In [1]:
```
# Importing necessary libraries
import yfinance as yf
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
import matplotlib.pyplot as plt
import seaborn as sns
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

```
C:\Users\mrmua\New folder\lib\site-packages\scipy\__init__.py:155: UserWarning:
A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (dete
cted version 1.26.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
WARNING:tensorflow:From C:\Users\mrmua\New folder\lib\site-packages\keras\src\lo
sses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Ple
ase use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

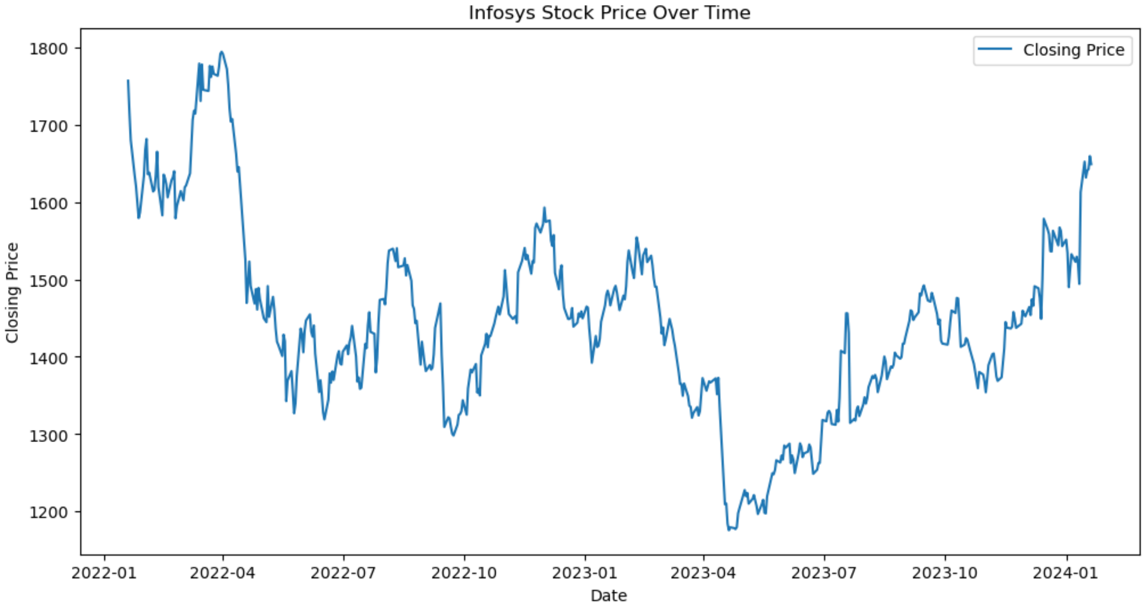
In [2]:
```
# Download historical data for Infosys (INFY)
ticker = yf.Ticker("INFY.NS")
df = ticker.history(period="2y", interval="1d")

# Display the first few rows of the dataframe
print("Dataframe head:")
df.head()
```

```
Dataframe head:
```

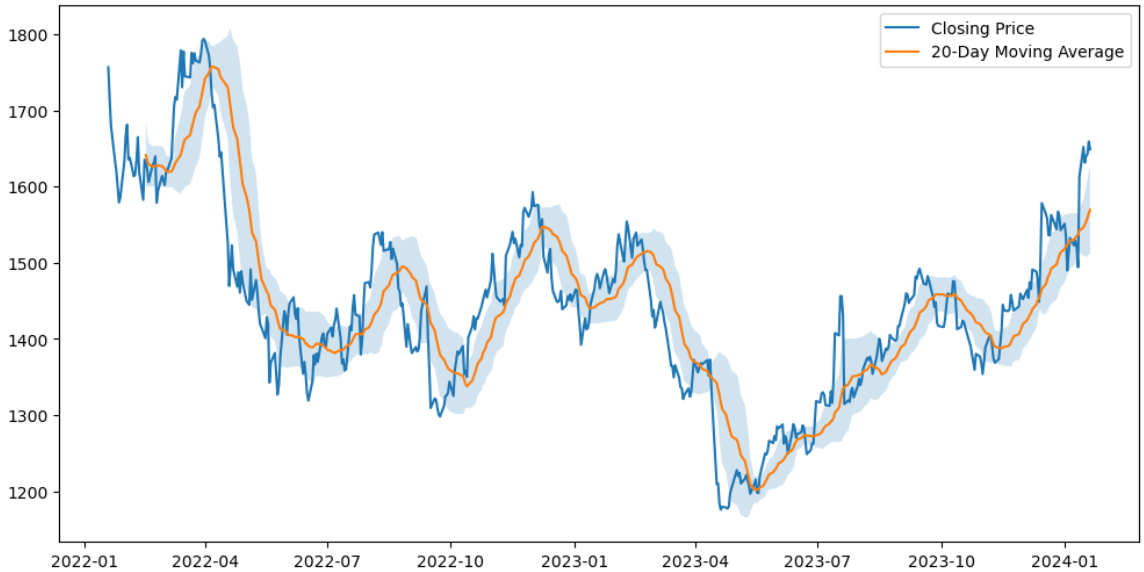
Out[2]:

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
2022-01-19 00:00:00+05:30	1802.654294	1802.654294	1752.648490	1756.600098	5747770	0.0	0.0
2022-01-20 00:00:00+05:30	1734.913602	1738.676972	1708.099590	1715.814453	5533463	0.0	0.0
2022-01-21 00:00:00+05:30	1688.812259	1701.043211	1670.936252	1680.062378	8252758	0.0	0.0
2022-01-24 00:00:00+05:30	1660.587258	1664.021357	1625.776081	1634.055542	7116712	0.0	0.0
2022-01-25 00:00:00+05:30	1624.458554	1637.065865	1599.902589	1620.271851	9137653	0.0	0.0

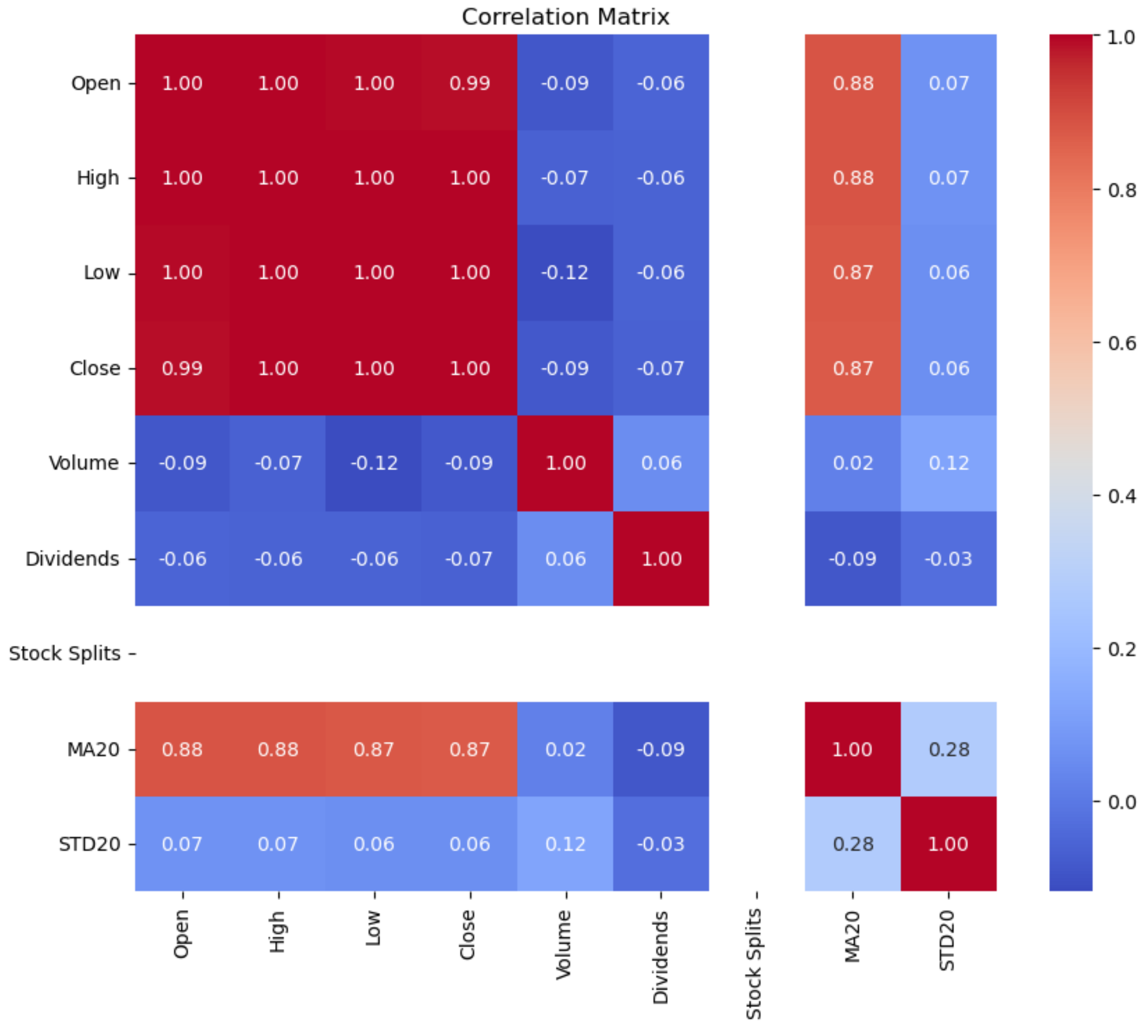
In [3]:
```
# Data visualization
plt.figure(figsize=(12, 6))
plt.plot(df['Close'], label='Closing Price')
plt.title('Infosys Stock Price Over Time')
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.legend()
plt.show()
```



In [4]:
```
# Moving averages
df['MA20'] = df['Close'].rolling(window=20).mean()
df['STD20'] = df['Close'].rolling(window=20).std()

plt.figure(figsize=(12, 6))
plt.plot(df['Close'], label='Closing Price')
plt.plot(df['MA20'], label='20-Day Moving Average')
plt.fill_between(df.index, df['MA20'] - df['STD20'], df['MA20'] + df['STD20'], a
plt.legend()
plt.show()
```



In [5]:
```
# Statistical analysis
correlation_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```



In [6]:
```
# Data preparation for LSTM
X = df["Close"].values.reshape(-1, 1)
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

sequence_length = 5
sequences = []
target = []

for i in range(len(X_scaled) - sequence_length):
    seq = X_scaled[i:i + sequence_length, 0]
    label = X_scaled[i + sequence_length, 0]
    sequences.append(seq)
    target.append(label)

X_seq, y_seq = np.array(sequences), np.array(target)
```

In [7]:
```
# Creating a new dataframe with sequences and target
df_sequences = pd.DataFrame(X_seq, columns=[f'Day_{i+1}' for i in range(sequence
df_sequences['Target'] = y_seq

# Displaying the new dataframe with sequences and target
print("\nDataframe with Sequences and Target:")
df_sequences.head()
```

```
Dataframe with Sequences and Target:
```

Out[7]:

	Day_1	Day_2	Day_3	Day_4	Day_5	Target
0	0.939420	0.873436	0.815596	0.741165	0.718866	0.652578
1	0.873436	0.815596	0.741165	0.718866	0.652578	0.664146
2	0.815596	0.741165	0.718866	0.652578	0.664146	0.740252
3	0.741165	0.718866	0.652578	0.664146	0.740252	0.794820
4	0.718866	0.652578	0.664146	0.740252	0.794820	0.817727

In [8]:

```
# Displaying the differences in the dataframe after creating sequences and targe
print("\nDifferences in the Dataframe:")
df_sequences.diff().dropna().head()
```

```
Differences in the Dataframe:
```

Out[8]:

	Day_1	Day_2	Day_3	Day_4	Day_5	Target
1	-0.065984	-0.057840	-0.074431	-0.022300	-0.066288	0.011568
2	-0.057840	-0.074431	-0.022300	-0.066288	0.011568	0.076105
3	-0.074431	-0.022300	-0.066288	0.011568	0.076105	0.054568
4	-0.022300	-0.066288	0.011568	0.076105	0.054568	0.022908
5	-0.066288	0.011568	0.076105	0.054568	0.022908	-0.073898

In [9]:

```
# Splitting the data into training, validation, and testing sets
split_ratio_train = 0.8
split_ratio_val = 0.9

split_index_train = int(split_ratio_train * len(X_seq))
split_index_val = int(split_ratio_val * len(X_seq))

X_train, X_val, X_test = X_seq[:split_index_train], X_seq[split_index_train:spli
y_train, y_val, y_test = y_seq[:split_index_train], y_seq[split_index_train:spli
```

In [10]:

```
# Creating LSTM model
model = Sequential([
    LSTM(units=64, return_sequences=True, input_shape=(sequence_length, 1)),
    Dropout(0.3),
    LSTM(units=32),
    Dropout(0.4),
    Dense(1)
])
```

```
WARNING:tensorflow:From C:\Users\mrmua\New folder\lib\site-packages\keras\src\la
yers\rnn\lstm.py:148: The name tf.executing_eagerly_outside_functions is depreca
ted. Please use tf.compat.v1.executing_eagerly_outside_functions instead.
```

In [11]:

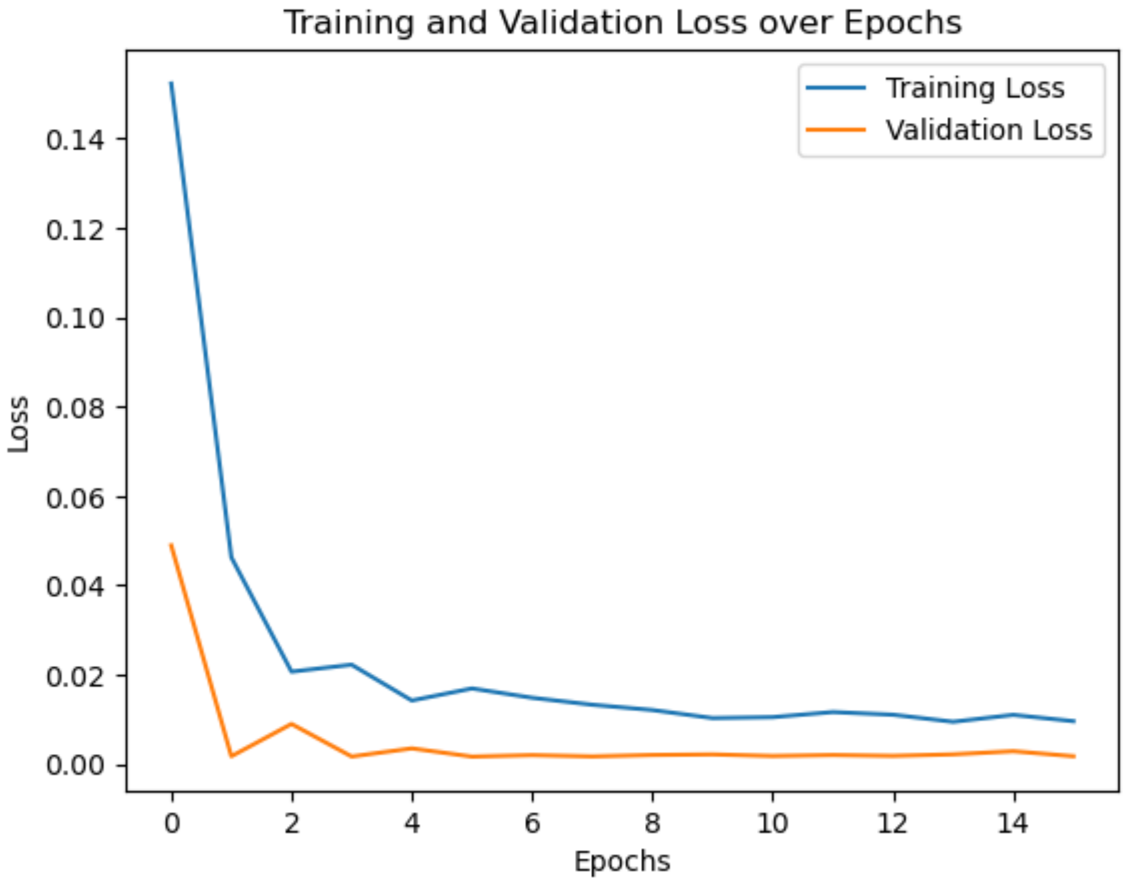
```
# Compile the model with learning rate and MAE metric
model.compile(loss='mean_squared_error', optimizer=Adam(learning_rate=0.001), me
```

In [12]:

```
# Train the model with validation data and early stopping
history = model.fit(X_train, y_train, epochs=100, batch_size=64, verbose=2,
                    validation_data=(X_val, y_val),
                    callbacks=[EarlyStopping(patience=10, restore_best_weights=T
```

```
Epoch 1/100
WARNING:tensorflow:From C:\Users\mrmua\New folder\lib\site-packages\keras\src\ut
ils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please
use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\mrmua\New folder\lib\site-packages\keras\src\en
gine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is
deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

7/7 - 7s - loss: 0.1524 - mae: 0.3414 - val_loss: 0.0490 - val_mae: 0.2158 - 7s/
epoch - 1s/step
Epoch 2/100
7/7 - 0s - loss: 0.0463 - mae: 0.1676 - val_loss: 0.0018 - val_mae: 0.0358 - 112
ms/epoch - 16ms/step
Epoch 3/100
7/7 - 0s - loss: 0.0207 - mae: 0.1143 - val_loss: 0.0090 - val_mae: 0.0855 - 122
ms/epoch - 17ms/step
Epoch 4/100
7/7 - 0s - loss: 0.0223 - mae: 0.1219 - val_loss: 0.0017 - val_mae: 0.0353 - 103
ms/epoch - 15ms/step
Epoch 5/100
7/7 - 0s - loss: 0.0142 - mae: 0.0940 - val_loss: 0.0035 - val_mae: 0.0489 - 98m
s/epoch - 14ms/step
Epoch 6/100
7/7 - 0s - loss: 0.0169 - mae: 0.1001 - val_loss: 0.0017 - val_mae: 0.0350 - 124
ms/epoch - 18ms/step
Epoch 7/100
7/7 - 0s - loss: 0.0149 - mae: 0.0947 - val_loss: 0.0020 - val_mae: 0.0369 - 110
ms/epoch - 16ms/step
Epoch 8/100
7/7 - 0s - loss: 0.0133 - mae: 0.0888 - val_loss: 0.0017 - val_mae: 0.0353 - 104
ms/epoch - 15ms/step
Epoch 9/100
7/7 - 0s - loss: 0.0121 - mae: 0.0846 - val_loss: 0.0021 - val_mae: 0.0375 - 109
ms/epoch - 16ms/step
Epoch 10/100
7/7 - 0s - loss: 0.0103 - mae: 0.0793 - val_loss: 0.0022 - val_mae: 0.0385 - 94m
s/epoch - 13ms/step
Epoch 11/100
7/7 - 0s - loss: 0.0105 - mae: 0.0775 - val_loss: 0.0018 - val_mae: 0.0358 - 119
ms/epoch - 17ms/step
Epoch 12/100
7/7 - 0s - loss: 0.0117 - mae: 0.0825 - val_loss: 0.0020 - val_mae: 0.0375 - 127
ms/epoch - 18ms/step
Epoch 13/100
7/7 - 0s - loss: 0.0111 - mae: 0.0808 - val_loss: 0.0019 - val_mae: 0.0360 - 125
ms/epoch - 18ms/step
Epoch 14/100
7/7 - 0s - loss: 0.0095 - mae: 0.0734 - val_loss: 0.0022 - val_mae: 0.0388 - 99m
s/epoch - 14ms/step
Epoch 15/100
7/7 - 0s - loss: 0.0110 - mae: 0.0794 - val_loss: 0.0029 - val_mae: 0.0446 - 120
ms/epoch - 17ms/step
Epoch 16/100
7/7 - 0s - loss: 0.0096 - mae: 0.0751 - val_loss: 0.0018 - val_mae: 0.0354 - 116
ms/epoch - 17ms/step
```

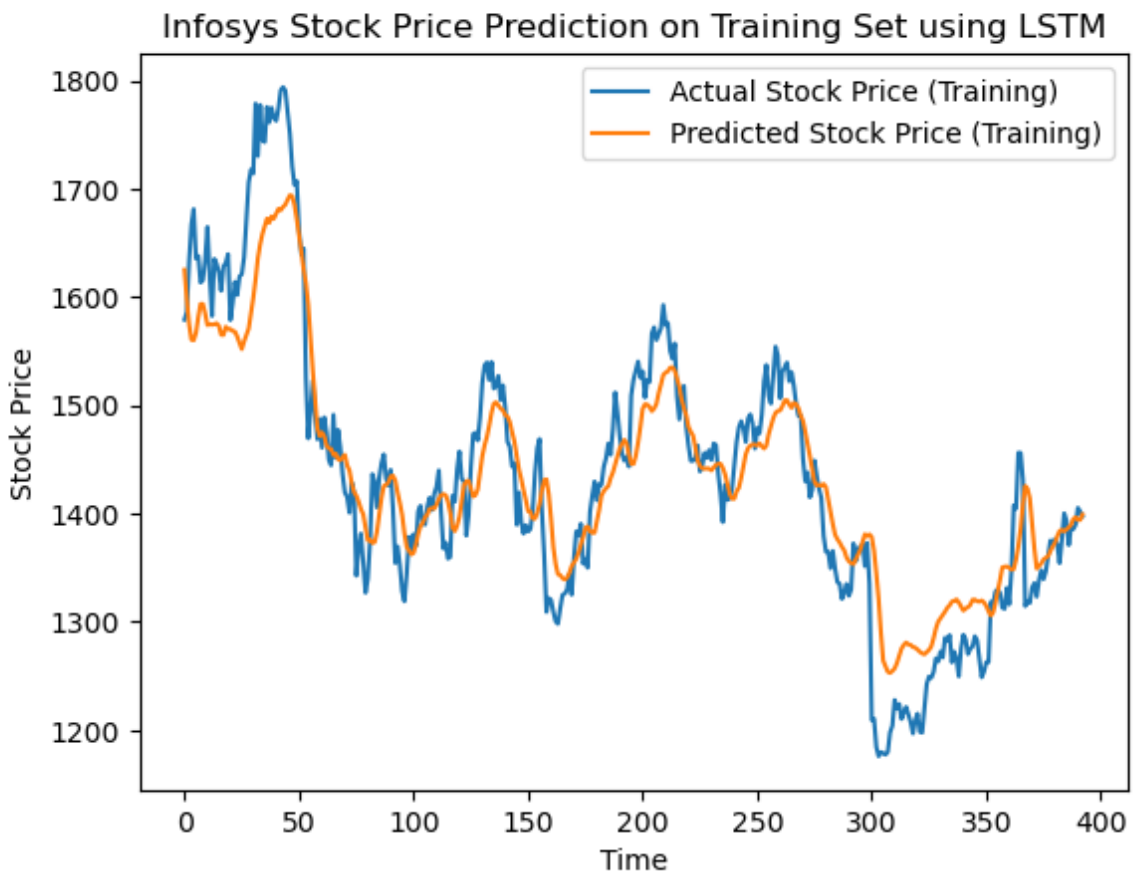
In [13]:
```
# Visualize the training and validation loss over epochs
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [14]:
```
# Evaluate on the training set
train_predictions = model.predict(X_train)
train_predictions = scaler.inverse_transform(train_predictions)
y_train_actual = scaler.inverse_transform(y_train.reshape(-1, 1))
```

```
13/13 [==============================] - 1s 4ms/step
```

In [15]:
```
# Visualize the predictions on the training set
plt.plot(y_train_actual, label='Actual Stock Price (Training)')
plt.plot(train_predictions, label='Predicted Stock Price (Training)')
plt.title('Infosys Stock Price Prediction on Training Set using LSTM')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```

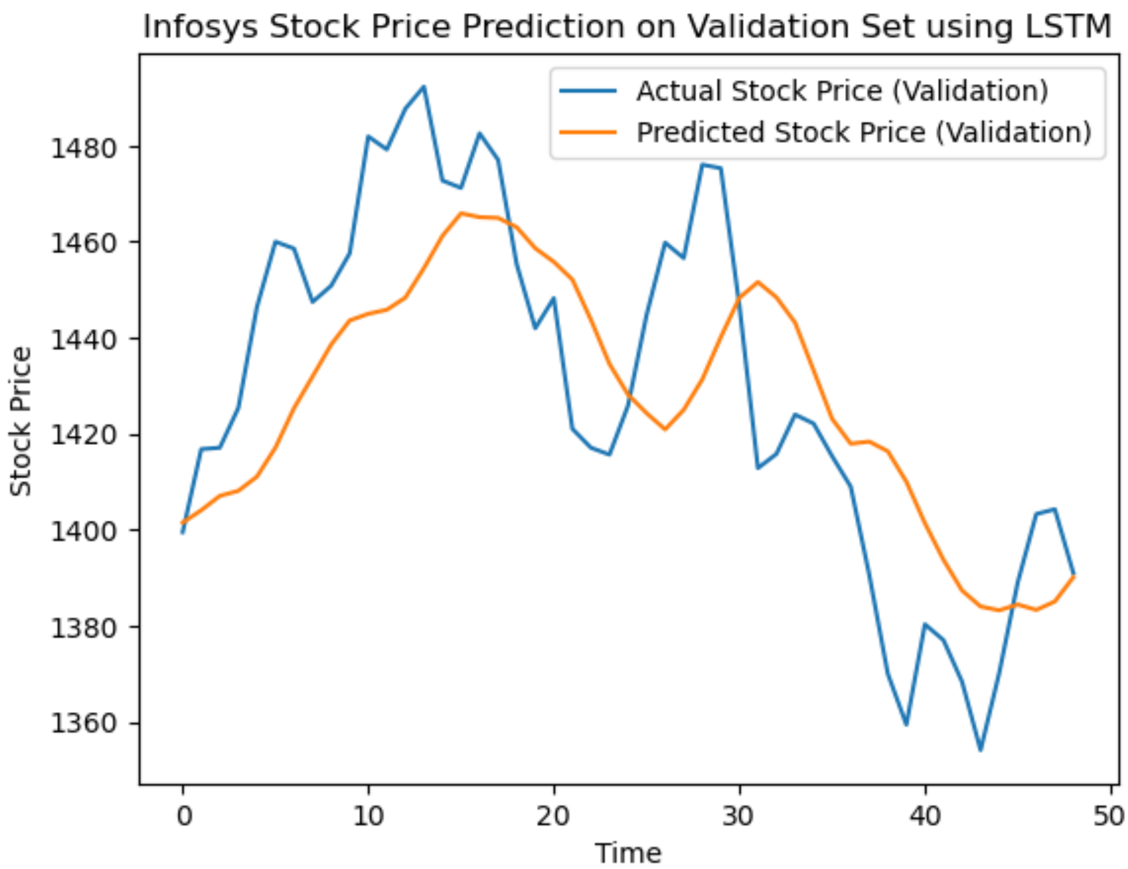


In [16]:
```python
# Calculate metrics for the training set
train_mae = mean_absolute_error(y_train_actual, train_predictions)
train_mse = mean_squared_error(y_train_actual, train_predictions)
train_r2 = r2_score(y_train_actual, train_predictions)*100

print('\nTraining Metrics:')
print(f'Mean Absolute Error (MAE): {train_mae:.2f}')
print(f'Mean Squared Error (MSE): {train_mse:.2f}')
print(f'R-squared (R2): {train_r2:.2f}%')
```

```
Training Metrics:
Mean Absolute Error (MAE): 39.02
Mean Squared Error (MSE): 2510.95
R-squared (R2): 85.69%
```

In [17]:
```python
# Evaluate on the validation set
val_predictions = model.predict(X_val)
val_predictions = scaler.inverse_transform(val_predictions)
y_val_actual = scaler.inverse_transform(y_val.reshape(-1, 1))

# Visualize the predictions on the validation set
plt.plot(y_val_actual, label='Actual Stock Price (Validation)')
plt.plot(val_predictions, label='Predicted Stock Price (Validation)')
plt.title('Infosys Stock Price Prediction on Validation Set using LSTM')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```

```
2/2 [==============================] - 0s 0s/step
```



In [18]:

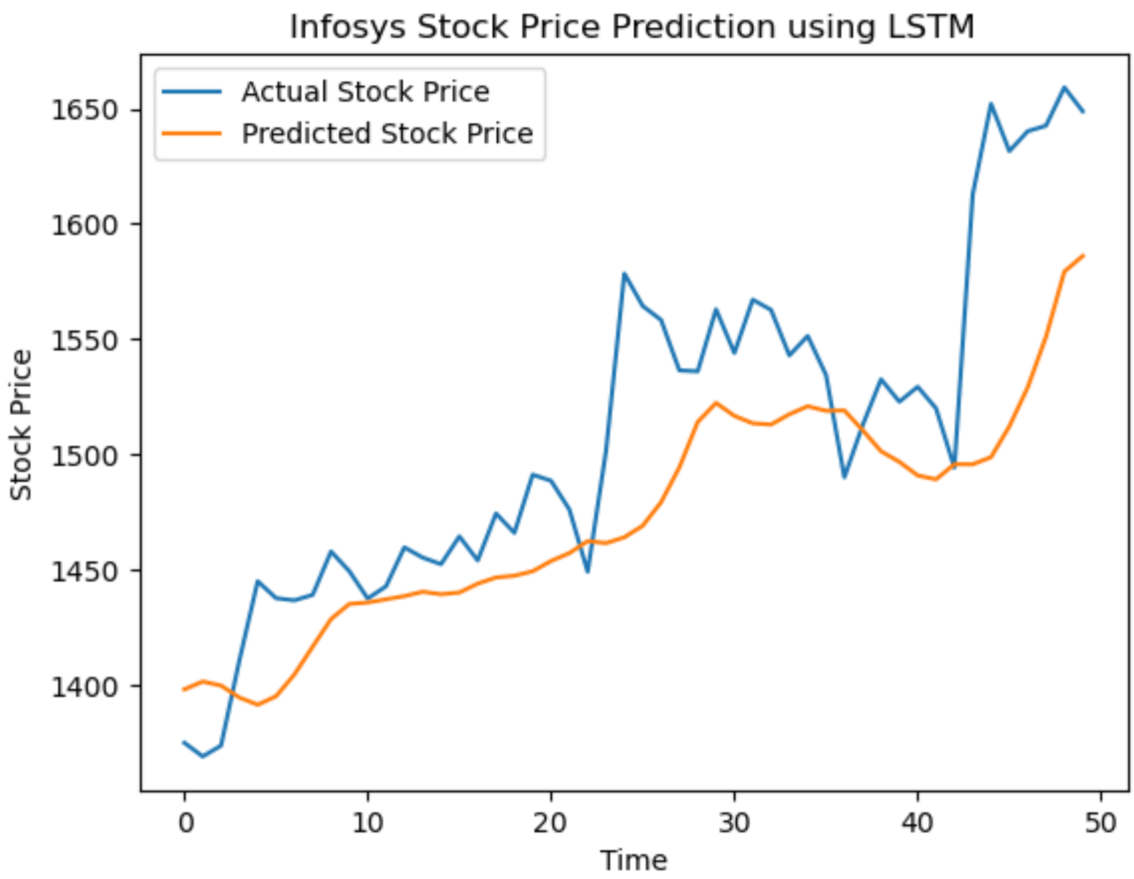
```
# Calculate metrics for the validation set
val_mae = mean_absolute_error(y_val_actual, val_predictions)
val_mse = mean_squared_error(y_val_actual, val_predictions)
val_r2 = r2_score(y_val_actual, val_predictions)*100

print('\nValidation Metrics:')
print(f'Mean Absolute Error (MAE): {val_mae:.2f}')
print(f'Mean Squared Error (MSE): {val_mse:.2f}')
print(f'R-squared (R2): {val_r2:.2f}%')
```

```
Validation Metrics:
Mean Absolute Error (MAE): 21.62
Mean Squared Error (MSE): 644.93
R-squared (R2): 53.38%
```

In [19]:

```
# Evaluate on the test set
test_predictions = model.predict(X_test)
test_predictions = scaler.inverse_transform(test_predictions)
y_test_actual = scaler.inverse_transform(y_test.reshape(-1, 1))

# Visualize the predictions on the test set
plt.plot(y_test_actual, label='Actual Stock Price')
plt.plot(test_predictions, label='Predicted Stock Price')
plt.title('Infosys Stock Price Prediction using LSTM')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```

```
2/2 [==============================] - 0s 16ms/step
```



In [20]:

```
# Evaluate different metrics on the test set
mae = mean_absolute_error(y_test_actual, test_predictions)
mse = mean_squared_error(y_test_actual, test_predictions)
rmse = np.sqrt(mse)
r2 = r2_score(y_test_actual, test_predictions) * 100

print('\nTest Set Metrics:')
print(f'Mean Absolute Error (MAE): {mae:.2f}')
print(f'Mean Squared Error (MSE): {mse:.2f}')
print(f'Root Mean Squared Error (RMSE): {rmse:.2f}')
print(f'R-squared (R2): {r2:.2f}%')
```

```
Test Set Metrics:
Mean Absolute Error (MAE): 41.38
Mean Squared Error (MSE): 2907.80
Root Mean Squared Error (RMSE): 53.92
R-squared (R2): 46.93%
```