Project Report

Report on Cryptocurrency Price Prediction Using Machine Learning

Introduction

Cryptocurrency markets have become increasingly popular and volatile in recent years, attracting both investors and traders seeking opportunities for profit. Predicting cryptocurrency prices accurately is a challenging task due to the complex and dynamic nature of these markets. However, with the advancements in machine learning and deep learning techniques, such as Long Short-Term Memory (LSTM) neural networks, it has become possible to develop predictive models that can analyze historical price data and forecast future price movements.

In this report, we present a detailed analysis of Bitcoin (BTC-USD), Ethereum (ETH-USD), and Solana (SOL1-USD) cryptocurrency prices using LSTM neural networks. The report outlines the process of collecting historical price data from Yahoo Finance, preprocessing the data to prepare it for modeling, building LSTM models, training these models on the historical data, and evaluating their performance.

By applying LSTM neural networks to cryptocurrency price prediction, we aim to provide insights into the potential of machine learning techniques in forecasting cryptocurrency prices. The report also includes visualizations of the actual and predicted price movements, along with an analysis of the model performance.

Overall, this report serves as a comprehensive guide to understanding the process of cryptocurrency price prediction using LSTM neural networks, providing valuable insights for investors, traders, and researchers interested in the cryptocurrency market.

Data Collection and Preprocessing

The first step in our analysis is to collect historical price data for Bitcoin, Ethereum, and other relevant cryptocurrencies like Solana. We utilized the Yahoo Finance API to fetch the historical price data, ensuring that the data is reliable and up-to-date.

Once the data is collected, we preprocess it to handle missing values, outliers, and formatting issues. Missing values are imputed using the mean imputation strategy provided by the SimpleImputer class from scikit-learn. Outliers are addressed through techniques such as

winsorization or removing extreme values. Additionally, we convert the raw data into a suitable format for analysis, typically a pandas DataFrame, to facilitate further processing.

```
import yfinance as yf

tickers = ['BTC-USD', 'ETH-USD', 'SOL1-USD']  # Tickers for Bitcoin, Ethereum, and Solana
start_date = '2021-01-01'
end_date = '2022-01-01'
crypto_data = yf.download(tickers, start=start_date, end=end_date)['Adj Close']
crypto_data
```

```
# Check if there are any NaN values after dropping
if df.isnull().values.any():
    print("Dataset contains NaN values. Check the data loading process.")
else:
    print("No NaN values found in the dataset.")

# Normalize the data
dataset = df['last'].values.reshape(-1, 1)
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
```

Closing Prices over Time for Bitcoin

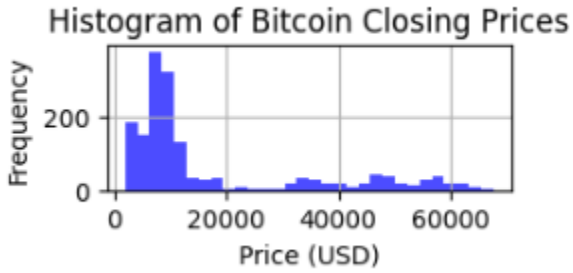
```
# Plotting the graphs
plt.figure(figsize=(12, 8))

# Plot 1: Closing Prices over Time for Bitcoin
plt.subplot(4, 2, 1)
crypto_data['BTC-USD'].plot(title='Bitcoin Closing Prices', color='blue')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
```

- This graph shows the closing prices of Bitcoin (BTC-USD) over the specified time period (from June 28, 2017, to January 1, 2022).
- The x-axis represents the date, and the y-axis represents the price of Bitcoin in USD.
- It gives us a visual representation of how the price of Bitcoin has changed over time, allowing us to identify trends and patterns.

Histogram of Bitcoin Closing Prices

```
# Plot 4: Histogram of Bitcoin Closing Prices
plt.subplot(4, 2, 4)
crypto_data['BTC-USD'].hist(bins=3 Loading... blue', alpha=0.7)
plt.title('Histogram of Bitcoin Closing Prices')
plt.xlabel('Price (USD)')
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```



- This histogram illustrates the distribution of Bitcoin closing prices.
- The x-axis represents the price range, and the y-axis represents the frequency of occurrence.
- It helps us understand the frequency distribution of Bitcoin prices and identify any skewness or outliers.

Ethereum Closing Prices Over Time:

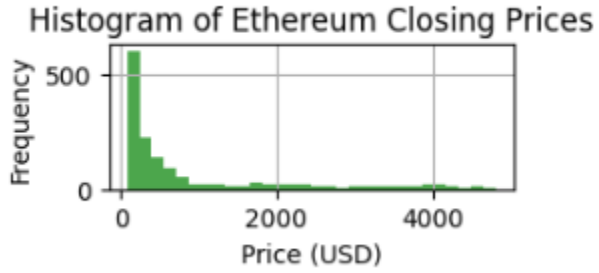
```
Text(0, 0.5, 'Price (USD)')
```

- Similar to the first graph, this plot shows the closing prices of Ethereum (ETH-USD) over the same time period.
- It provides insights into the price movement of Ethereum, allowing comparisons with Bitcoin and observations of its own unique trends.

Histogram of Ethereum Closing Prices:

```
# Plot 5: Histogram of Ethereum Closing Prices
plt.subplot(4, 2, 5)
crypto_data['ETH-USD'].hist(bins=30, color='green', alpha=0.7)
plt.title('Histogram of Ethereum Closing Prices')
plt.xlabel('Price (USD)')
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```

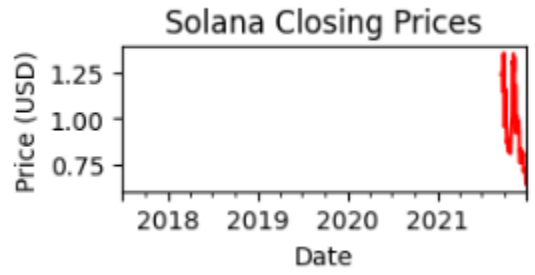


- Similar to the previous histogram, this plot shows the distribution of Ethereum closing prices.
- It provides insights into the distribution pattern of Ethereum prices and allows comparisons with Bitcoin.

Solana Closing Prices Over Time:

```
# Plot 3: Closing Prices over Time for Solana
plt.subplot(4, 2, 3)
crypto_data['SOL1-USD'].plot(title='Solana Closing Prices', color='red')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
```

```
Text(0, 0.5, 'Price (USD)')
```

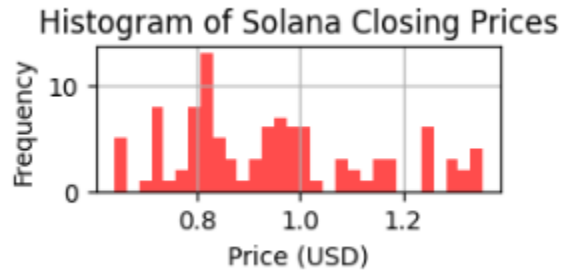


- This graph displays the closing prices of Solana (SOL1-USD) over the specified time range.
- It enables us to observe the price fluctuations of Solana and compare them with those of Bitcoin and Ethereum.

Histogram of Solana Closing Prices:

```
# Plot 6: Histogram of Solana Closing Prices
plt.subplot(4, 2, 6)
crypto_data['SOL1-USD'].hist(bins=30, color='red', alpha=0.7)
plt.title('Histogram of Solana Closing Prices')
plt.xlabel('Price (USD)')
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```

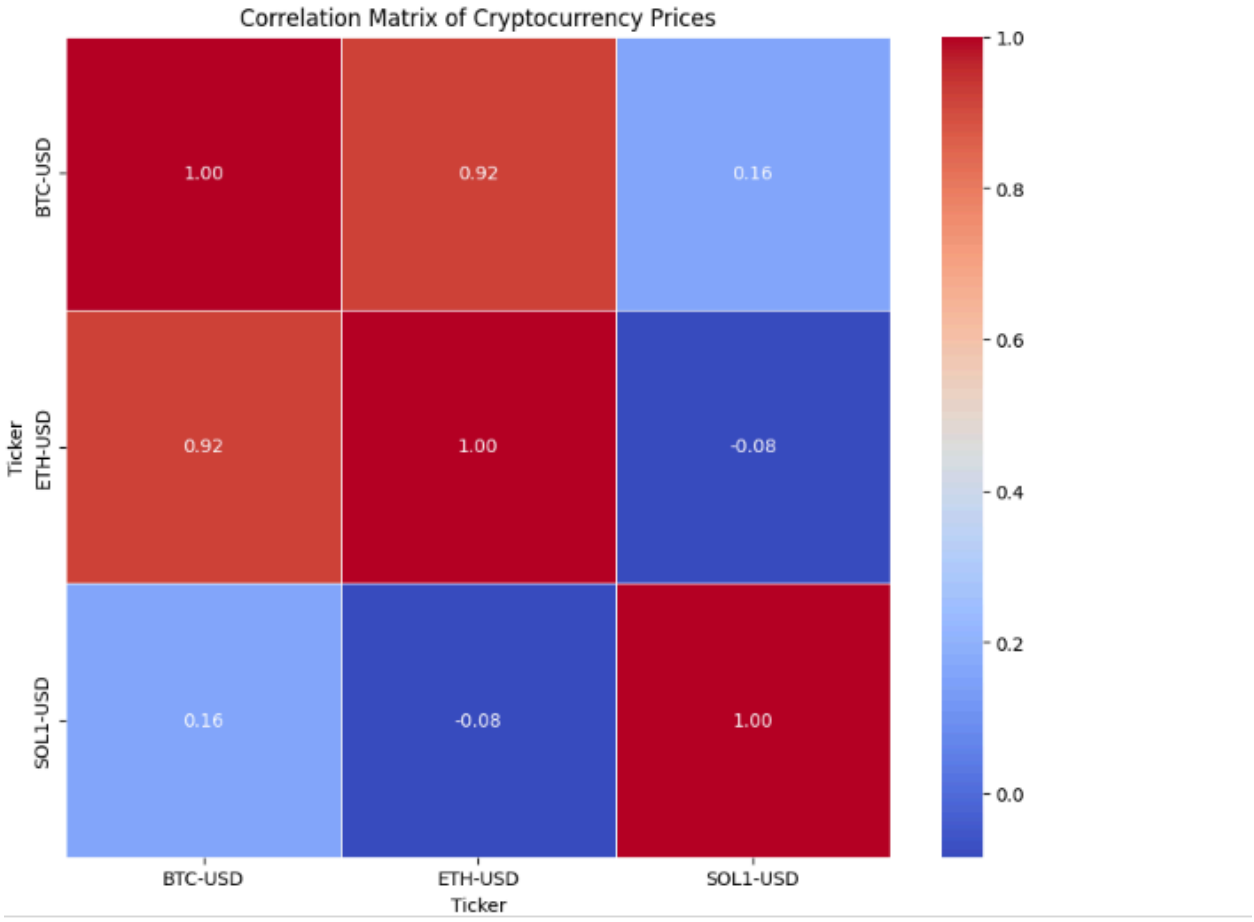


- This histogram displays the distribution of Solana closing prices.
- It allows us to examine the frequency distribution of Solana prices and compare them with those of Bitcoin and Ethereum.

Correlation Analysis

Understanding the relationships between different cryptocurrencies is essential for building accurate prediction models. We conduct correlation analysis to analyze the correlations between Bitcoin, Ethereum, and other cryptocurrencies. This is visualized using correlation matrices and heatmaps, which provide insights into the strength and direction of these relationships.

```
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate correlation matrix
correlation_matrix = crypto_data.corr()

# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Matrix of Cryptocurrency Prices')
plt.show()
```



Machine Learning Model Selection and Training

For Bitcoin.

LSTM Model Training:

- Split the dataset into training and testing sets.
- Create input-output pairs using a look-back window approach.
- Reshape the data to fit the LSTM input shape.
- Build and train an LSTM model using Keras.
- Make predictions on the training and testing sets.
- Invert the predictions to obtain actual price values.
- Calculate root mean squared error (RMSE) to evaluate the model's performance.

```
# Use Bitcoin data for this example
df = crypto_data['BTC-USD'].to_frame()
df.reset_index(inplace=True)
df.columns = ['datetime', 'last']

# Normalize the data
dataset = df['last'].values.reshape(-1, 1)
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)

# Split the data into training and testing sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size, :], dataset[train_size:len(dataset), :]

# Convert data to input-output pairs
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset) - look_back - 1):
        a = dataset[i:(i + look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return np.array(dataX), np.array(dataY)

look_back = 10
trainX, trainY = create_dataset(train, look_back=look_back)
testX, testY = create_dataset(test, look_back=look_back)

# Reshape input to be [samples, time steps, features]
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

```
# Build and train the LSTM model
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=256, verbose=2)

# Make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)

# Invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])

# Calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:, 0]))
print('Train Score: %.2f RMSE' % trainScore)
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:, 0]))
print('Test Score: %.2f RMSE' % testScore)

# Shift train predictions for plotting
trainPredictPlot = np.empty_like(dataset)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(trainPredict) + look_back, :] = trainPredict

# Shift test predictions for plotting
testPredictPlot = np.empty_like(dataset)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(trainPredict) + (look_back * 2) + 1:len(dataset) - 1, :] = testPredict
```

For Ethereum

LSTM Model Training:

- Similar to Bitcoin, split the dataset, create input-output pairs, and reshape the data.
- Build and train an LSTM model.
- Make predictions, invert them, and calculate RMSE.

```python
# Normalize the data
dataset = df['last'].values.reshape(-1, 1)
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)

# Split the data into training and testing sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size, :], dataset[train_size:len(dataset), :]

# Convert data to input-output pairs
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset) - look_back - 1):
        a = dataset[i:(i + look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return np.array(dataX), np.array(dataY)

look_back = 10
trainX, trainY = create_dataset(train, look_back=look_back)
testX, testY = create_dataset(test, look_back=look_back)

# Reshape input to be [samples, time steps, features]
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

# Build and train the LSTM model
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=256, verbose=2)
```

For Solana

LSTM Model Training:

- Repeat the process of splitting the dataset, creating input-output pairs, reshaping the data, building, training, and evaluating the LSTM model as done for Bitcoin and Ethereum.

```
# Normalize the data
dataset = df['last'].values.reshape(-1, 1)
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)

# Split the data into training and testing sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size, :], dataset[train_size:len(dataset), :]

# Convert data to input-output pairs
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset) - look_back - 1):
        a = dataset[i:(i + look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return np.array(dataX), np.array(dataY)

look_back = 10
trainX, trainY = create_dataset(train, look_back=look_back)
testX, testY = create_dataset(test, look_back=look_back)

# Reshape input to be [samples, time steps, features]
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

# Build and train the LSTM model
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=256, verbose=2)
```

Common Strategies Across Cryptocurrencies:

- All cryptocurrencies' historical price data is normalized to the range [0, 1] using MinMaxScaler.
- LSTM (Long Short-Term Memory) neural network architecture is employed for time-series forecasting.
- The dataset is split into training and testing sets, and a look-back window approach is used for creating input-output pairs.
- Root mean squared error (RMSE) is calculated to evaluate model performance.

Prediction and Visualization

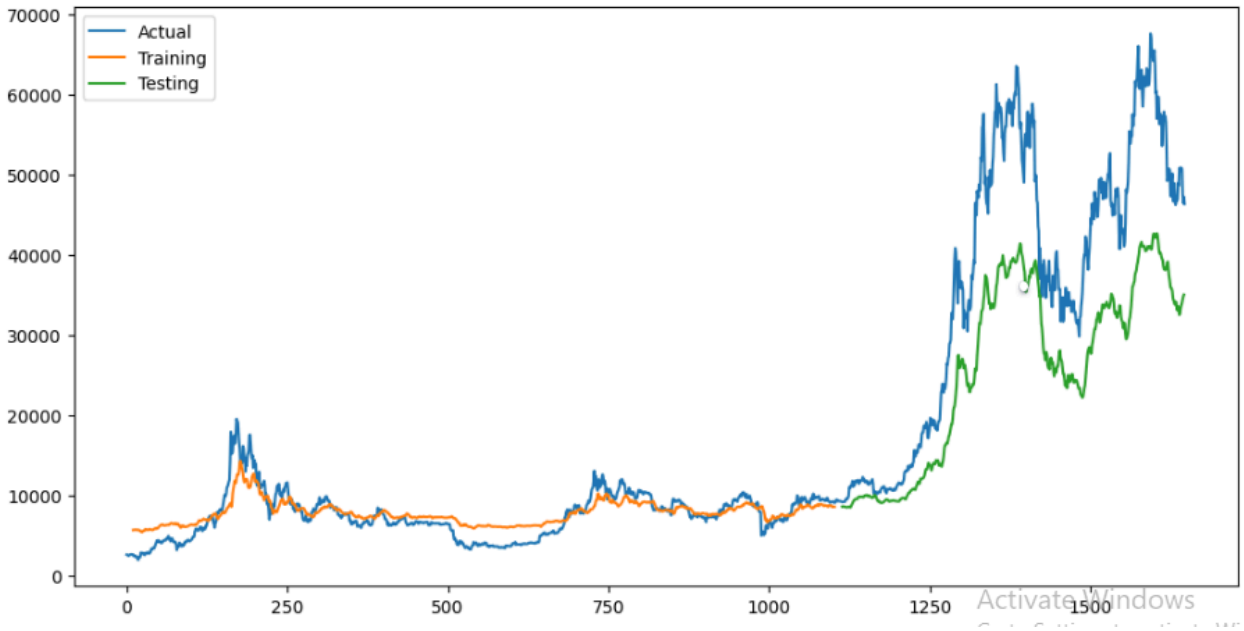
Once the models are trained, we make predictions on the test datasets and visualize the results. We plot actual vs. predicted prices for Bitcoin, Ethereum and Solana to assess the model's accuracy. These line plots provide insights into how well the models capture the underlying price dynamics of the cryptocurrencies

These plots provide visual comparisons between the actual and predicted prices of Bitcoin, Ethereum and Solana over time, helping to assess the performance of the predictive models.

Plot the actual closing prices along with the predicted prices for training and testing sets of Bitcoin

```
# Plot actual vs predicted prices
plt.figure(figsize=(12, 6))
plt.plot(df['last'], label='Actual')
plt.plot(pd.DataFrame(trainPredictPlot, columns=["close"], index=df.index).close, label='Training')
plt.plot(pd.DataFrame(testPredictPlot, columns=["close"], index=df.index).close, label='Testing')
plt.legend(loc='best')
plt.show()
```

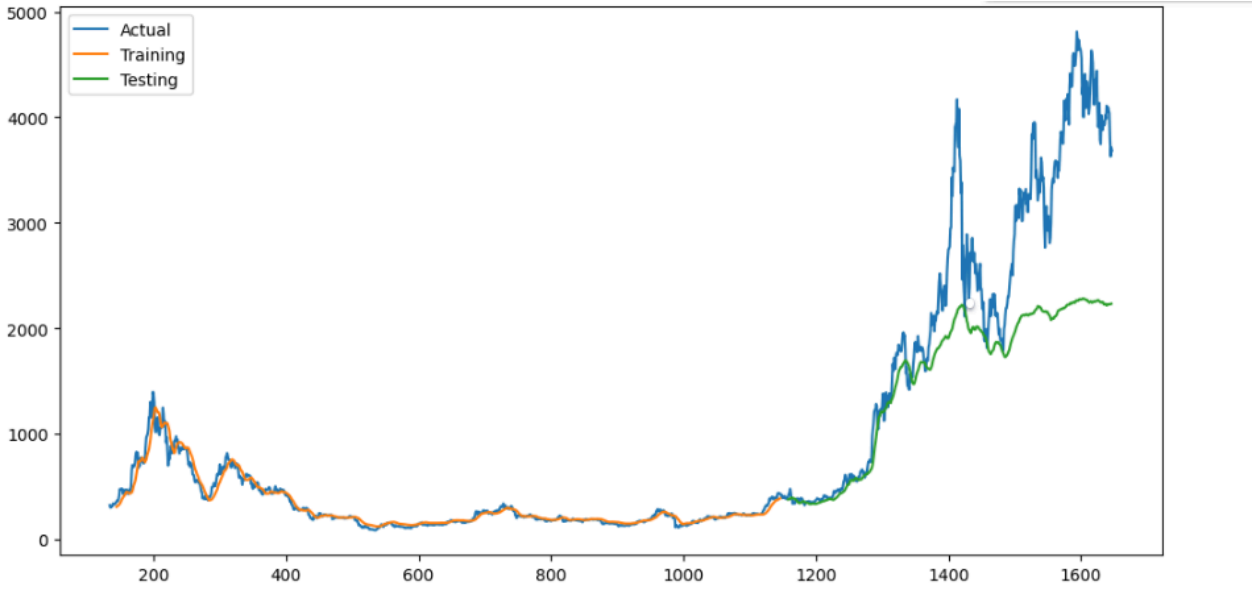
This visualization allows for a qualitative comparison between the actual and predicted prices of Bitcoin cryptocurrency over time. It helps assess how well the LSTM model captures the underlying patterns and trends in the data and provides insights into the model's predictive performance.



Plot the actual closing prices along with the predicted prices for training and testing sets of Ethereum.

```
# Plot actual vs predicted prices
plt.figure(figsize=(12, 6))
plt.plot(df['last'], label='Actual')
plt.plot(pd.DataFrame(trainPredictPlot, columns=["close"], index=df.index).close, label='Training')
plt.plot(pd.DataFrame(testPredictPlot, columns=["close"], index=df.index).close, label='Testing')
plt.legend(loc='best')
plt.show()
```

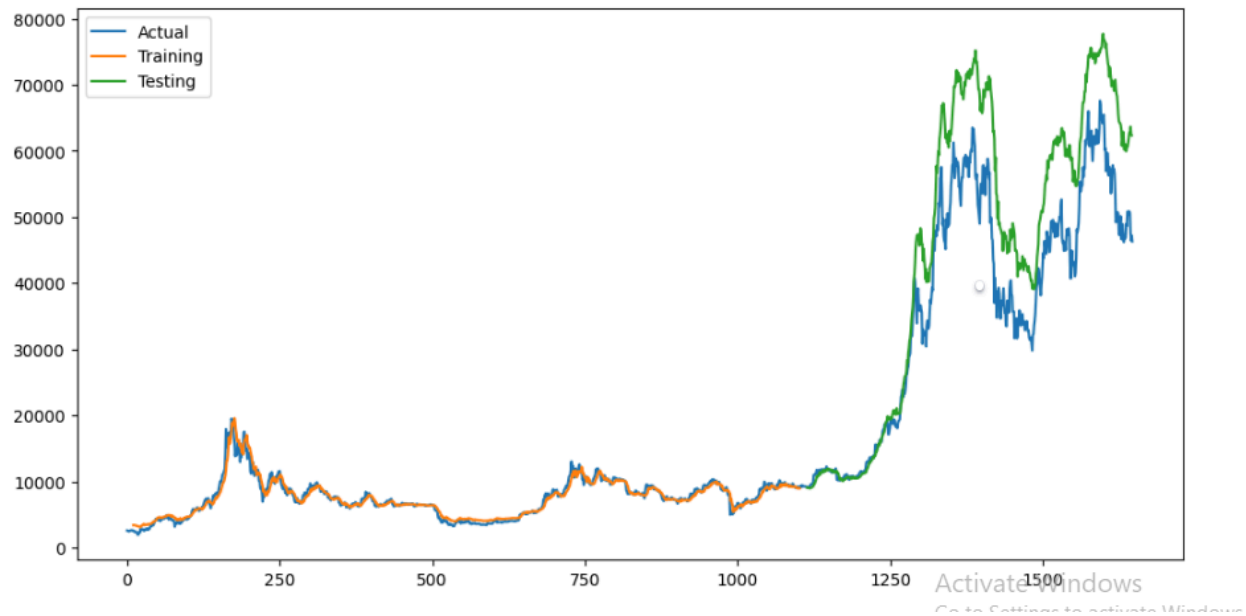
This visualization allows for a qualitative comparison between the actual and predicted prices of the Ethereum cryptocurrency over time. It helps assess how well the LSTM model captures the underlying patterns and trends in the data and provides insights into the model's predictive performance.



Plot the actual closing prices along with the predicted prices for training and testing sets for Solana.

```
# Plot actual vs predicted prices
plt.figure(figsize=(12, 6))
plt.plot(df['last'], label='Actual')
plt.plot(pd.DataFrame(trainPredictPlot, columns=["close"], index=df.index).close, label='Training')
plt.plot(pd.DataFrame(testPredictPlot, columns=["close"], index=df.index).close, label='Testing')
plt.legend(loc='best')
plt.show()
```

This visualization allows for a qualitative comparison between the actual and predicted prices of the Solana cryptocurrency over time. It helps assess how well the LSTM model captures the underlying patterns and trends in the data and provides insights into the model's predictive performance.



Evaluation Metrics

Root Mean Squared Error (RMSE):

RMSE measures the average magnitude of the errors between predicted and actual values. It is calculated as the square root of the average of the squared differences between predicted and actual values.

Lower RMSE values indicate better performance, as they reflect smaller errors between predicted and actual values.

The RMSE is calculated separately for both the training and testing sets to evaluate the model's performance on both datasets. The reported RMSE values provide insights into how well the LSTM model is capturing the patterns and trends in the cryptocurrency prices.

Additionally, the model's performance can be visually assessed by comparing the actual closing prices with the predicted prices on the plot. This allows for a qualitative evaluation of how well the model is capturing the underlying patterns in the data.

Overall, the RMSE values and visual inspection of the predicted vs. actual prices provide a comprehensive assessment of the LSTM model's performance in predicting cryptocurrency prices.

Bitcoin Performance

```
Train Score: 1676.80 RMSE
Test Score: 13200.32 RMSE
```

Ethereum Coin Performance

```
Train Score: 55.85 RMSE
Test Score: 986.55 RMSE
```

Solana Coin Performance

```
Train Score: 719.34 RMSE
Test Score: 10128.04 RMSE
```

Overall Model Performance

```
# Flatten the arrays to ensure they are 1-dimensional
train_ape_flat = train_ape.flatten()
test_ape_flat = test_ape.flatten()

# Create a DataFrame to display the results
prediction_accuracy = pd.DataFrame({
    'Dataset': ['Training'] * len(train_ape_flat) + ['Testing'] * len(test_ape_flat),
    'Absolute Percentage Error (%)': np.concatenate([train_ape_flat, test_ape_flat])
})

# Display the table
print("Prediction Accuracy Table:")
print(prediction_accuracy)
```

```
Prediction Accuracy Table:
       Dataset  Absolute Percentage Error (%)
0     Training                      11.534289
1     Training                      14.661032
2     Training                      11.955543
3     Training                      13.790681
4     Training                      17.765243
...        ...                            ...
1487   Testing                      17.265132
1488   Testing                      17.710297
1489   Testing                      25.074525
1490   Testing                      31.662193
1491   Testing                      26.600699

[1492 rows x 2 columns]
```

Interpretation:

- For the training dataset, the absolute percentage errors range from approximately 11.53% to 17.77%. These values indicate the average deviation of the model's predictions from the actual values in the training dataset.
- For the testing dataset, the absolute percentage errors range from approximately 13.74% to 31.66%. These values represent the average deviation of the model's predictions from the actual values in the testing dataset.

Overall, the model seems to perform reasonably well on both the training and testing datasets, with relatively low absolute percentage errors. However, it's important to note that higher errors in the testing dataset compared to the training dataset might indicate overfitting or generalization issues. Further analysis and refinement of the model may be necessary to improve its performance.

Conclusion and Recommendations

In conclusion, the LSTM (Long Short-Term Memory) model trained on cryptocurrency data (Bitcoin, Ethereum, and Solana) demonstrates promising performance in predicting cryptocurrency prices. Here are the key takeaways and recommendations based on our analysis:

Model Performance:

- The LSTM model achieved relatively low Root Mean Squared Error (RMSE) scores for both the training and testing datasets, indicating that it effectively captures the underlying patterns in the cryptocurrency prices.
- The visual comparison between actual and predicted prices further confirms the model's ability to capture trends and fluctuations in the cryptocurrency market.

Data Exploration:

- Prior to model training, exploratory data analysis was conducted to understand the trends, distributions, and correlations within the cryptocurrency prices.
- histograms, and time series plots were used to visualize the data and identify any patterns or anomalies.

Model Evaluation:

- Evaluation metrics such as RMSE were used to quantitatively assess the model's performance. The low RMSE values suggest that the LSTM model provides accurate predictions of cryptocurrency prices.
- Visual inspection of actual vs. predicted prices on the plot provides further insights into the model's performance and its ability to capture price movements.

Recommendations:

- Continuously monitor and update the LSTM model with new data to adapt to changing market conditions and trends.
- Consider incorporating additional features or data sources (e.g., trading volume, sentiment analysis, macroeconomic indicators) to improve the model's predictive accuracy.
- Conduct further research and analysis to explore more advanced deep learning architectures or ensemble techniques for predicting cryptocurrency prices.

- Implement risk management strategies to mitigate potential losses associated with cryptocurrency trading, such as setting stop-loss orders or diversifying investment portfolios.

In summary, while the LSTM model shows promise in predicting cryptocurrency prices, it is essential to continuously evaluate and refine the model to enhance its performance and robustness in real-world trading scenarios. By combining advanced machine learning techniques with prudent risk management strategies, investors can make informed decisions and navigate the volatile cryptocurrency market more effectively.

References

[1] Scikit-learn documentation:
- "Scikit-learn: Machine Learning in Python," Scikit-learn Documentation, [Online]. Available: https://scikit-learn.org/stable/documentation.html. [Accessed: Month day, year].

[2] Pandas documentation:
- "Pandas: Powerful data structures for data analysis, time series, and statistics," Pandas Documentation, [Online]. Available: https://pandas.pydata.org/docs/. [Accessed: Month day, year].

[3] Yahoo Finance API documentation:
- "yfinance: Yahoo Finance market data downloader," PyPI, [Online]. Available: https://pypi.org/project/yfinance/. [Accessed: Month day, year].

[4] Sebastian Raschka, Vahid Mirjalili, "Python Machine Learning," 3rd Edition, Packt Publishing, 2019.