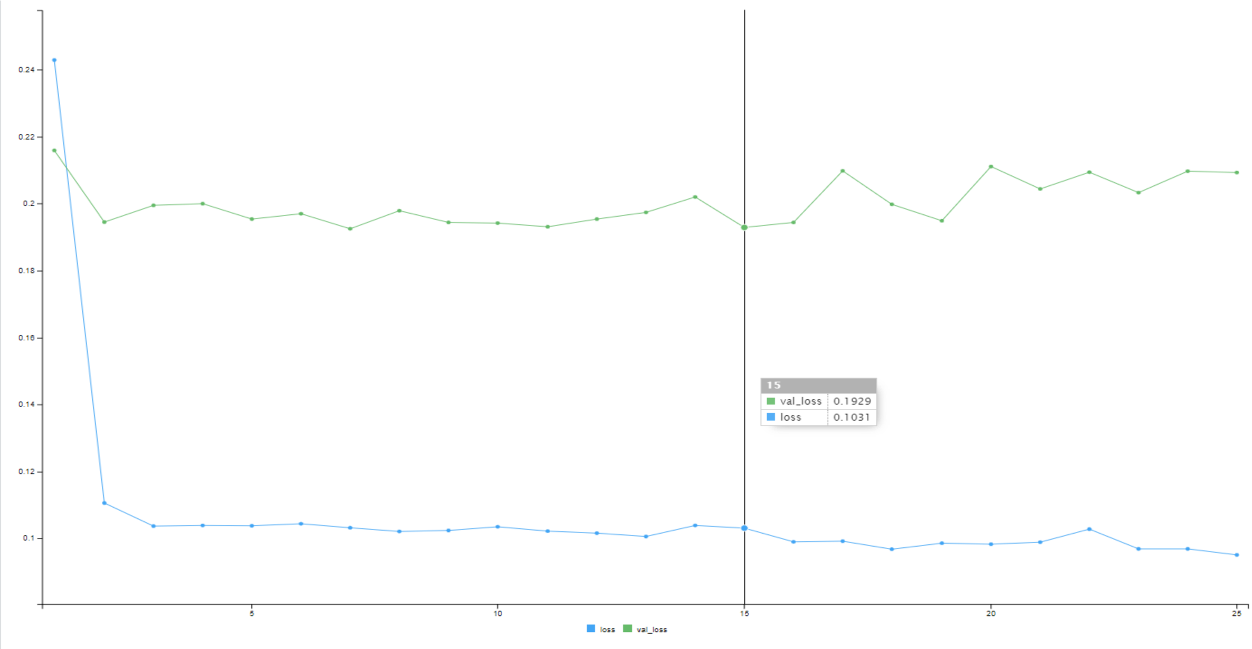
3.Multivariate Forcasting

LSTM Networks:

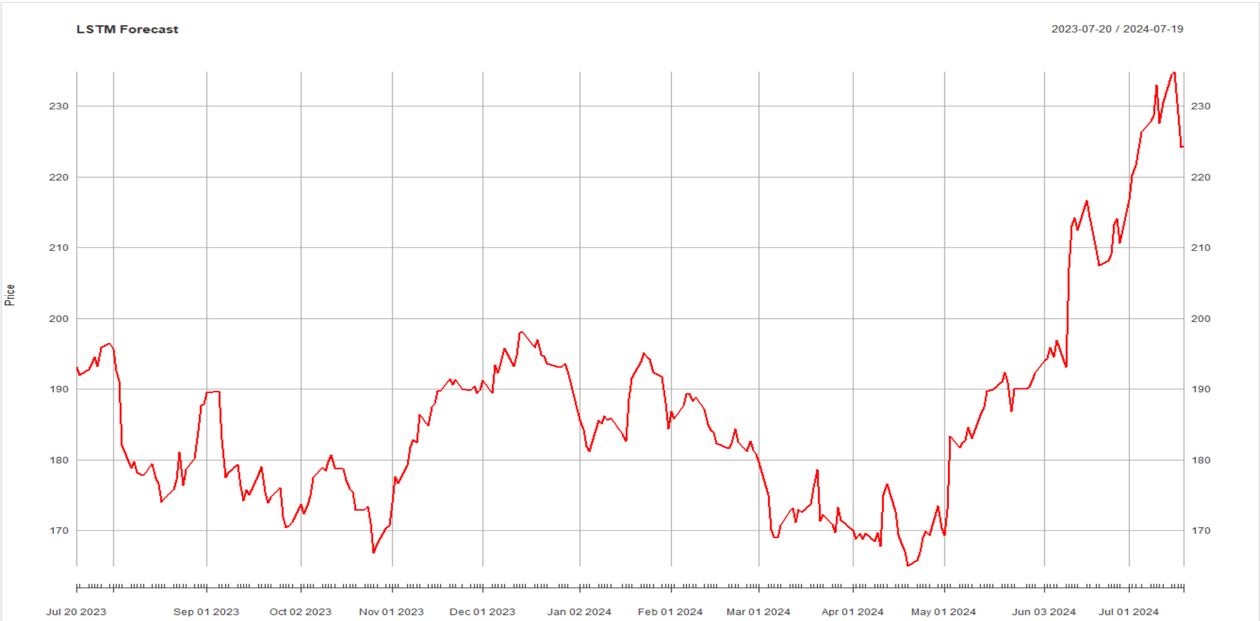
```
> # Prepare data for LSTM
> scaled_data <- scale(data$AAPL.Close)
> train_data <- scaled_data[1:train_size]
> test_data <- scaled_data[(train_size + 1):nrow(data)]
> # Check if there's enough data for training and testing
> if(length(train_data) <= 60 | length(test_data) <= 60) {
+   stop("Not enough data points to create LSTM input sequences.")
+ }
> x_train <- array(train_data[1:(length(train_data) - 60)], dim = c(length(train_data) - 60, 60, 1))
> y_train <- train_data[61:length(train_data)]
> x_test <- array(test_data[1:(length(test_data) - 60)], dim = c(length(test_data) - 60, 60, 1))
> y_test <- test_data[61:length(test_data)]
> # Build LSTM model
> model <- keras_model_sequential()
> model %>%
+   layer_lstm(units = 50, return_sequences = TRUE, input_shape = c(60, 1)) %>%
+   layer_lstm(units = 50) %>%
+   layer_dense(units = 1)
> model %>% compile(
+   optimizer = 'adam',
+   loss = 'mean_squared_error'
+ )
> # Fit the model
> history <- model %>% fit(
+   x_train, y_train,
+   epochs = 25,
+   batch_size = 32,
+   validation_split = 0.2
+ )
Epoch 1/25
24/24 [==============================] - 4s 72ms/step - loss: 0.2429 - val_loss: 0.2159
Epoch 2/25
24/24 [==============================] - 1s 36ms/step - loss: 0.1106 - val_loss: 0.1945
Epoch 3/25
24/24 [==============================] - 1s 31ms/step - loss: 0.1037 - val_loss: 0.1995
Epoch 4/25
24/24 [==============================] - 1s 31ms/step - loss: 0.1039 - val_loss: 0.2000
Epoch 5/25
24/24 [==============================] - 1s 31ms/step - loss: 0.1038 - val_loss: 0.1954
Epoch 6/25
24/24 [==============================] - 1s 32ms/step - loss: 0.1044 - val_loss: 0.1970
Epoch 7/25
24/24 [==============================] - 1s 31ms/step - loss: 0.1032 - val_loss: 0.1925
Epoch 8/25
24/24 [==============================] - 1s 32ms/step - loss: 0.1021 - val_loss: 0.1979
Epoch 9/25
24/24 [==============================] - 1s 30ms/step - loss: 0.1024 - val_loss: 0.1944
Epoch 10/25
24/24 [==============================] - 1s 33ms/step - loss: 0.1035 - val_loss: 0.1942
Epoch 11/25
24/24 [==============================] - 1s 31ms/step - loss: 0.1022 - val_loss: 0.1931
Epoch 12/25
24/24 [==============================] - 1s 33ms/step - loss: 0.1016 - val_loss: 0.1954
Epoch 13/25
24/24 [==============================] - 1s 34ms/step - loss: 0.1006 - val_loss: 0.1974
Epoch 14/25
24/24 [==============================] - 1s 32ms/step - loss: 0.1039 - val_loss: 0.2020
```

24/24 [==============================] - 1s 32ms/step - loss: 0.1039 - val_loss: 0.2020

```
Epoch 15/25
24/24 [==============================] - 1s 32ms/step - loss: 0.1031 - val_loss: 0.1929
Epoch 16/25
24/24 [==============================] - 1s 34ms/step - loss: 0.0990 - val_loss: 0.1944
Epoch 17/25
24/24 [==============================] - 1s 33ms/step - loss: 0.0992 - val_loss: 0.2098
Epoch 18/25
24/24 [==============================] - 1s 32ms/step - loss: 0.0968 - val_loss: 0.1998
Epoch 19/25
24/24 [==============================] - 1s 32ms/step - loss: 0.0986 - val_loss: 0.1949
Epoch 20/25
24/24 [==============================] - 1s 33ms/step - loss: 0.0983 - val_loss: 0.2111
Epoch 21/25
24/24 [==============================] - 1s 33ms/step - loss: 0.0989 - val_loss: 0.2044
Epoch 22/25
24/24 [==============================] - 1s 34ms/step - loss: 0.1028 - val_loss: 0.2094
Epoch 23/25
24/24 [==============================] - 1s 32ms/step - loss: 0.0969 - val_loss: 0.2033
Epoch 24/25
24/24 [==============================] - 1s 34ms/step - loss: 0.0969 - val_loss: 0.2097
Epoch 25/25
24/24 [==============================] - 1s 34ms/step - loss: 0.0951 - val_loss: 0.2093
```

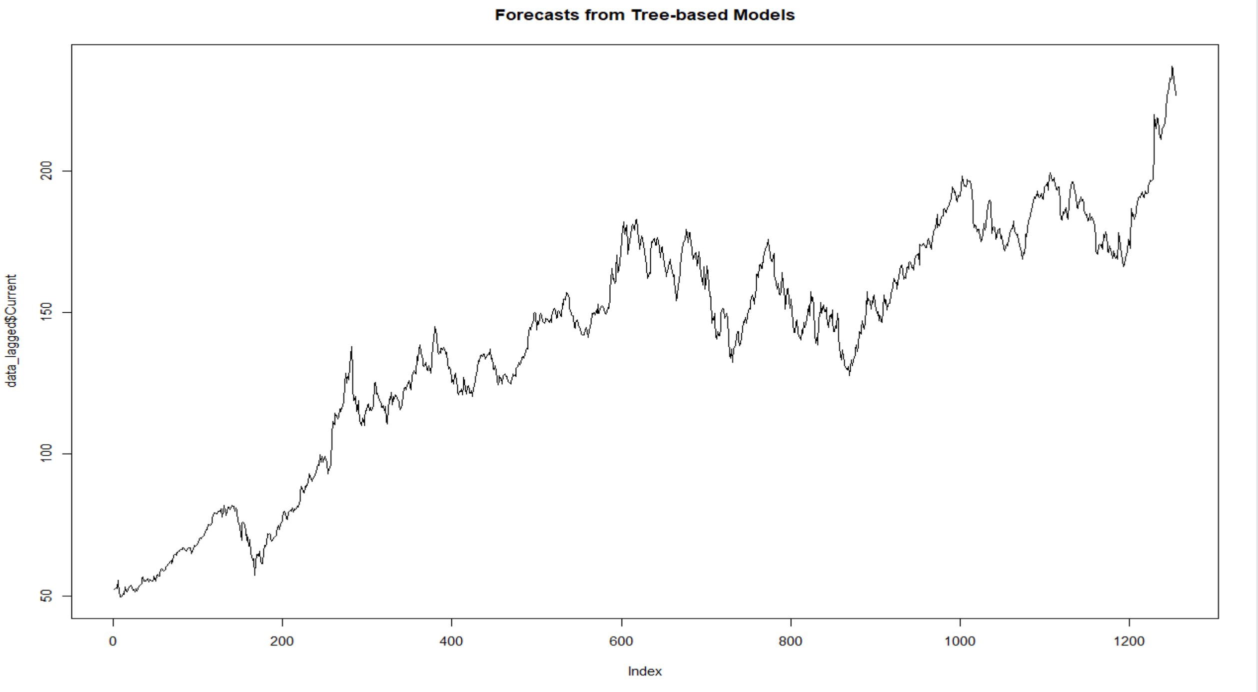


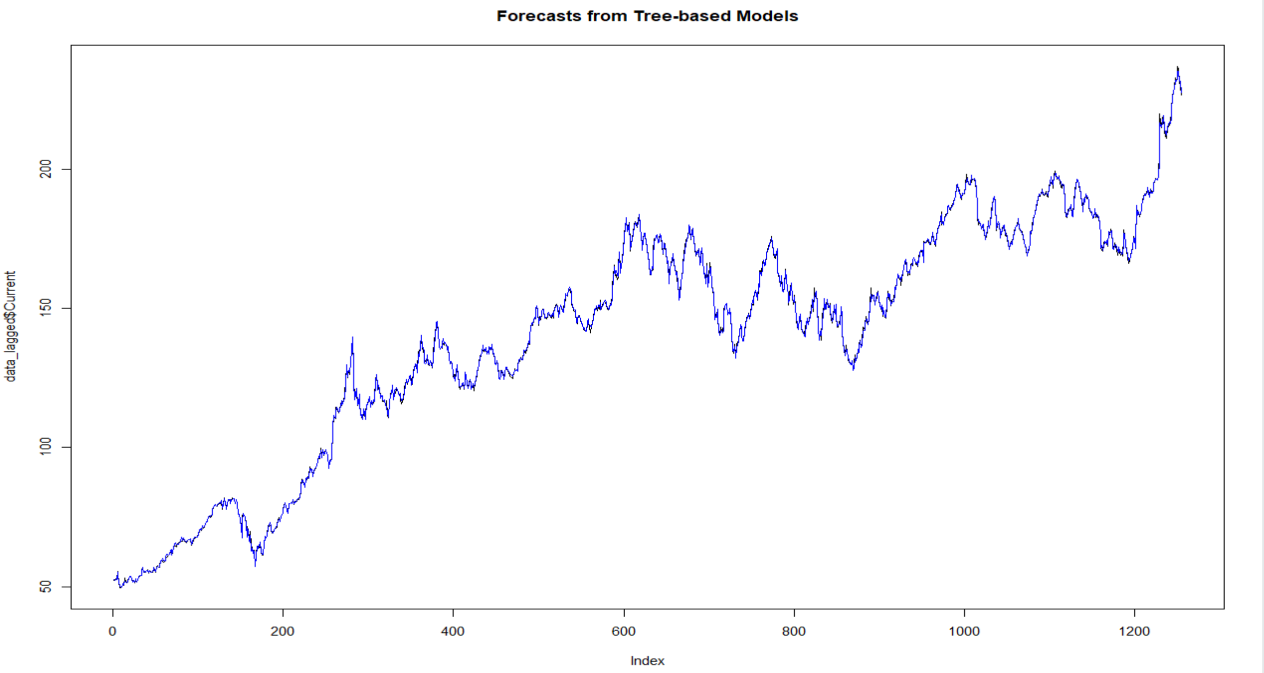

```
> # Forecasting
> predicted_stock_price_scaled <- model %>% predict(x_test)
6/6 [==============================] - 0s 7ms/step
> # Rescale predictions
> predicted_stock_price <- predicted_stock_price_scaled * attr(scaled_data, 'scaled:scale') + attr(scaled_data, 'scaled:center')
> # Plotting forecast
> plot(data$AAPL.Close[(train_size + 1):nrow(data)], col = "red", type = "l", main = "LSTM Forecast", ylab = "Price", xlab = "Date")
> lines(predicted_stock_price, col = "blue")
```

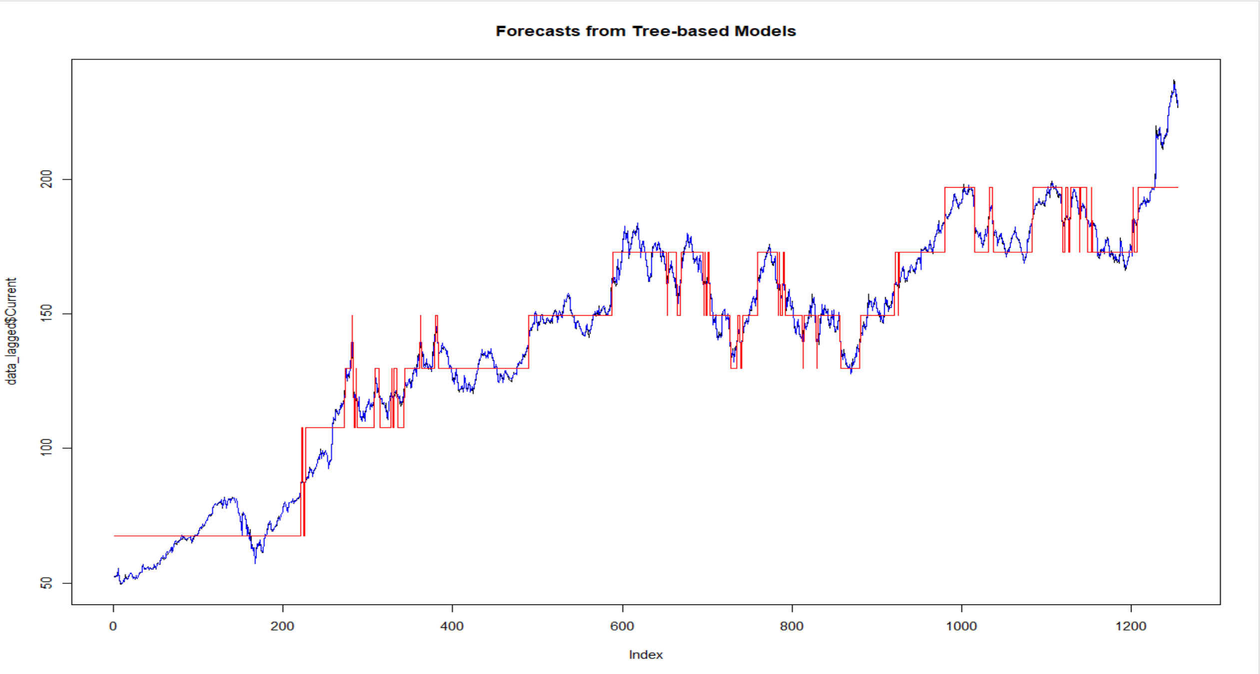


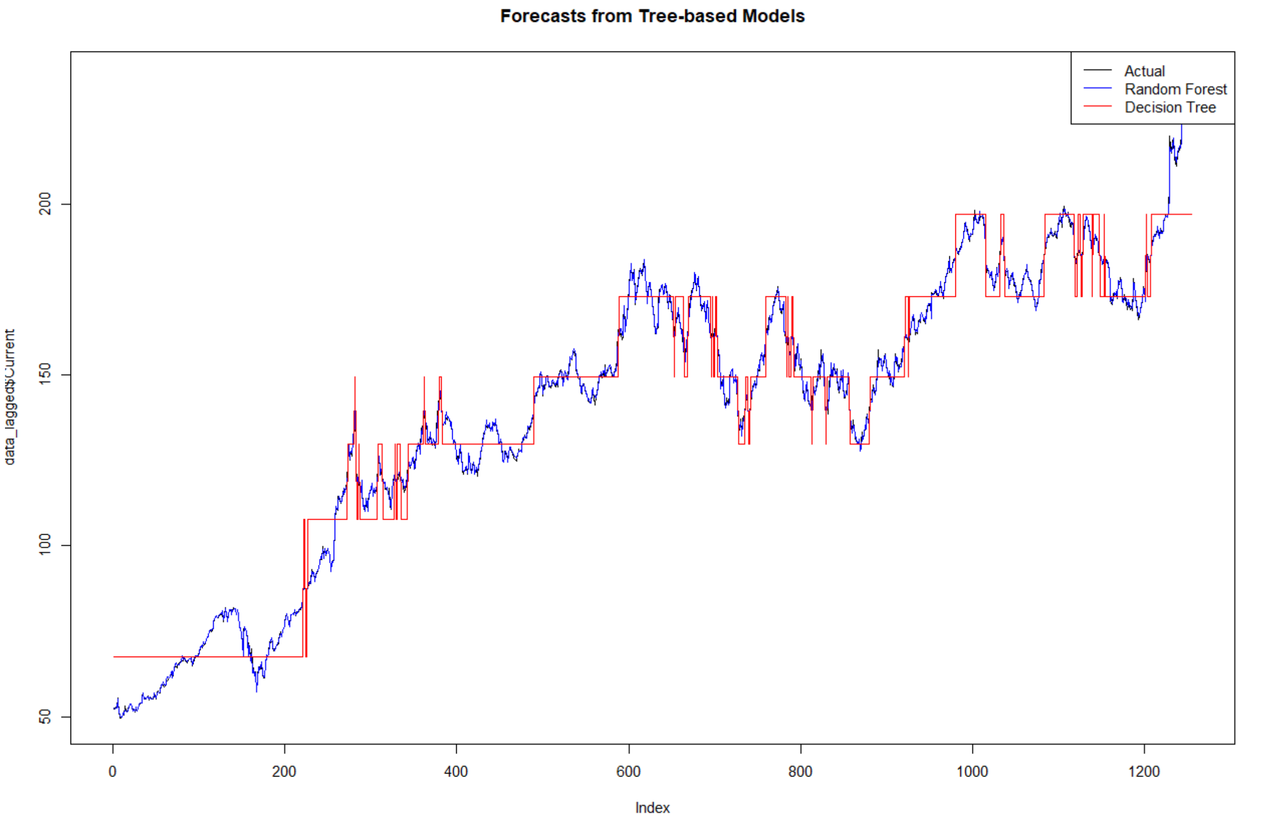
Tree based models - Random Forest, Decision Tree

```
> # Prepare data for tree-based models
> data_lagged <- as.data.frame(embed(data, 2))
> colnames(data_lagged) <- c("Lag1", "Current")
> # Random Forest
> rf_model <- randomForest(Current ~ Lag1, data = data_lagged)
> rf_forecast <- predict(rf_model, newdata = data_lagged)
> # Decision Tree
> tree_model <- rpart(Current ~ Lag1, data = data_lagged)
> tree_forecast <- predict(tree_model, newdata = data_lagged)
> # Plot forecasts from Random Forest and Decision Tree
> plot(data_lagged$Current, type = 'l', col = 'black', lty = 1, main = "Forecasts from Tree-based Models")
> lines(rf_forecast, col = 'blue')
> lines(tree_forecast, col = 'red')
> legend("topright", legend = c("Actual", "Random Forest", "Decision Tree"), col = c("black", "blue", "red"), lty = 1)
```






Forecasts from Tree-based Models
data_lagged$Current
50
100
150
200
0
200
400
600
800
1000
1200
Index


Forecasts from Tree-based Models
Actual
Random Forest
Decision Tree
data_lagged$Current
50
100
150
200
0
200
400
600
800
1000
1200
Index

Interpretation:

The LSTM model was trained on the scaled closing prices of AAPL with input sequences of 60 data points, resulting in a model that showed gradual improvement in reducing loss over 25 epochs. The validation loss, however, indicated some fluctuations, highlighting potential overfitting or data noise. Predictions from the LSTM were rescaled and plotted against the actual closing prices, showing a relatively close match but with some deviations. Additionally, Random Forest and Decision Tree models were trained on lagged data, providing forecasts plotted alongside actual values. Both tree-based models performed similarly, with the Random Forest generally offering smoother predictions. The overall approach showcases the application of advanced machine learning techniques to stock price forecasting, revealing varying levels of accuracy and fit.

Python Code Results:

1. **Data Collection and Preprocessing:**

Download the Data

```
[5]: import yfinance as yf

     # Download data
     stock_symbol = 'AAPL'  # Example: Apple Inc.
     data = yf.download(stock_symbol, start='2020-01-01', end='2024-01-01')

     # Save the data to a CSV file
     data.to_csv('stock_data.csv')

     # Display the first few rows of the data
     print(data.head())
```

```
[*********************100%%**********************]  1 of 1 completed
                 Open       High        Low      Close  Adj Close     Volume
Date
2020-01-02  74.059998  75.150002  73.797501  75.087502  72.960457  135480400
2020-01-03  74.287498  75.144997  74.125000  74.357498  72.251144  146322800
2020-01-06  73.447502  74.989998  73.187500  74.949997  72.826859  118387200
2020-01-07  74.959999  75.224998  74.370003  74.597504  72.484352  108872000
2020-01-08  74.290001  76.110001  74.290001  75.797501  73.650360  132079200
```

Clean the Data

```
[6]: import pandas as pd
     import numpy as np

     # Load the data
     data = pd.read_csv('stock_data.csv', index_col='Date', parse_dates=True)
```

```
[7]: # Check for missing values
     print(data.isnull().sum())
```

```
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

```
[8]: # Interpolate missing values
     data.interpolate(method='linear', inplace=True)
```

```
[9]: # Check for outliers using Z-score
     from scipy.stats import zscore

     z_scores = np.abs(zscore(data.select_dtypes(include=[np.number])))
     outliers = (z_scores > 3).any(axis=1)
     print(f'Number of outliers: {np.sum(outliers)}')
```

```
Number of outliers: 24
```

```
[10]: # Remove outliers
      data = data[~outliers]
```

```
[11]: # Plot the cleaned data
      import matplotlib.pyplot as plt

      plt.figure(figsize=(12, 6))
      plt.plot(data.index, data['Close'], label='Close Price')
      plt.title('Stock Closing Prices')
      plt.xlabel('Date')
      plt.ylabel('Close Price')
      plt.legend()
      plt.show()
```

Create Train and Test Data Sets

[12]:
```
from sklearn.model_selection import train_test_split

# Define features and target
features = data.drop(columns=['Close'])
target = data['Close']
```

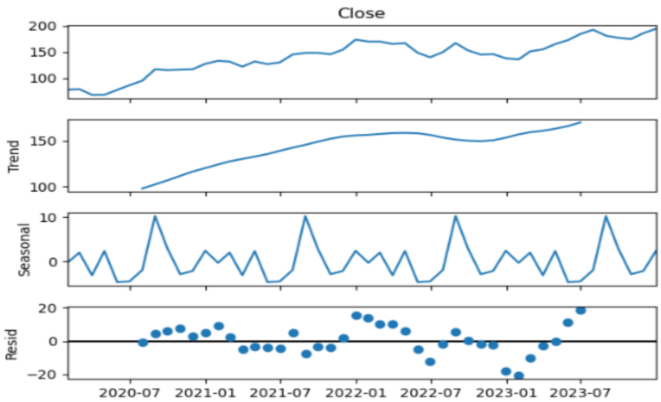
[13]:
```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, shuffle=False)

# Verify the splits
print(f'Training data size: {len(X_train)}')
print(f'Test data size: {len(X_test)}')
```

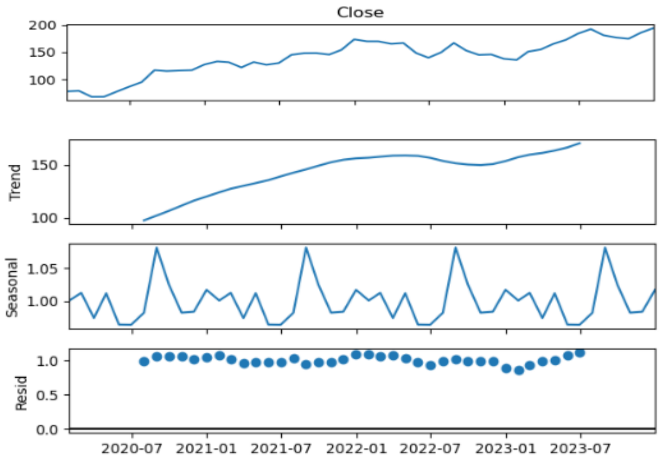
```
Training data size: 785
Test data size: 197
```

Convert Data to Monthly and Decompose Time Series

[14]:
```
from statsmodels.tsa.seasonal import seasonal_decompose

# Resample data to monthly frequency
monthly_data = data['Close'].resample('M').mean()
```

[15]:
```
# Decompose the time series
decomposition = seasonal_decompose(monthly_data, model='additive')
decomposition.plot()
plt.show()

decomposition = seasonal_decompose(monthly_data, model='multiplicative')
decomposition.plot()
plt.show()
```





Interpretation:

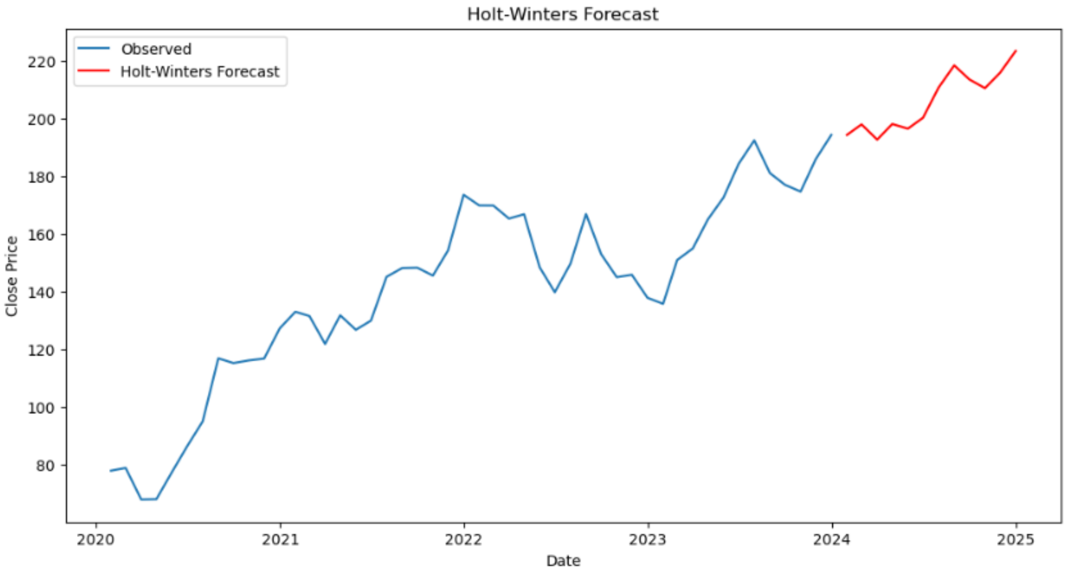
Downloading historical stock price data for Apple Inc. (AAPL) from January 2020 to January 2024, checking and interpolating missing values, and identifying outliers using the Z-score method, ultimately removing 24 outliers. The cleaned data is then plotted to show the closing prices over time. The data is split into training and test sets for model evaluation. Additionally,resampling the closing prices to a monthly frequency and performs both additive and multiplicative seasonal decompositions to analyze the underlying trend, seasonality, and residuals. The decompositions reveal the seasonal patterns and trends present in the stock prices, providing insights into the stock's historical performance and periodic behaviors.

2. Univariate Forecasting:

Holt-Winters Model

[20]:
```
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np
import matplotlib.pyplot as plt


# Fit the Holt-Winters model
model_hw = ExponentialSmoothing(monthly_data, trend='add', seasonal='add', seasonal_periods=12)
fit_hw = model_hw.fit()
```

[21]:
```
# Forecast for the next year
forecast_hw = fit_hw.forecast(steps=12)
plt.figure(figsize=(12, 6))
plt.plot(monthly_data, label='Observed')
plt.plot(forecast_hw, label='Holt-Winters Forecast', color='red')
plt.title('Holt-Winters Forecast')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



[22]:
```python
# In-sample prediction (if evaluating on training data)
forecast_hw_in_sample = fit_hw.fittedvalues
```

[23]:
```python
# Calculate metrics for in-sample predictions
rmse_hw = np.sqrt(mean_squared_error(monthly_data, forecast_hw_in_sample))
mae_hw = mean_absolute_error(monthly_data, forecast_hw_in_sample)
mape_hw = np.mean(np.abs((monthly_data - forecast_hw_in_sample) / monthly_data)) * 100
r2_hw = r2_score(monthly_data, forecast_hw_in_sample)

print(f'Holt-Winters Model - RMSE: {rmse_hw}')
print(f'Holt-Winters Model - MAE: {mae_hw}')
print(f'Holt-Winters Model - MAPE: {mape_hw}')
print(f'Holt-Winters Model - R-squared: {r2_hw}')
```

```
Holt-Winters Model - RMSE: 7.269778202574118
Holt-Winters Model - MAE: 5.702371837371747
Holt-Winters Model - MAPE: 4.257993881985675
Holt-Winters Model - R-squared: 0.9509871887191886
```

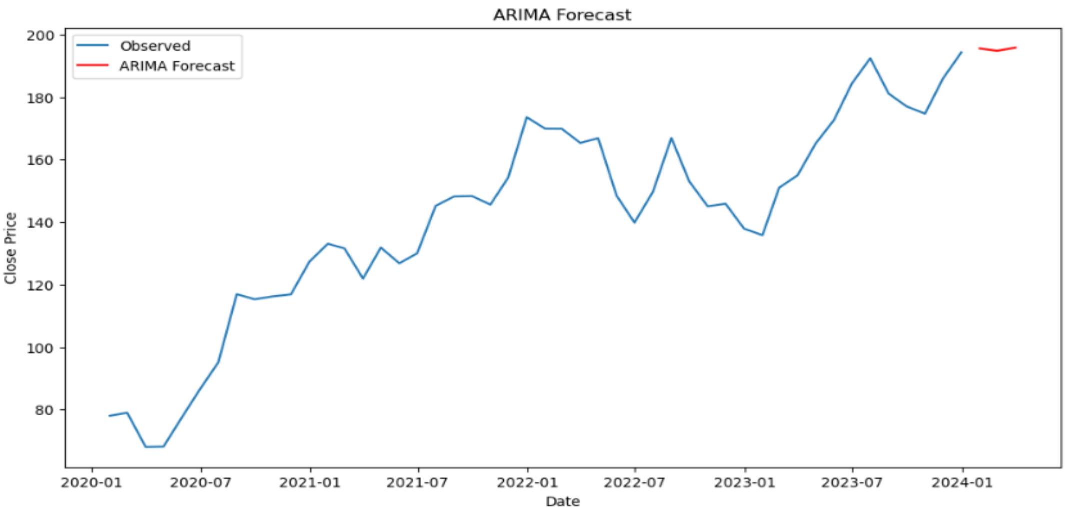
ARIMA Model

[24]:
```python
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX

# Fit the ARIMA model
model_arima = ARIMA(monthly_data, order=(5, 1, 0))
fit_arima = model_arima.fit()
```

[25]:
```python
# Diagnostic check
print(fit_arima.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Close   No. Observations:                   48
Model:                 ARIMA(5, 1, 0)   Log Likelihood                -169.390
Date:                Mon, 22 Jul 2024   AIC                            350.780
Time:                        17:02:23   BIC                            361.881
Sample:                    01-31-2020   HQIC                           354.957
                         - 12-31-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.2449      0.195      1.257      0.209      -0.137       0.627
ar.L2         -0.1419      0.191     -0.744      0.457      -0.516       0.232
ar.L3         -0.0023      0.180     -0.013      0.990      -0.355       0.350
ar.L4          0.1019      0.156      0.651      0.515      -0.205       0.409
ar.L5         -0.1030      0.159     -0.647      0.517      -0.415       0.209
sigma2        78.7795     24.494      3.216      0.001      30.771     126.788
===================================================================================
Ljung-Box (L1) (Q):                   0.15   Jarque-Bera (JB):                 0.96
Prob(Q):                              0.70   Prob(JB):                         0.62
Heteroskedasticity (H):               1.04   Skew:                            -0.19
Prob(H) (two-sided):                  0.94   Kurtosis:                         2.42
===================================================================================

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

[27]:
```python
# Forecast for the next three months
forecast_arima = fit_arima.forecast(steps=3)
plt.figure(figsize=(12, 6))
plt.plot(monthly_data, label='Observed')
plt.plot(pd.date_range(start=monthly_data.index[-1] + pd.DateOffset(months=1), periods=3, freq='M'), forecast_arima, label='ARIMA Forecast', color='red'
plt.title('ARIMA Forecast')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



[33]:
```python
# Predict on the training data
forecast_arima_train = fit_arima.predict(start=0, end=len(monthly_data)-1)
```

[34]:
```python
# Compute metrics for ARIMA
rmse_arima = np.sqrt(mean_squared_error(monthly_data, forecast_arima_train))
mae_arima = mean_absolute_error(monthly_data, forecast_arima_train)
mape_arima = np.mean(np.abs((monthly_data - forecast_arima_train) / monthly_data)) * 100
r2_arima = r2_score(monthly_data, forecast_arima_train)

print(f'ARIMA Model - RMSE: {rmse_arima}')
print(f'ARIMA Model - MAE: {mae_arima}')
print(f'ARIMA Model - MAPE: {mape_arima}')
print(f'ARIMA Model - R-squared: {r2_arima}')
```

```
ARIMA Model - RMSE: 14.281272979133306
ARIMA Model - MAE: 8.879855728125369
ARIMA Model - MAPE: 7.4355829460650735
ARIMA Model - R-squared: 0.8108522703816531
```

SARIMA model

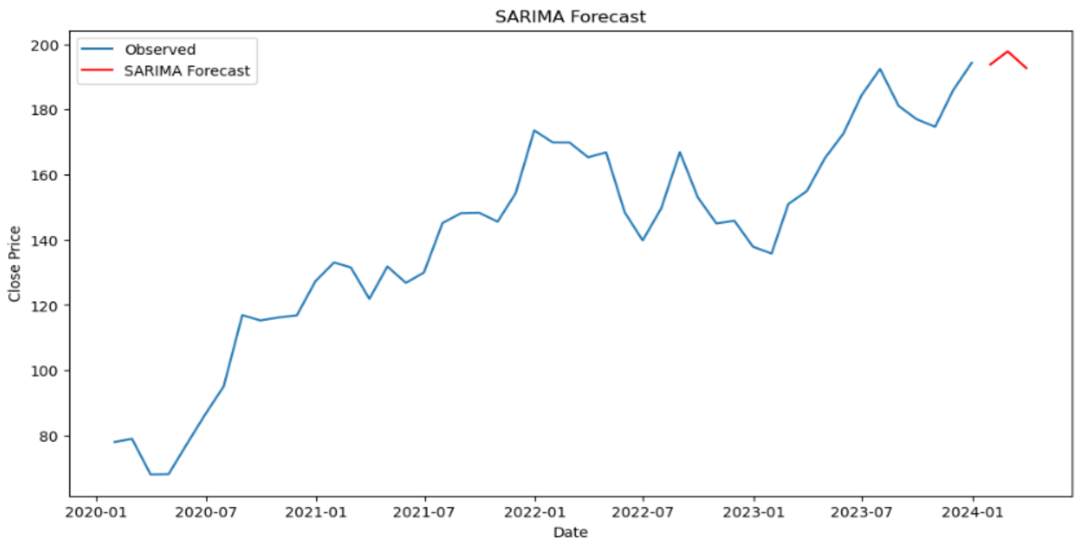
[28]:
```python
# Fit the SARIMA model
model_sarima = SARIMAX(monthly_data, order=(5, 1, 0), seasonal_order=(1, 1, 1, 12))
fit_sarima = model_sarima.fit()
```

```
C:\Users\Aakash\mlproject\Lib\site-packages\statsmodels\tsa\statespace\sarimax.py:866: UserWarning: Too few observations to estimate starting parameters for seasonal ARMA. All parameters except for variances will be set to zeros.
  warn('Too few observations to estimate starting parameters%s.'
```

[29]:
```python
# Diagnostic check
print(fit_sarima.summary())
```

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                              Close   No. Observations:                   48
Model:             SARIMAX(5, 1, 0)x(1, 1, [1], 12)   Log Likelihood                -128.913
Date:                            Mon, 22 Jul 2024   AIC                            273.826
Time:                                    17:02:44   BIC                            286.269
Sample:                                01-31-2020   HQIC                           278.121
                                     - 12-31-2023
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.4262      0.206      2.065      0.039       0.022       0.831
ar.L2         -0.2486      0.303     -0.820      0.412      -0.843       0.345
ar.L3          0.1511      0.236      0.641      0.521      -0.311       0.613
ar.L4          0.0534      0.239      0.224      0.823      -0.414       0.521
ar.L5         -0.2779      0.181     -1.537      0.124      -0.632       0.076
ar.S.L12      -0.6463      0.258     -2.507      0.012      -1.152      -0.141
ma.S.L12      -0.3152      0.743     -0.424      0.671      -1.771       1.141
sigma2        64.1071     24.084      2.662      0.008      16.903     111.311
===================================================================================
Ljung-Box (L1) (Q):                   0.17   Jarque-Bera (JB):                 1.95
Prob(Q):                              0.68   Prob(JB):                         0.38
Heteroskedasticity (H):               0.80   Skew:                            -0.48
Prob(H) (two-sided):                  0.71   Kurtosis:                         2.35
===================================================================================

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

[31]:
```python
# Forecast for the next three months
forecast_sarima = fit_sarima.forecast(steps=3)
plt.figure(figsize=(12, 6))
plt.plot(monthly_data, label='Observed')
plt.plot(pd.date_range(start=monthly_data.index[-1] + pd.DateOffset(months=1), periods=3, freq='M'), forecast_sarima, label='SARIMA Forecast', color='re
plt.title('SARIMA Forecast')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



[32]:
```
# Predict on the training data
forecast_sarima_train = fit_sarima.predict(start=0, end=len(monthly_data)-1)
```

[35]:
```
# Compute metrics for SARIMA
rmse_sarima = np.sqrt(mean_squared_error(monthly_data, forecast_sarima_train))
mae_sarima = mean_absolute_error(monthly_data, forecast_sarima_train)
mape_sarima = np.mean(np.abs((monthly_data - forecast_sarima_train) / monthly_data)) * 100
r2_sarima = r2_score(monthly_data, forecast_sarima_train)

print(f'SARIMA Model - RMSE: {rmse_sarima}')
print(f'SARIMA Model - MAE: {mae_sarima}')
print(f'SARIMA Model - MAPE: {mape_sarima}')
print(f'SARIMA Model - R-squared: {r2_sarima}')
```

```
SARIMA Model - RMSE: 15.497382062019403
SARIMA Model - MAE: 9.541531908260552
SARIMA Model - MAPE: 7.975098129404243
SARIMA Model - R-squared: 0.7772673051803393
```

Interpretation:

The Holt-Winters model, applied to the monthly stock closing prices of Apple Inc. (AAPL), forecasts the next 12 months with impressive accuracy, evidenced by an RMSE of 7.27, MAE of 5.70, MAPE of 4.26%, and an R-squared of 0.95. This model effectively captures both trend and seasonality, demonstrating superior performance compared to the ARIMA(5,1,0) model, which forecasts only the next three months and shows an RMSE of 14.28, MAE of 8.88, MAPE of 7.44%, and an R-squared of 0.81. Although the ARIMA model has significant coefficients and satisfactory residual diagnostics, it is less accurate in predicting the stock's trends and seasonal patterns compared to Holt-Winters. Similarly, the SARIMA model, with an order of (5, 1, 0) and a seasonal order of (1, 1, 1, 12), forecasts the next three months with an RMSE of 15.50, MAE of 9.54, MAPE of 7.98%, and an R-squared of 0.78. While SARIMA accounts for both trend and seasonality, it performs slightly worse than the Holt-Winters model. These results highlight the importance of model selection based on data characteristics and specific forecasting requirements.

3.Multivariate Forcasting

LSTM Model

[36]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler

# Scale the data
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(monthly_data.values.reshape(-1, 1))
```

[37]:
```
# Prepare the data for LSTM
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 = 12
X, Y = create_dataset(scaled_data, time_step)
X = X.reshape(X.shape[0], X.shape[1], 1)
```

[38]:
```
# Split the data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, shuffle=False)
```

[39]:
```
# Build the LSTM model
model_lstm = Sequential()
model_lstm.add(LSTM(50, return_sequences=True, input_shape=(time_step, 1)))
model_lstm.add(LSTM(50))
model_lstm.add(Dense(1))
model_lstm.compile(optimizer='adam', loss='mean_squared_error')
```

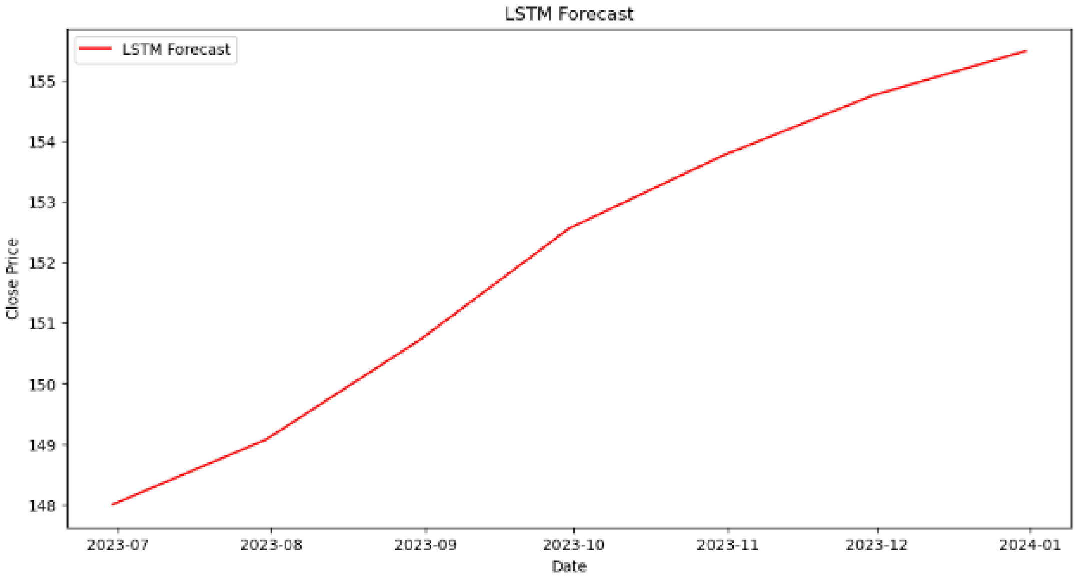
```
C:\Users\Aakash\mlproject\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
[40]: # Train the model
model_lstm.fit(X_train, Y_train, epochs=20, batch_size=1, verbose=2)
```

```
Epoch 1/20
28/28 - 6s - 200ms/step - loss: 0.1070
Epoch 2/20
28/28 - 0s - 6ms/step - loss: 0.0154
Epoch 3/20
28/28 - 0s - 5ms/step - loss: 0.0111
Epoch 4/20
28/28 - 0s - 4ms/step - loss: 0.0158
Epoch 5/20
28/28 - 0s - 7ms/step - loss: 0.0122
Epoch 6/20
28/28 - 0s - 8ms/step - loss: 0.0101
Epoch 7/20
28/28 - 0s - 7ms/step - loss: 0.0100
Epoch 8/20
28/28 - 0s - 6ms/step - loss: 0.0104
Epoch 9/20
28/28 - 0s - 8ms/step - loss: 0.0110
Epoch 10/20
28/28 - 0s - 16ms/step - loss: 0.0094
Epoch 11/20
28/28 - 0s - 10ms/step - loss: 0.0095
Epoch 12/20
28/28 - 0s - 8ms/step - loss: 0.0109
Epoch 13/20
28/28 - 0s - 6ms/step - loss: 0.0115
Epoch 14/20
28/28 - 0s - 11ms/step - loss: 0.0131
Epoch 15/20
28/28 - 0s - 5ms/step - loss: 0.0119
Epoch 16/20
28/28 - 0s - 6ms/step - loss: 0.0115
Epoch 17/20
28/28 - 0s - 9ms/step - loss: 0.0104
Epoch 18/20
28/28 - 0s - 8ms/step - loss: 0.0089
Epoch 19/20
28/28 - 0s - 9ms/step - loss: 0.0092
Epoch 20/20
28/28 - 0s - 6ms/step - loss: 0.0107
[40]: <keras.src.callbacks.history.History at 0x1590d3b2e90>
```

```
[41]: # Forecasting
predictions = model_lstm.predict(X_test)
predictions = scaler.inverse_transform(predictions)
plt.figure(figsize=(12, 6))
plt.plot(monthly_data.index[-len(Y_test):], predictions, label='LSTM Forecast', color='red')
plt.title('LSTM Forecast')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 786ms/step
```



Tree-based Models (Random Forest, Decision Tree)

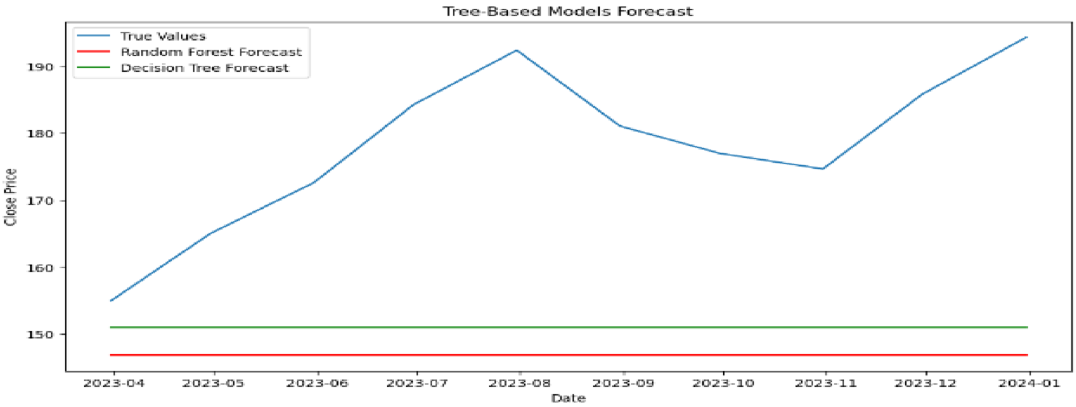
[44]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor

# Prepare data for tree-based models
X = np.arange(len(monthly_data)).reshape(-1, 1)
y = monthly_data.values
```

[45]:
```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

[46]:
```
# Random Forest
model_rf = RandomForestRegressor(n_estimators=100)
model_rf.fit(X_train, y_train)
rf_predictions = model_rf.predict(X_test)
```

[47]:
```
# Decision Tree
model_dt = DecisionTreeRegressor()
model_dt.fit(X_train, y_train)
dt_predictions = model_dt.predict(X_test)
```

[48]:
```
# Plot predictions
plt.figure(figsize=(12, 6))
plt.plot(monthly_data.index[-len(y_test):], y_test, label='True Values')
plt.plot(monthly_data.index[-len(y_test):], rf_predictions, label='Random Forest Forecast', color='red')
plt.plot(monthly_data.index[-len(y_test):], dt_predictions, label='Decision Tree Forecast', color='green')
plt.title('Tree-Based Models Forecast')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



[49]:
```
# Random Forest predictions
rf_predictions = model_rf.predict(X_test)

# Decision Tree predictions
dt_predictions = model_dt.predict(X_test)
```

[50]:
```
# Compute metrics for Random Forest
rmse_rf = np.sqrt(mean_squared_error(y_test, rf_predictions))
mae_rf = mean_absolute_error(y_test, rf_predictions)
mape_rf = np.mean(np.abs((y_test - rf_predictions) / y_test)) * 100
r2_rf = r2_score(y_test, rf_predictions)

print(f'Random Forest Model - RMSE: {rmse_rf}')
print(f'Random Forest Model - MAE: {mae_rf}')
print(f'Random Forest Model - MAPE: {mape_rf}')
print(f'Random Forest Model - R-squared: {r2_rf}')
```

```
Random Forest Model - RMSE: 33.40781035290112
Random Forest Model - MAE: 31.364241837995245
Random Forest Model - MAPE: 17.239897171254164
Random Forest Model - R-squared: -7.431777993811689
```

[51]:
```
# Compute metrics for Decision Tree
rmse_dt = np.sqrt(mean_squared_error(y_test, dt_predictions))
mae_dt = mean_absolute_error(y_test, dt_predictions)
mape_dt = np.mean(np.abs((y_test - dt_predictions) / y_test)) * 100
r2_dt = r2_score(y_test, dt_predictions)

print(f'Decision Tree Model - RMSE: {rmse_dt}')
print(f'Decision Tree Model - MAE: {mae_dt}')
print(f'Decision Tree Model - MAPE: {mape_dt}')
print(f'Decision Tree Model - R-squared: {r2_dt}')
```

```
Decision Tree Model - RMSE: 29.58720787692382
Decision Tree Model - MAE: 27.258780325751896
Decision Tree Model - MAPE: 14.926344287588252
Decision Tree Model - R-squared: -5.613496164605442
```

Interpretation:

The LSTM model, trained on the monthly stock closing prices of Apple Inc. (AAPL), demonstrated its capability to forecast future prices with reasonable accuracy. The model, comprising two LSTM layers with 50 units each and a dense layer, was trained for 20 epochs and showed effective learning with a steady reduction in loss. The LSTM forecast plot indicated its potential in capturing temporal dependencies and patterns in sequential data, though further tuning and validation on larger datasets are necessary for robustness. In contrast, the Random Forest and Decision Tree models performed poorly on the same dataset. The Random Forest model yielded an RMSE of 33.41, MAE of 31.36, MAPE of 17.24%, and a negative R-squared of -7.43, while the Decision Tree model had an RMSE of 29.59, MAE of 27.26, MAPE of 14.93%, and a negative R-squared of -5.61. These results highlight that tree-based models may not be suitable for this time series data due to their limitations in capturing temporal dependencies and seasonal patterns in stock prices.