

**VIRGINIA COMMONWEALTH UNIVERSITY**

**Statistical Analysis and Modelling (SCMA 632)**

A6: Time series

**Nitheesh MK**

**V01107616**

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

# CONTENTS

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **TITLE** | **Page No.** |
| **I.** | Introduction | 3 |
| **II.** | Objective | 3 |
| **III.** | Business Significance | 4 |
| **IV.** | Results and Interpretation | 5 |
| **V.** | Recommendations | 12 |
| **VI.** | Conclusion | 13 |

| **Open** | **High** | **Low** | **Close** | **Adj Close** | **Volume** |
| --- | --- | --- | --- | --- | --- |
| **Date** |  |  |  |  |  |  |
| **2021-04-01** | 306.750000 | 309.850006 | 303.049988 | 307.750000 | 305.850006 | 44088352 |
| **2021-04-05** | 306.799988 | 311.700012 | 297.200012 | 305.049988 | 303.166656 | 66178755 |
| **2021-04-06** | 306.149994 | 313.799988 | 304.799988 | 307.750000 | 305.850006 | 63031783 |
| **2021-04-07** | 306.750000 | 310.649994 | 305.100006 | 307.799988 | 305.899689 | 39073986 |
| **2021-04-08** | 307.899994 | 319.799988 | 307.500000 | 313.950012 | 312.011719 | 62459774 |

In [213]:

*# 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 [220]:

**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: (710, 30, 6)

Shape of y: (710,)

In [224]:

*# 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))

In [225]:

model.summary()

**Model: "sequential\_3"**

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓

┃ **Layer (type)** ┃ **Output Shape** ┃ **Param #** ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩

│ lstm\_6 (LSTM) │ (None, 30, 50) │ 11,400 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ dropout\_6 (Dropout) │ (None, 30, 50) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ lstm\_7 (LSTM) │ (None, 50) │ 20,200 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ dropout\_7 (Dropout) │ (None, 50) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ dense\_3 (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)

In [226]:

*# 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}")

Epoch 1/20

**18/18** ━━━━━━━━━━━━━━━━━━━━ **6s** 63ms/step - loss: 0.0092 - val\_loss: 0.0410

Epoch 2/20

**18/18** ━━━━━━━━━━━━━━━━━━━━ **0s** 24ms/step - loss: 0.0048 - val\_loss: 0.0086

Epoch 3/20

**18/18** ━━━━━━━━━━━━━━━━━━━━ **0s** 25ms/step - loss: 0.0024 - val\_loss: 0.0051

Epoch 4/20

**18/18** ━━━━━━━━━━━━━━━━━━━━ **0s** 22ms/step - loss: 0.0019 - val\_loss: 0.0027

Epoch 5/20

**18/18** ━━━━━━━━━━━━━━━━━━━━ **0s** 23ms/step - loss: 0.0015 - val\_loss: 0.0019

Epoch 6/20

**18/18** ━━━━━━━━━━━━━━━━━━━━ **0s** 23ms/step - loss: 0.0015 - val\_loss: 0.0014

Epoch 7/20

**18/18** ━━━━━━━━━━━━━━━━━━━━ **0s** 23ms/step - loss: 0.0016 - val\_loss: 0.0013

Epoch 8/20

**18/18** ━━━━━━━━━━━━━━━━━━━━ **0s** 24ms/step - loss: 0.0019 - val\_loss: 0.0019

Epoch 9/20

**18/18** ━━━━━━━━━━━━━━━━━━━━ **0s** 24ms/step - loss: 0.0015 - val\_loss: 0.0014

Epoch 10/20

**18/18** ━━━━━━━━━━━━━━━━━━━━ **0s** 23ms/step - loss: 0.0016 - val\_loss: 0.0011

Epoch 11/20

**18/18** ━━━━━━━━━━━━━━━━━━━━ **0s** 24ms/step - loss: 0.0015 - val\_loss: 0.0011

Epoch 12/20

**18/18** ━━━━━━━━━━━━━━━━━━━━ **0s** 24ms/step - loss: 0.0012 - val\_loss: 0.0012

Epoch 13/20

**18/18** ━━━━━━━━━━━━━━━━━━━━ **0s** 22ms/step - loss: 9.8967e-04 - val\_loss: 0.0010

Epoch 14/20

**18/18** ━━━━━━━━━━━━━━━━━━━━ **0s** 23ms/step - loss: 0.0011 - val\_loss: 0.0011

Epoch 15/20

**18/18** ━━━━━━━━━━━━━━━━━━━━ **0s** 21ms/step - loss: 0.0010 - val\_loss: 0.0011

Epoch 16/20

**18/18** ━━━━━━━━━━━━━━━━━━━━ **1s** 29ms/step - loss: 9.2151e-04 - val\_loss: 0.0012

Epoch 17/20

**18/18** ━━━━━━━━━━━━━━━━━━━━ **0s** 22ms/step - loss: 9.2854e-04 - val\_loss: 0.0011

Epoch 18/20

**18/18** ━━━━━━━━━━━━━━━━━━━━ **0s** 22ms/step - loss: 0.0012 - val\_loss: 0.0013

Epoch 19/20

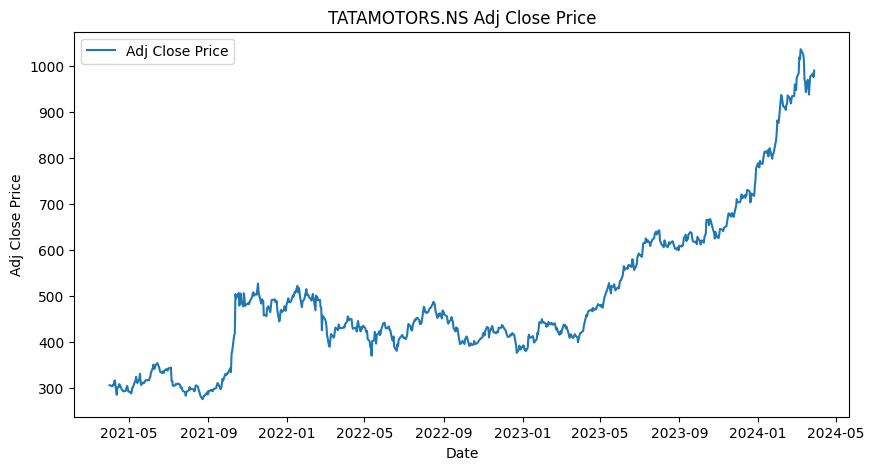
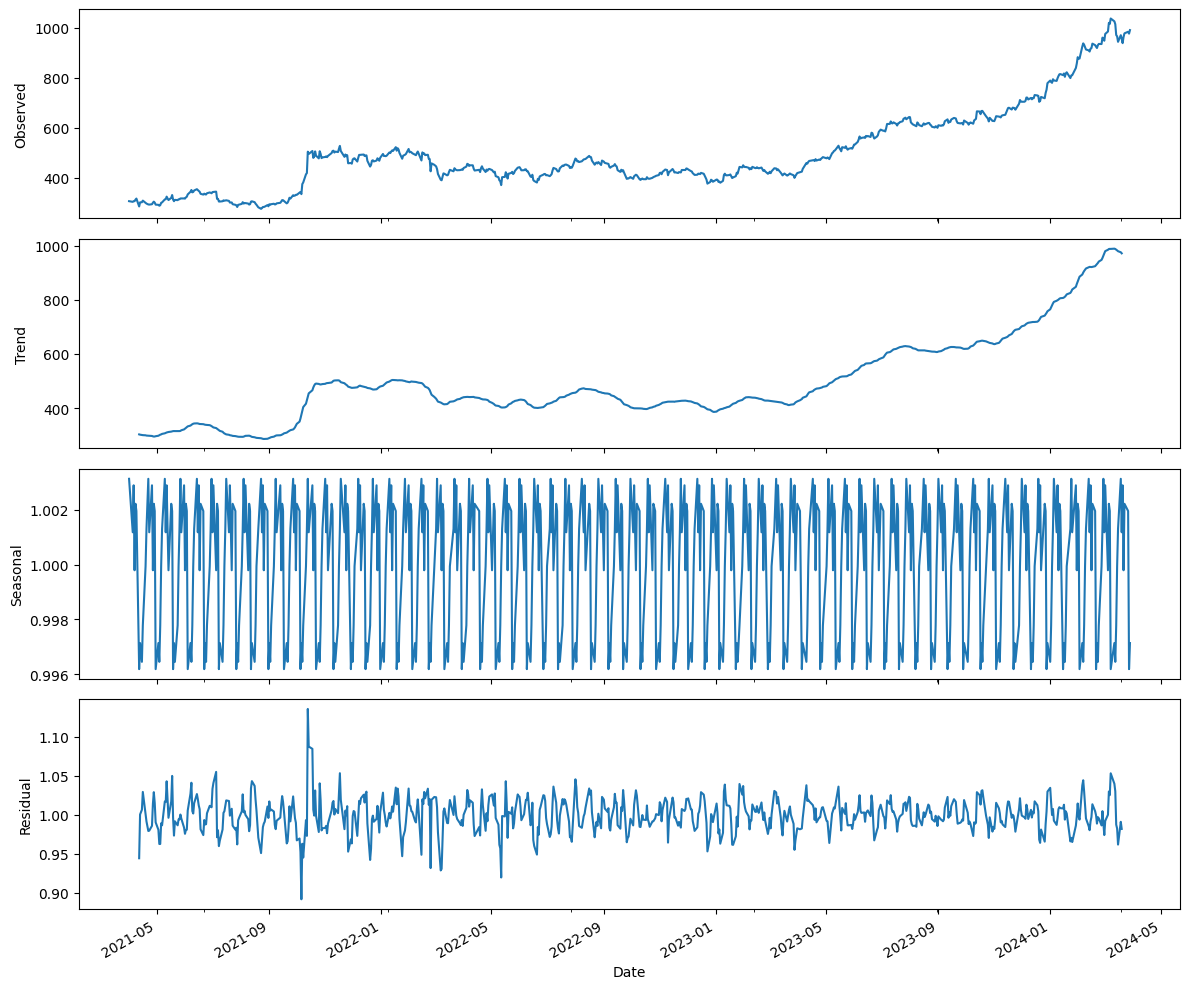
**18/18** ━━━━━━━━━━━━━━━━━━━━ **0s** 22ms/step - loss: 0.0015 - val\_loss: 8.6428e-04

Epoch 20/20

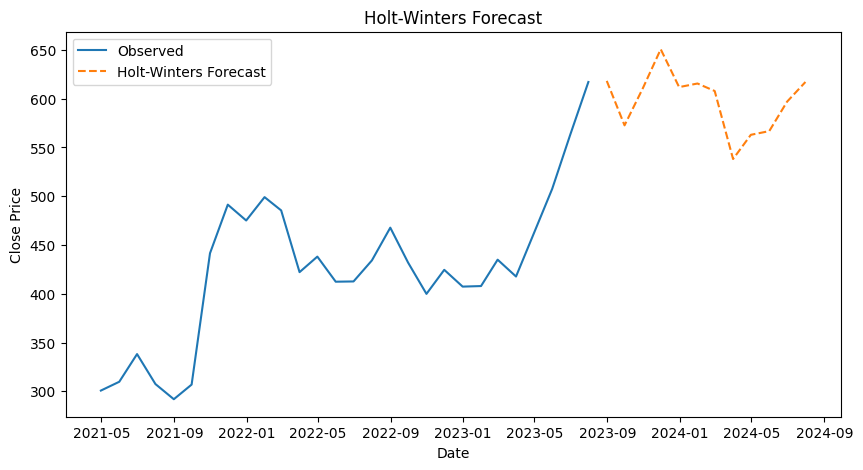
**18/18** ━━━━━━━━━━━━━━━━━━━━ **0s** 21ms/step - loss: 0.0017 - val\_loss: 0.0020

**5/5** ━━━━━━━━━━━━━━━━━━━━ **0s** 8ms/step - loss: 0.0014

Test Loss: 0.001983438851311803



* + 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()



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')

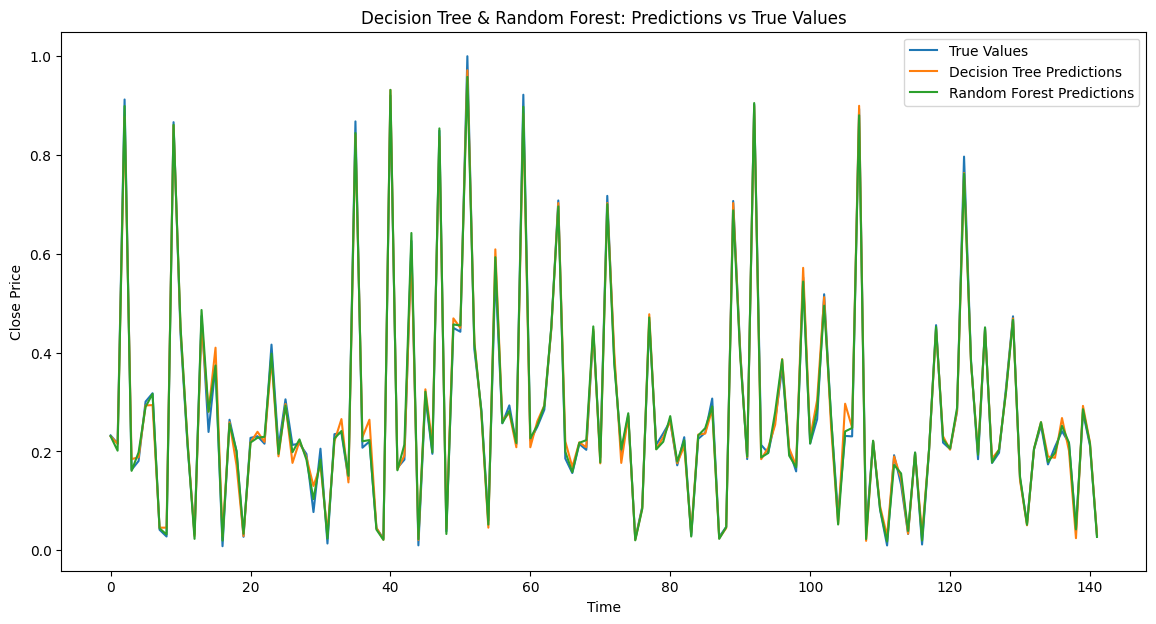
plt.title('Decision Tree & Random Forest: Predictions vs True Values')

plt.xlabel('Time')

plt.ylabel('Close Price')

plt.legend()

plt.show()



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')

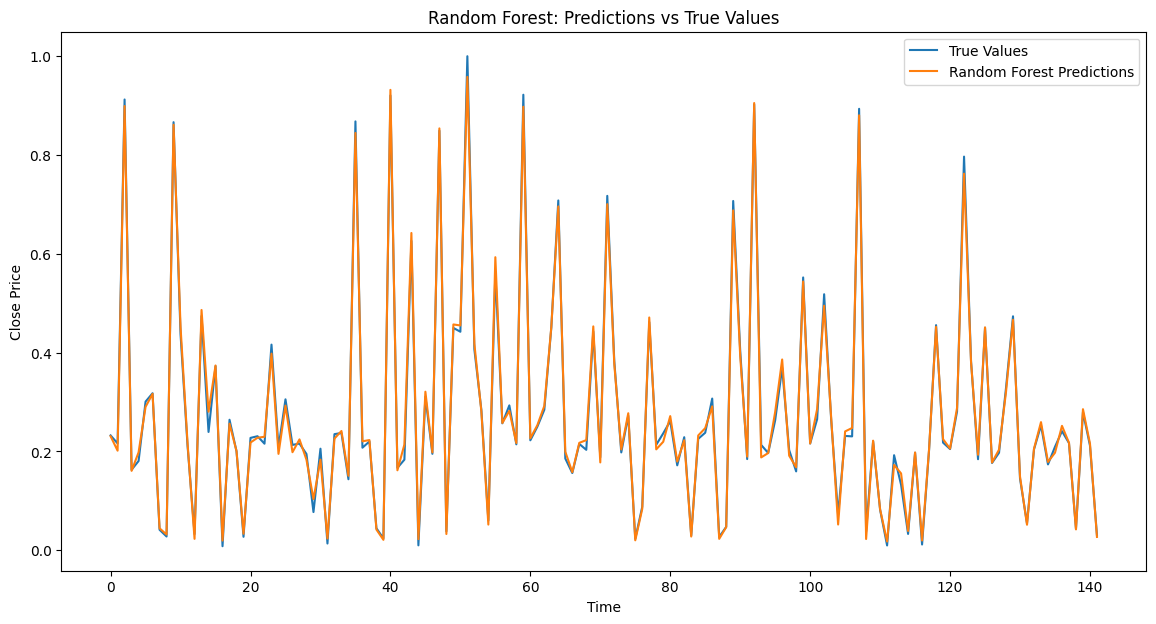
plt.title('Random Forest: Predictions vs True Values')

plt.xlabel('Time')

plt.ylabel('Close Price')

plt.legend()

plt.show()



Plot 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')

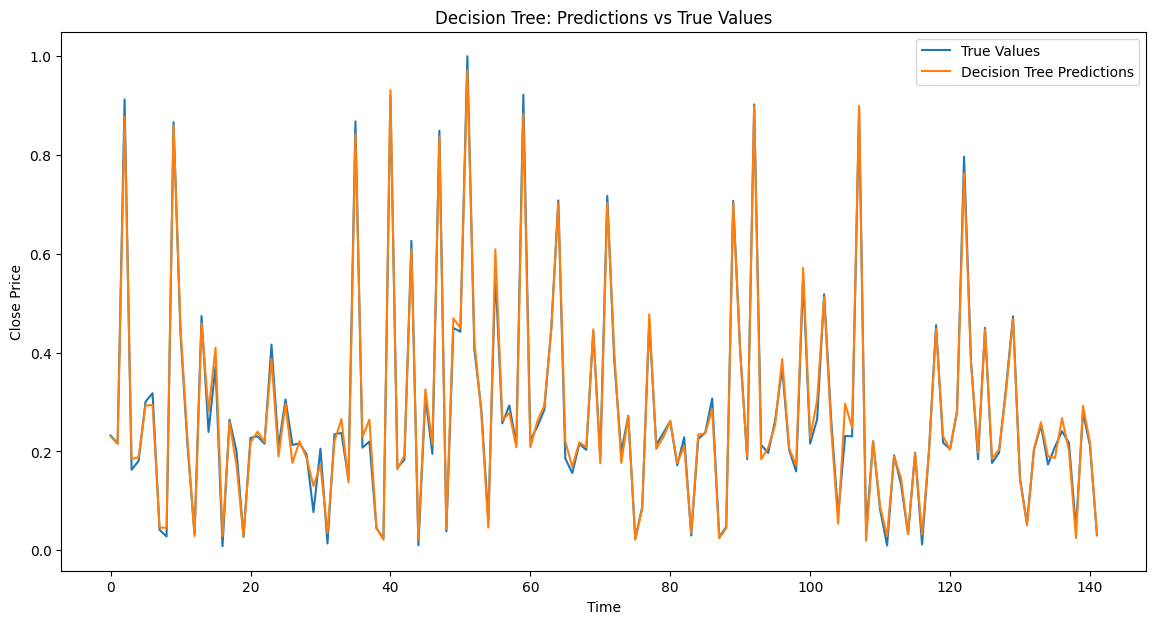
plt.title('Decision Tree: Predictions vs True Values')

plt.xlabel('Time')

plt.ylabel('Close Price')

plt.legend()

plt.show()



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()

