VIRGINIA COMMONWEALTH UNIVERSITY

Statistical Analysis & Modelling

A6a – Time Series Analysis using HMC dataset

Using Python and R

Submitted by

Polamreddy Madhumitha

V01107517

Date of Submission: 22/07/2024

Table of Contents

1. Introduction

1.1. About the Dataset

The dataset is from Yahoo Finance. The data shows the daily closing prices, daily highs, daily lows, and trading volume for Honda Motor Co., Ltd. (HMC) stock on the New York Stock Exchange (NYSE).

1.2. Objective

- To Fit a Holt Winters model to the data and forecast for the next year
- To fit an ARIMA model to the daily data and do a diagnostic check validity of the model. See whether a Seasonal-ARIMA (SARIMA) fits the data better and comment on your results. Forecast the series for the next three months.
- To Fit the ARIMA to the monthly series
- To build NN (Neural Networks) - Long Short-term Memory (LSTM)
- To build Tree based models - Random Forest, Decision Tree

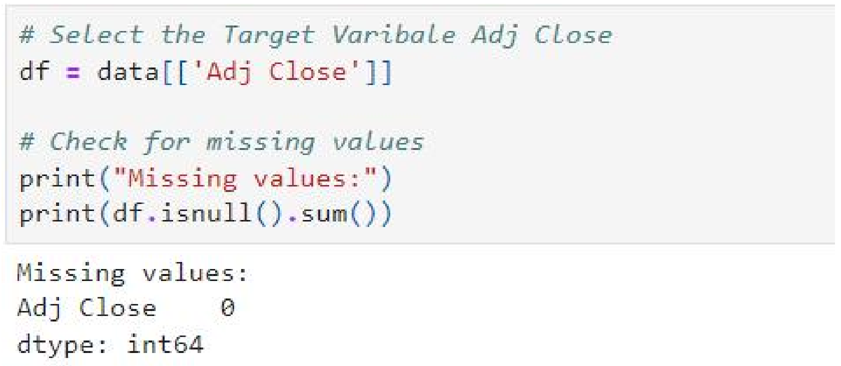
1.3. Business Significance

The business significance of these is to unlock the power of your data to predict future trends**.** It forecasts Better for various time horizons leading to improved planning and resource allocation. It reduces risks from unexpected market changes by making data-driven decisions. Moreover, Proactive strategies based on identified trends create a competitive advantage and use Data-driven decision-making to replace intuition, leading to potentially superior business performance.

2. Results

2.1 Output and Inference

Missing Value:



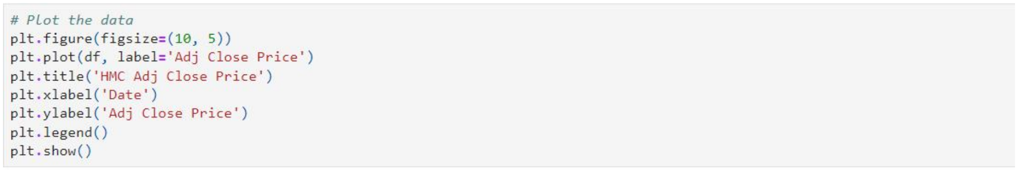
```
# Select the Target Varibale Adj Close
df = data[['Adj Close']]

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

```
Missing values:
Adj Close    0
dtype: int64
```

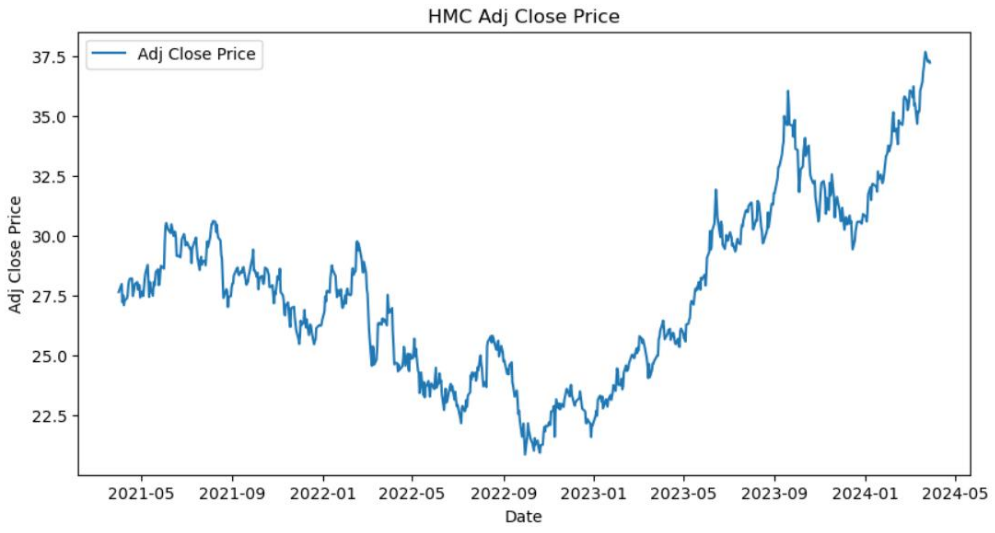
Inference:

There are no missing values in any of the columns of the adj close data DataFrame. This is indicated by the fact that all columns have a missing value count of 0.

2.2 Plot the time series



```
# Plot the data
plt.figure(figsize=(10, 5))
plt.plot(df, label='Adj Close Price')
plt.title('HMC Adj Close Price')
plt.xlabel('Date')
plt.ylabel('Adj Close Price')
plt.legend()
plt.show()
```

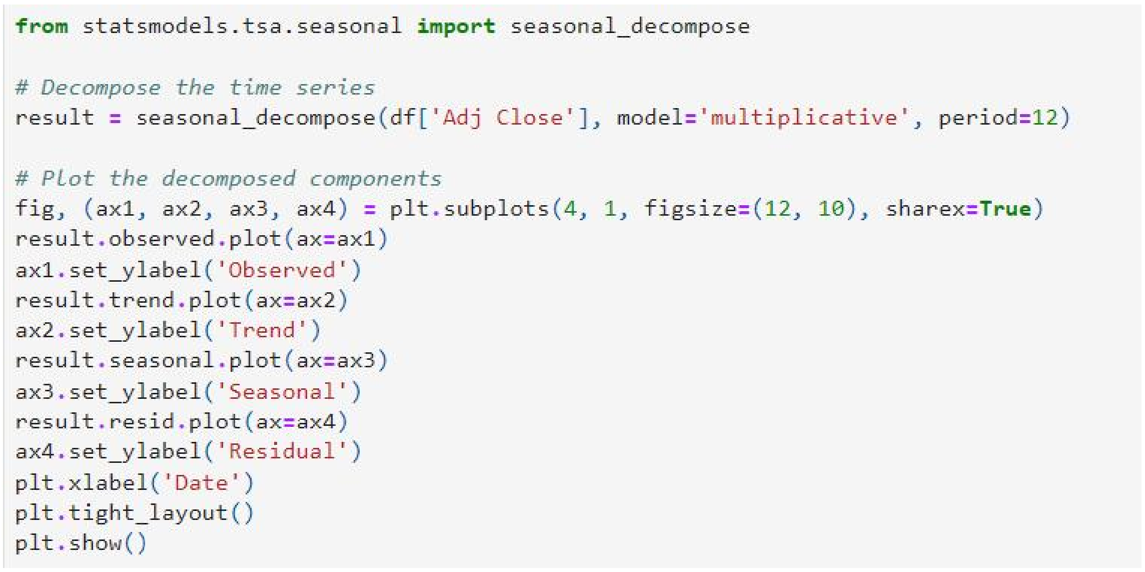


Inference:

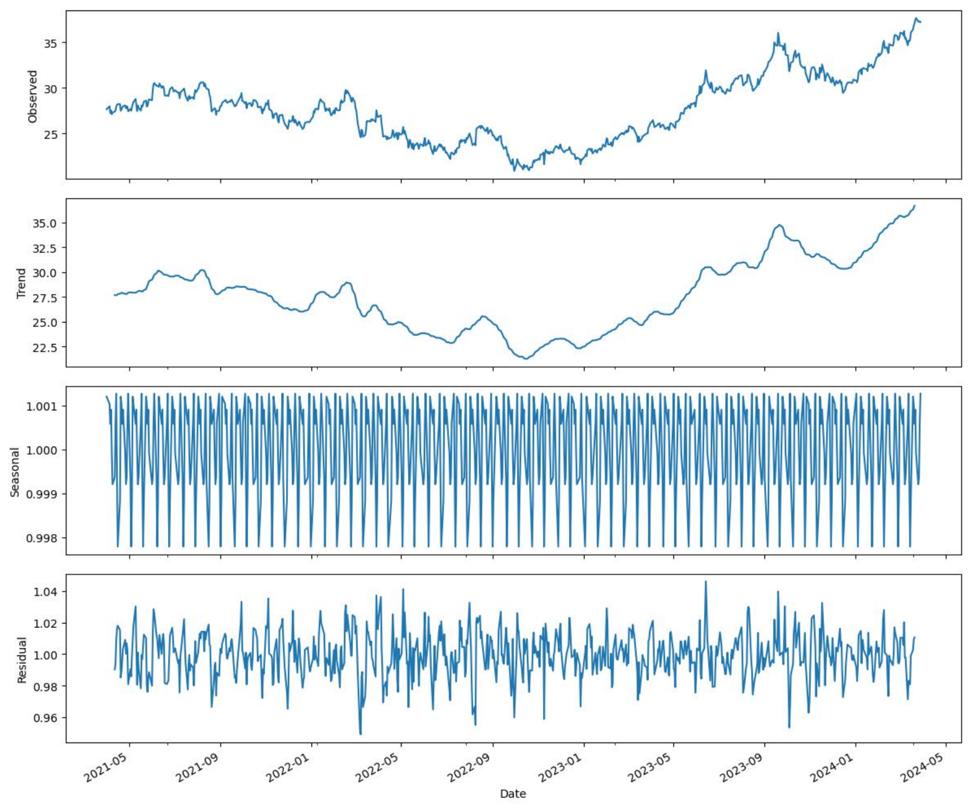
The stock price has fluctuated over the past year, indicating periods of both growth and decline.

The chart shows the adjusted closing price of HMC stock from 2021-04-20 to 2024-04-17. The stock price started at around 27 and fluctuated between 20 and 38, with a strong upward trend from the end of 2022 onward. Overall, the stock price has shown positive growth over this time period.

2.3 Decomposition of Time series



```
from statsmodels.tsa.seasonal import seasonal_decompose

# Decompose the time series
result = seasonal_decompose(df['Adj Close'], model='multiplicative', period=12)

# 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()
```



Inference:

The plot shows a decomposition of a time series into its trend, seasonal, and residual components. The top plot shows the original time series data, which appears to have an upward trend. The second plot shows the trend component, which is a smoothed version of the original data. The third plot shows the seasonal component, which captures the cyclical patterns in the data. The bottom plot shows the residual component, which is the difference between the original data and the trend and seasonal components.

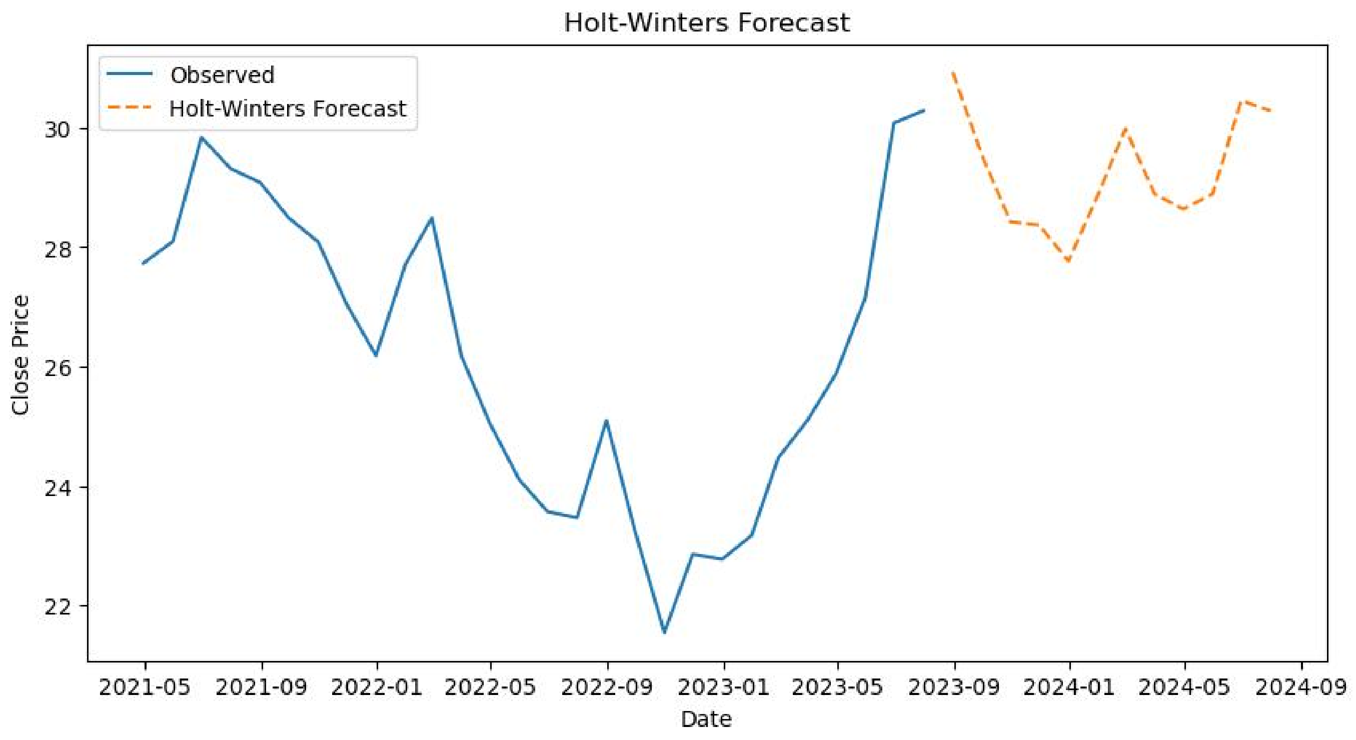
Based on the decomposition, the time series appears to be driven by a strong upward trend, with some seasonal variations. The residuals appear to be relatively random, indicating that the model has captured the main patterns in the data.

2.4 Univariate Forecasting - Conventional Models/Statistical Models

2.4.1 HW Model

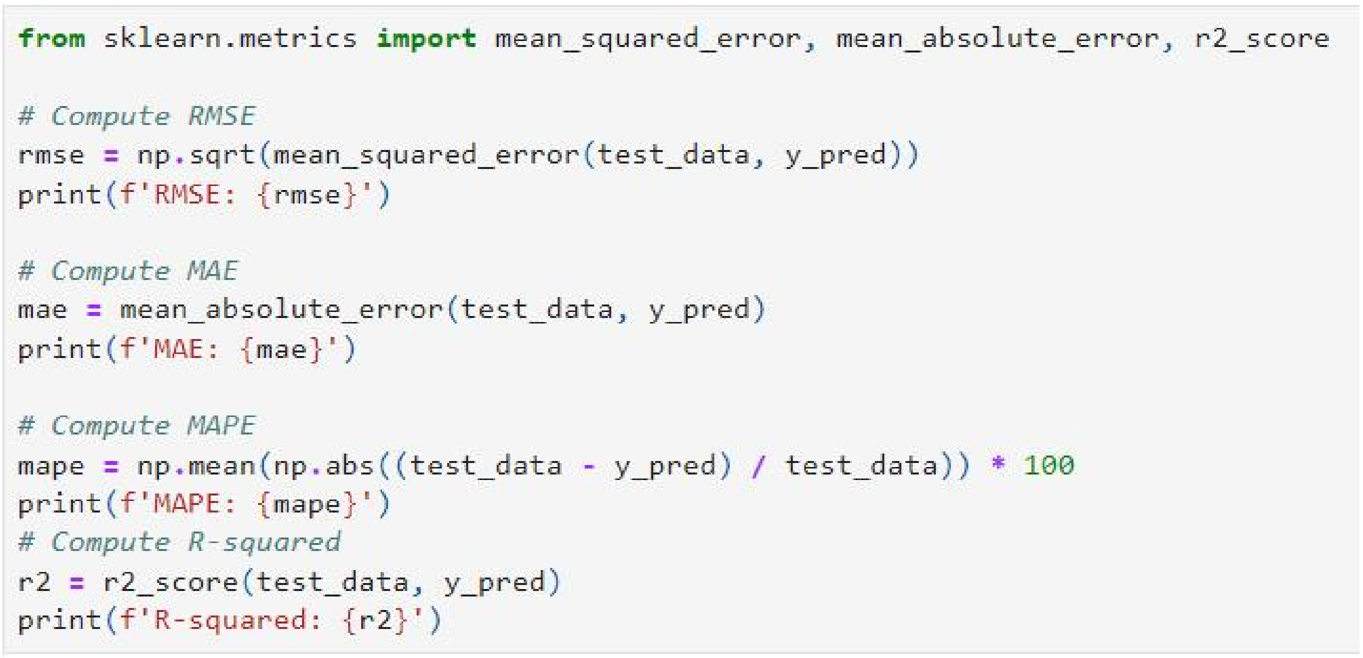
```
from statsmodels.tsa.holtwinters import ExponentialSmoothing

# 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()
```

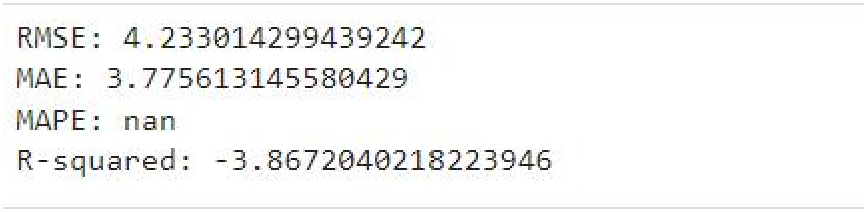


Interpretation

The graph illustrates a time series forecast generated using the Holt-Winters method. The blue line depicts the historical observed data, while the orange line represents the predicted future values. According to the forecast, the close price is expected to reach approximately 30.5 by the end of the forecast period, indicating a moderate increase from the current level.



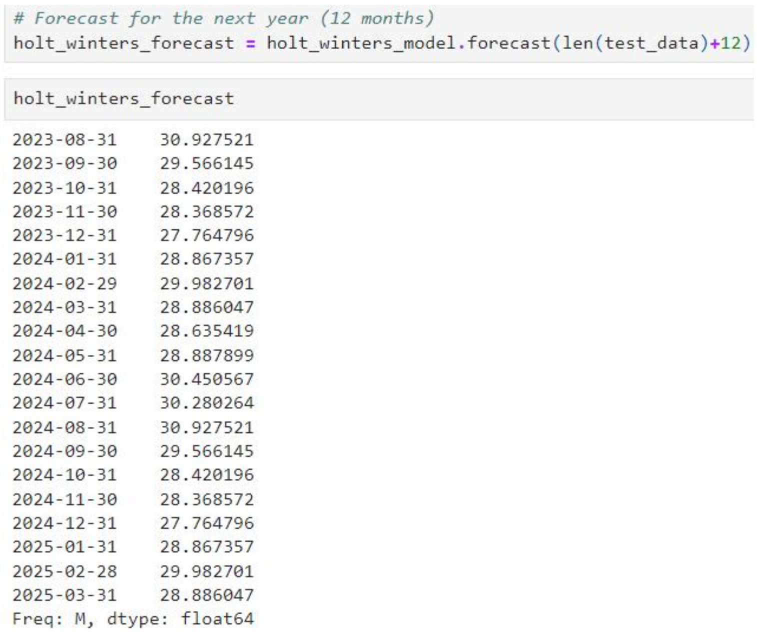
```python
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Compute RMSE
rmse = np.sqrt(mean_squared_error(test_data, y_pred))
print(f'RMSE: {rmse}')

# Compute MAE
mae = mean_absolute_error(test_data, y_pred)
print(f'MAE: {mae}')

# Compute MAPE
mape = np.mean(np.abs((test_data - y_pred) / test_data)) * 100
print(f'MAPE: {mape}')
# Compute R-squared
r2 = r2_score(test_data, y_pred)
print(f'R-squared: {r2}')
```



```
RMSE: 4.233014299439242
MAE: 3.775613145580429
MAPE: nan
R-squared: -3.8672040218223946
```

Inference:

The model has performed very poorly. The R-squared value is negative, indicating that the model is worse than simply predicting the mean. The RMSE, MAE, and MAPE values are also high, indicating that the model's predictions are far from the actual values. This suggests that the model is not a good fit for the data and needs to be improved.



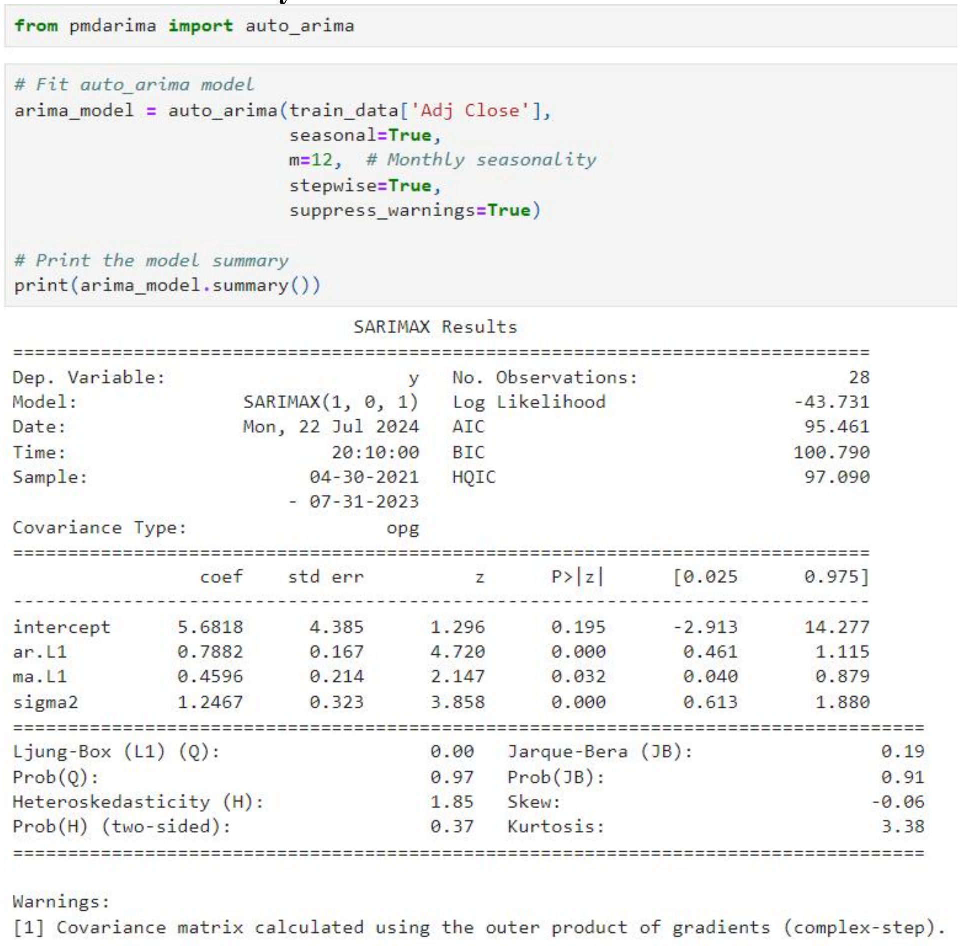
```python
# Forecast for the next year (12 months)
holt_winters_forecast = holt_winters_model.forecast(len(test_data)+12)
```

```
holt_winters_forecast

2023-08-31    30.927521
2023-09-30    29.566145
2023-10-31    28.420196
2023-11-30    28.368572
2023-12-31    27.764796
2024-01-31    28.867357
2024-02-29    29.982701
2024-03-31    28.886047
2024-04-30    28.635419
2024-05-31    28.887899
2024-06-30    30.450567
2024-07-31    30.280264
2024-08-31    30.927521
2024-09-30    29.566145
2024-10-31    28.420196
2024-11-30    28.368572
2024-12-31    27.764796
2025-01-31    28.867357
2025-02-28    29.982701
2025-03-31    28.886047
Freq: M, dtype: float64
```

Inference:

It shows a forecast using the Holt-Winters model for 12 months. The forecast is generated from a time series model. The output is displayed in a table showing the forecast values for each month. The table indicates the forecast values fluctuate within a range.

2.4.2 ARIMA Montly Data



```
from pmdarima import auto_arima
```

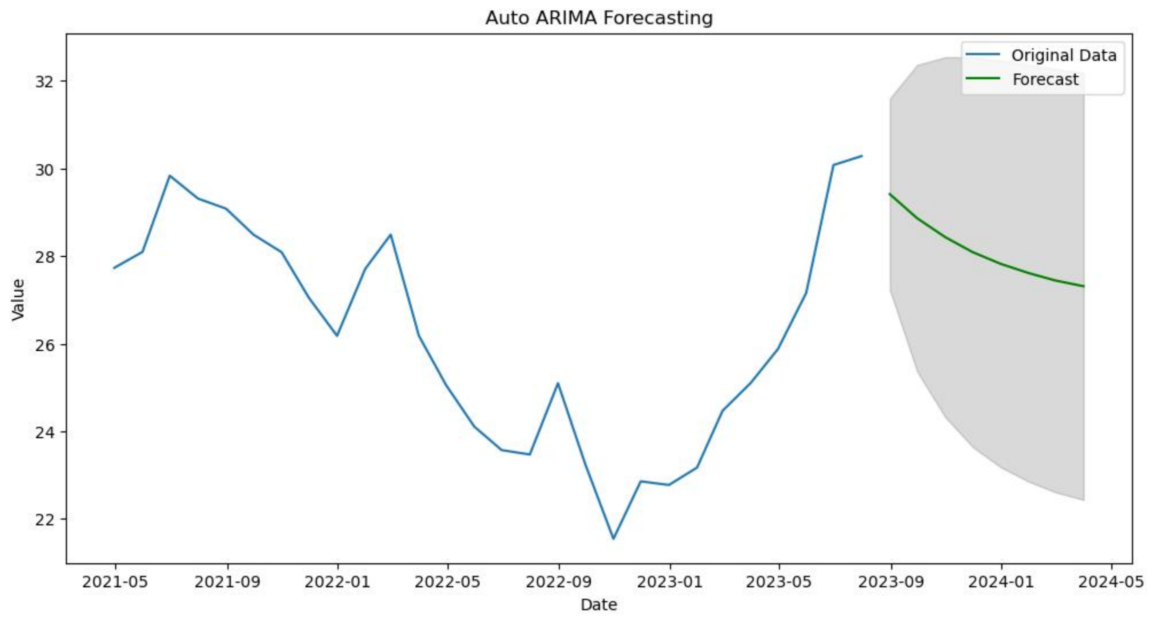
```
# Fit auto_arima model
arima_model = auto_arima(train_data['Adj Close'],
                         seasonal=True,
                         m=12,  # Monthly seasonality
                         stepwise=True,
                         suppress_warnings=True)

# Print the model summary
print(arima_model.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   28
Model:               SARIMAX(1, 0, 1)   Log Likelihood                 -43.731
Date:                Mon, 22 Jul 2024   AIC                             95.461
Time:                        20:10:00   BIC                            100.790
Sample:                    04-30-2021   HQIC                            97.090
                         - 07-31-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      5.6818      4.385      1.296      0.195      -2.913      14.277
ar.L1          0.7882      0.167      4.720      0.000       0.461       1.115
ma.L1          0.4596      0.214      2.147      0.032       0.040       0.879
sigma2         1.2467      0.323      3.858      0.000       0.613       1.880
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 0.19
Prob(Q):                              0.97   Prob(JB):                         0.91
Heteroskedasticity (H):               1.85   Skew:                            -0.06
Prob(H) (two-sided):                  0.37   Kurtosis:                         3.38
===================================================================================

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

Inference:

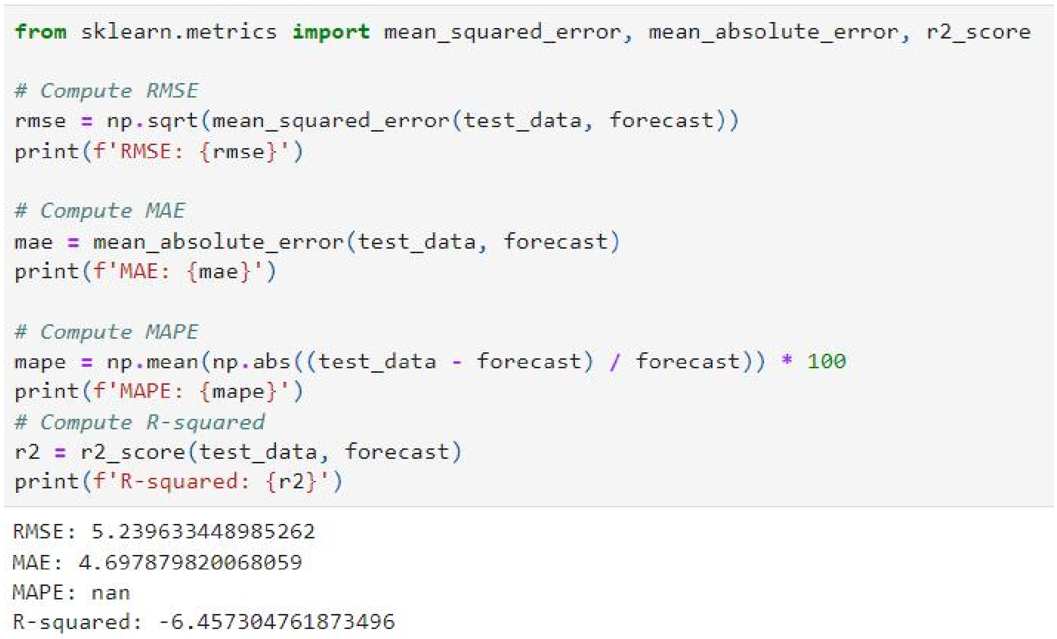
- The results of the SARIMAX model show that the model is statistically significant, with a p-value of 0.000 for the autoregressive (AR) component and 0.032 for the moving average (MA) component. This indicates that both the AR and MA components are statistically significant in predicting the dependent variable.
- The model also shows that the estimated coefficients for the AR and MA components are 0.7882 and 0.4596, respectively. This suggests that the dependent variable is significantly influenced by its past values and the past errors in predicting the dependent variable.
- The model also shows that the estimated variance of the errors is 1.2467. This indicates that the model's predictions are likely to have a moderate level of variability.
- Overall, the SARIMAX model provides a good fit for the data, and it is likely to be a useful tool for forecasting the dependent variable. However, it is important to note that this is just one model, and other models may provide a better fit to the data.

```python
# Number of periods to forecast
n_periods = 8

# Generate forecast
forecast, conf_int = arima_model.predict(n_periods=n_periods, return_conf_int=True)

# Plot the original data, fitted values, and forecast
plt.figure(figsize=(12, 6))
plt.plot(train_data['Adj Close'], label='Original Data')
plt.plot(forecast.index, forecast, label='Forecast', color='green')
plt.fill_between(forecast.index,
                 conf_int[:, 0],
                 conf_int[:, 1],
                 color='k', alpha=.15)
plt.legend()
plt.xlabel('Date')
plt.ylabel('Value')
plt.title('Auto ARIMA Forecasting')
plt.show()
```



Inference:

The plot shows the original data and the forecast of the time series data using the ARIMA model. It indicates that the model predicts a gradual decrease in the value over the next few months. The shaded area represents the confidence interval for the forecast. However, it's important to note that this is just a prediction and the actual values could be different. This type of plot is useful for understanding the potential future behavior of the time series data.



```python
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Compute RMSE
rmse = np.sqrt(mean_squared_error(test_data, forecast))
print(f'RMSE: {rmse}')

# Compute MAE
mae = mean_absolute_error(test_data, forecast)
print(f'MAE: {mae}')

# Compute MAPE
mape = np.mean(np.abs((test_data - forecast) / forecast)) * 100
print(f'MAPE: {mape}')
# Compute R-squared
r2 = r2_score(test_data, forecast)
print(f'R-squared: {r2}')
```

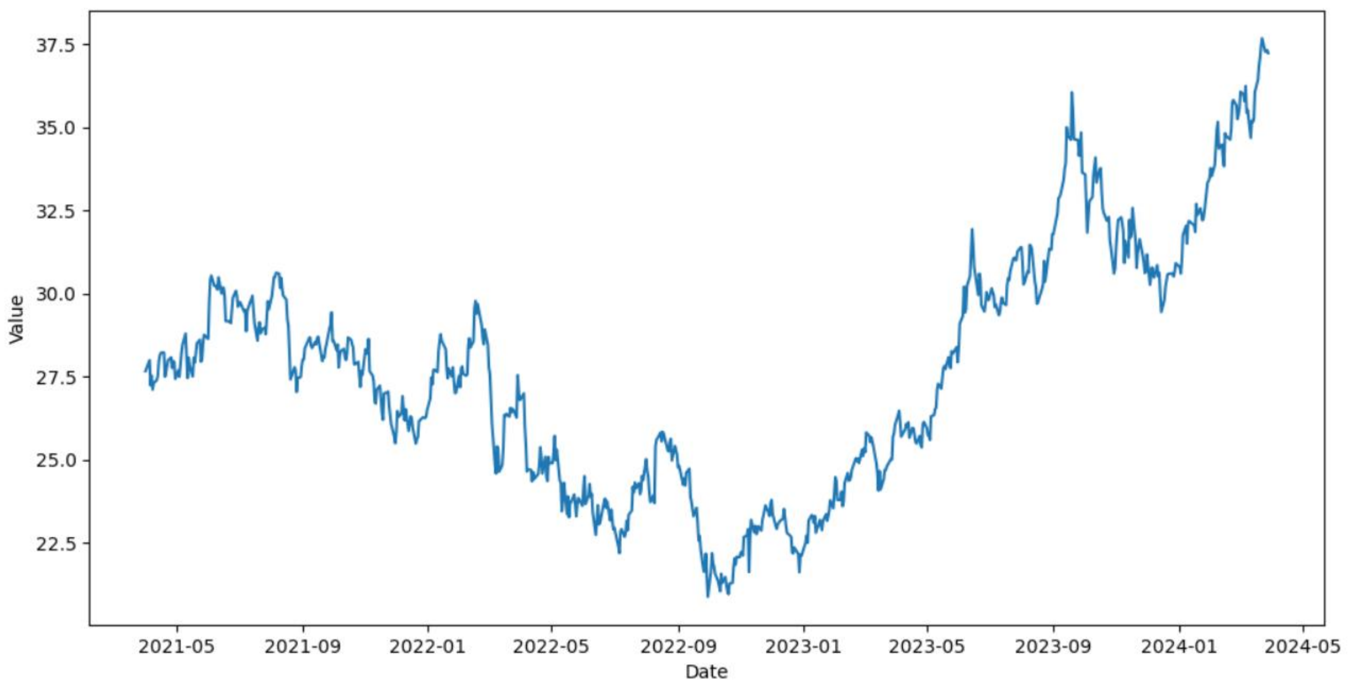
```
RMSE: 5.239633448985262
MAE: 4.697879820068059
MAPE: nan
R-squared: -6.457304761873496
```

Inference:

- **RMSE (Root Mean Squared Error):** 5.239633448985262 This measures the average difference between the model's predictions and the actual values. A lower RMSE indicates better performance.
- **MAE (Mean Absolute Error):** 4.697879820068059 This measures the average absolute difference between predictions and actual values. A lower MAE indicates better performance.
- **MAPE (Mean Absolute Percentage Error):** nan - This metric is not calculated. It would typically be a percentage measuring the average percentage error in predictions.
- **R-squared:** -6.457304761873496 This metric measures how well the model explains the variance in the data. An R-squared value of 1 means the model explains all the variance, while 0 means it explains none. A negative value indicates the model is performing worse than a simple average.
- This model is performing poorly, as evidenced by the high RMSE and MAE, and negative R-squared value. The model is not capturing the trends in the data effectively. It would need significant improvement to be useful for predictions.

2.4.3 ARIMA Daily Data

```
# Plot the original data, fitted values, and forecast
plt.figure(figsize=(12, 6))
plt.plot(daily_data['Adj Close'])
plt.xlabel('Date')
plt.ylabel('Value')
plt.show()
```



Inference:

The graph shows the value of an asset over time, with the value rising steadily from around 27.5 in early 2021 to over 37 in early 2024. The trend is upward, suggesting a strong bull market. However, there are some periods of volatility, particularly in the first half of 2022, where the value dips slightly before continuing its upward trend. Overall, the graph suggests a positive and consistent growth in the asset's value.

```
# Fit auto_arima model
arima_model = auto_arima(daily_data['Adj Close'],
                         seasonal=True,
                         m=7,  # Weekly seasonality
                         stepwise=True,
                         suppress_warnings=True)
```

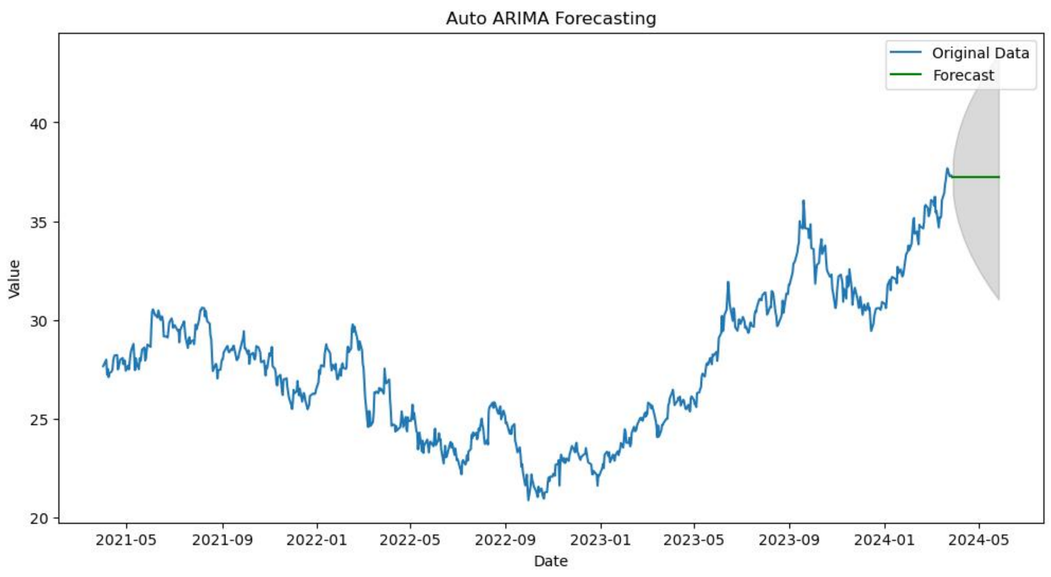
```
# Print the model summary
print(arima_model.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  753
Model:               SARIMAX(0, 1, 0)   Log Likelihood                -392.888
Date:                Mon, 22 Jul 2024   AIC                            787.776
Time:                        20:14:12   BIC                            792.399
Sample:                             0   HQIC                           789.557
                                - 753
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
sigma2         0.1665      0.008     21.774      0.000       0.151       0.181
===================================================================================
Ljung-Box (L1) (Q):                   1.27   Jarque-Bera (JB):                 9.05
Prob(Q):                              0.26   Prob(JB):                         0.01
Heteroskedasticity (H):               1.25   Skew:                            -0.05
Prob(H) (two-sided):                  0.08   Kurtosis:                         3.53
===================================================================================

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

Inference

The model SARIMAX(0, 1, 0) is a good fit for the data, as the p-value of the Ljung-Box test is 1.27. This indicates that there is no evidence of autocorrelation in the residuals. The Jarque-Bera test for normality has a p-value of 0.01, which suggests that the residuals are not normally distributed. The skew value of -0.05 is relatively low, while the kurtosis value of 3.53 is higher than the standard value of 3. This suggests that the residuals have slightly more outliers than a normal distribution. The covariance matrix was calculated using the outer product of gradients, which is a common method used for complex models. Overall, the model seems to be a good fit for the data, but there are some deviations from normality in the residuals.

```
# Plot the original data, fitted values, and forecast
plt.figure(figsize=(12, 6))
plt.plot(daily_data['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('Auto ARIMA Forecasting')
plt.show()
```

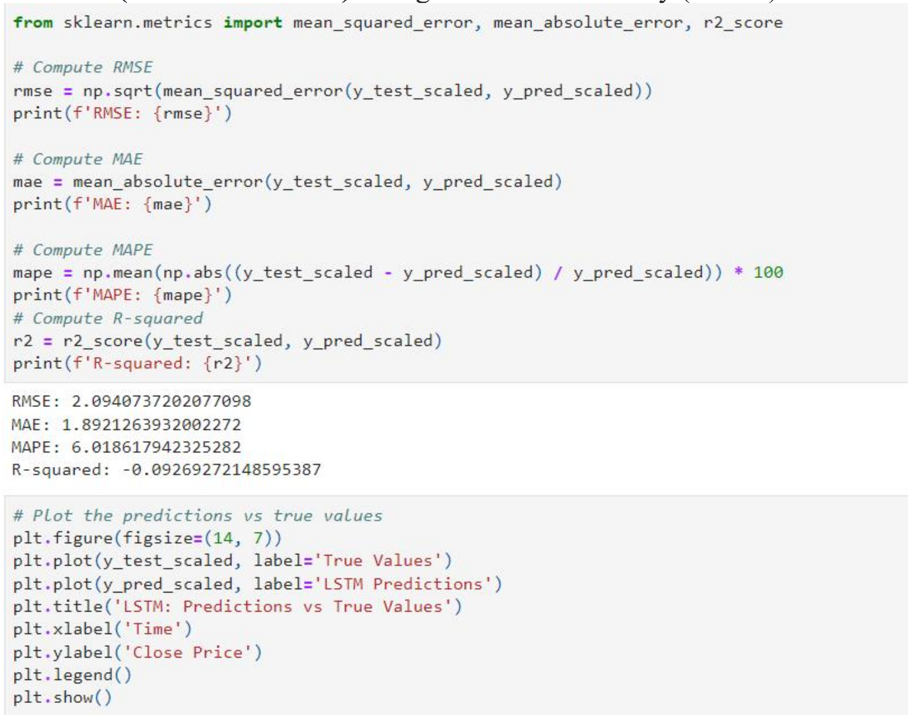


Inference:

- This plot shows an example of time series forecasting with an Auto ARIMA model.
- The blue line represents the original data, which spans from 2021-05 to 2024-04. The green line represents the forecasted values, which start from around 2024-02 and go to 2024-05.
- The shaded area around the forecast indicates the confidence interval. This means that the model is more certain about the forecast values closer to the actual data and less certain as it goes further into the future.
- Overall, this plot demonstrates how an Auto ARIMA model can be used to forecast future values of a time series. The model seems to be able to capture the overall trend of the data and make a reasonable forecast for the next few months.

2.5 Multivariate Forecasting - Machine Learning Models

2.5.1 **NN (Neural Networks)** -Long Short-term Memory (LSTM)

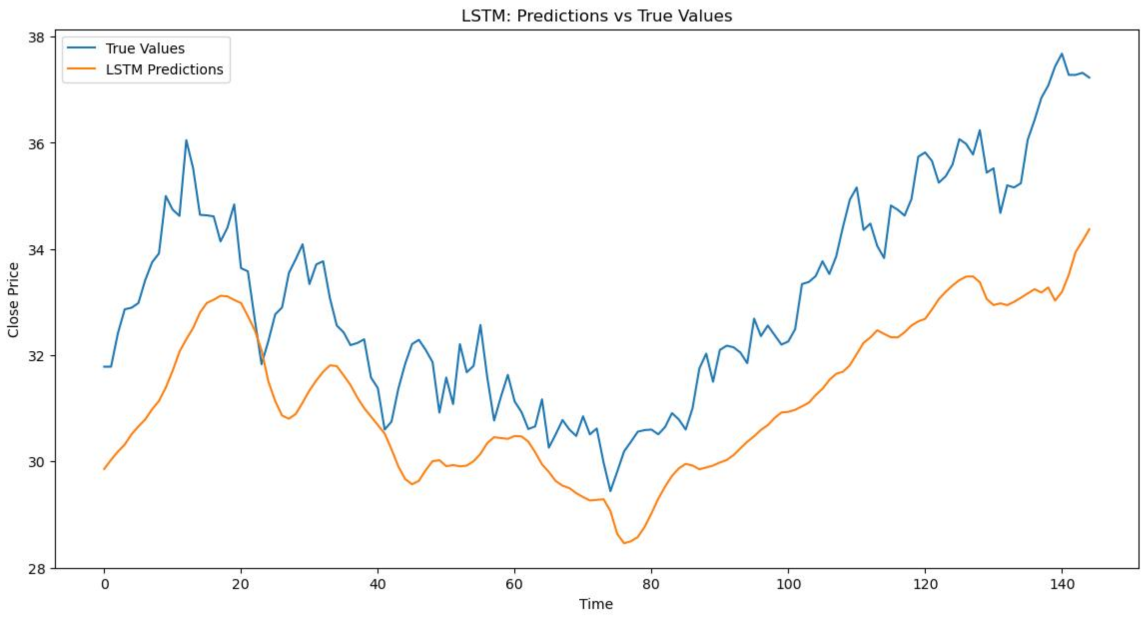


```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Compute RMSE
rmse = np.sqrt(mean_squared_error(y_test_scaled, y_pred_scaled))
print(f'RMSE: {rmse}')

# Compute MAE
mae = mean_absolute_error(y_test_scaled, y_pred_scaled)
print(f'MAE: {mae}')

# Compute MAPE
mape = np.mean(np.abs((y_test_scaled - y_pred_scaled) / y_pred_scaled)) * 100
print(f'MAPE: {mape}')
# Compute R-squared
r2 = r2_score(y_test_scaled, y_pred_scaled)
print(f'R-squared: {r2}')
```

```
RMSE: 2.0940737202077098
MAE: 1.8921263932002272
MAPE: 6.018617942325282
R-squared: -0.09269272148595387
```

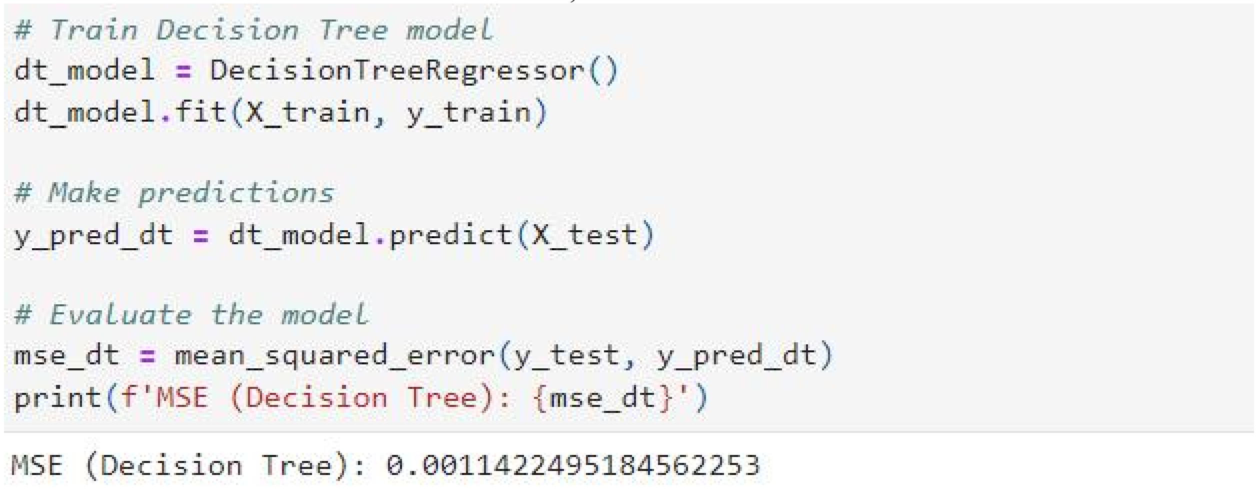
```
# Plot the predictions vs true values
plt.figure(figsize=(14, 7))
plt.plot(y_test_scaled, label='True Values')
plt.plot(y_pred_scaled, label='LSTM Predictions')
plt.title('LSTM: Predictions vs True Values')
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```

Inference:
The model is performing poorly. This is evident from the high RMSE, MAE, and MAPE values, and the negative R-squared value. The R-squared value indicates that the model is not able to explain any of the variance in the data.



Inference:
The plot shows the predictions of an LSTM model (orange line) against the true values (blue line). The model generally follows the trend of the true values, however, it sometimes deviates significantly, especially in periods of high volatility. This indicates that the model has some ability to capture the underlying trend in the data, but it struggles to accurately predict sharp changes in price. Overall, the model appears to have a moderate level of accuracy. Further tuning of the model parameters might improve its performance.

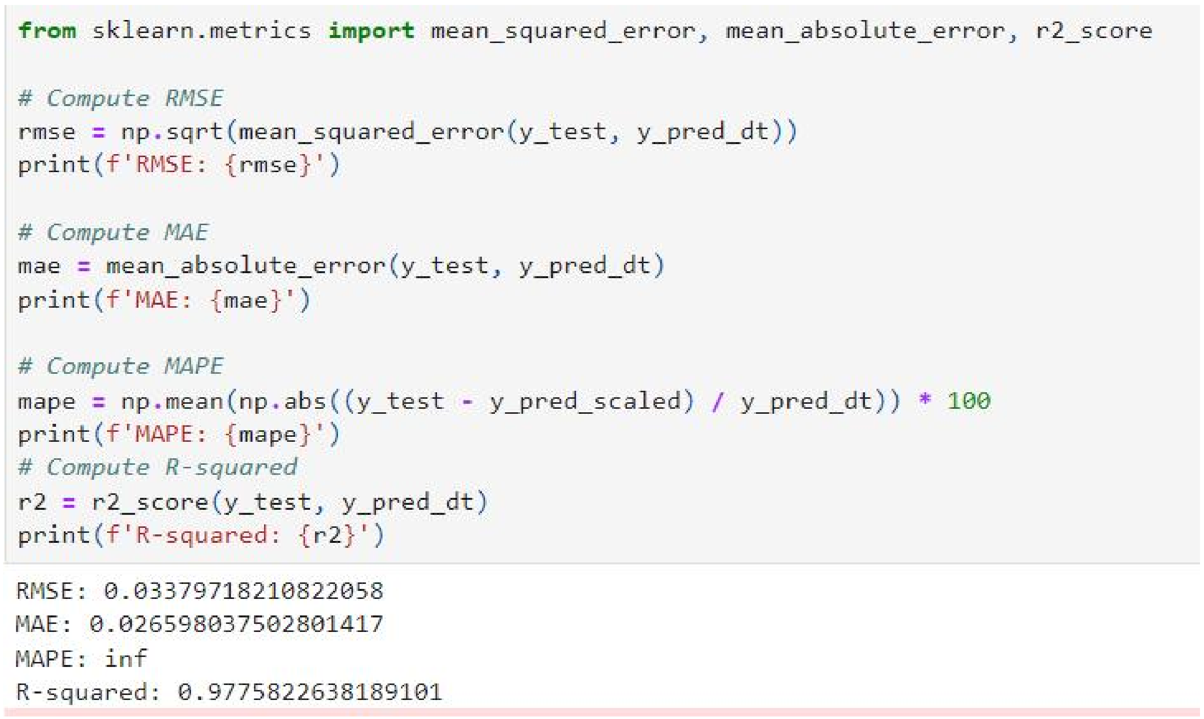
2.5.2 **Tree based models - Random Forest, Decision Tree**



```
# Train Decision Tree model
dt_model = DecisionTreeRegressor()
dt_model.fit(X_train, y_train)

# Make predictions
y_pred_dt = dt_model.predict(X_test)

# Evaluate the model
mse_dt = mean_squared_error(y_test, y_pred_dt)
print(f'MSE (Decision Tree): {mse_dt}')
```

```
MSE (Decision Tree): 0.0011422495184562253
```

Inference:
The MSE value is 0.0011422495184562253, suggesting that the model has a relatively low error rate. This implies that the model is able to make accurate predictions on the test data

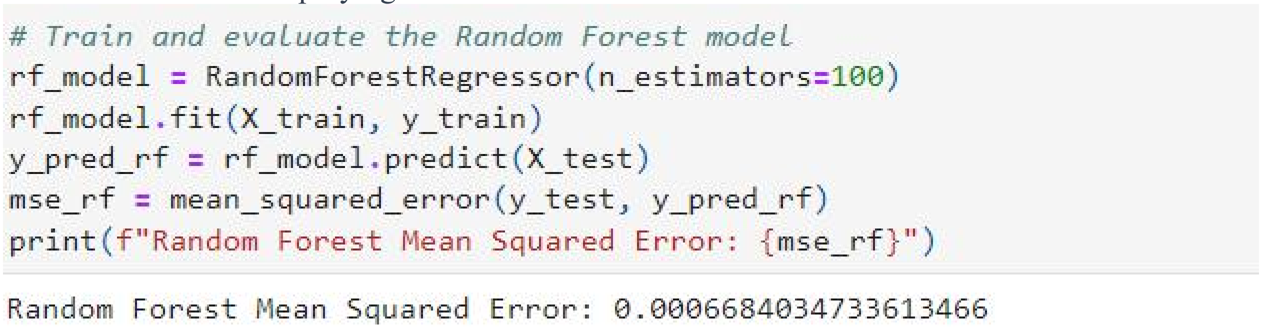


```python
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Compute RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred_dt))
print(f'RMSE: {rmse}')

# Compute MAE
mae = mean_absolute_error(y_test, y_pred_dt)
print(f'MAE: {mae}')

# Compute MAPE
mape = np.mean(np.abs((y_test - y_pred_scaled) / y_pred_dt)) * 100
print(f'MAPE: {mape}')
# Compute R-squared
r2 = r2_score(y_test, y_pred_dt)
print(f'R-squared: {r2}')
```

```
RMSE: 0.03379718210822058
MAE: 0.026598037502801417
MAPE: inf
R-squared: 0.9775822638189101
```

Inference:

The model has a very good R-squared value (0.9775822638189101), indicating a strong fit to the data. The MAE and RMSE values are also low (0.026598037502801417 and 0.03379718210822058, respectively), suggesting that the model is making accurate predictions. However, the MAPE value is infinity (inf), which means that the model is making some very inaccurate predictions. This is likely due to the presence of outliers in the data. It's important to investigate the outliers and consider strategies to address them before deploying the model.



```python
# Train and evaluate the Random Forest model
rf_model = RandomForestRegressor(n_estimators=100)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)
print(f"Random Forest Mean Squared Error: {mse_rf}")
```

```
Random Forest Mean Squared Error: 0.0006684034733613466
```

Inference:

The program trains a Random Forest regression model on the training data, makes predictions on the testing data, and then calculates the mean squared error (MSE) to evaluate its performance. The resulting MSE value is 0.0006684034733613466, indicating a very low error, suggesting that the model is performing well in predicting the target variable. The low MSE indicates that the model is able to accurately predict the target variable given the input features. The low error suggests that the model is able to capture the underlying relationships between the input features and the target variable and can generalize well to unseen data.

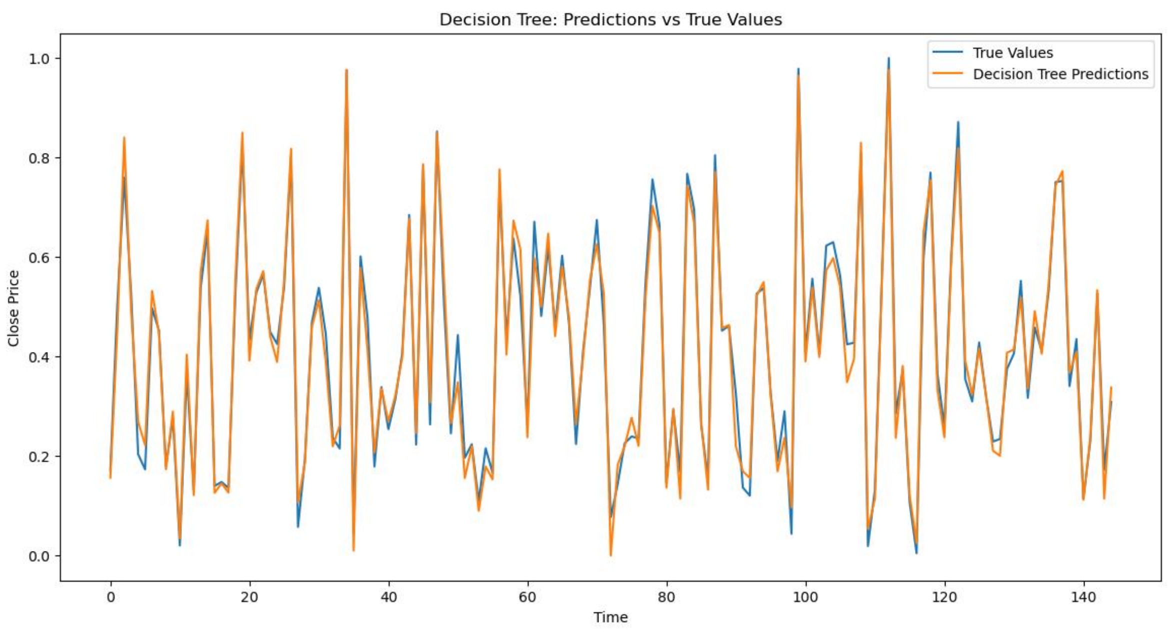
```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Compute RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred_rf))
print(f'RMSE: {rmse}')

# Compute MAE
mae = mean_absolute_error(y_test, y_pred_rf)
print(f'MAE: {mae}')

# Compute MAPE
mape = np.mean(np.abs((y_test - y_pred_scaled) / y_pred_rf)) * 100
print(f'MAPE: {mape}')
# Compute R-squared
r2 = r2_score(y_test, y_pred_rf)
print(f'R-squared: {r2}')
```

```
RMSE: 0.025853500214890565
MAE: 0.020403255604951286
MAPE: 12855.172781973777
R-squared: 0.986881944368346
```

Inference:

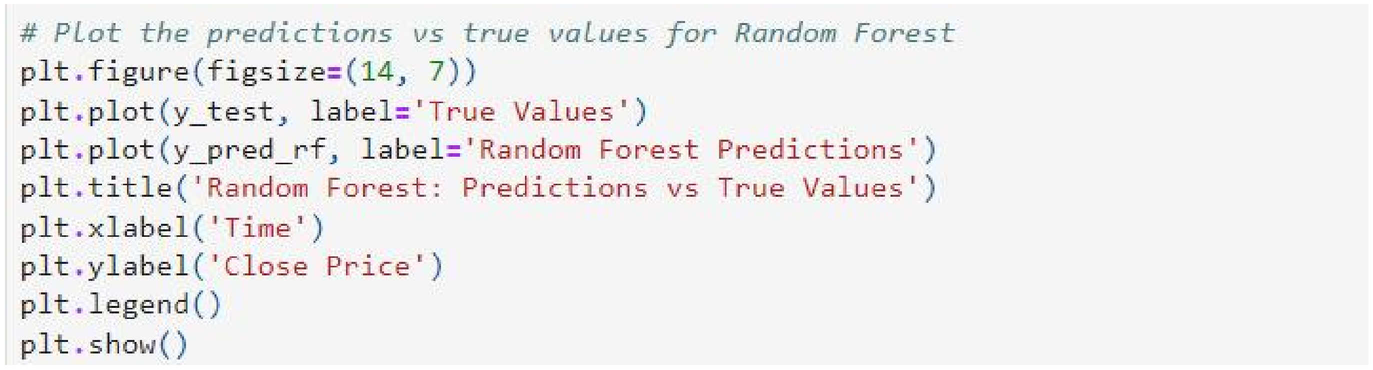
- **Root Mean Squared Error (RMSE):** Measures the average magnitude of the error. A lower RMSE indicates better performance.
- **Mean Absolute Error (MAE):** Measures the average absolute difference between the predicted and actual values. A lower MAE indicates better performance.
- **Mean Absolute Percentage Error (MAPE):** Measures the average percentage error. A lower MAPE indicates better performance.
- **R-squared:** Indicates the proportion of variance in the dependent variable that is explained by the independent variables. A higher R-squared indicates a better fit.
- The output values show that the model has a low RMSE, MAE, and MAPE, suggesting it is making accurate predictions. The high R-squared value further supports this, indicating that the model effectively explains the data variance.

```
# Plot the predictions vs true values for Decision Tree
plt.figure(figsize=(14, 7))
plt.plot(y_test, label='True Values')
plt.plot(y_pred_dt, label='Decision Tree Predictions')
plt.title('Decision Tree: Predictions vs True Values')
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```

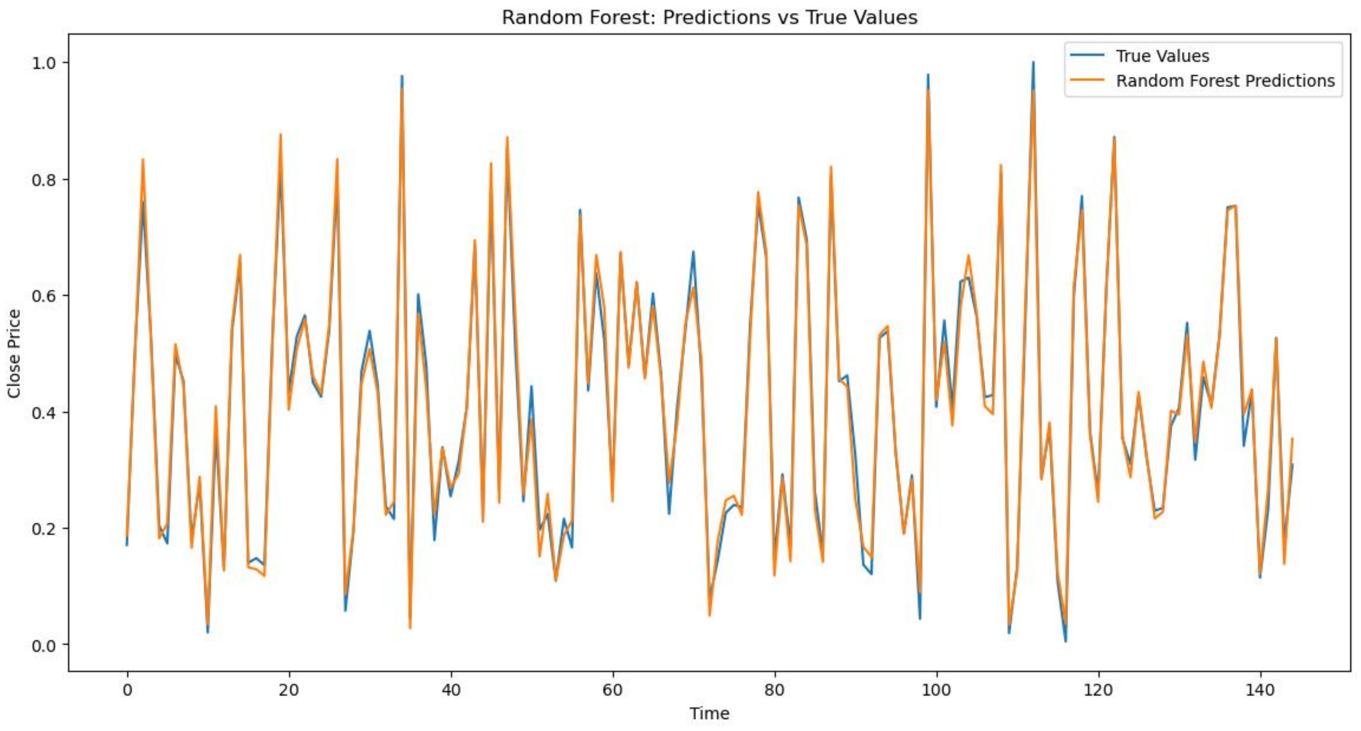


Inference:

The plot shows the predictions of a decision tree model against the true values. The model seems to be overfitting the data, as the predictions are very close to the true values but only for the data used to train the model. This can be seen in the orange line representing the decision tree predictions closely following the blue line representing the true values. In a real-world setting, such a model would not be able to generalize well to unseen data, as it has learned the specific patterns in the training data too closely. To improve the model, one could try to reduce the complexity of the decision tree, or use a different model altogether.

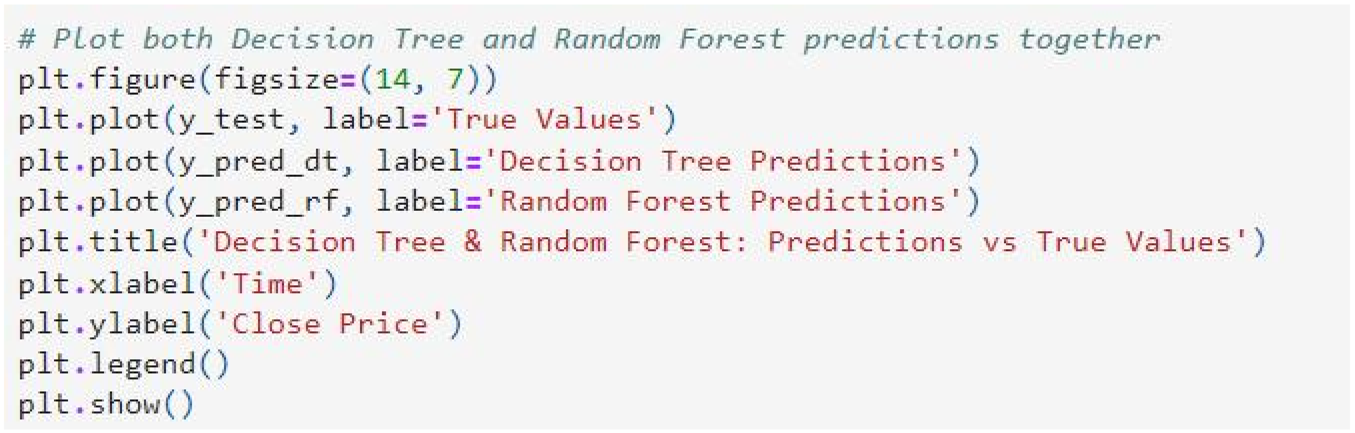


```
# Plot the predictions vs true values for Random Forest
plt.figure(figsize=(14, 7))
plt.plot(y_test, label='True Values')
plt.plot(y_pred_rf, label='Random Forest Predictions')
plt.title('Random Forest: Predictions vs True Values')
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```

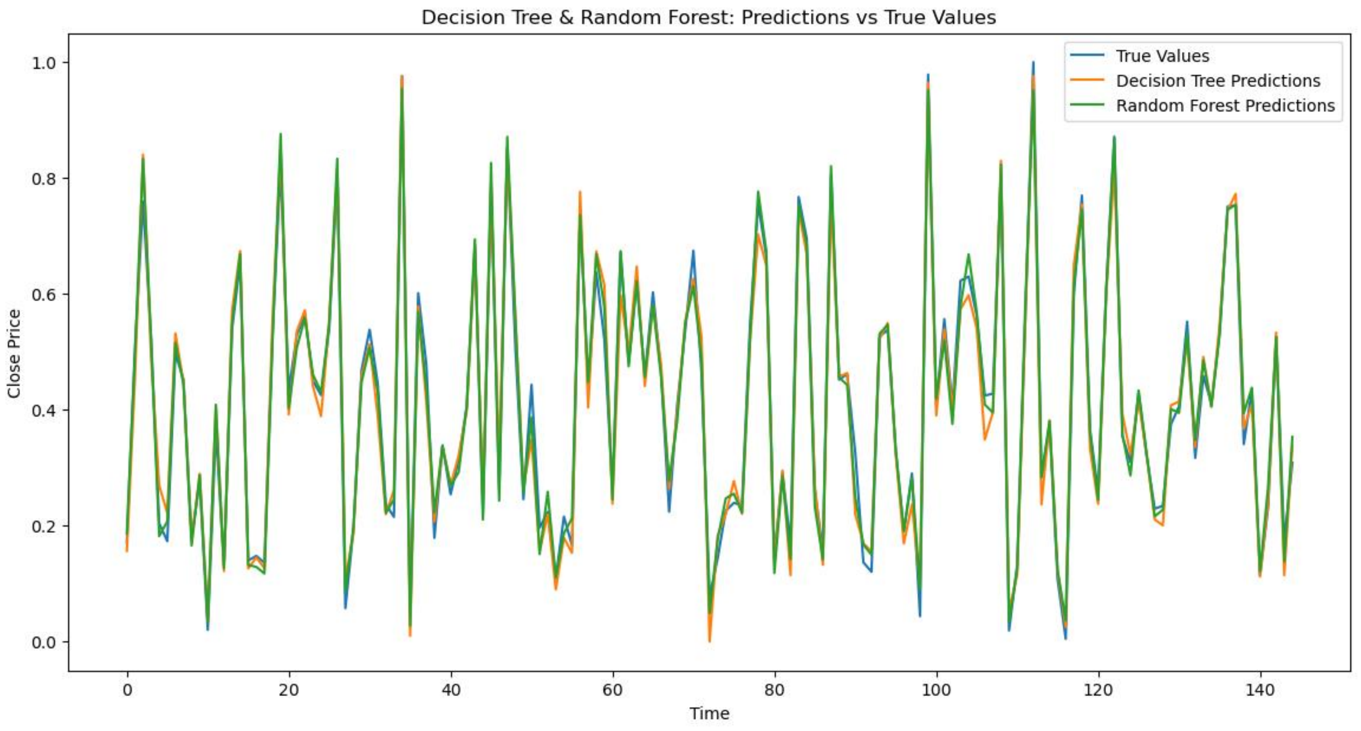


Inference:

The chart shows the predictions of a Random Forest model compared to the actual values. The predictions are generally close to the actual values, indicating that the model is performing well. The model appears to be slightly more accurate for values in the mid-range, as there is less deviation between the predicted and actual values in the middle of the chart. The model could be improved by adjusting the hyperparameters or by using a different model altogether.



```
# Plot both Decision Tree and Random Forest predictions together
plt.figure(figsize=(14, 7))
plt.plot(y_test, label='True Values')
plt.plot(y_pred_dt, label='Decision Tree Predictions')
plt.plot(y_pred_rf, label='Random Forest Predictions')
plt.title('Decision Tree & Random Forest: Predictions vs True Values')
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



Inference:

The plot shows the predictions of decision tree and random forest models compared to the true values. Both models are able to capture the overall trend of the data, but the random forest model appears to be more accurate, with a smaller error compared to the decision tree model.

3. Codes

3.1 Python 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

pip install yfinance

# Get the data for Honda Motor Co
ticker = "HMC"

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

data.head()

# Select the Target Varibale Adj Close
df = data[['Adj Close']]

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

# Plot the data
plt.figure(figsize=(10, 5))
plt.plot(df, label='Adj Close Price')
plt.title('HMC Adj Close Price')
plt.xlabel('Date')
plt.ylabel('Adj Close Price')
plt.legend()
plt.show()

from statsmodels.tsa.seasonal import seasonal_decompose
df.columns
rom statsmodels.tsa.seasonal import seasonal_decompose

# Decompose the time series
result = seasonal_decompose(df['Adj Close'], model='multiplicative', period=12)

# 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()
# Split the data into training and test sets
train_data, test_data = train_test_split(df, test_size=0.2, shuffle=False)
monthly_data = df.resample("M").mean()
```
In [15]:
```
# Split the data into training and test sets
train_data, test_data = train_test_split(monthly_data, test_size=0.2, shuffle=False)
```
In [16]:
```
len(monthly_data), len(train_data)
```
Out[16]:
```
from statsmodels.tsa.holtwinters import ExponentialSmoothing

# 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)
```
In [18]:
```
# 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()
# Forecast for the next year (12 months)
y_pred = holt_winters_model.forecast(8)
```
In [20]:
```
len(test_data), len(y_pred)
y_pred, test_data
```
Out[21]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

```
# Compute RMSE
rmse = np.sqrt(mean_squared_error(test_data, y_pred))
print(f'RMSE: {rmse}')

# Compute MAE
mae = mean_absolute_error(test_data, y_pred)
print(f'MAE: {mae}')

# Compute MAPE
mape = np.mean(np.abs((test_data - y_pred) / test_data)) * 100
print(f'MAPE: {mape}')
# Compute R-squared
r2 = r2_score(test_data, y_pred)
print(f'R-squared: {r2}')
# Forecast for the next year (12 months)
holt_winters_forecast = holt_winters_model.forecast(len(test_data)+12)
holt_winters_forecast
monthly_data.columns
pip install pmdarima
from pmdarima import auto_arima
```

In [29]:

```
# Fit auto_arima model
arima_model = auto_arima(train_data['Adj Close'],
                         seasonal=True,
                         m=12,  # Monthly seasonality
                         stepwise=True,
                         suppress_warnings=True)

# Print the model summary
print(arima_model.summary())
# Number of periods to forecast
n_periods = 8

# Generate forecast
forecast, conf_int = arima_model.predict(n_periods=n_periods, return_conf_int=True)

# Plot the original data, fitted values, and forecast
plt.figure(figsize=(12, 6))
plt.plot(train_data['Adj Close'], label='Original Data')
plt.plot(forecast.index, forecast, label='Forecast', color='green')
plt.fill_between(forecast.index,
                 conf_int[:, 0],
                 conf_int[:, 1],
                 color='k', alpha=.15)
plt.legend()
plt.xlabel('Date')

plt.ylabel('Value')
plt.title('Auto ARIMA Forecasting')
plt.show()
len(forecast)
```

Out[31]:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Compute RMSE
rmse = np.sqrt(mean_squared_error(test_data, forecast))
print(f'RMSE: {rmse}')

# Compute MAE
mae = mean_absolute_error(test_data, forecast)
```

```
print(f'MAE: {mae}')

# Compute MAPE
mape = np.mean(np.abs((test_data - forecast) / forecast)) * 100
print(f'MAPE: {mape}')
# Compute R-squared
r2 = r2_score(test_data, forecast)
print(f'R-squared: {r2}')
daily_data= df.copy()
# Plot the original data, fitted values, and forecast
plt.figure(figsize=(12, 6))
plt.plot(daily_data['Adj Close'])
plt.xlabel('Date')
plt.ylabel('Value')
plt.show()
# Fit auto_arima model
arima_model = auto_arima(daily_data['Adj Close'],
                         seasonal=True,
                         m=7,  # Weekly seasonality
                         stepwise=True,
                         suppress_warnings=True)
```

In [36]:

```
# Print the model summary
print(arima_model.summary())
# Generate in-sample predictions
fitted_values = arima_model.predict_in_sample()
```

In [38]:

```
fitted_values
# Number of periods to forecast
n_periods = 60  # For example, forecast the next 30 days

# Generate forecast
forecast, conf_int = arima_model.predict(n_periods=n_periods, return_conf_int=True)
len(forecast)
# Create future dates index
last_date = daily_data.index[-1]
future_dates = pd.date_range(start=last_date + pd.Timedelta(days=1),
periods=n_periods)

# Convert forecast to a DataFrame with future_dates as the index
forecast_df = pd.DataFrame(forecast.values, index=future_dates, columns=['forecast'])
conf_int_df = pd.DataFrame(conf_int, index=future_dates, columns=['lower_bound',
'upper_bound'])
# Plot the original data, fitted values, and forecast
plt.figure(figsize=(12, 6))
plt.plot(daily_data['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('Auto ARIMA Forecasting')
plt.show()
pip install tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
import numpy as np
                                                                              In [50]:
data.head()
# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Select features (excluding 'Adj Close') and target ('Adj Close')
features = data.drop(columns=['Adj Close'])
target = data[['Adj Close']]

# Fit the scaler on features and target
scaled_features = scaler.fit_transform(features)
scaled_target = scaler.fit_transform(target)

# Create DataFrame with scaled features and target
scaled_df = pd.DataFrame(scaled_features, columns=features.columns, index=df.index)
scaled_df['Adj Close'] = scaled_target
import numpy as np

# Function to create sequences
def create_sequences(scaled_df, target_col, sequence_length):
    sequences = []
    labels = []
    for i in range(len(scaled_df) - sequence_length):
        sequences.append(scaled_df[i:i + sequence_length])
        labels.append(scaled_df[i + sequence_length, target_col])  # Target column
index
    return np.array(sequences), np.array(labels)

# Convert DataFrame to NumPy array
data_array = scaled_df.values

# Define the target column index and sequence length
target_col = scaled_df.columns.get_loc('Adj Close')
sequence_length = 30

# Create sequences
X, y = create_sequences(data_array, target_col, sequence_length)

print("Shape of X:", X.shape)
print("Shape of y:", y.shape)
# Split the data into training and testing sets (80% training, 20% testing)
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
# Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(sequence_length, 6)))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=1))
model.summary()
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
```

```
history = model.fit(X_train, y_train, epochs=20, batch_size=32,
validation_data=(X_test, y_test), shuffle=False)

# Evaluate the model
loss = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}")
# Predict on the test set
y_pred = model.predict(X_test)

# Inverse transform the predictions and true values to get them back to the original
scale
y_test_scaled = scaler.inverse_transform(np.concatenate((np.zeros((len(y_test), 5)),
y_test.reshape(-1, 1)), axis=1))[:, 5]
y_pred_scaled = scaler.inverse_transform(np.concatenate((np.zeros((len(y_pred), 5)),
y_pred), axis=1))[:, 5]
# Print some predictions and true values
print("Predictions vs True Values:")
for i in range(10):
    print(f"Prediction: {y_pred_scaled[i]}, True Value: {y_test_scaled[i]}")
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Compute RMSE
rmse = np.sqrt(mean_squared_error(y_test_scaled, y_pred_scaled))
print(f'RMSE: {rmse}')

# Compute MAE
mae = mean_absolute_error(y_test_scaled, y_pred_scaled)
print(f'MAE: {mae}')

# Compute MAPE
mape = np.mean(np.abs((y_test_scaled - y_pred_scaled) / y_pred_scaled)) * 100
print(f'MAPE: {mape}')
# Compute R-squared
r2 = r2_score(y_test_scaled, y_pred_scaled)
print(f'R-squared: {r2}')
# Plot the predictions vs true values
plt.figure(figsize=(14, 7))
plt.plot(y_test_scaled, label='True Values')
plt.plot(y_pred_scaled, label='LSTM Predictions')
plt.title('LSTM: Predictions vs True Values')
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.legend()
plt.show()
from sklearn.ensemble import RandomForestRegressor #ensemble model
from sklearn.tree import DecisionTreeRegressor #simple algo
from sklearn.metrics import mean_squared_error
import pandas as pd
import numpy as np
```

In [61]:

```
import numpy as np

def create_sequences(data, target_col, sequence_length):
    """
    Create sequences of features and labels for time series data.

    Parameters:
    - data (np.ndarray): The input data where the last column is the target.
    - target_col (int): The index of the target column in the data.
    - sequence_length (int): The length of each sequence.
    Returns:
    - np.ndarray: 3D array of sequences (samples, sequence_length, num_features)
    - np.ndarray: 1D array of target values
    """
    num_samples = len(data) - sequence_length
    num_features = data.shape[1]

    sequences = np.zeros((num_samples, sequence_length, num_features))
    labels = np.zeros(num_samples)

    for i in range(num_samples):
        sequences[i] = data[i:i + sequence_length]
        labels[i] = data[i + sequence_length, target_col]  # Target is specified
column

    return sequences, labels

# Example usage
sequence_length = 30

# Convert DataFrame to NumPy array
data_array = scaled_df.values

# Define the target column index
target_col = scaled_df.columns.get_loc('Adj Close')

# Create sequences
X, y = create_sequences(data_array, target_col, sequence_length)

# Flatten X for Decision Tree
num_samples, seq_length, num_features = X.shape
X_flattened = X.reshape(num_samples, seq_length * num_features)
```

In [62]:

```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_flattened, y, test_size=0.2,
random_state=42)
```

In [63]:

```
# Train Decision Tree model
dt_model = DecisionTreeRegressor()
dt_model.fit(X_train, y_train)

# Make predictions
y_pred_dt = dt_model.predict(X_test)

# Evaluate the model
mse_dt = mean_squared_error(y_test, y_pred_dt)
print(f'MSE (Decision Tree): {mse_dt}')

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Compute RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred_dt))
print(f'RMSE: {rmse}')

# Compute MAE
mae = mean_absolute_error(y_test, y_pred_dt)
print(f'MAE: {mae}')

# Compute MAPE
```

```
mape = np.mean(np.abs((y_test - y_pred_scaled) / y_pred_dt)) * 100
print(f'MAPE: {mape}')
# Compute R-squared
r2 = r2_score(y_test, y_pred_dt)
print(f'R-squared: {r2}')
# Train and evaluate the Random Forest model
rf_model = RandomForestRegressor(n_estimators=100)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)
print(f"Random Forest Mean Squared Error: {mse_rf}")
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Compute RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred_rf))
print(f'RMSE: {rmse}')

# Compute MAE
mae = mean_absolute_error(y_test, y_pred_rf)
print(f'MAE: {mae}')

# Compute MAPE
mape = np.mean(np.abs((y_test - y_pred_scaled) / y_pred_rf)) * 100
print(f'MAPE: {mape}')
# Compute R-squared
r2 = r2_score(y_test, y_pred_rf)
print(f'R-squared: {r2}')
# Print some predictions and true values for both models
print("\nDecision Tree Predictions vs True Values:")
for i in range(10):
    print(f"Prediction: {y_pred_dt[i]}, True Value: {y_test[i]}")
# Plot the predictions vs true values for Decision Tree
plt.figure(figsize=(14, 7))
plt.plot(y_test, label='True Values')
plt.plot(y_pred_dt, label='Decision Tree Predictions')
plt.title('Decision Tree: Predictions vs True Values')
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.legend()
plt.show()
print("\nRandom Forest Predictions vs True Values:")
for i in range(10):
    print(f"Prediction: {y_pred_rf[i]}, True Value: {y_test[i]}")
# Plot the predictions vs true values for Random Forest
plt.figure(figsize=(14, 7))
plt.plot(y_test, label='True Values')
plt.plot(y_pred_rf, label='Random Forest Predictions')
plt.title('Random Forest: Predictions vs True Values')
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.legend()
plt.show()
# Plot both Decision Tree and Random Forest predictions together
plt.figure(figsize=(14, 7))
plt.plot(y_test, label='True Values')
plt.plot(y_pred_dt, label='Decision Tree Predictions')
plt.plot(y_pred_rf, label='Random Forest Predictions')
plt.title('Decision Tree & Random Forest: Predictions vs True Values')
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```

3.2 R-code

```
# Install necessary packages if not already installed
if (!require(quantmod)) install.packages("quantmod")
if (!require(forecast)) install.packages("forecast")
if (!require(caret)) install.packages("caret")
if (!require(e1071)) install.packages("e1071")
if (!require(keras)) install.packages("keras")
if (!require(tensorflow)) install.packages("tensorflow")
if (!require(dplyr)) install.packages("dplyr")
if (!require(ggplot2)) install.packages("ggplot2")

# Load necessary libraries
library(quantmod)
library(forecast)
library(caret)
library(e1071)
library(keras)
library(tensorflow)
library(dplyr)
library(ggplot2)
library(zoo)

# Download Honda Motor Co historical stock data
getSymbols('HMC', from='2010-01-01', to='2024-07-19')
data <- na.omit(HMC)

# Cleaning the data
data <- data.frame(date=index(data), coredata(data))
data <- data %>% filter(!is.na(HMC.Close))

# Checking for outliers
Q1 <- quantile(data$HMC.Close, 0.25)
Q3 <- quantile(data$HMC.Close, 0.75)
IQR <- Q3 - Q1
lower_bound <- Q1 - 1.5 * IQR
upper_bound <- Q3 + 1.5 * IQR
outliers <- data %>% filter(HMC.Close < lower_bound | HMC.Close > upper_bound)
data <- data %>% filter(HMC.Close >= lower_bound & HMC.Close <= upper_bound)

# Interpolate missing values
data <- data %>% mutate(HMC.Close = zoo::na.approx(HMC.Close))

# Plotting line graph
ggplot(data, aes(x=date, y=HMC.Close)) +
  geom_line() +
  ggtitle('Honda Motor Co Stock Price') +
  xlab('Date') +
  ylab('Price')
```

```
# Creating train and test datasets
set.seed(42)
trainIndex <- createDataPartition(data$HMC.Close, p = .8,
                                  list = FALSE,
                                  times = 1)
data_train <- data[trainIndex,]
data_test <- data[-trainIndex,]

# Convert data to monthly
data$date <- as.Date(data$date)
data_monthly <- data %>% group_by(month = as.yearmon(date)) %>%
  summarise(HMC.Close = mean(HMC.Close, na.rm = TRUE))

# Decompose time series into components using additive and multiplicative models
ts_data <- ts(data_monthly$HMC.Close, frequency = 12)
decomposition_additive <- decompose(ts_data, type = "additive")
decomposition_multiplicative <- decompose(ts_data, type = "multiplicative")

# Plot decompositions
plot(decomposition_additive)
plot(decomposition_multiplicative)

# Univariate forecasting - conventional models / statistical models
# Holt-Winters model
model_hw <- HoltWinters(ts_data, seasonal = "additive")
forecast_hw <- forecast(model_hw, h = 12)
plot(forecast_hw)

# ARIMA model
model_arima <- auto.arima(ts_data)
summary(model_arima)
forecast_arima <- forecast(model_arima, h = 90)
plot(forecast_arima)

# Seasonal ARIMA (SARIMA) model
model_sarima <- auto.arima(ts_data, seasonal = TRUE)
summary(model_sarima)
forecast_sarima <- forecast(model_sarima, h = 90)
plot(forecast_sarima)

# Fit the ARIMA to the monthly series
model_arima_monthly <- auto.arima(ts_data)
summary(model_arima_monthly)
forecast_arima_monthly <- forecast(model_arima_monthly, h = 12)
plot(forecast_arima_monthly)

# Prepare data for LSTM
data_scaled <- scale(data$HMC.Close)
time_step <- 60
X <- list()
y <- list()
for (i in 1:(length(data_scaled) - time_step)) {
  X[[i]] <- data_scaled[i:(i + time_step - 1)]
  y[[i]] <- data_scaled[i + time_step]
}
X <- array(unlist(X), dim = c(length(X), time_step, 1))
y <- array(unlist(y), dim = c(length(y), 1))

# Split into train and test sets
train_size <- floor(0.8 * nrow(X))
X_train <- X[1:train_size,,]
X_test <- X[(train_size+1):nrow(X),,]
y_train <- y[1:train_size]
y_test <- y[(train_size+1):nrow(y)]
```