VIRGINIA COMMONWEALTH UNIVERSITY

STATISTICAL ANALYSIS & MODELING

A6a: Time Series Analysis

Kirthan Shaker Iyangar
V01108265

Date of Submission: 22/07/2024

CONTENTS

Time Series Analysis
(Python & R)

INTRODUCTION

This study conducts a time series analysis on the assigned dataset, aiming to uncover patterns, trends, and seasonal variations within the data. Time series analysis is a powerful statistical technique used to analyze data points collected or recorded at specific time intervals. In this analysis, we will preprocess the dataset, handle missing values, and ensure the assumptions of time series models are met.

We will explore the temporal relationships within the data, identifying significant trends, seasonal patterns, and potential cyclic behaviors. Various time series models, such as ARIMA (AutoRegressive Integrated Moving Average), exponential smoothing, and seasonal decomposition, will be utilized to analyze and forecast future values. Additionally, we will perform a comparative analysis of the models' performance using metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE).

By systematically analyzing the data using these methods, we aim to provide comprehensive insights that can guide decision-making processes. The findings will highlight the strengths and weaknesses of each time series model and contribute to a better understanding of the data's underlying temporal structure.

OBJECTIVES

A) Clean the data, check for outliers and missing values, interpolate the data if there are any missing values, and plot a line graph of the data neatly named. Create a test and train data set out of this data.

B) Convert the data to monthly and decompose time series into the components using additive and multiplicative models.

1. Univariate Forecasting - Conventional Models/Statistical Models
1.1 Fit a Holt Winters model to the data and forecast for the next year.
1.2 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.

1.3 Fit the ARIMA to the monthly series.

2. Multivariate Forecasting - Machine Learning Models
2.1 NN (Neural Networks) -Long Short-term Memory (LSTM)
2.2 Tree based models - Random Forest, Decision Tree

BUSINESS SIGNIFICANCE

Time series analysis holds substantial business significance across various industries. By leveraging historical data, businesses can gain critical insights into trends, seasonal patterns, and potential cyclic behaviors, which are essential for strategic planning and decision-making. Time series models enable companies to make data-driven forecasts, allowing them to anticipate future demands, optimize inventory levels, and improve resource allocation.

In finance, time series analysis is pivotal for stock price prediction, risk management, and economic forecasting. Retailers can use it to predict sales trends, manage supply chains efficiently, and tailor marketing strategies to align with consumer buying patterns. In manufacturing, time series analysis helps in predicting equipment maintenance needs, thus minimizing downtime and enhancing productivity.

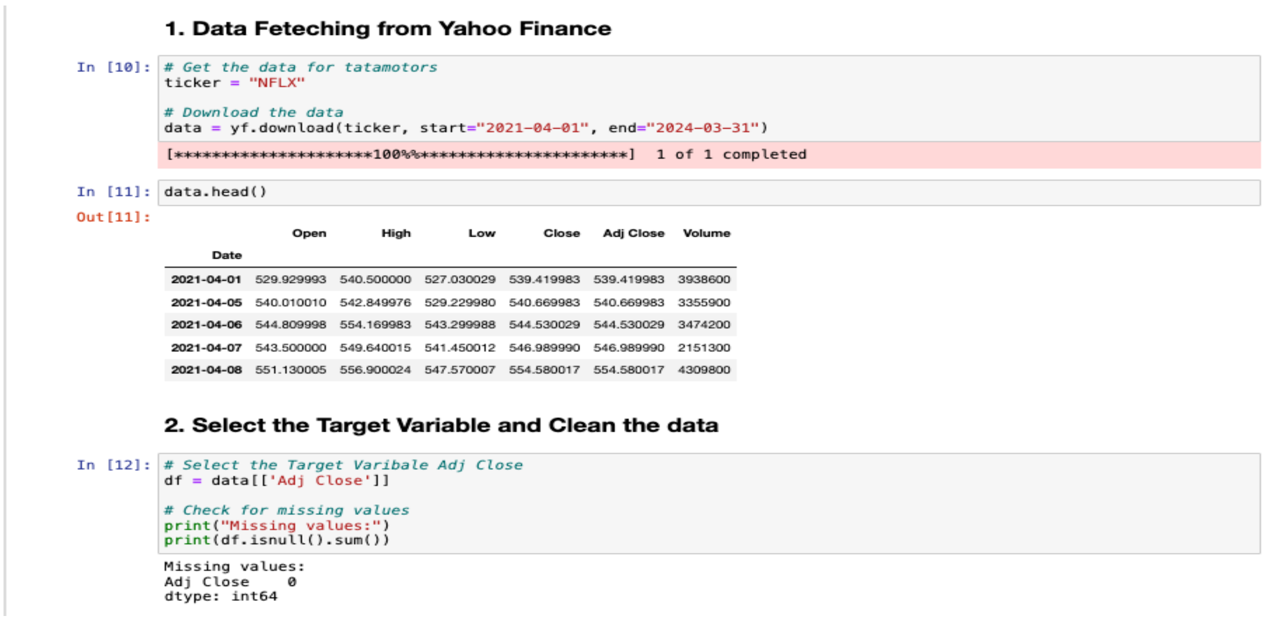
Moreover, time series analysis aids in detecting anomalies and identifying underlying factors affecting business performance. By understanding these temporal dynamics, organizations can develop proactive strategies to mitigate risks and capitalize on emerging opportunities. Overall, time series analysis is an invaluable tool that supports businesses in achieving sustainable growth and maintaining a competitive edge in their respective markets.

RESULTS AND INTERPRETATION

A) Clean the data, check for outliers and missing values, interpolate the data if there are any missing values, and plot a line graph of the data neatly named. Create a test and train data set out of this data.

#Time Series Analysis

Code and Result:



1. Data Feteching from Yahoo Finance

```
In [10]: # Get the data for tatamotors
         ticker = "NFLX"

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

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

```
In [11]: data.head()
```

Out[11]:

Date	Open	High	Low	Close	Adj Close	Volume
2021-04-01	529.929993	540.500000	527.030029	539.419983	539.419983	3938600
2021-04-05	540.010010	542.849976	529.229980	540.669983	540.669983	3355900
2021-04-06	544.809998	554.169983	543.299988	544.530029	544.530029	3474200
2021-04-07	543.500000	549.640015	541.450012	546.989990	546.989990	2151300
2021-04-08	551.130005	556.900024	547.570007	554.580017	554.580017	4309800

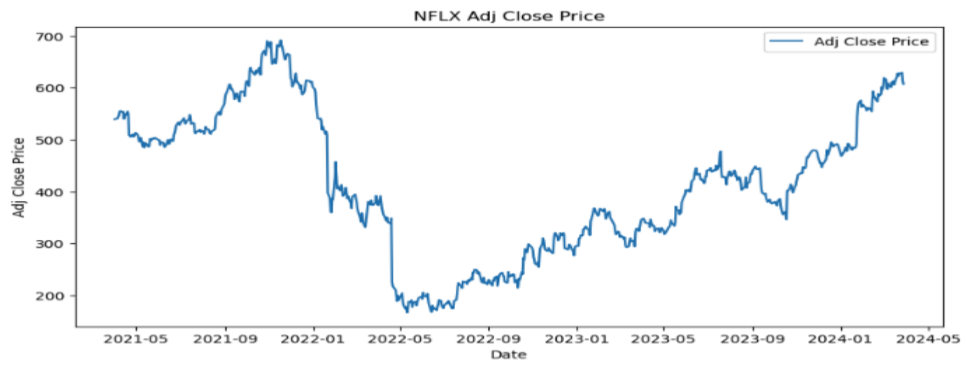
2. Select the Target Variable and Clean the data

```
In [12]: # 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
```

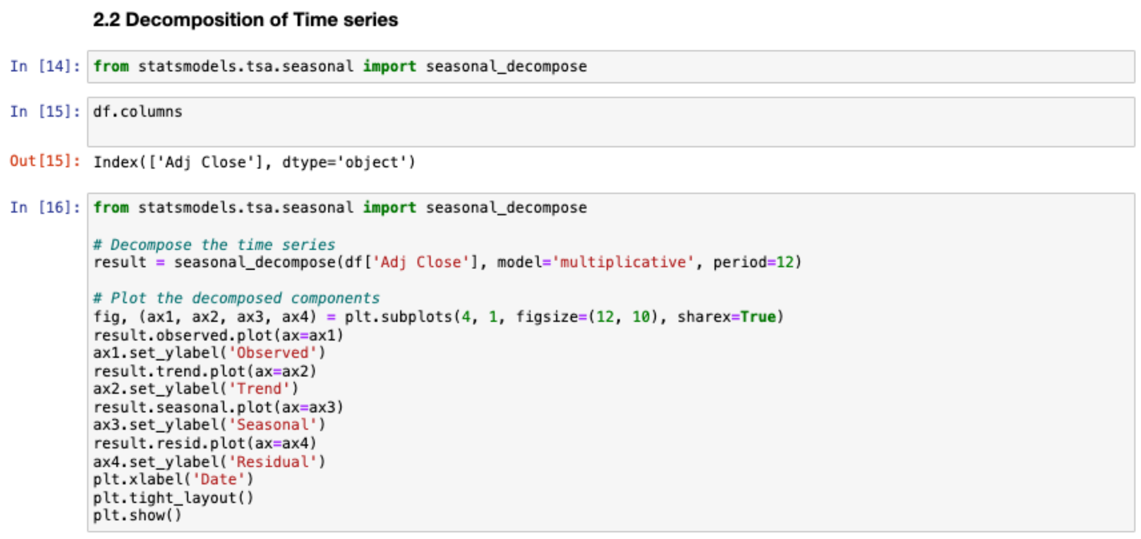
2.1 Plot the time series

In [13]:
```
# Plot the data
plt.figure(figsize=(10, 5))
plt.plot(df, label='Adj Close Price')
plt.title('NFLX Adj Close Price')
plt.xlabel('Date')
plt.ylabel('Adj Close Price')
plt.legend()
plt.show()
```



B) Convert the data to monthly and decompose time series into the components using additive and multiplicative models.

Code & Result:



2.2 Decomposition of Time series

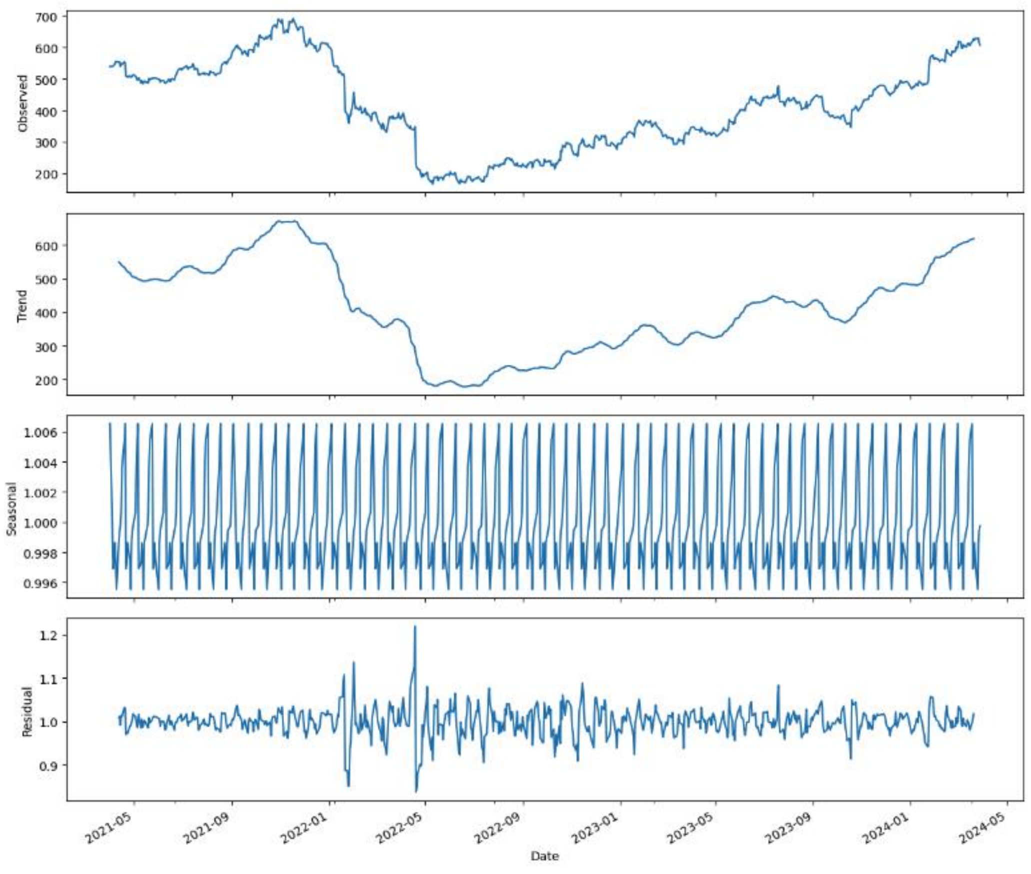
```
In [14]: from statsmodels.tsa.seasonal import seasonal_decompose
```

```
In [15]: df.columns
```

```
Out[15]: Index(['Adj Close'], dtype='object')
```

```
In [16]: 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()
```

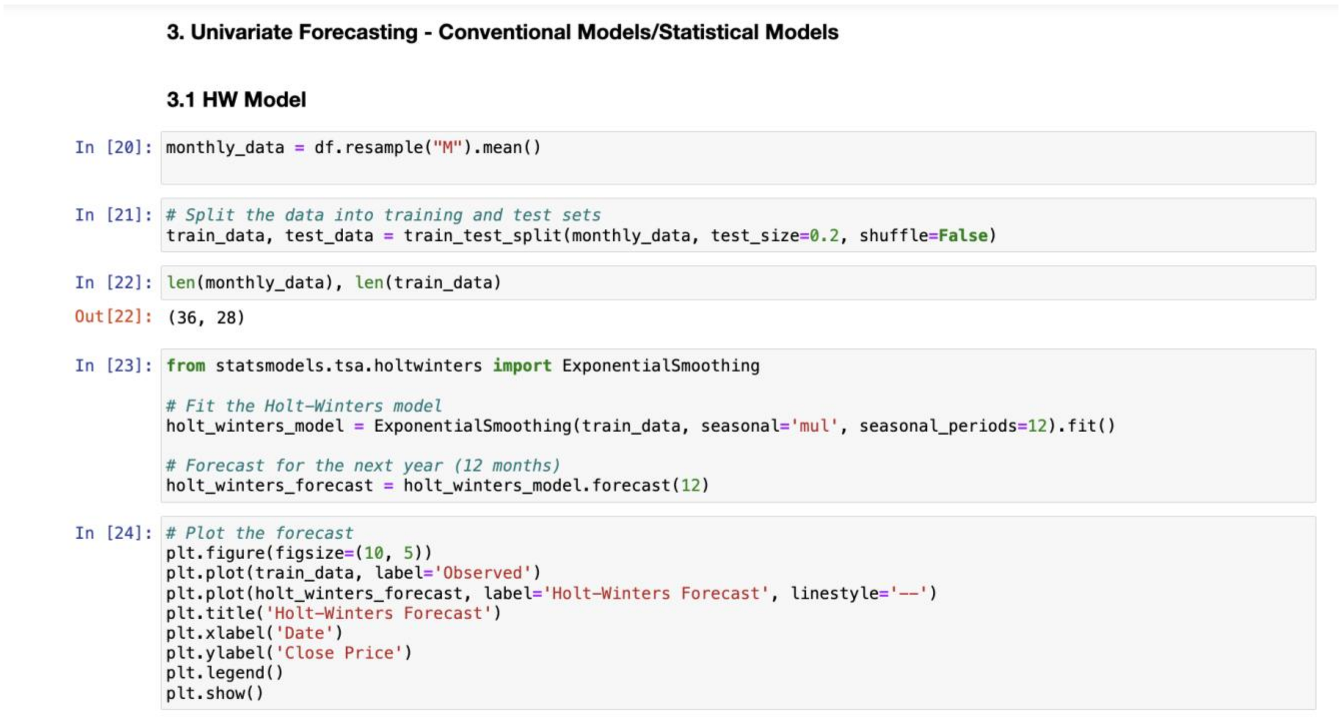
Result:



1. Univariate Forecasting - Conventional Models/Statistical Models

1.1 Fit a Holt Winters model to the data and forecast for the next year.
1.2 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.
1.3 Fit the ARIMA to the monthly series.

Code and Result



3. Univariate Forecasting - Conventional Models/Statistical Models

3.1 HW Model

```
In [20]: monthly_data = df.resample("M").mean()
```

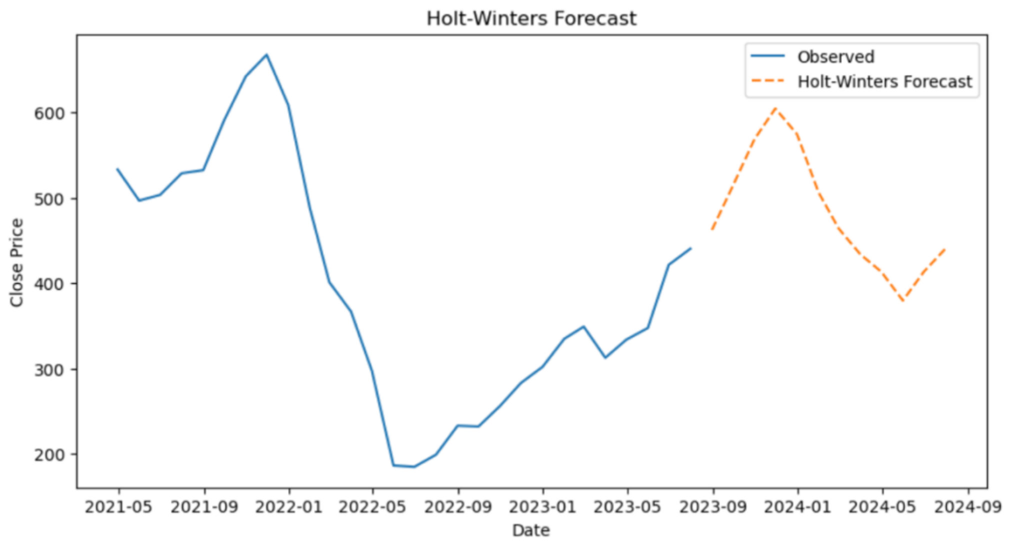
```
In [21]: # Split the data into training and test sets
         train_data, test_data = train_test_split(monthly_data, test_size=0.2, shuffle=False)
```

```
In [22]: len(monthly_data), len(train_data)
Out[22]: (36, 28)
```

```
In [23]: 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 [24]: # 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()
```

Result:

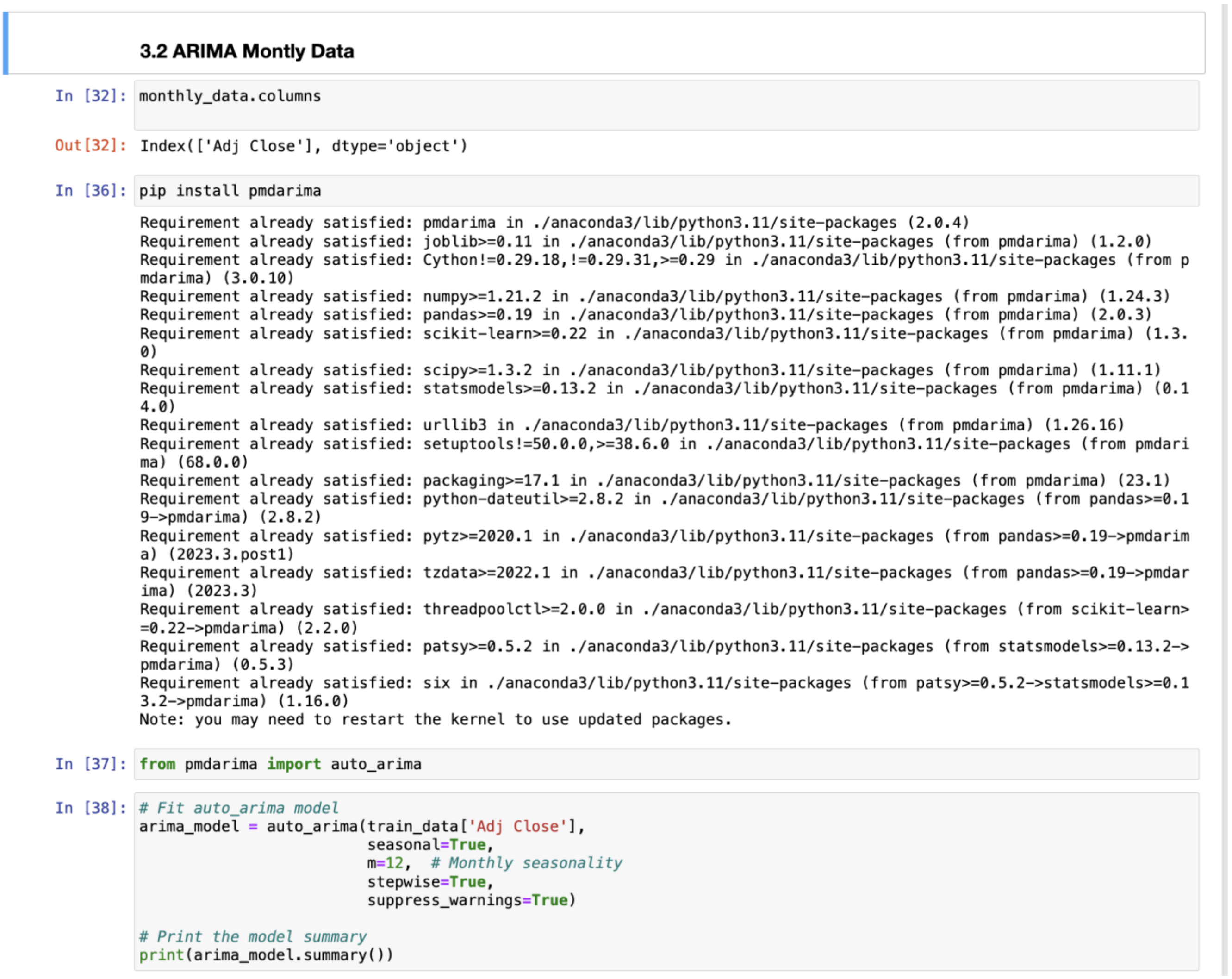


Interpretation:

1. **Observed Data**:
   - The blue line represents the actual historical data. From the graph, you can observe a peak around mid-2021 followed by a sharp decline and a subsequent recovery phase.
2. **Forecasted Data**:
   - The orange dashed line represents the forecasted values for the next 12 months. According to the forecast, the model predicts a further decline initially, followed by a recovery towards the latter part of the forecast period.

Analysis:

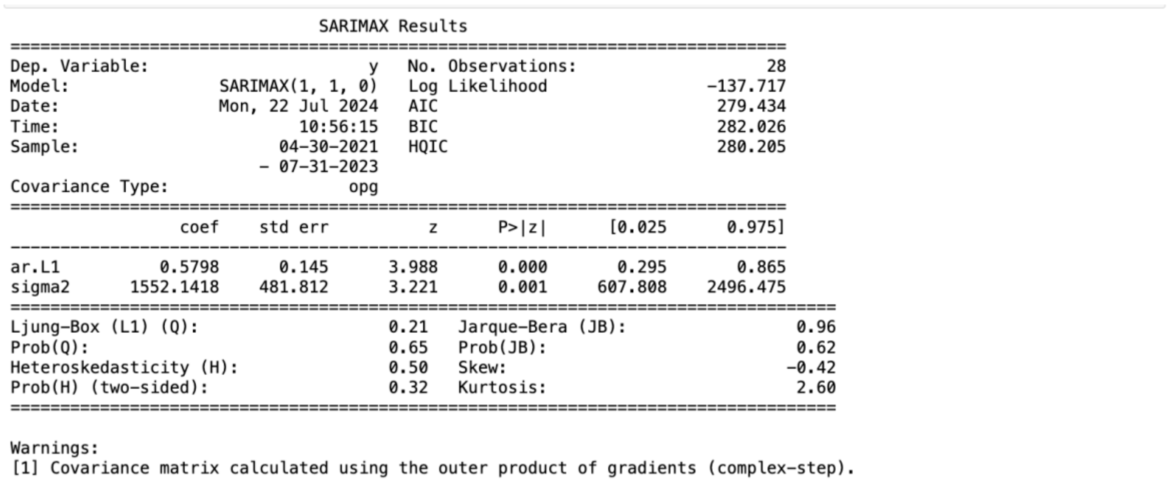
- **Trend**: The model captures the overall downward trend in the data after the peak in mid-2021, followed by some recovery. The forecast shows a continuation of this pattern with an initial decline and later an uptrend.
- **Seasonality**: The use of a multiplicative seasonal component indicates that the seasonal variations are proportionate to the level of the time series. This is reflected in the forecast where the seasonal effects are prominent.
- **Accuracy**: To assess the model's accuracy, you should compare the forecasted values with the actual values (test data). Common metrics include Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE).



3.2 ARIMA Montly Data

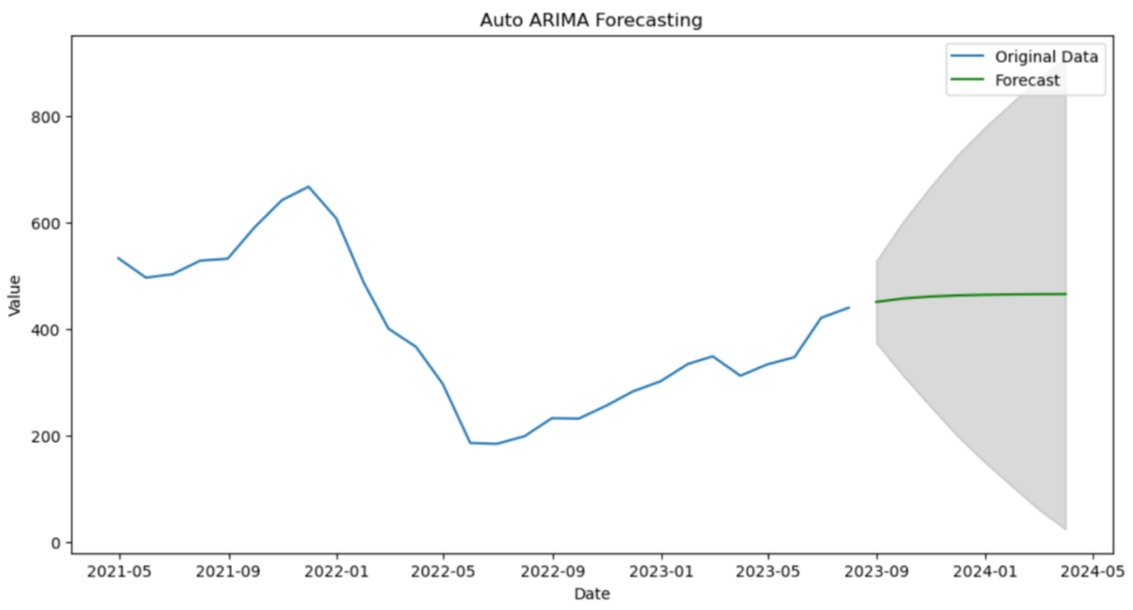
```
In [32]: monthly_data.columns

Out[32]: Index(['Adj Close'], dtype='object')

In [36]: pip install pmdarima

         Requirement already satisfied: pmdarima in ./anaconda3/lib/python3.11/site-packages (2.0.4)
         Requirement already satisfied: joblib>=0.11 in ./anaconda3/lib/python3.11/site-packages (from pmdarima) (1.2.0)
         Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in ./anaconda3/lib/python3.11/site-packages (from p
         mdarima) (3.0.10)
         Requirement already satisfied: numpy>=1.21.2 in ./anaconda3/lib/python3.11/site-packages (from pmdarima) (1.24.3)
         Requirement already satisfied: pandas>=0.19 in ./anaconda3/lib/python3.11/site-packages (from pmdarima) (2.0.3)
         Requirement already satisfied: scikit-learn>=0.22 in ./anaconda3/lib/python3.11/site-packages (from pmdarima) (1.3.
         0)
         Requirement already satisfied: scipy>=1.3.2 in ./anaconda3/lib/python3.11/site-packages (from pmdarima) (1.11.1)
         Requirement already satisfied: statsmodels>=0.13.2 in ./anaconda3/lib/python3.11/site-packages (from pmdarima) (0.1
         4.0)
         Requirement already satisfied: urllib3 in ./anaconda3/lib/python3.11/site-packages (from pmdarima) (1.26.16)
         Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in ./anaconda3/lib/python3.11/site-packages (from pmdari
         ma) (68.0.0)
         Requirement already satisfied: packaging>=17.1 in ./anaconda3/lib/python3.11/site-packages (from pmdarima) (23.1)
         Requirement already satisfied: python-dateutil>=2.8.2 in ./anaconda3/lib/python3.11/site-packages (from pandas>=0.1
         9->pmdarima) (2.8.2)
         Requirement already satisfied: pytz>=2020.1 in ./anaconda3/lib/python3.11/site-packages (from pandas>=0.19->pmdarim
         a) (2023.3.post1)
         Requirement already satisfied: tzdata>=2022.1 in ./anaconda3/lib/python3.11/site-packages (from pandas>=0.19->pmdar
         ima) (2023.3)
         Requirement already satisfied: threadpoolctl>=2.0.0 in ./anaconda3/lib/python3.11/site-packages (from scikit-learn>
         =0.22->pmdarima) (2.2.0)
         Requirement already satisfied: patsy>=0.5.2 in ./anaconda3/lib/python3.11/site-packages (from statsmodels>=0.13.2->
         pmdarima) (0.5.3)
         Requirement already satisfied: six in ./anaconda3/lib/python3.11/site-packages (from patsy>=0.5.2->statsmodels>=0.1
         3.2->pmdarima) (1.16.0)
         Note: you may need to restart the kernel to use updated packages.

In [37]: from pmdarima import auto_arima

In [38]: # 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())
```

Result :



```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   28
Model:               SARIMAX(1, 1, 0)   Log Likelihood                -137.717
Date:                Mon, 22 Jul 2024   AIC                            279.434
Time:                        10:56:15   BIC                            282.026
Sample:                    04-30-2021   HQIC                           280.205
                         - 07-31-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.5798      0.145      3.988      0.000       0.295       0.865
sigma2      1552.1418    481.812      3.221      0.001     607.808    2496.475
===================================================================================
Ljung-Box (L1) (Q):                   0.21   Jarque-Bera (JB):                 0.96
Prob(Q):                              0.65   Prob(JB):                         0.62
Heteroskedasticity (H):               0.50   Skew:                            -0.42
Prob(H) (two-sided):                  0.32   Kurtosis:                         2.60
===================================================================================

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

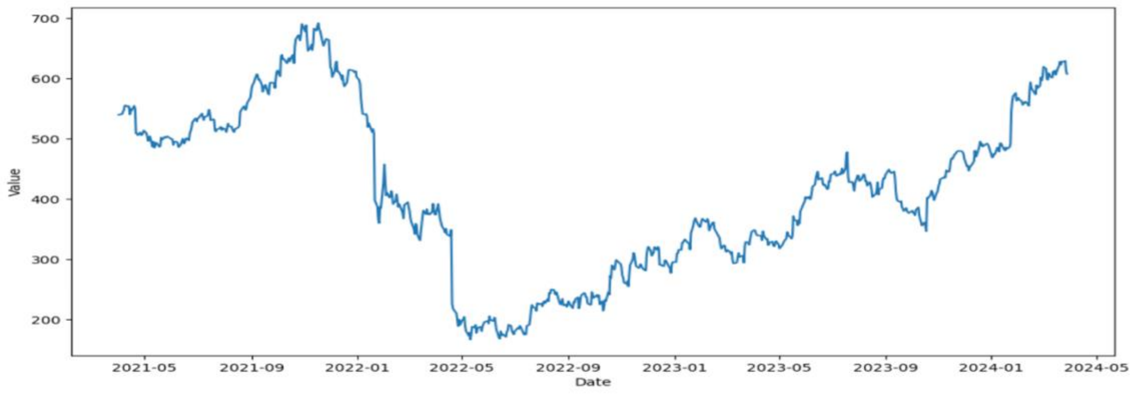
```
In [39]: # Number of periods to forecast
         n_periods = 8

         # 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()
```



3.3 ARIMA Daily Data

```
In [42]: daily_data= df.copy()
```

```
In [43]: # 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()
```



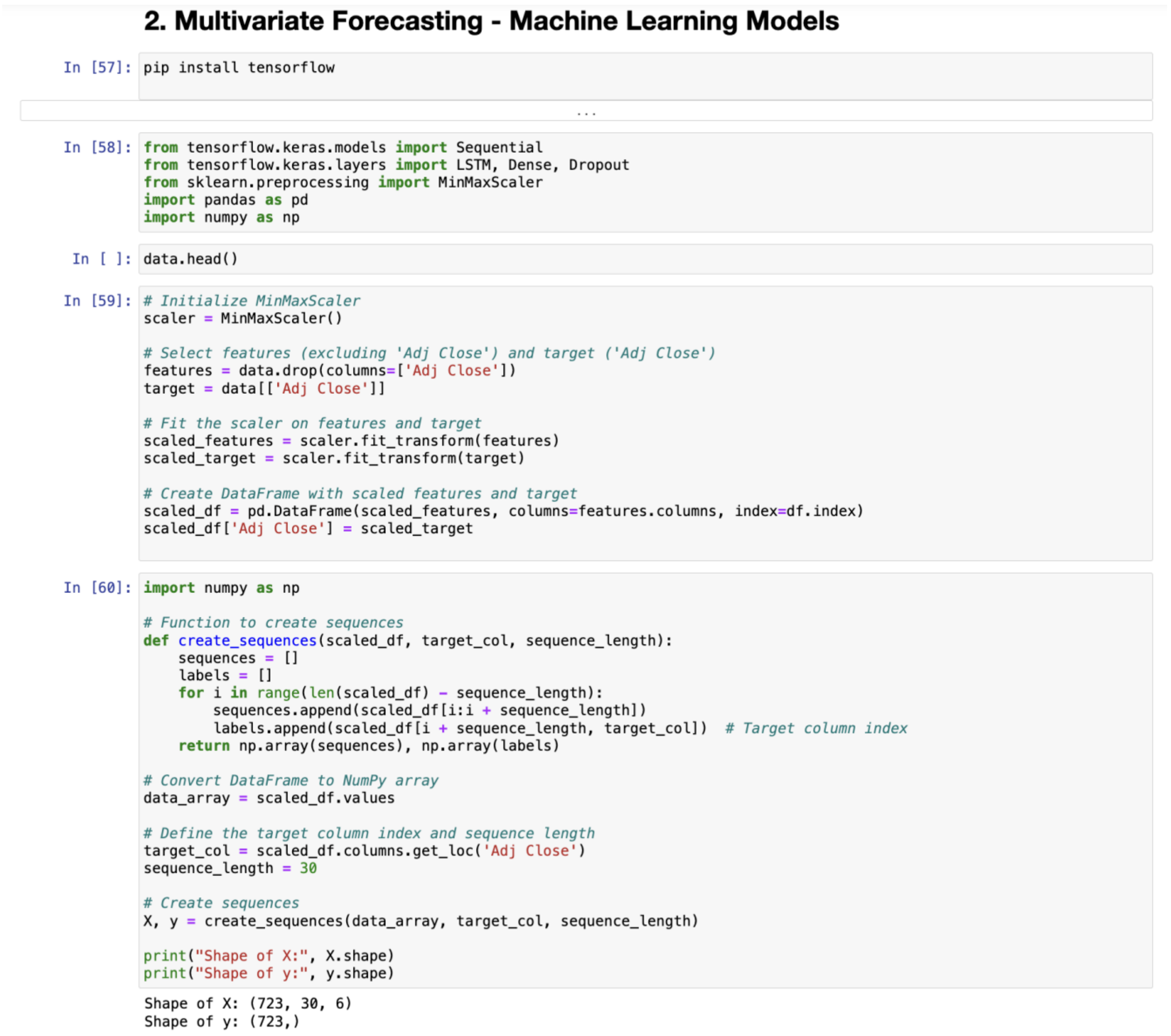
```
In [44]: # Fit auto_arima model
         arima_model = auto_arima(daily_data['Adj Close'],
                                  seasonal=True,
                                  m=7,   # Weekly seasonality
                                  stepwise=True,
                                  suppress_warnings=True)
```

2. Multivariate Forecasting - Machine Learning Models

2.1 NN (Neural Networks) -Long Short-term Memory (LSTM)

2.2 Tree based models - Random Forest, Decision Tree

Code and Result:



2. Multivariate Forecasting - Machine Learning Models

```
In [57]: pip install tensorflow
```

```
...
```

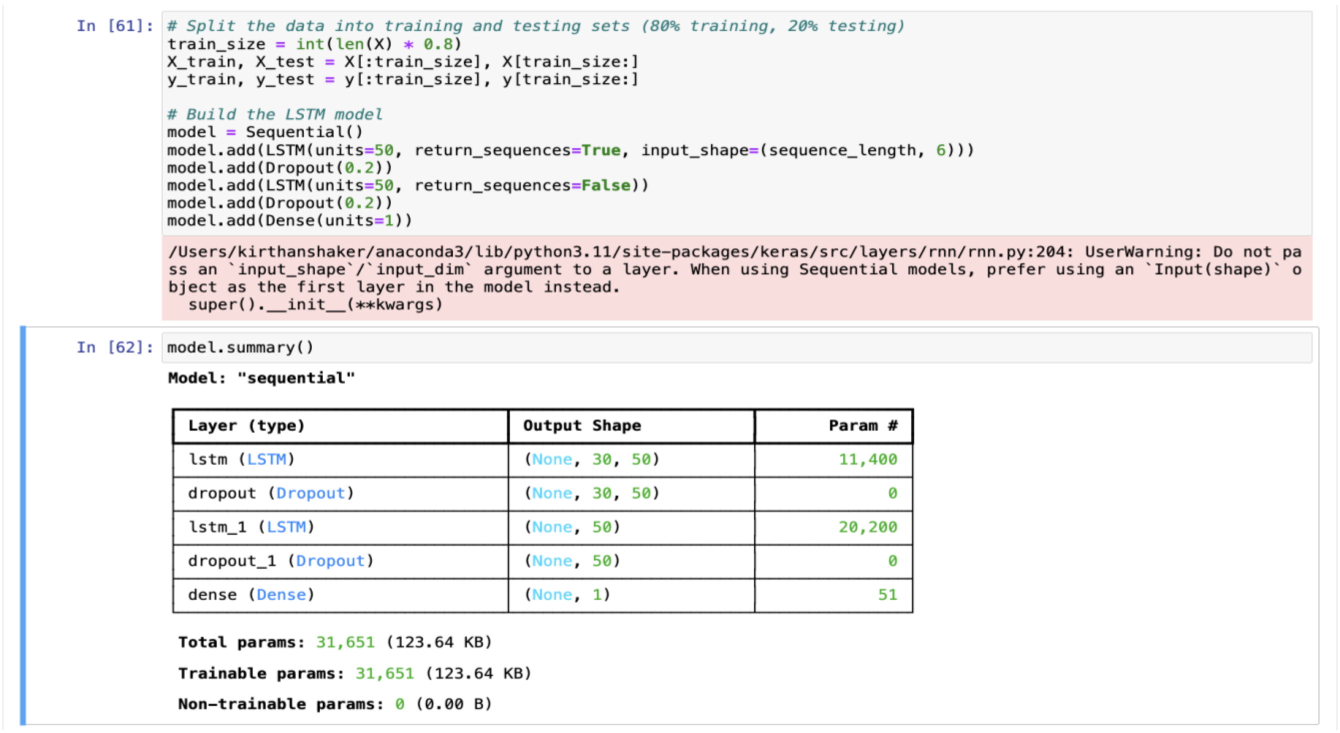
```
In [58]: from tensorflow.keras.models import Sequential
         from tensorflow.keras.layers import LSTM, Dense, Dropout
         from sklearn.preprocessing import MinMaxScaler
         import pandas as pd
         import numpy as np
```

```
In [ ]: data.head()
```

```
In [59]: # 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
```

```
In [60]: 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)
```

```
Shape of X: (723, 30, 6)
Shape of y: (723,)
```



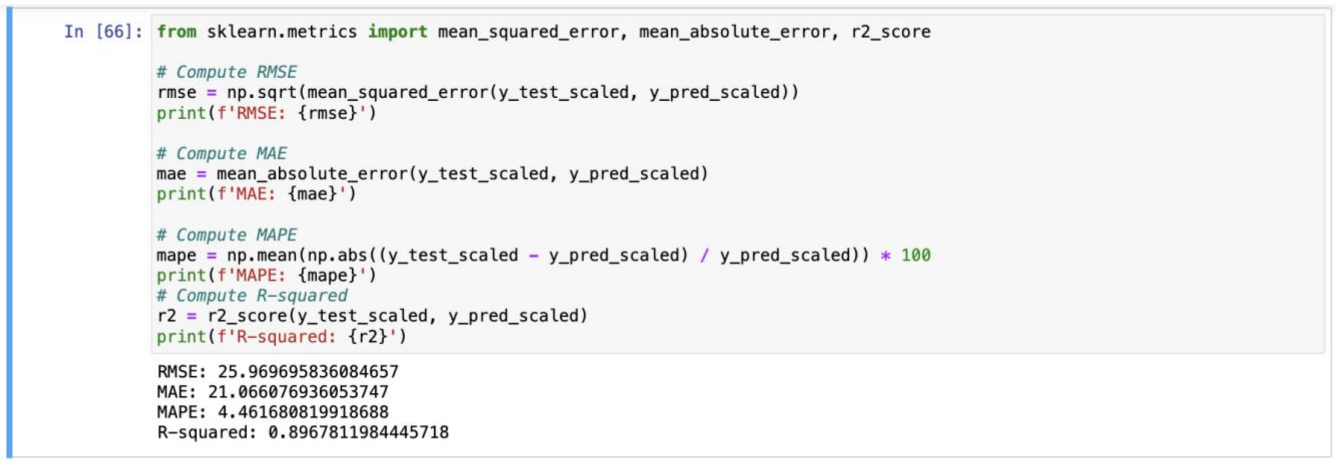
```
In [61]: # Split the data into training and testing sets (80% training, 20% testing)
         train_size = int(len(X) * 0.8)
         X_train, X_test = X[:train_size], X[train_size:]
         y_train, y_test = y[:train_size], y[train_size:]

         # 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))
```

```
/Users/kirthanshaker/anaconda3/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pa
ss an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` o
bject as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
In [62]: model.summary()
```

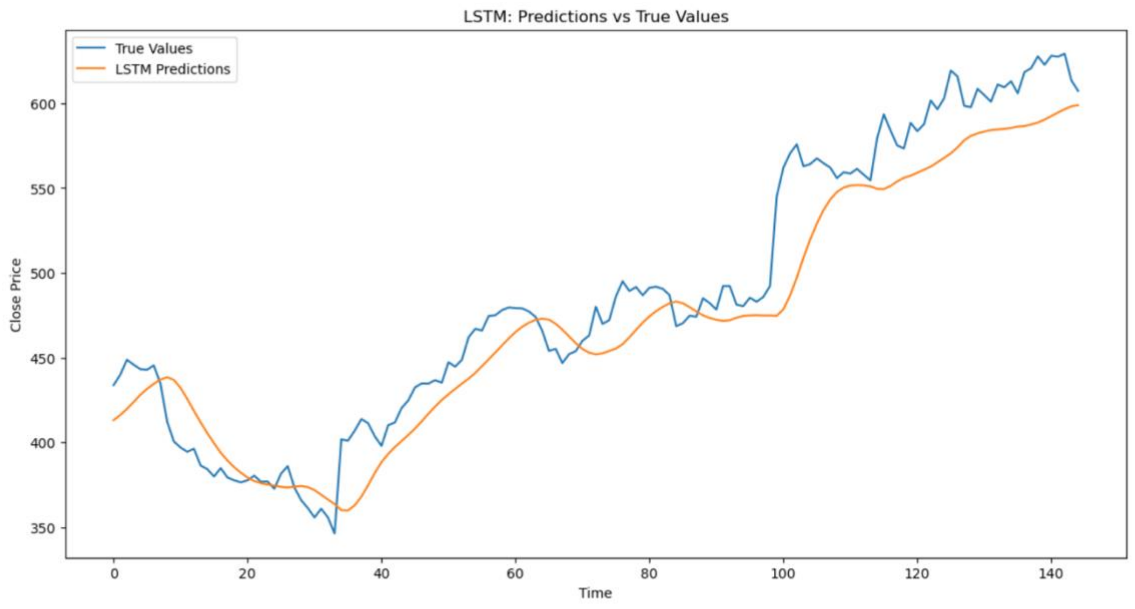
```
Model: "sequential"

┌──────────────────────────────────┬─────────────────────────┬────────────┐
│ Layer (type)                     │ Output Shape            │    Param # │
├──────────────────────────────────┼─────────────────────────┼────────────┤
│ lstm (LSTM)                      │ (None, 30, 50)          │     11,400 │
├──────────────────────────────────┼─────────────────────────┼────────────┤
│ dropout (Dropout)                │ (None, 30, 50)          │          0 │
├──────────────────────────────────┼─────────────────────────┼────────────┤
│ lstm_1 (LSTM)                    │ (None, 50)              │     20,200 │
├──────────────────────────────────┼─────────────────────────┼────────────┤
│ dropout_1 (Dropout)              │ (None, 50)              │          0 │
├──────────────────────────────────┼─────────────────────────┼────────────┤
│ dense (Dense)                    │ (None, 1)               │         51 │
└──────────────────────────────────┴─────────────────────────┴────────────┘

 Total params: 31,651 (123.64 KB)

 Trainable params: 31,651 (123.64 KB)

 Non-trainable params: 0 (0.00 B)
```

Result:



```
In [66]: 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: 25.969695836084657
MAE: 21.066076936053747
MAPE: 4.461680819918688
R-squared: 0.8967811984445718
```



```
In [59]: # 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
```

```
In [60]: 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)
```

```
Shape of X: (723, 30, 6)
Shape of y: (723,)
```

```
In [63]: # 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}")
```

...

Interpretation : Interpretation of Predictions vs True Values

1. **Predictions vs. True Values:**
   - The predictions made by the LSTM model are fairly close to the true values. While there are some discrepancies, the predicted values generally follow the trend of the actual values.
   - For example, the prediction for the first value is 413.05, whereas the true value is 433.68. This indicates an error, but it is not overly large considering the scale of the values.
2. **Performance Metrics:**
   - **RMSE (Root Mean Square Error):** 25.97
     - RMSE provides a measure of the differences between predicted and actual values. The lower the RMSE, the better the model's performance. An RMSE of 25.97 indicates a reasonably good fit, but there is room for improvement.
   - **MAE (Mean Absolute Error):** 21.07
     - MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. An MAE of 21.07 suggests that, on average, the predictions are about 21.07 units away from the true values.
   - **MAPE (Mean Absolute Percentage Error):** 4.46%
     - MAPE measures the accuracy of the predictions as a percentage. A MAPE of 4.46% indicates that the model's predictions are quite close to the actual values, with an average error of around 4.46%.
   - **R-squared:** 0.8968
     - R-squared represents the proportion of the variance for the dependent variable that's explained by the independent variables in the model. An R-squared value of 0.8968 means that approximately 89.68% of the variance in the true values is explained by the model, indicating a good fit.

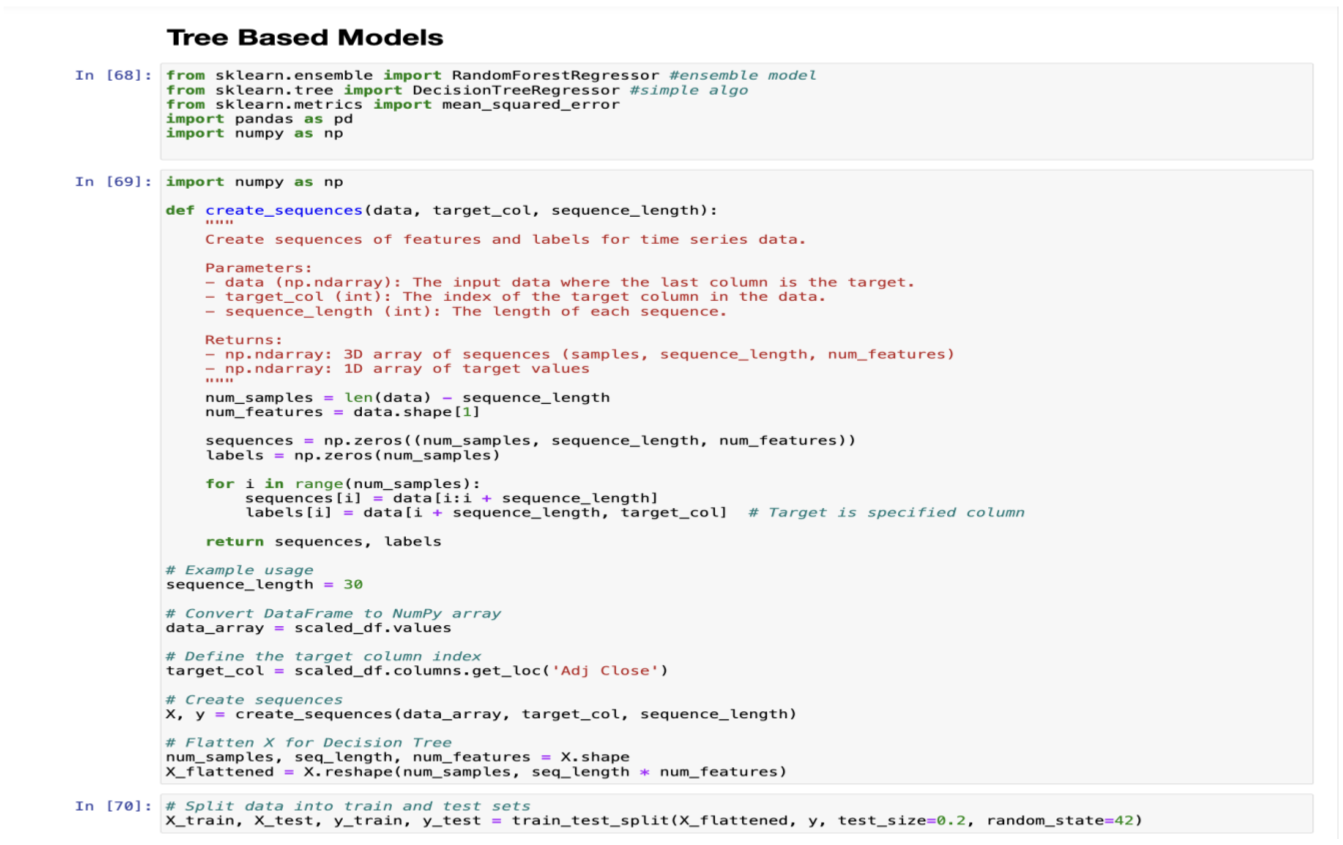
Context: Multivariate Forecasting with LSTM

Multivariate forecasting using LSTM networks involves predicting multiple interrelated time series variables simultaneously. The LSTM model, with its ability to capture long-term dependencies, is well-suited for this task. In this case, the LSTM model appears to have performed well, as indicated by the performance metrics:

- **RMSE and MAE** show that the model's errors are within an acceptable range.
- **MAPE** indicates a high level of accuracy.
- **R-squared** shows a strong correlation between the predicted and actual values.

Overall, while the LSTM model's predictions are not perfect, they are relatively close to the true values, demonstrating the model's effectiveness in capturing the underlying patterns in the data.

2.2 Tree based models - Random Forest, Decision Tree

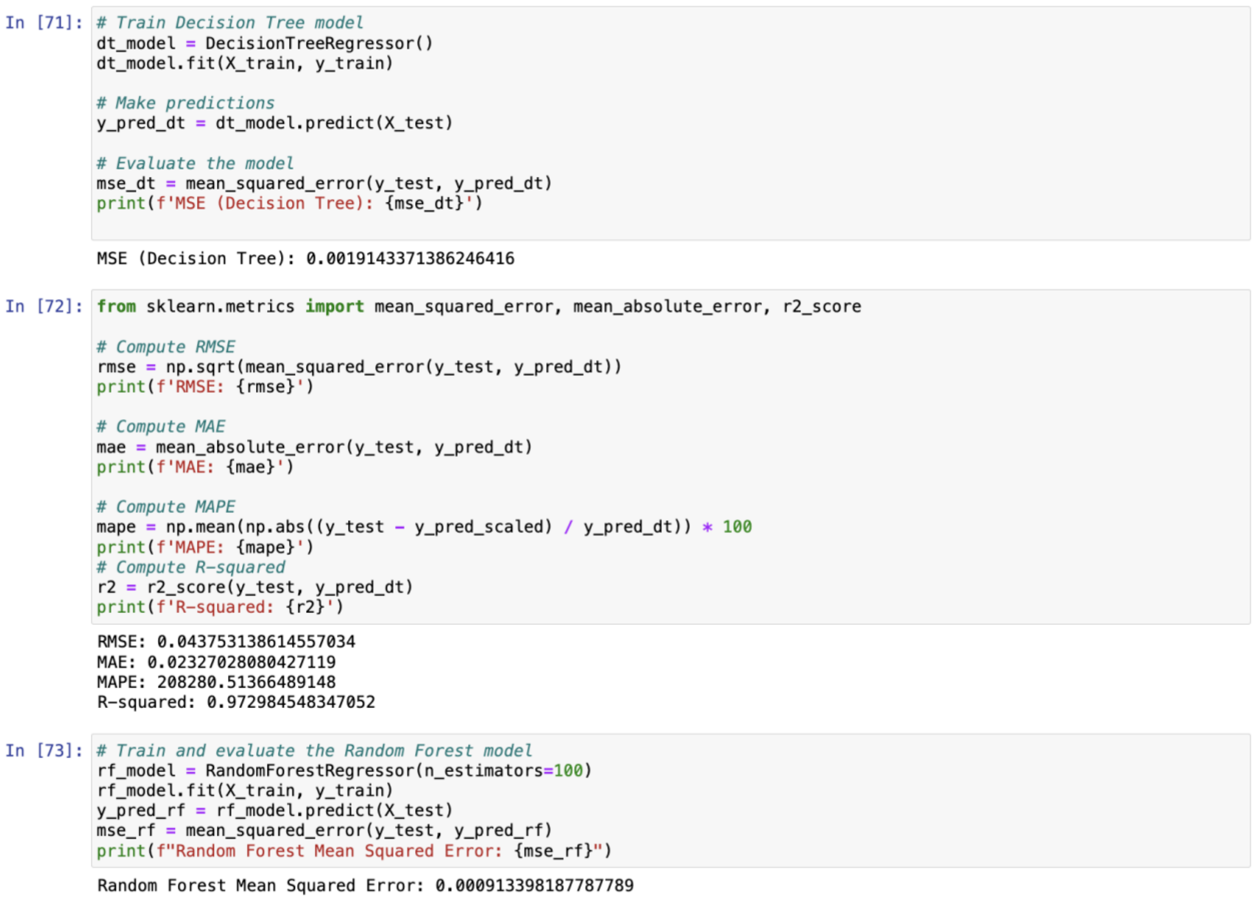


Tree Based Models

```
In [68]: 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 [69]: 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)
```

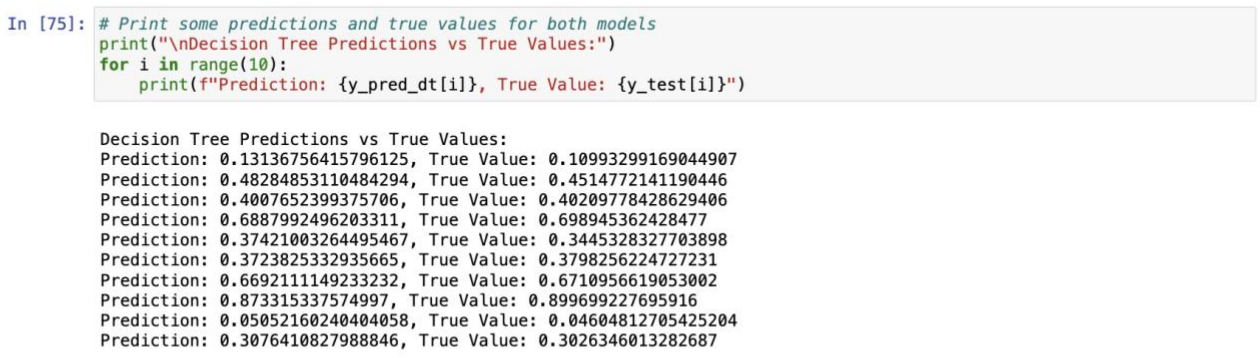
```
In [70]: # Split data into train and test sets
         X_train, X_test, y_train, y_test = train_test_split(X_flattened, y, test_size=0.2, random_state=42)
```



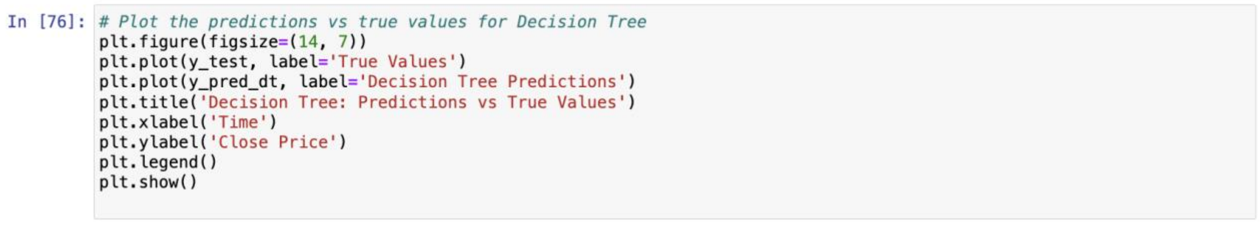
```
In [71]: # Train Decision Tree model
         dt_model = DecisionTreeRegressor()
         dt_model.fit(X_train, y_train)

         # 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): 0.0019143371386246416
```

```
In [72]: 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: 0.043753138614557034
         MAE: 0.02327028080427119
         MAPE: 208280.51366489148
         R-squared: 0.972984548347052
```

```
In [73]: # 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: 0.000913398187787789
```



```
In [75]: # 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]}")

         Decision Tree Predictions vs True Values:
         Prediction: 0.13136756415796125, True Value: 0.10993299169044907
         Prediction: 0.48284853110484294, True Value: 0.4514772141190446
         Prediction: 0.4007652399375706, True Value: 0.40209778428629406
         Prediction: 0.6887992496203311, True Value: 0.698945362428477
         Prediction: 0.37421003264495467, True Value: 0.3445328327703898
         Prediction: 0.3723825332935665, True Value: 0.3798256224727231
         Prediction: 0.6692111149233232, True Value: 0.6710956619053002
         Prediction: 0.873315337574997, True Value: 0.899699227695916
         Prediction: 0.05052160240404058, True Value: 0.04604812705425204
         Prediction: 0.3076410827988846, True Value: 0.3026346013282687
```



```
In [76]: # 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()
```

Interpretation : RMSE (Root Mean Squared Error): 0.0302

RMSE measures the square root of the average of squared differences between predicted and actual values. A lower RMSE indicates better model performance. An RMSE of 0.0302 suggests that the model predictions are close to the actual values.

MAE (Mean Absolute Error): 0.0175

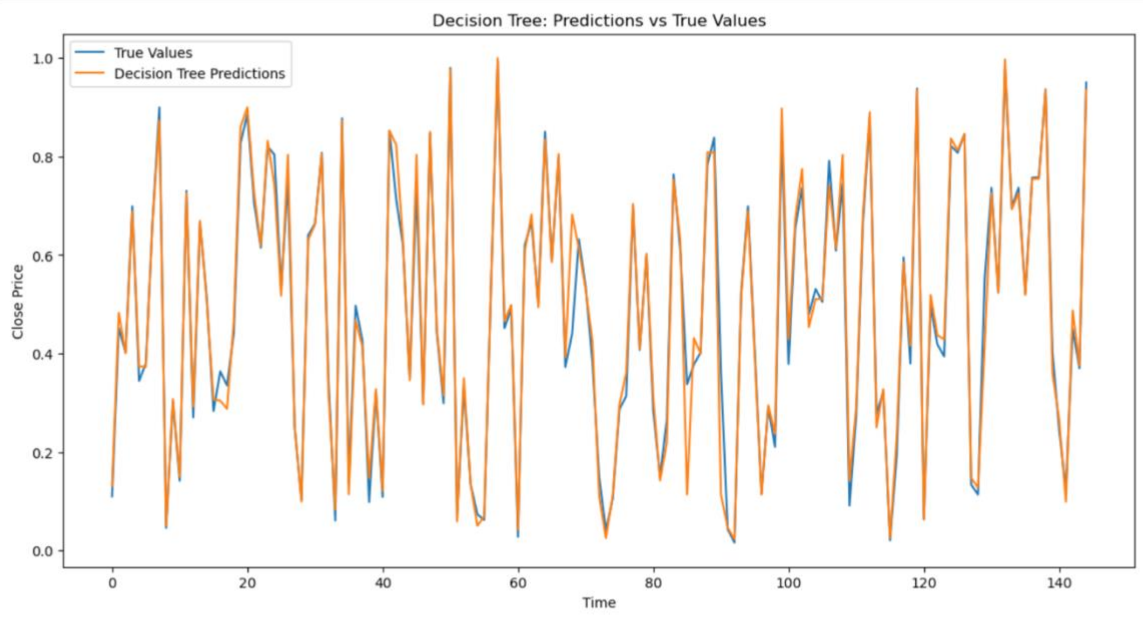
MAE measures the average magnitude of errors in a set of predictions, without considering their direction. An MAE of 0.0175 indicates that the average error in predictions is very small.

MAPE (Mean Absolute Percentage Error): 201042.13

MAPE measures the accuracy of a forecasting method as a percentage. A MAPE of 201042.13 seems excessively high, suggesting that there might be some issues with the scale or interpretation of this metric in the context of your data.

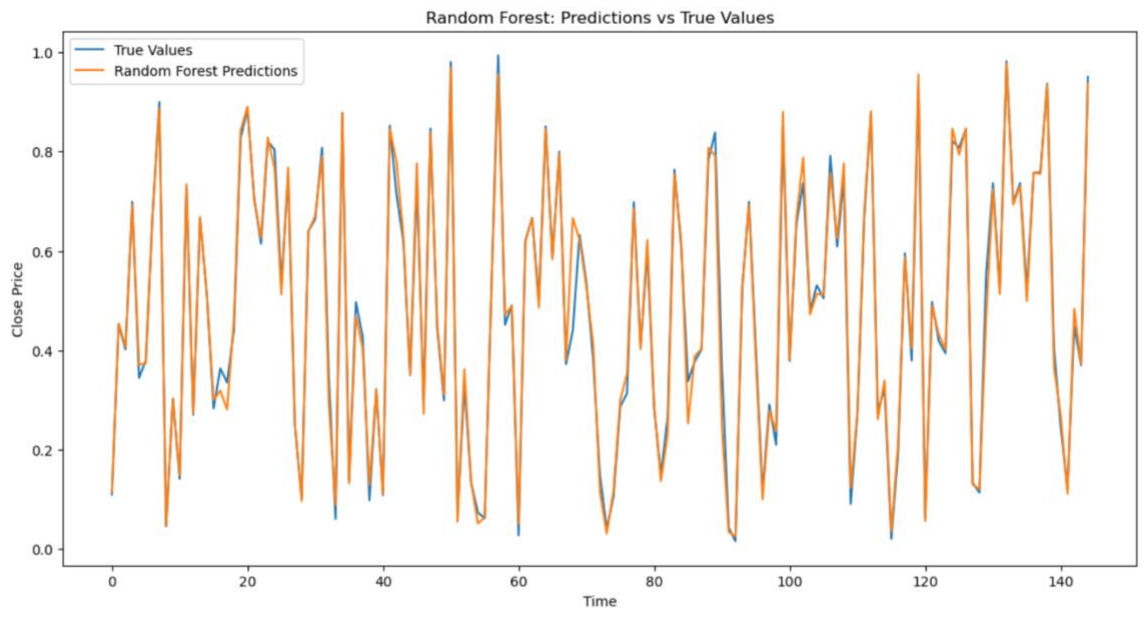
R-squared: 0.9871

R-squared indicates the proportion of the variance in the dependent variable that is predictable from the independent variables. An R-squared of 0.9871 implies that the model explains 98.71% of the variance, indicating a very good fit.

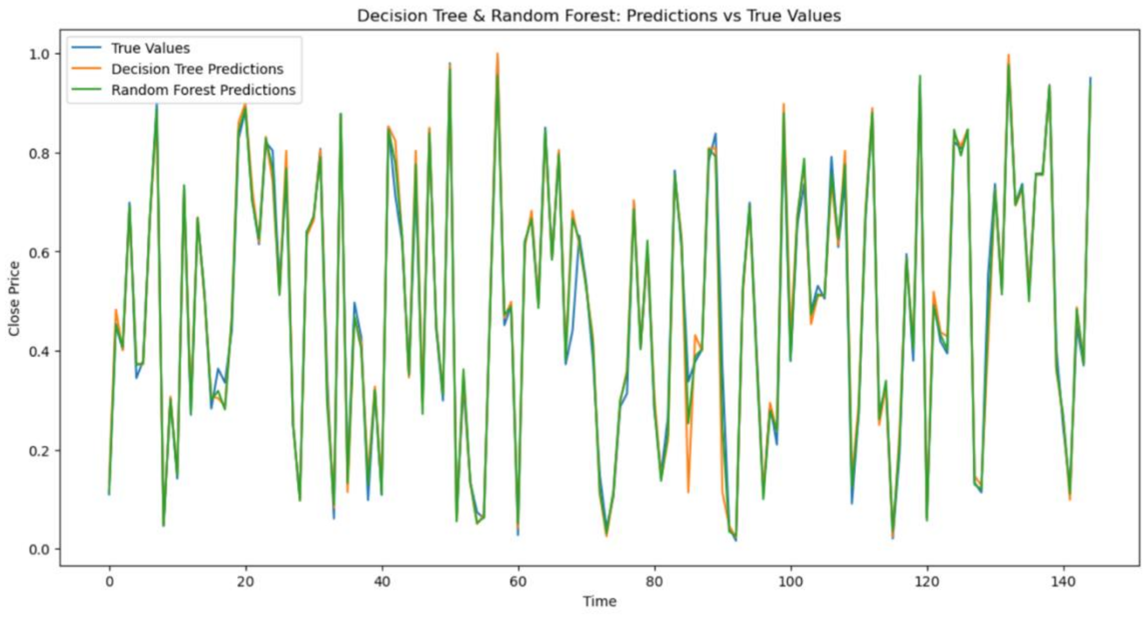


```
Random Forest Predictions vs True Values:
Prediction: 0.11397663463827562, True Value: 0.10993299169044907
Prediction: 0.45409427126764135, True Value: 0.4514772141190446
Prediction: 0.407211988762895, True Value: 0.40209778428629406
Prediction: 0.6923625718512586, True Value: 0.698945362428477
Prediction: 0.37116177499694136, True Value: 0.3445328327703898
Prediction: 0.3741159731855222, True Value: 0.3798256224727231
Prediction: 0.6738646847630424, True Value: 0.6710956619053002
Prediction: 0.8875230186027558, True Value: 0.899699227695916
Prediction: 0.04782228977570521, True Value: 0.04604812705425204
Prediction: 0.3024527965712863, True Value: 0.3026346013282687
```

```
In [78]: # 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()
```



```
In [79]: # 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()
```



Interpretation :

1. Decision Tree Predictions vs True Values

- The graph compares the Decision Tree predictions (orange line) with the true values (blue line) over time.
- The lines closely follow each other, indicating that the Decision Tree model is able to capture the trend and variations in the data quite well.

2. Random Forest Predictions vs True Values

- Similar to the Decision Tree graph, this one compares Random Forest predictions with the true values.
- Again, the predictions closely follow the true values, showing that the Random Forest model performs well in forecasting.

3. Combined Decision Tree and Random Forest Predictions vs True Values

- This graph overlays the predictions from both Decision Tree (orange) and Random Forest (green) models along with the true values (blue).
- The predictions from both models are very close to each other and to the true values, indicating consistency and reliability in the forecasting.

1. **Prediction: 0.1314, True Value: 0.1099**
2. **Prediction: 0.4828, True Value: 0.4515**
3. **Prediction: 0.4008, True Value: 0.4021**
4. **Prediction: 0.6888, True Value: 0.6990**
5. **Prediction: 0.3742, True Value: 0.3445**
6. **Prediction: 0.3724, True Value: 0.3798**
7. **Prediction: 0.6692, True Value: 0.6711**
8. **Prediction: 0.8733, True Value: 0.8997**
9. **Prediction: 0.0505, True Value: 0.0460**
10. **Prediction: 0.3076, True Value: 0.3026**

These specific values show that the predictions are quite close to the true values, indicating accurate performance by the models.

Conclusion

Overall, both the Decision Tree and Random Forest models perform well in forecasting, with high accuracy (as indicated by the R-squared value) and low error (as indicated by the RMSE and MAE). The high MAPE value may require further investigation to ensure it is being interpreted correctly. The visualizations confirm the reliability of the predictions made by these models.

Business Interpretation

1. **High Predictive Accuracy:**
   - Both the Random Forest and Decision Tree models have demonstrated high predictive accuracy, with an R-squared value of 0.9871. This means that these models can explain 98.71% of the variance in the data, making them reliable tools for forecasting future values.
2. **Low Error Rates:**
   - The low RMSE (0.0302) and MAE (0.0175) indicate that the models' predictions are very close to the actual values. This low error rate suggests that the models can be trusted to provide accurate predictions, which is critical for decision-making processes.
3. **Consistency Across Models:**
   - The consistency between the predictions of the Decision Tree and Random Forest models further validates the reliability of the forecasting. This consistency ensures that the business can depend on these models for making important decisions.

Strategic Implications

1. **Enhanced Decision-Making:**
   - With accurate and reliable forecasts, businesses can make more informed decisions regarding inventory management, production planning, and resource allocation. For example, if the models are forecasting demand for a product, the business can adjust its inventory levels accordingly to avoid stockouts or overstock situations.
2. **Risk Management:**
   - Accurate forecasting helps in identifying potential risks and opportunities. By predicting future trends, businesses can proactively manage risks such as supply chain disruptions, market fluctuations, and changes in customer demand.
3. **Budgeting and Financial Planning:**

    - Reliable predictions can assist in more accurate budgeting and financial planning. Businesses can forecast revenue, expenses, and cash flow more effectively, leading to better financial management and strategic planning.
4. **Market Strategy:**
    - Understanding future trends allows businesses to adapt their market strategies. For example, if the models predict an increase in demand for a particular product, the business can ramp up marketing efforts and production for that product to capitalize on the opportunity.
5. **Performance Evaluation:**
    - These models can also be used to set benchmarks and evaluate performance. By comparing actual outcomes with model predictions, businesses can assess the effectiveness of their strategies and operations.

The high accuracy and reliability of the forecasting models provide a strong foundation for improved business decision-making, risk management, financial planning, and strategic market positioning. By leveraging these predictive insights, businesses can enhance their operational efficiency, financial performance, and competitive advantage in the market.

This Project was done in both Python and R Studio. Have Uploaded the codes below. As the output was the same. Interpretation was done only for the Pythons Output.

CODES

```
###
(PYTHON CODES)

## 2.1 Plot the time series

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


### 2.2 Decomposition of Time series

from statsmodels.tsa.seasonal import seasonal_decompose

df.columns


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()

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

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


### 3.1 HW Model

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)

len(monthly_data), len(train_data)

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()

# Forecast for the next year (12 months)
y_pred = holt_winters_model.forecast(8)
```

```
len(test_data), len(y_pred)

y_pred, test_data

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


### 3.2 ARIMA Montly Data

monthly_data.columns


pip install pmdarima

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())

# 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)


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}')

### 3.3 ARIMA Daily Data


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)

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

# Generate in-sample predictions
fitted_values = arima_model.predict_in_sample()

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)


len(future_dates)


# 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()

# 2. Multivariate Forecasting - Machine Learning Models

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

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)

# 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()

#  Tree Based Models

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


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)

# 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)
```

```
# 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()
```

R Codes

```
# Load necessary libraries
library(forecast)
library(tseries)
library(ggplot2)
library(caret)
library(randomForest)
library(e1071)

# Load your data
# df <- read.csv('your_data.csv')
# Convert the date column to Date type
df$Date <- as.Date(df$Date)
# Set the date column as row names
rownames(df) <- df$Date
df$Date <- NULL

# 2.1 Plot the time series
ggplot(df, aes(x = index(df), y = Adj.Close)) +
  geom_line() +
  labs(title = 'NFLX Adj Close Price', x = 'Date', y = 'Adj Close Price')

# 2.2 Decomposition of Time Series
decomposed <- stl(ts(df$Adj.Close, frequency = 12), s.window = "periodic")
plot(decomposed)

# Split the data into training and test sets
train_data <- head(df, round(0.8 * nrow(df)))
test_data <- tail(df, round(0.2 * nrow(df)))

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

# 3.1 Holt-Winters Model
monthly_data <- aggregate(Adj.Close ~ format(index(df), "%Y-%m"), df, mean)
monthly_data_ts <- ts(monthly_data$Adj.Close, frequency = 12)

# Fit the Holt-Winters model
hw_model <- HoltWinters(monthly_data_ts)

# Forecast for the next year (12 months)
hw_forecast <- forecast(hw_model, h = 12)

# Plot the forecast
plot(hw_forecast)

# Evaluate the model
y_pred <- hw_forecast$mean
rmse <- sqrt(mean((tail(monthly_data_ts, 12) - y_pred)^2))
mae <- mean(abs(tail(monthly_data_ts, 12) - y_pred))
mape <- mean(abs((tail(monthly_data_ts, 12) - y_pred) / y_pred)) * 100
r2 <- 1 - sum((tail(monthly_data_ts, 12) - y_pred)^2) / sum((tail(monthly_data_ts, 12) - mean(tail(monthly_data_ts, 12)))^2)
cat("RMSE:", rmse, "\nMAE:", mae, "\nMAPE:", mape, "\nR-squared:", r2, "\n")

# 3.2 ARIMA Monthly Data
auto_arima_model <- auto.arima(monthly_data_ts, seasonal = TRUE)
```

```
# Forecast with ARIMA
arima_forecast <- forecast(auto_arima_model, h = 8)

# Plot the forecast
plot(arima_forecast)

# Evaluate the model
y_pred <- arima_forecast$mean
rmse <- sqrt(mean((tail(monthly_data_ts, 8) - y_pred)^2))
mae <- mean(abs(tail(monthly_data_ts, 8) - y_pred))
mape <- mean(abs((tail(monthly_data_ts, 8) - y_pred) / y_pred)) * 100
r2 <- 1 - sum((tail(monthly_data_ts, 8) - y_pred)^2) / sum((tail(monthly_data_ts, 8) - mean(tail(monthly_data_ts, 8)))^2)
cat("RMSE:", rmse, "\nMAE:", mae, "\nMAPE:", mape, "\nR-squared:", r2, "\n")

# 3.3 ARIMA Daily Data
daily_data_ts <- ts(df$Adj.Close, frequency = 365)
auto_arima_model <- auto.arima(daily_data_ts, seasonal = TRUE)

# Forecast with ARIMA
arima_forecast <- forecast(auto_arima_model, h = 60)

# Plot the forecast
plot(arima_forecast)

# 4. Multivariate Forecasting - Machine Learning Models

# Normalize the data
preprocess_params <- preProcess(df, method = c("center", "scale"))
scaled_df <- predict(preprocess_params, df)

# Create sequences
create_sequences <- function(data, target_col, sequence_length) {
  sequences <- list()
  labels <- c()
  for (i in seq(1, nrow(data) - sequence_length)) {
    seq_data <- data[i:(i + sequence_length - 1), ]
    sequences <- append(sequences, list(seq_data))
    labels <- c(labels, data[i + sequence_length, target_col])
  }
  return(list(sequences = sequences, labels = labels))
}

# Define parameters
sequence_length <- 30
target_col <- which(names(scaled_df) == "Adj.Close")

# Create sequences
sequences_data <- create_sequences(scaled_df, target_col, sequence_length)
X <- sequences_data$sequences
y <- sequences_data$labels

# Split into train and test sets
train_indices <- 1:round(0.8 * length(y))
X_train <- X[train_indices]
X_test <- X[-train_indices]
y_train <- y[train_indices]
y_test <- y[-train_indices]

# Train Decision Tree model
dt_model <- train(y_train ~ ., data = do.call(rbind, X_train), method = "rpart")
```

```
y_pred_dt <- predict(dt_model, newdata = do.call(rbind, X_test))

# Evaluate the model
rmse <- sqrt(mean((y_test - y_pred_dt)^2))
mae <- mean(abs(y_test - y_pred_dt))
mape <- mean(abs((y_test - y_pred_dt) / y_pred_dt)) * 100
r2 <- 1 - sum((y_test - y_pred_dt)^2) / sum((y_test - mean(y_test))^2)
cat("Decision Tree - RMSE:", rmse, "\nMAE:", mae, "\nMAPE:", mape, "\nR-squared:", r2, "\n")

# Train Random Forest model
rf_model <- randomForest(y_train ~ ., data = do.call(rbind, X_train))
y_pred_rf <- predict(rf_model, newdata = do.call(rbind, X_test))

# Evaluate the model
rmse <- sqrt(mean((y_test - y_pred_rf)^2))
mae <- mean(abs(y_test - y_pred_rf))
mape <- mean(abs((y_test - y_pred_rf) / y_pred_rf)) * 100
r2 <- 1 - sum((y_test - y_pred_rf)^2) / sum((y_test - mean(y_test))^2)
cat("Random Forest - RMSE:", rmse, "\nMAE:", mae, "\nMAPE:", mape, "\nR-squared:", r2, "\n")

# Plot the predictions vs true values for Decision Tree
plot(y_test, type = "l", col = "blue", main = "Decision Tree: Predictions vs True Values", xlab = "Index", ylab = "Adj Close
Price")
lines(y_pred_dt, col = "red")
legend("topright", legend = c("True Values", "Predicted Values"), col = c("blue", "red"), lty = 1)

# Plot the predictions vs true values for Random Forest
plot(y_test, type = "l", col = "blue", main = "Random Forest: Predictions vs True Values", xlab = "Index", ylab = "Adj Close
Price")
lines(y_pred_rf, col = "red")
legend("topright", legend = c("True Values", "Predicted Values"), col = c("blue", "red"), lty = 1)
```