Predicting Stock Prices using a Recurrent Neural Network

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout
import numpy as np
import matplotlib.pyplot as plt
```

Data Preprocessing

```
# Load the dataset
df = pd.read_csv("https://raw.githubusercontent.com/sumeir/data-
mining-project/main/DowJones.csv")

values = df["Value"].values.reshape(-1, 1)

# Normalize the values
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_values = scaler.fit_transform(values)

def create_sequences(data, sequence_length):
    X, y = [], []
    for i in range(len(data) - sequence_length):
        X.append(data[i : (i + sequence_length), 0])
        y.append(data[i + sequence_length, 0])
    return np.array(X), np.array(y)

sequence_length = 4
X, y = create_sequences(scaled_values, sequence_length)

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:]

# Reshaping for the LSTM layer
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

Training the Model

```
# Building the RNN model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True,
input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))  # 20% dropout
```

```python
model.add(LSTM(units=50))
model.add(Dropout(0.2))  # 20% dropout
model.add(Dense(1))

model.compile(optimizer="adam", loss="mean_squared_error")
model.fit(X_train, y_train, epochs=100, batch_size=32)
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/rnn/
rnn.py:199: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)

Epoch 1/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0412
Epoch 2/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - loss: 0.0012
Epoch 3/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 9.2929e-04
Epoch 4/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 8.8374e-04
Epoch 5/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 8.8938e-04
Epoch 6/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 7.9608e-04
Epoch 7/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 7.6572e-04
Epoch 8/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 7.6978e-04
Epoch 9/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 7.7768e-04
Epoch 10/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 7.3056e-04
Epoch 11/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 6.7829e-04
Epoch 12/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 6.4175e-04
Epoch 13/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5.6098e-04
Epoch 14/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 6.4701e-04
Epoch 15/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 6.2881e-04
Epoch 16/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5.6752e-04
Epoch 17/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 6.0792e-04
Epoch 18/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 6.1040e-04
Epoch 19/100
```

```
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 7.0396e-04
Epoch 20/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 5.4316e-04
Epoch 21/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 5.9858e-04
Epoch 22/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 6.3998e-04
Epoch 23/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 4.9927e-04
Epoch 24/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5.0628e-04
Epoch 25/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5.2089e-04
Epoch 26/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5.0977e-04
Epoch 27/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5.1292e-04
Epoch 28/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5.0180e-04
Epoch 29/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 4.8887e-04
Epoch 30/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 5.8777e-04
Epoch 31/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 5.0132e-04
Epoch 32/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 5.3041e-04
Epoch 33/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 5.2168e-04
Epoch 34/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 5.0385e-04
Epoch 35/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 5.1831e-04
Epoch 36/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 5.0505e-04
Epoch 37/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 5.0225e-04
Epoch 38/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 4.6794e-04
Epoch 39/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5.3056e-04
Epoch 40/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 4.9158e-04
Epoch 41/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 4.3769e-04
Epoch 42/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 4.9149e-04
Epoch 43/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 4.0623e-04
```

```
Epoch 44/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 4.0751e-04
Epoch 45/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 5.0144e-04
Epoch 46/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 4.6896e-04
Epoch 47/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 4.2438e-04
Epoch 48/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 4.4910e-04
Epoch 49/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 4.1082e-04
Epoch 50/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 4.2219e-04
Epoch 51/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 4.2388e-04
Epoch 52/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 4.5087e-04
Epoch 53/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 4.0986e-04
Epoch 54/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 4.1420e-04
Epoch 55/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 4.1259e-04
Epoch 56/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 4.3454e-04
Epoch 57/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 3.7076e-04
Epoch 58/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 4.4468e-04
Epoch 59/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 4.1068e-04
Epoch 60/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 4.7518e-04
Epoch 61/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 3.8848e-04
Epoch 62/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 3.9968e-04
Epoch 63/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 3.8026e-04
Epoch 64/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 3.7294e-04
Epoch 65/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 3.6289e-04
Epoch 66/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 4.2156e-04
Epoch 67/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 3.7839e-04
Epoch 68/100
```

```
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 4.3541e-04
Epoch 69/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 3.8431e-04
Epoch 70/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 3.4460e-04
Epoch 71/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 3.9795e-04
Epoch 72/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 3.5704e-04
Epoch 73/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 4.2344e-04
Epoch 74/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 4.1443e-04
Epoch 75/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 4.0988e-04
Epoch 76/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 3.9764e-04
Epoch 77/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 3.5054e-04
Epoch 78/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 3.7472e-04
Epoch 79/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 4.1332e-04
Epoch 80/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 4.3426e-04
Epoch 81/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 3.6221e-04
Epoch 82/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 3.5431e-04
Epoch 83/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 3.7245e-04
Epoch 84/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 3.9528e-04
Epoch 85/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 4.0122e-04
Epoch 86/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 3.5485e-04
Epoch 87/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 3.8182e-04
Epoch 88/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 3.8892e-04
Epoch 89/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 4.1214e-04
Epoch 90/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 3.3967e-04
Epoch 91/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 3.7456e-04
Epoch 92/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 3.7451e-04
```

```
Epoch 93/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 4.4930e-04
Epoch 94/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 3.8448e-04
Epoch 95/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 3.3965e-04
Epoch 96/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 3.9899e-04
Epoch 97/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 3.4636e-04
Epoch 98/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 3.6563e-04
Epoch 99/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 3.9023e-04
Epoch 100/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 3.7168e-04

<keras.src.callbacks.history.History at 0x78c41c6c1dc0>
```

```
# Making predictions
predicted_stock_price = model.predict(X_test)
predicted_stock_price = scaler.inverse_transform(
    predicted_stock_price
)  # Invert scaling

y_test = scaler.inverse_transform(y_test.reshape(-1, 1))  # Invert
scaling
y_sub = y_test.reshape(-1, 1)
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```

Evaluation

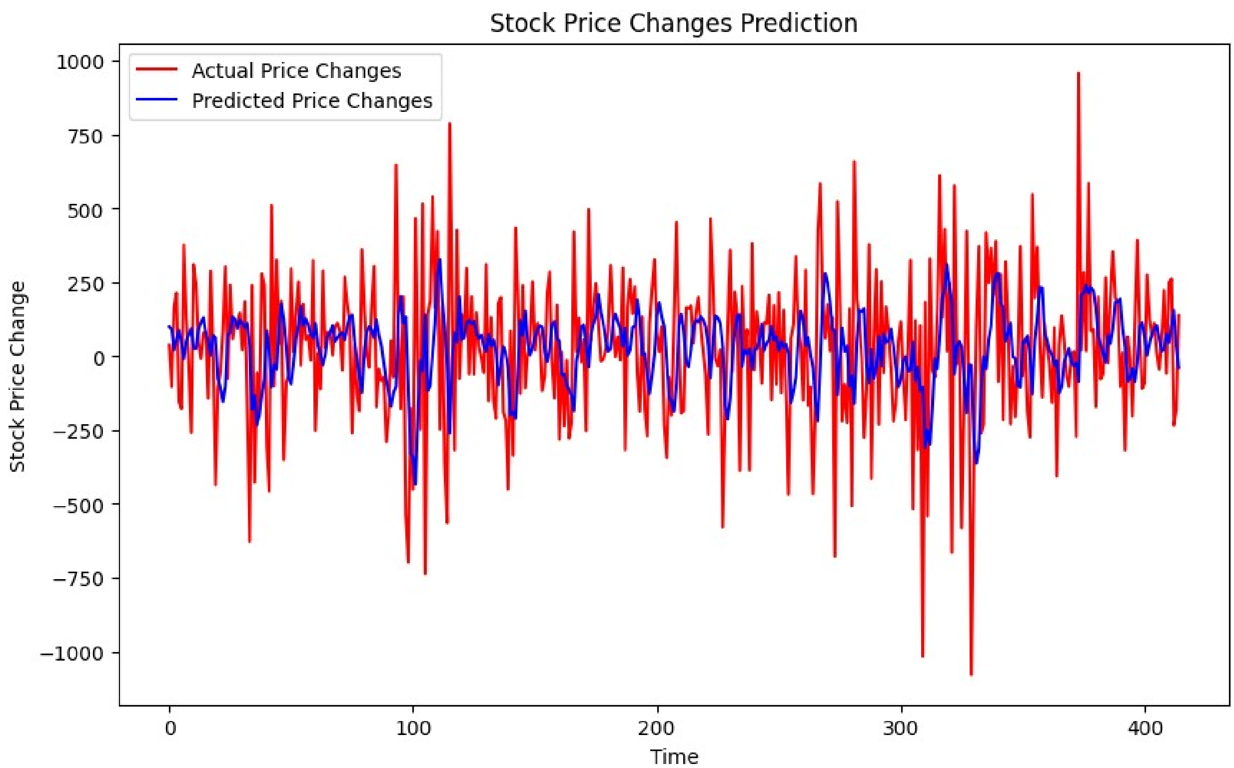
```
print(np.mean(np.abs((y_test - predicted_stock_price) / y_test)))

# Calculate deltas for actual and predicted prices
actual_deltas = np.diff(y_test.reshape(-1, 1), axis=0)
predicted_deltas = np.diff(predicted_stock_price, axis=0)

# Determine the sign (positive or negative) of each delta
actual_signs = actual_deltas > 0
predicted_signs = predicted_deltas > 0

# Count the number of times the deltas have the same sign
# Both positive or both negative
same_direction_count = np.sum((actual_signs ==
predicted_signs).astype(int))
print(same_direction_count)
```

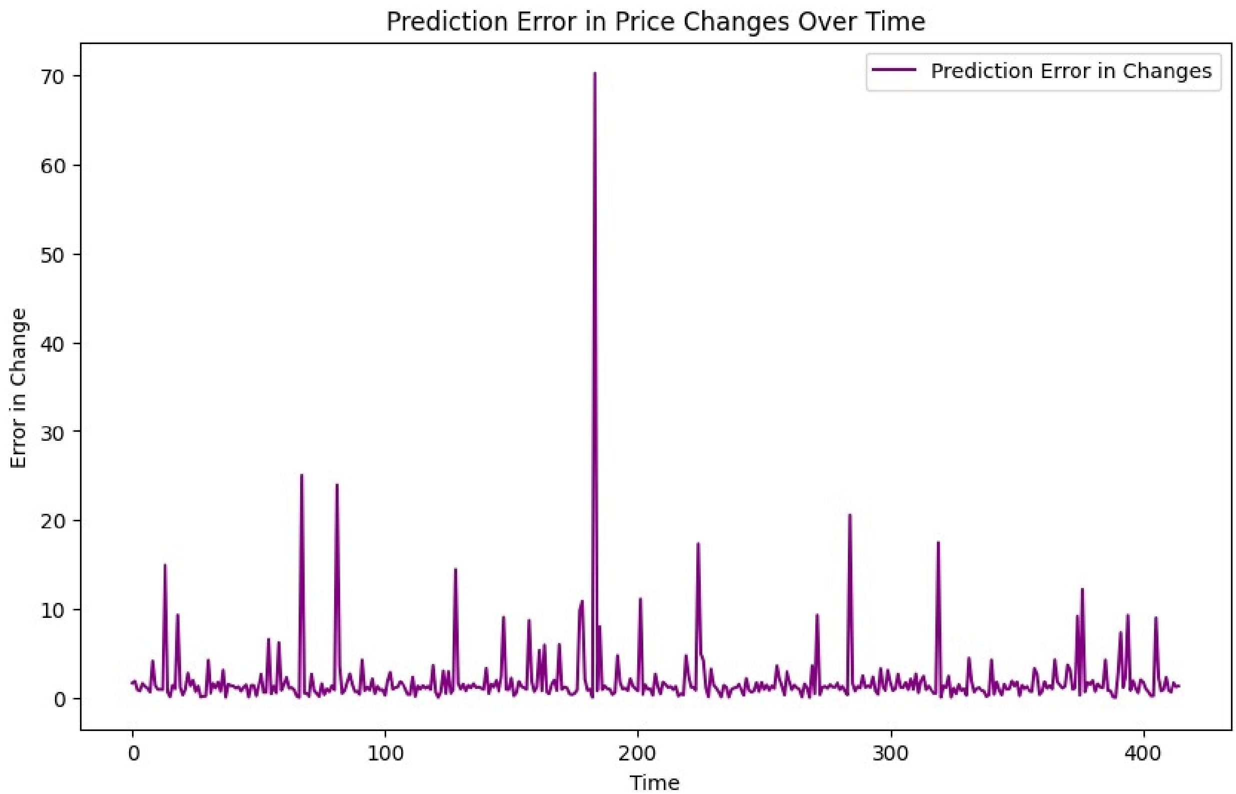
```python
# Calculate errors between predicted and actual deltas
delta_errors = np.abs(predicted_deltas - actual_deltas)
print("Mean Absolute Error for Deltas:", np.mean(delta_errors))

# Plotting actual vs predicted deltas
plt.figure(figsize=(10, 6))
plt.plot(actual_deltas, color='red', label='Actual Price Changes')
plt.plot(predicted_deltas, color='blue', label='Predicted Price
Changes')
plt.title('Stock Price Changes Prediction')
plt.xlabel('Time')
plt.ylabel('Stock Price Change')
plt.legend()
plt.show()

# Optionally, plot the error in deltas over time
plt.figure(figsize=(10, 6))
plt.plot(abs(delta_errors/actual_deltas), color='purple',
label='Prediction Error in Changes')
plt.title('Prediction Error in Price Changes Over Time')
plt.xlabel('Time')
plt.ylabel('Error in Change')
plt.legend()
plt.show()


plt.figure(figsize=(10, 6))
plt.hist(delta_errors, bins=50, color='skyblue', edgecolor='black')
plt.title('Histogram of Prediction Delta Errors')
plt.xlabel('Delta Error')
plt.ylabel('Frequency')
plt.show()

plt.figure(figsize=(10, 6))
plt.scatter(actual_deltas, predicted_deltas, color='green')
plt.title('Actual vs. Predicted Price Changes')
plt.xlabel('Actual Deltas')
plt.ylabel('Predicted Deltas')
plt.axline((0, 0), slope=1, color="red", linestyle="--")  # Adds a
reference line for perfect predictions
plt.grid(True)
plt.show()

directional_agreement = [same_direction_count, len(actual_deltas) -
same_direction_count]

plt.figure(figsize=(8, 5))
bar_labels = ['Same Direction', 'Different Direction']
plt.bar(bar_labels, directional_agreement, color=['blue', 'orange'])
plt.title('Directional Agreement of Price Changes')
```

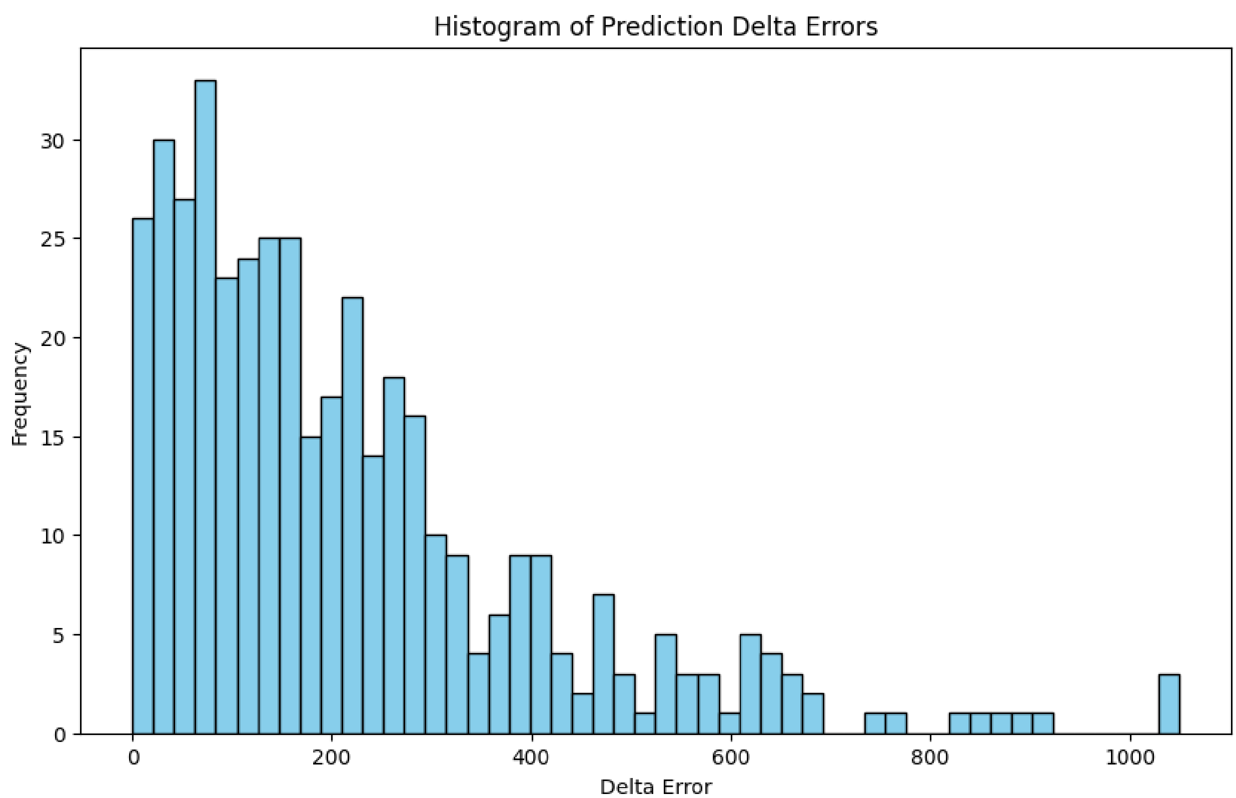
```
plt.ylabel('Count')
plt.show()

# Identifying indices where directions are the same and where they are different
same_direction_indices = np.where(actual_signs == predicted_signs)[0]
different_direction_indices = np.where(actual_signs != predicted_signs)[0]

# Separating delta errors based on the direction agreement
errors_same_direction = delta_errors[same_direction_indices]
errors_different_direction = delta_errors[different_direction_indices]

# Plotting histograms
plt.figure(figsize=(12, 6))
plt.hist(errors_same_direction, bins=50, alpha=0.5, label='Same Direction', color='green')
plt.hist(errors_different_direction, bins=50, alpha=0.5, label='Different Direction', color='red')
plt.title('Comparison of Predicted Delta Errors: Same vs. Different Directions')
plt.xlabel('Predicted Delta Error')
plt.ylabel('Frequency')
plt.legend()
plt.show()


relative_errors = np.where(actual_deltas != 0, delta_errors / np.abs(actual_deltas), 0)

plt.figure(figsize=(10, 6))
plt.plot(relative_errors, color='purple', label='Relative Prediction Error in Changes')
plt.title('Relative Prediction Error in Price Changes Over Time')
plt.xlabel('Time')
plt.ylabel('Relative Error in Change')
plt.legend()
plt.show()


# Initialize profit and cumulative profit lists
profit = []
cumulative_profit = []

# Simulate trading strategy
for i in range(len(predicted_deltas)):
    # Assuming we 'buy' if the prediction is for the price to go up, and 'sell' otherwise
    # Profit is calculated as the actual change in price
    profit.append(actual_deltas[i] * (predicted_deltas[i] > 0))
```

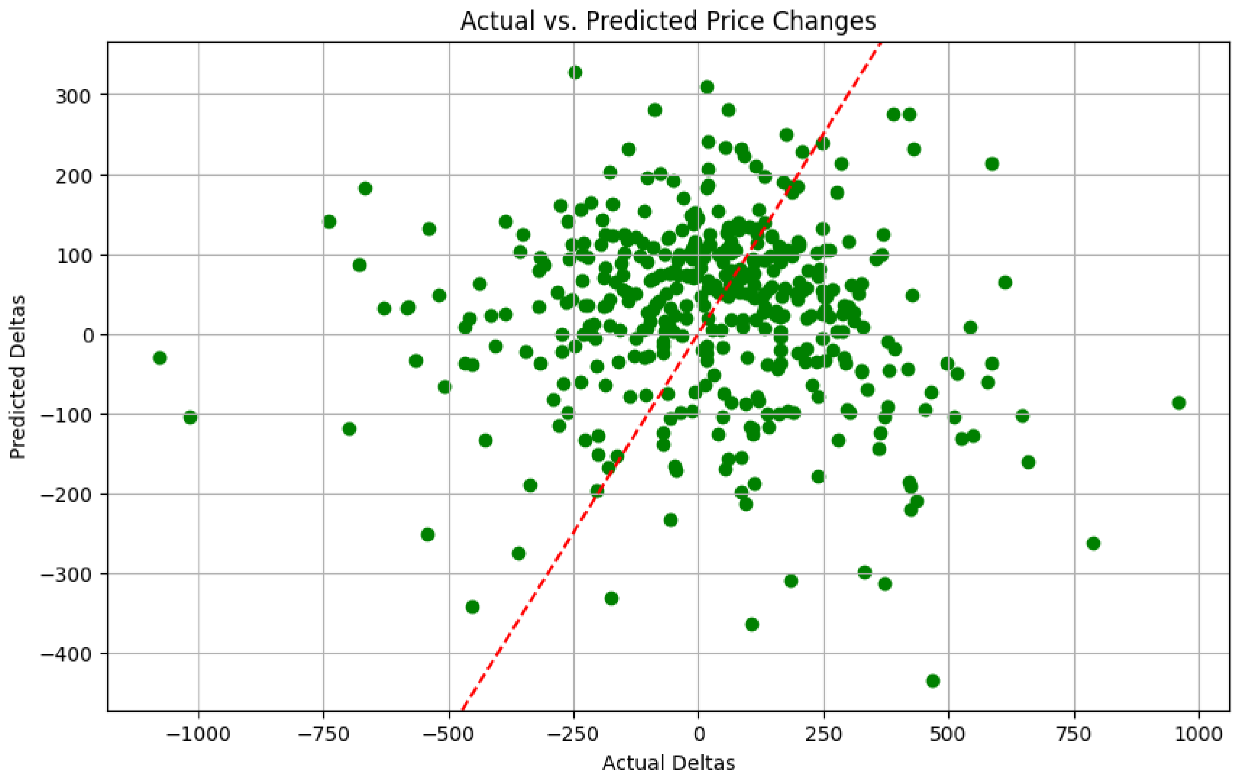
```
    cumulative_profit.append(np.sum(profit))

# Convert lists to numpy arrays for easier manipulation
profit = np.array(profit)
cumulative_profit = np.array(cumulative_profit)

# Plotting cumulative profit over time
plt.figure(figsize=(10, 6))
plt.plot(cumulative_profit, color='blue', label='Cumulative Profit')
plt.title('Cumulative Profit Over Time')
plt.xlabel('Time')
plt.ylabel('Cumulative Profit')
plt.legend()
plt.show()

# Printing the final cumulative profit
print(f"Final Cumulative Profit: {cumulative_profit[-1]}")
```

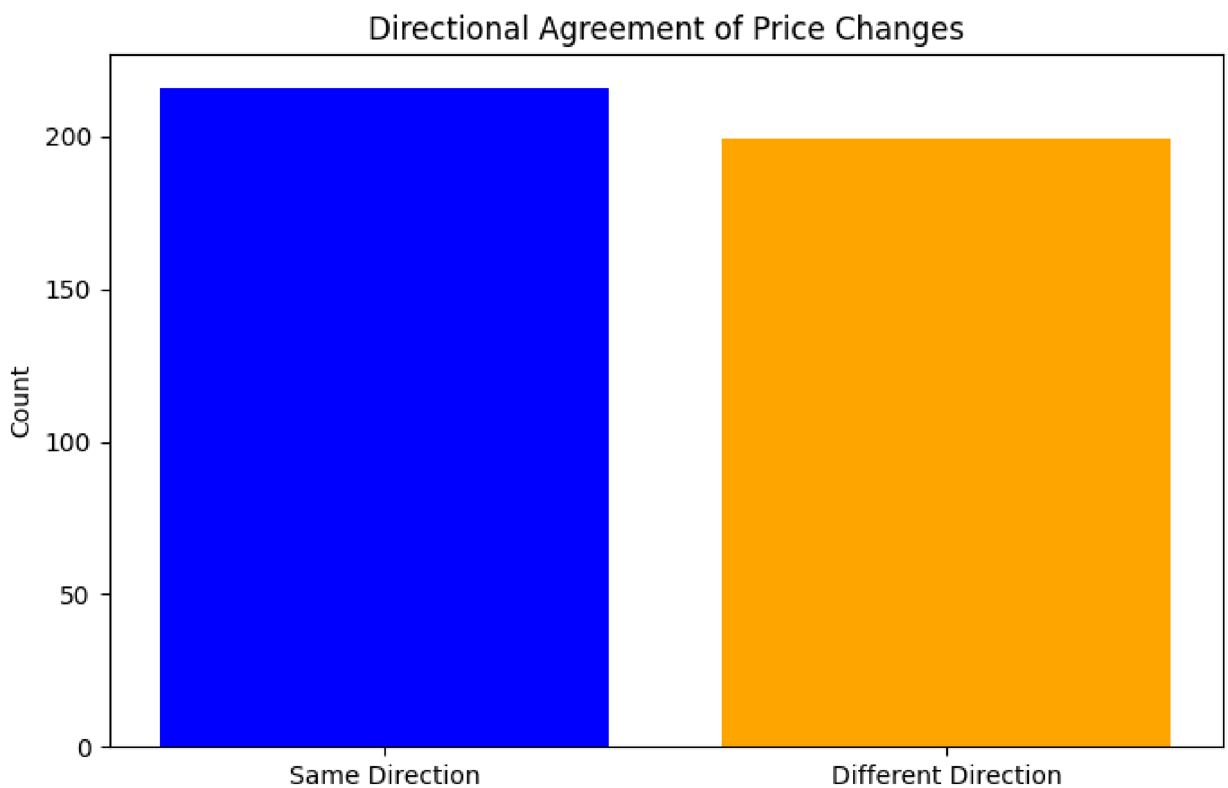
```
0.019488935202633832
216
Mean Absolute Error for Deltas: 217.60921187876508
```

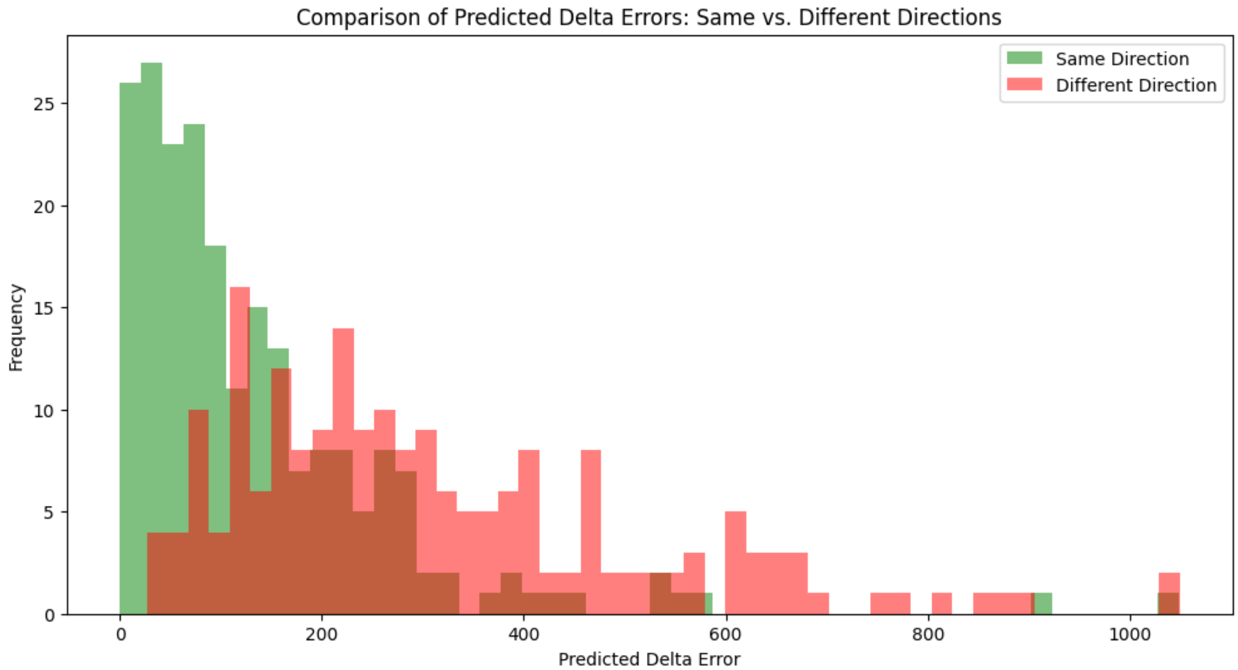


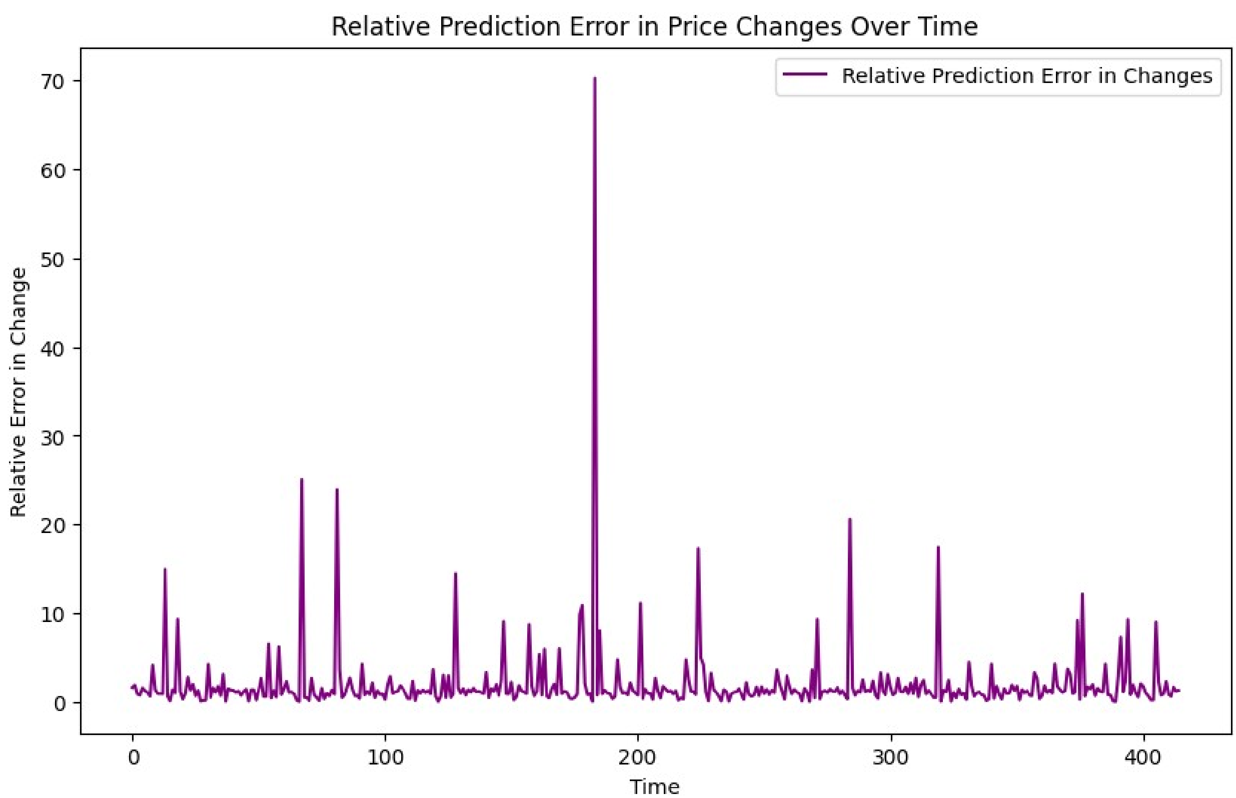

Prediction Error in Price Changes Over Time
Prediction Error in Changes
Error in Change
Time
0
10
20
30
40
50
60
70
0
100
200
300
400

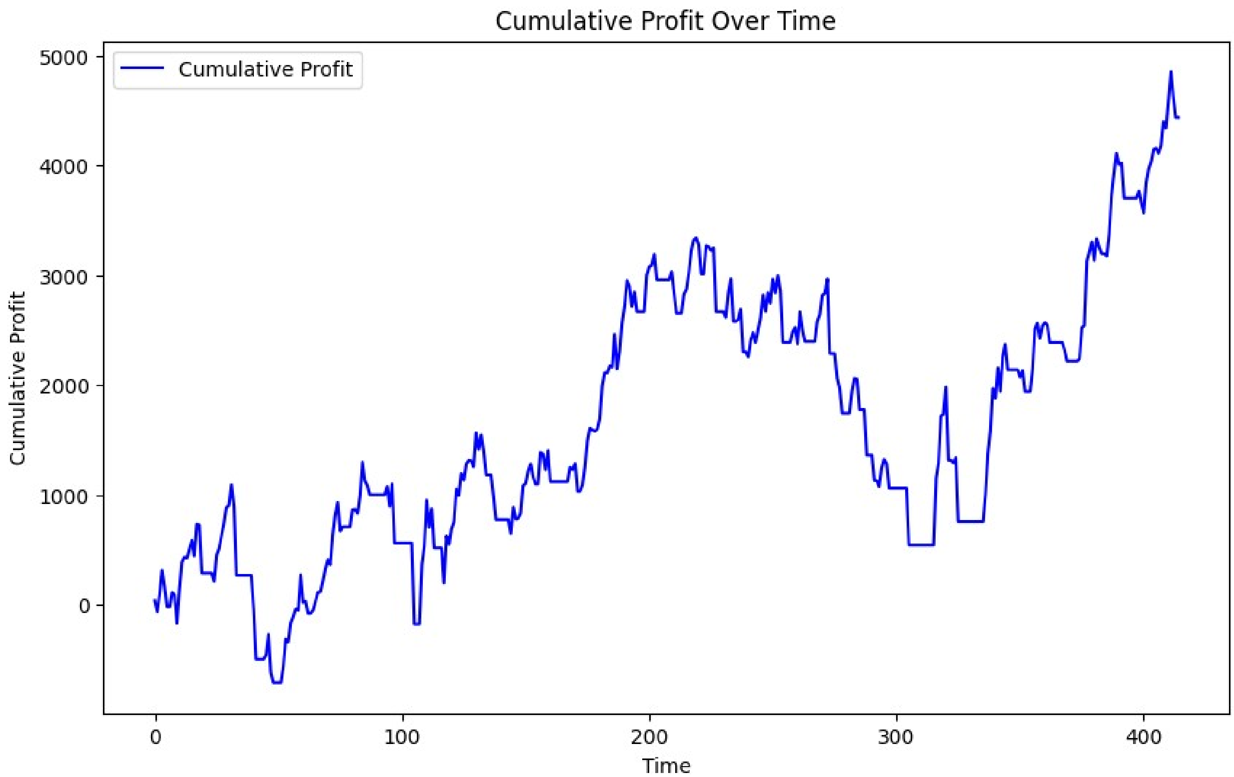

Histogram of Prediction Delta Errors
Frequency
Delta Error
0
5
10
15
20
25
30
0
200
400
600
800
1000


Actual vs. Predicted Price Changes
300
200
100
0
−100
−200
−300
−400
Predicted Deltas
−1000
−750
−500
−250
0
250
500
750
1000
Actual Deltas


Directional Agreement of Price Changes
200
150
100
50
0
Count
Same Direction
Different Direction


Comparison of Predicted Delta Errors: Same vs. Different Directions
Same Direction
Different Direction
Frequency
Predicted Delta Error
0
5
10
15
20
25
0
200
400
600
800
1000

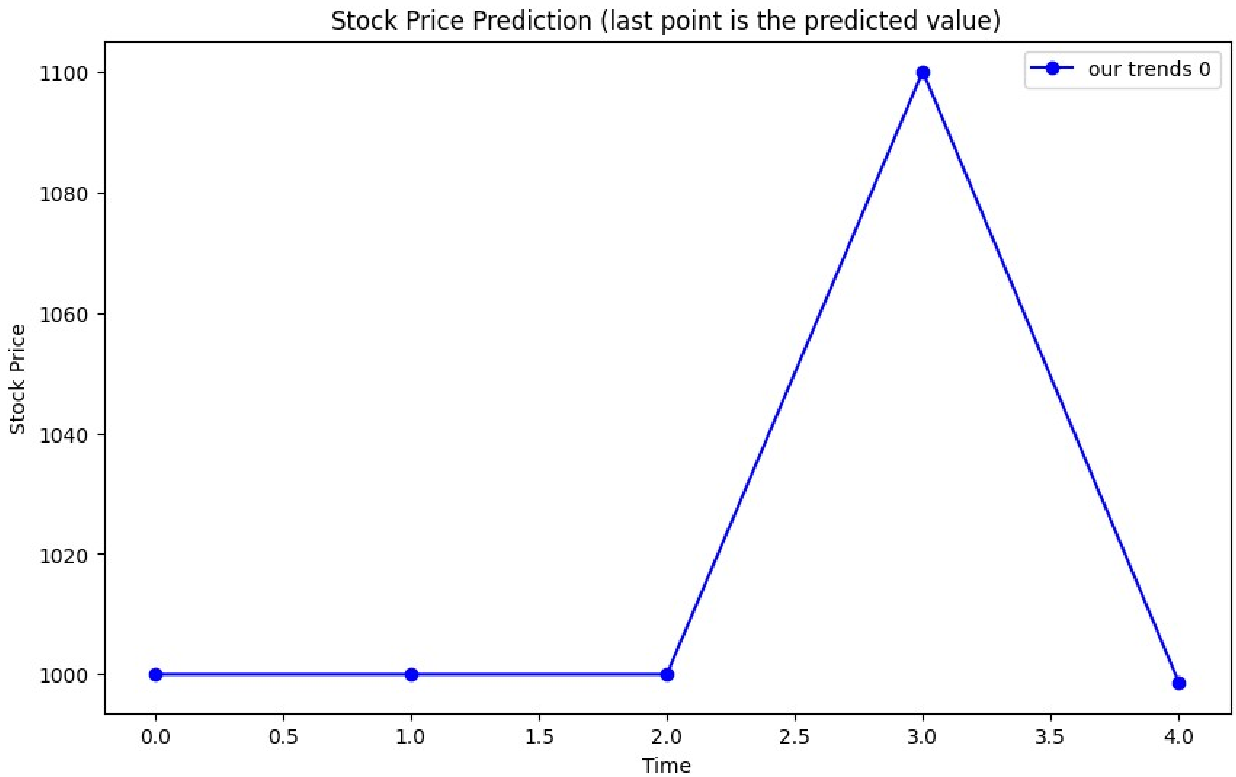

Relative Prediction Error in Price Changes Over Time
Relative Prediction Error in Changes
Relative Error in Change
Time
0
10
20
30
40
50
60
70
0
100
200
300
400

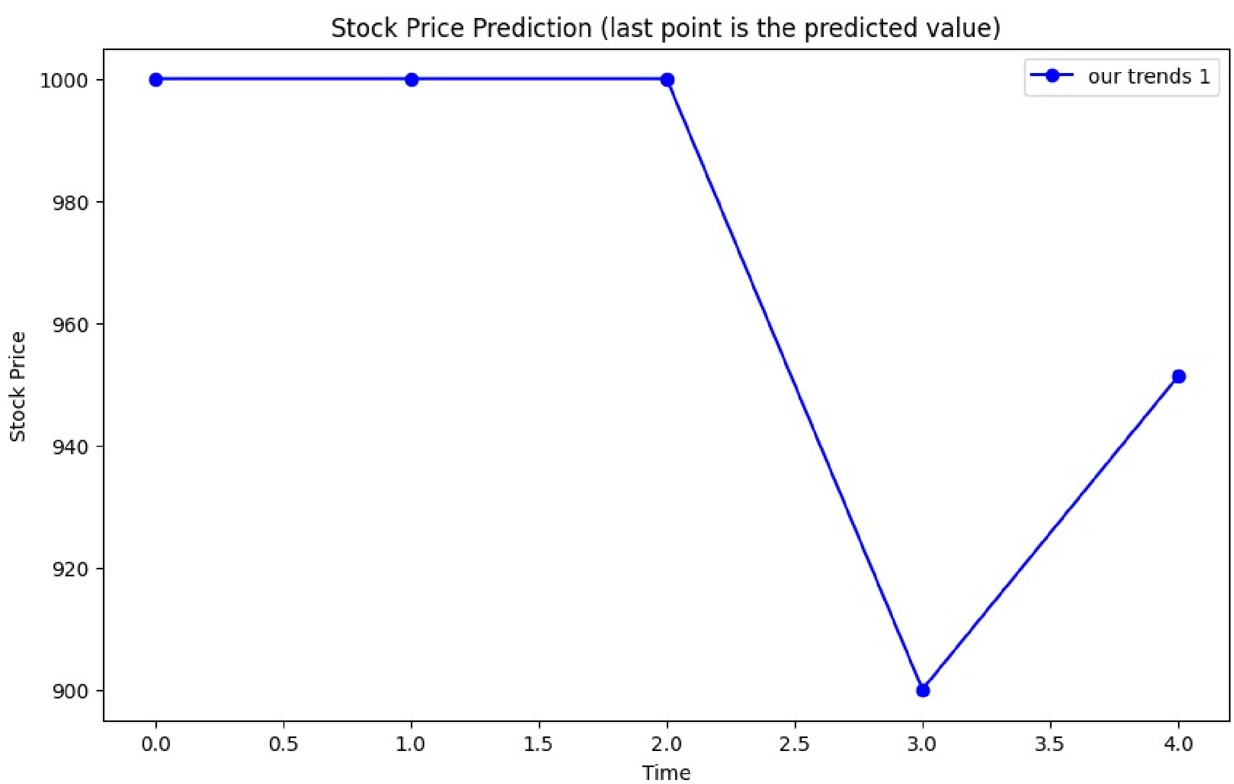


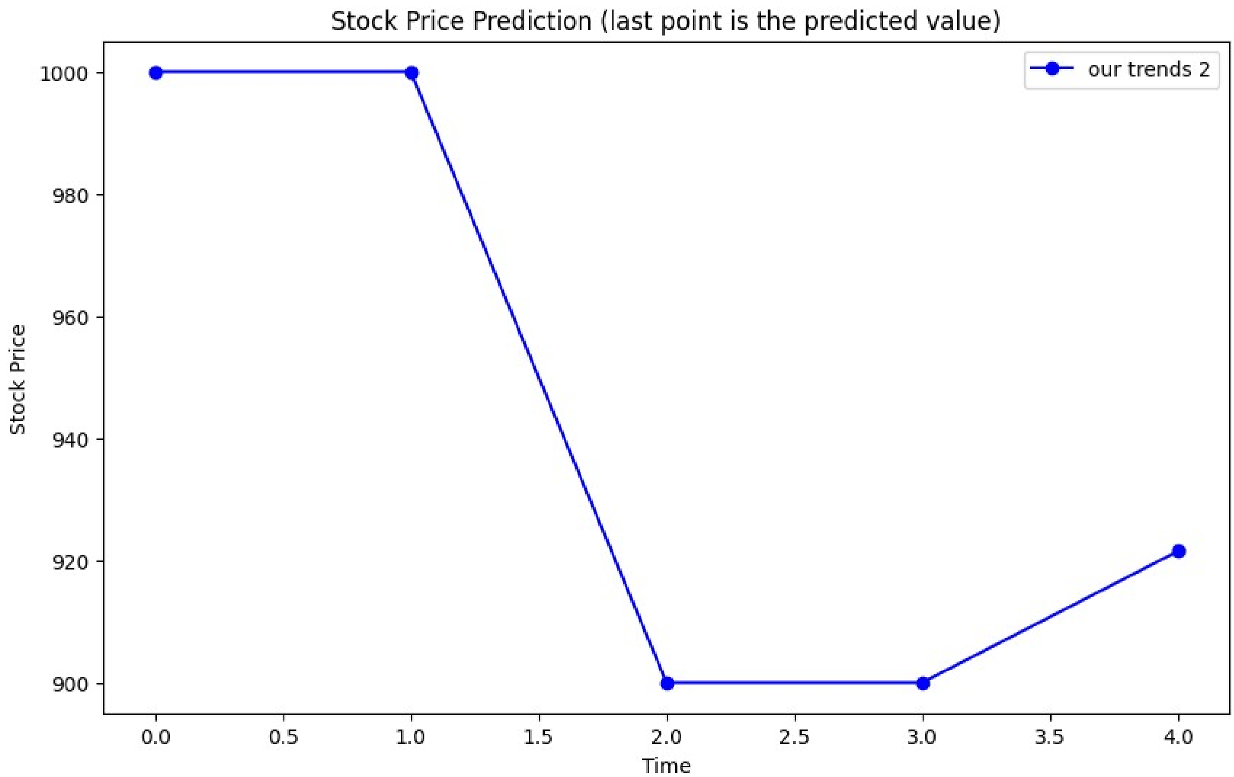

```
Final Cumulative Profit: 4434.059999999994

# Define your data
data = [
    [[1000], [1000], [1000], [1100]],
    [[1000], [1000], [1000], [900]],
    [[1000], [1000], [900], [900]],
    [[1000], [900], [900], [950]],
]

# Convert list to numpy array
np_array = np.array(data)
np_array_reshaped = np_array.reshape(-1, 1)
np_array_scaled = scaler.transform(np_array_reshaped)
np_array_scaled = np_array_scaled.reshape(np_array.shape)


predicted_stock_price = model.predict(np_array_scaled)

np_array_scaled_reshaped = np_array_scaled.reshape(-1, 1)
predicted_stock_price =
scaler.inverse_transform(predicted_stock_price)

ourTrends = data
for i in range(len(predicted_stock_price)):
    ourTrends[i].append([predicted_stock_price[i][0]])
```

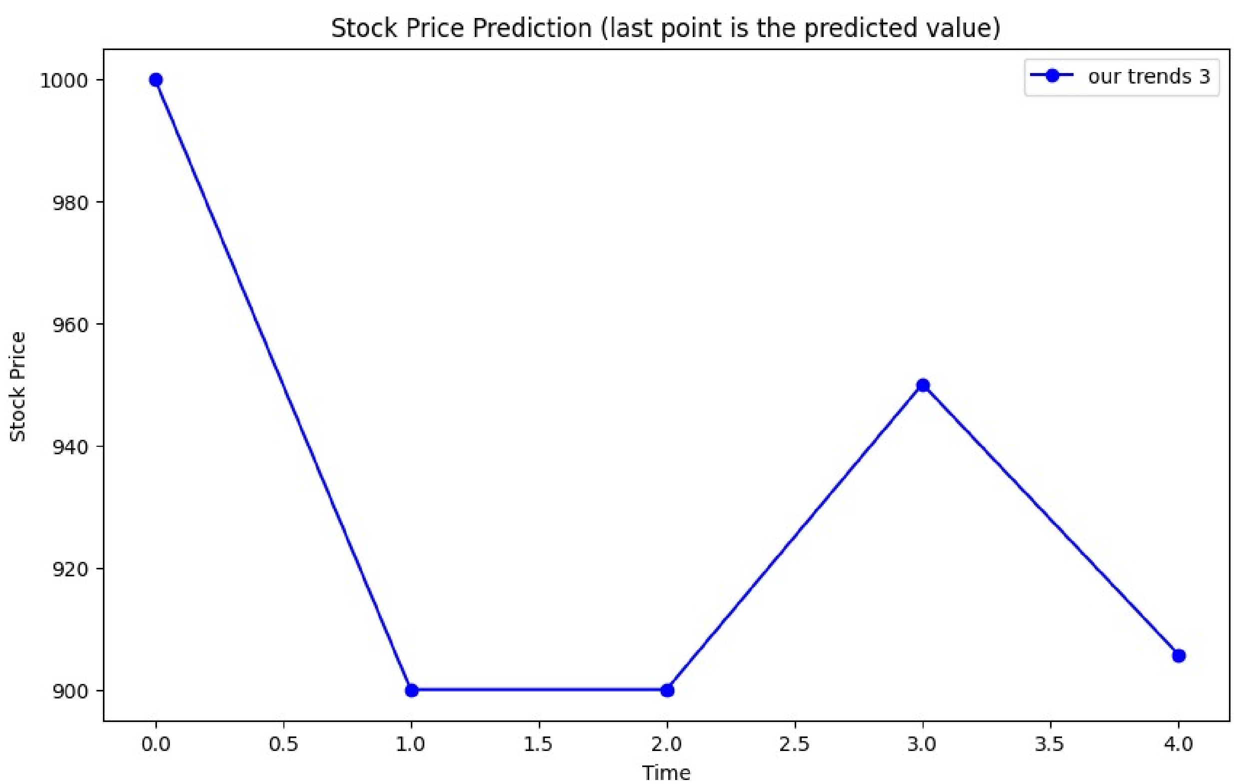
```python
# Plotting the data and predictions
plt.figure(figsize=(10, 6))
plt.plot(ourTrends[0], color="blue", label="our trends 0", marker="o")
plt.title("Stock Price Prediction (last point is the predicted
value)")
plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.legend()
plt.show()
plt.figure(figsize=(10, 6))
plt.plot(ourTrends[1], color="blue", label="our trends 1", marker="o")
plt.title("Stock Price Prediction (last point is the predicted
value)")
plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.legend()
plt.show()
plt.figure(figsize=(10, 6))
plt.plot(ourTrends[2], color="blue", label="our trends 2", marker="o")
plt.title("Stock Price Prediction (last point is the predicted
value)")
plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.legend()
plt.show()
plt.figure(figsize=(10, 6))
plt.plot(ourTrends[3], color="blue", label="our trends 3", marker="o")
plt.title("Stock Price Prediction (last point is the predicted
value)")
plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.legend()
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 295ms/step
```


Stock Price Prediction (last point is the predicted value)
our trends 0
1100
1080
1060
1040
1020
1000
Stock Price
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Time


Stock Price Prediction (last point is the predicted value)
our trends 1
1000
980
960
940
920
900
Stock Price
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Time


Stock Price Prediction (last point is the predicted value)
our trends 2
1000
980
960
940
920
900
Stock Price
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Time


Stock Price Prediction (last point is the predicted value)
our trends 3
1000
980
960
940
920
900
Stock Price
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Time