Cryptocurrency Price Prediction Web App

A Project Report

Submitted in partial fulfillment of the

Requirements for the award of the Degree of

BACHELOR OF SCIENCE
(COMPUTER SCIENCE)

By

Satyam Sharma
Seat Number: 444

Under the esteemed guidance of
Prof. Madhav Mishra

DEPARTMENT OF COMPUTER SCIENCE

GURU NANAK KHALSA COLLEGE
OF
ARTS, SCIENCE & COMMERCE
(*Autonomous)*
MATUNGA, MUMBAI - 400 019
AY 2023-2024

GURU NANAK KHALSA COLLEGE

OF

ARTS, SCIENCE & COMMERCE

(*Autonomous)*

MATUNGA, MUMBAI, MAHARASHTRA – 400 019

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

This is to certify that the entitled, **"Cryptocurrency Price Prediction Web App"** is bonafied work of **Satyam Sharma** bearing Seat No. **444** submitted in partial fulfilment of the requirements for the award of degree of **BACHELOR OF SCIENCE in COMPUTER SCIENCE** from University of Mumbai.

Internal Guide **Coordinator**

External Examiner

Date: **College Seal**

APPROVAL OF PROJECT PROPOSAL

PRN No: 2021016401287432 **Roll No: 444**

1. **Name of the Student**

   Satyam Sharma

2. **Title of the Project**

   Cryptocurrency Price Prediction Web App

3. **Name of the Project Guide**

   Prof. Madhav Mishra

Signature of the Student **Signature of the Guide**

Date: **Date:**

Signature of the Head of Dept. **Signature of the Examiner**

Date: **Date:**

ACKNOWLEDGEMENT

I would like to express my thanks to the people who have helped me most throughout my project

I am grateful to my **Prof. Madhav Mishra** for nonstop support for the project. I cannot say thank you enough for him tremendous support and help.

I owe my deep gratitude to our HOD of Computer Science Department **Mrs. Jasmeet Kaur Ghai** who took keen interest on our project work and guided us all along, till the completion of our project work by providing all the necessary information for developing a good system.

At last, but not the least I want to thank all my friends who helped/treasured me out in completing the project, where they all exchanged their own interesting ideas, thoughts and made this possible to complete my project with all accurate information. I wish to thank my parents for their personal support or attention who inspired me to go my own way.

DECLARATION

I hereby declare that the project entitled, "**Cryptocurrency Price Prediction Web App**" done at **Guru Nanak Khalsa College**, has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfilment of the requirements for the award of degree of **BACHELOR OF SCIENCE** (Computer Science) to be submitted as final semester project as part of our curriculum

Satyam Sharma

Table of Contents

Title

Cryptocurrency Price Prediction Web App

Problem Statement

Bitcoin is a crypto currency which is used worldwide for digital payment or simply for investment purposes. Bitcoin is decentralized i.e. it is not owned by anyone. Transactions made by Bitcoins are easy as they are not tied to any country. Investment can be done through various marketplaces known as “bitcoin exchanges.” These allow people to sell/buy Bitcoins using different currencies. The largest Bitcoin exchange is Mt Gox.

The Bitcoin market’s financial analogy is, of course, a cryptocurrency market. To maximize financial reward, the field of cryptocurrency market prediction has grown over the past decades, and has more recently exploded with the advent of high-frequency, low-latency trading hardware coupled with robust machine learning algorithms. Thus, it makes sense that this prediction methodology is replicated in the world of Bitcoin, as the network gains greater liquidity and more people develop an interest in investing profitably in the system. To do so, we feel it is necessary to leverage machine learning technology to predict the price of cryptocurrency.

Abstract

The purpose of this project is to find out with what accuracy the direction of the price of cryptocurrency can be predicted using machine learning methods. This is basically a time series prediction problem. While much research exists surrounding the use of different machine learning.

Techniques for time series prediction, research in this area relating specifically to Bitcoin is lacking. In addition, Bitcoin as a currency is in a transient stage and as a result is considerably more volatile than other currencies such as the USD. Interestingly, it is the top performing currency four out of the last five years. Thus, its prediction offers great potential and this provides motivation for research in the area.

Finally, in analysing the chosen dependent variables, each variable's importance is assessed using a random forest algorithm. In addition, the ability to predict the direction of the price of an asset such as Bitcoin offers the opportunity for profit to be made by trading the asse

Objective

The main aim is to predict Cryptocurrency Price using different Machine learning algorithms. It must calculate the estimated price of cryptocurrency based on the historical data. To predict cryptocurrency price with maximum efficiency using LSTM.

Make a Full-stack website widely used for real-time data display and predictions of the profits and prices of cryptocurrencies using machine learning. ensuring less risk and more profit for investor.

Our main goal is to build a web page that would predict cryptocurrency prices. This will be achieved using machine learning algorithms. The first step towards Bitcoin prediction is database collection. For this project I have collected data from **Yfinance. Yfinance** is a Python library that provides a simple interface to interact with Yahoo Finance API to fetch historical market data, real-time data, and other financial information for stocks, cryptocurrencies, indices, and more. It allows users to easily access and download historical market data, including open, high, low, close, and volume (OHLCV) data, as well as dividends and stock splits.

The next step is database normalization. We basically perform this step to achieve consistency i.e. reduce or eliminate duplicate data, insignificant points, and other redundancies. Then we used LSTM model for prediction.

Data Description

data.info()

```
data.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3453 entries, 2014-09-17 to 2024-02-29
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Open       3453 non-null   float64
 1   High       3453 non-null   float64
 2   Low        3453 non-null   float64
 3   Close      3453 non-null   float64
 4   Adj Close  3453 non-null   float64
 5   Volume     3453 non-null   int64  
dtypes: float64(5), int64(1)
memory usage: 188.8 KB
```

data.head()

```
data.head()
```

✓ 0.0s

Date	Open	High	Low	Close	Adj Close	Volume
2014-09-17	465.864014	468.174011	452.421997	457.334015	457.334015	21056800
2014-09-18	456.859985	456.859985	413.104004	424.440002	424.440002	34483200
2014-09-19	424.102997	427.834991	384.532013	394.795990	394.795990	37919700
2014-09-20	394.673004	423.295990	389.882996	408.903992	408.903992	36863600
2014-09-21	408.084991	412.425995	393.181000	398.821014	398.821014	26580100

Column Description:

Date: This column represents the date associated with the corresponding financial data. Each row typically corresponds to a specific trading day in the market.

Open: The opening price of the financial instrument (e.g., stock, cryptocurrency) at the beginning of the trading day.

High: The highest price reached by the financial instrument during the trading day.

Low: The lowest price reached by the financial instrument during the trading day.

Close: The closing price of the financial instrument at the end of the trading day.

Adj Close (Adjusted Close): The adjusted closing price of the financial instrument, which considers any corporate actions such as dividends, stock splits, or other adjustments that may affect the closing price.

Volume: The total number of shares or units of the financial instrument traded during the trading day. It represents the liquidity and activity level in the market for that instrument on that day.

Proposed Libraries

NumPy: NumPy is a fundamental package for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

Pandas: Pandas is a powerful data manipulation and analysis library for Python. It provides data structures and functions for efficiently handling structured data, such as data frames, series, and powerful tools for data cleaning, filtering, grouping, and analysis.

yfinance: yfinance is a Python library that provides an easy-to-use interface for accessing financial data from Yahoo Finance. It allows users to fetch historical market data, real-time data, and other financial information for stocks, cryptocurrencies, indices, and more.

Datetime: Datetime is a module in Python's standard library that provides classes and functions for manipulating dates and times. It allows users to create, format, and perform calculations with dates and times.

Plotly: Plotly is a Python graphing library that allows users to create interactive and publication-quality plots and dashboards. It supports a wide range of plot types, including scatter plots, line plots, bar charts, heatmaps, and more.

Scikit-learn: Scikit-learn is a machine learning library for Python that provides simple and efficient tools for data mining and data analysis. It includes a wide range of supervised and unsupervised learning algorithms, as well as tools for model selection, evaluation, and preprocessing.

TensorFlow: TensorFlow is an open-source machine learning framework developed by Google. It provides comprehensive support for building and deploying machine learning models, including deep learning models, across a wide range of platforms and devices.

Streamlit: Streamlit is a Python library that allows users to create interactive web applications for data science and machine learning projects. It simplifies the process of building and deploying web apps by providing a clean and intuitive API.

Warning: The warning library manages and controls warning messages in Python programs, enabling developers to handle potential issues or anomalies gracefully during runtime.

Keras: Keras is a user-friendly deep learning library in Python, offering a high-level interface for building and experimenting with various neural network architectures efficiently.

UML Diagrams

Unified Modeling Language (UML) diagrams are a standardized way to visualize, document, and communicate the different aspects of a software system or a complex process. UML diagrams are widely used in software engineering and system design to represent various elements and relationships within a system. There are several types of UML diagrams, each serving a specific purpose. Here are some UML diagrams:

- Use Case Diagram:

  Represents the functional requirements of a system.

  Shows how users (actors) interact with the system through various use cases (functionalities).

- Class Diagram:

  Represents the static structure of a system.

  Shows classes, their attributes, methods, and the relationships between classes.

- Activity Diagram:

  Represents the workflow or business process of a system.

  Shows the flow of activities and decisions in a process or use case.

UML diagrams are a powerful tool for system design, analysis, and documentation.

The choice of which type of UML diagram to use depends on the specific aspect of the system or process you want to model or communicate.

Depending on the complexity of your project, you may use one or more of these diagram types to represent different aspects of your system.

Class Diagram

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

Class diagram describes the attributes and operations of a class and the constraints imposed on the system. The class diagrams are widely used in the modeling of object-oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.

Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.

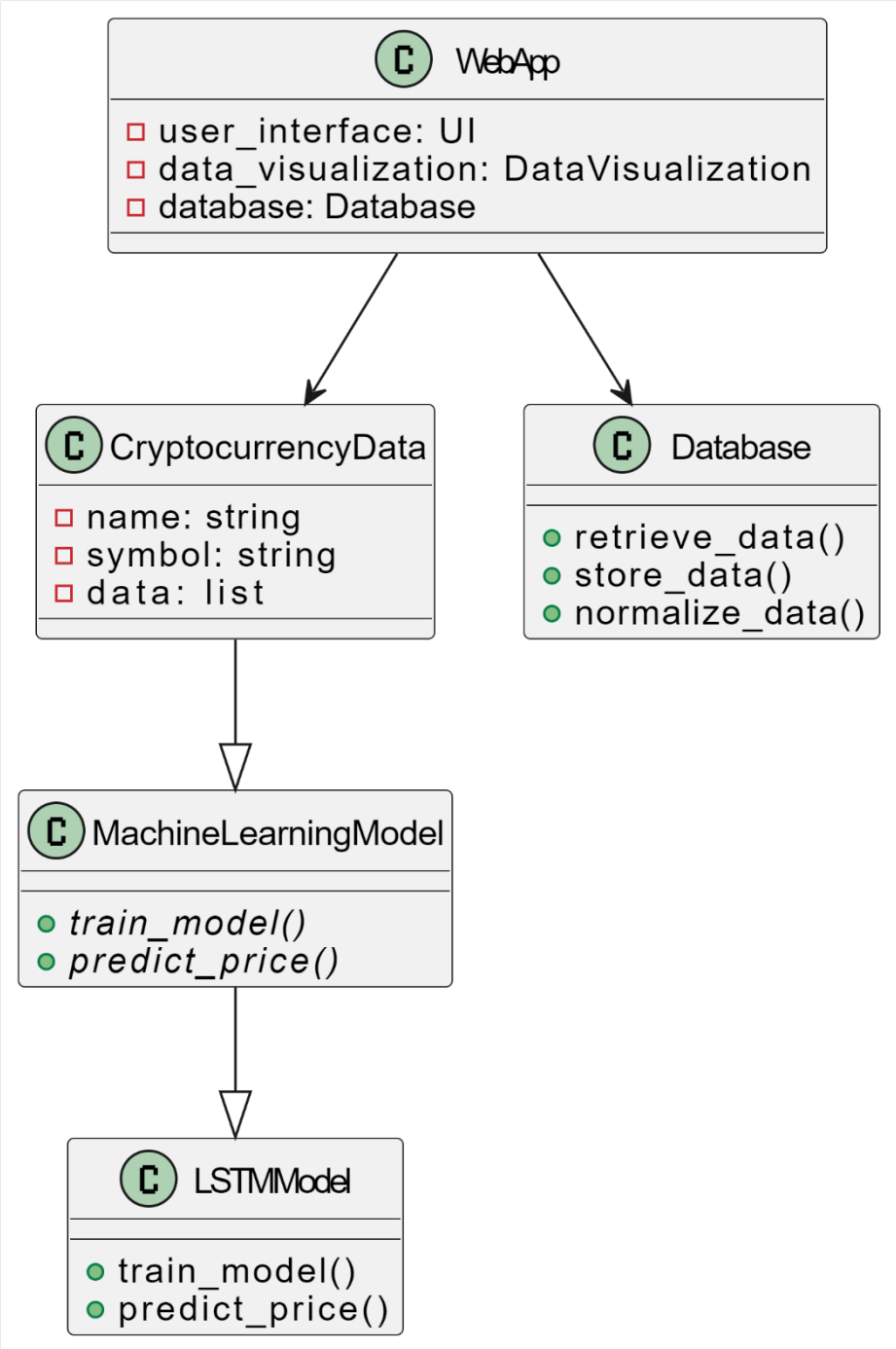
CryptocurrencyData: Represents the cryptocurrency data, including name, symbol, and historical data.

MachineLearningModel: Abstract class representing a machine learning model. It defines methods for training and predicting cryptocurrency prices.

LSTMModel: Concrete class representing the LSTM model. It implements methods for training and predicting cryptocurrency prices using LSTM algorithm.

Database: Represents the database for storing and retrieving cryptocurrency data. It includes methods for data retrieval, storage, and normalization.

WebApp: Represents the web application for predicting cryptocurrency prices. It interacts with the user interface, data visualization, database, and prediction comparison functionalities.


WebApp
user_interface: UI
data_visualization: DataVisualization
database: Database
CryptocurrencyData
name: string
symbol: string
data: list
Database
retrieve_data()
store_data()
normalize_data()
MachineLearningModel
train_model()
predict_price()
LSTMModel
train_model()
predict_price()

Activity Diagram

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system.

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.

Start: The process begins with the start symbol.

Collect Cryptocurrency Data: This activity represents the initial step of collecting cryptocurrency data for analysis and prediction.

Data Exists? This decision point checks if the data already exists in the database.

Retrieve Data: If the data exists, this activity retrieves it from the database.

Download Data from Yfinance: If the data does not exist, this activity downloads it from Yfinance, which provides historical market data for cryptocurrencies.

Store Data in Database: After downloading the data, it is stored in the database for future use.

Normalize Data: This activity represents the process of normalizing the collected data to ensure consistency and eliminate redundancies.

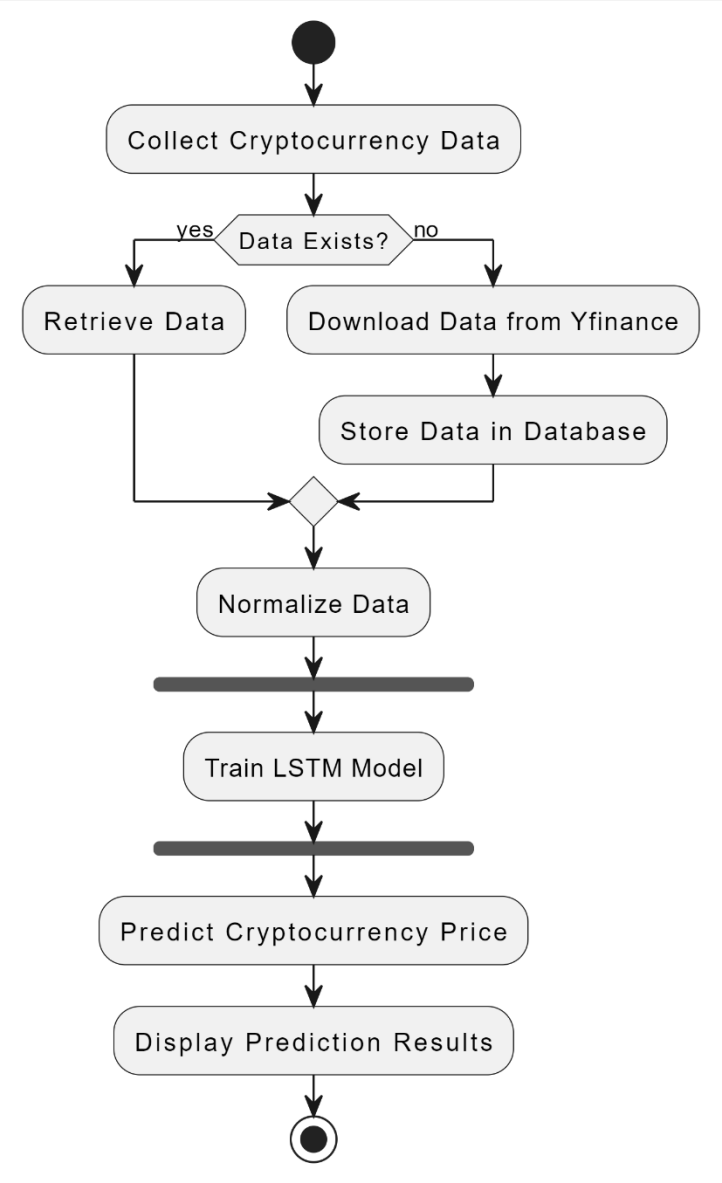
Fork: This symbol represents a fork in the process, indicating that the following activities will be performed concurrently.

Train LSTM Model: This activity involves training the LSTM (Long Short-Term Memory) model, a type of recurrent neural network (RNN), using the normalized cryptocurrency data.

Predict Cryptocurrency Price: Using the selected model, this activity predicts the future prices of cryptocurrencies.

Display Prediction Results: Finally, this activity involves displaying the predicted cryptocurrency prices to the user, allowing them to analyse and make informed decisions.

Stop: The process ends at the stop symbol.



Use-Case Diagram

Actors:

User: Represents a user who interacts with the Cryptocurrency Prediction System.

Website Visitor: Represents a visitor accessing the system through the website.

Use Cases:

Collect Cryptocurrency Data: Use case for collecting cryptocurrency data. It includes retrieving existing data, downloading new data if needed, storing the data, and normalizing it.

Retrieve Cryptocurrency Data: Sub-use case for retrieving existing cryptocurrency data from the database.

Download Cryptocurrency Data: Sub-use case for downloading new cryptocurrency data from an external source.

Store Cryptocurrency Data: Sub-use case for storing cryptocurrency data in the database.

Normalize Cryptocurrency Data: Sub-use case for normalizing cryptocurrency data to achieve consistency.

Train LSTM Model: Use case for training the LSTM (Long Short-Term Memory) model for predicting cryptocurrency prices.

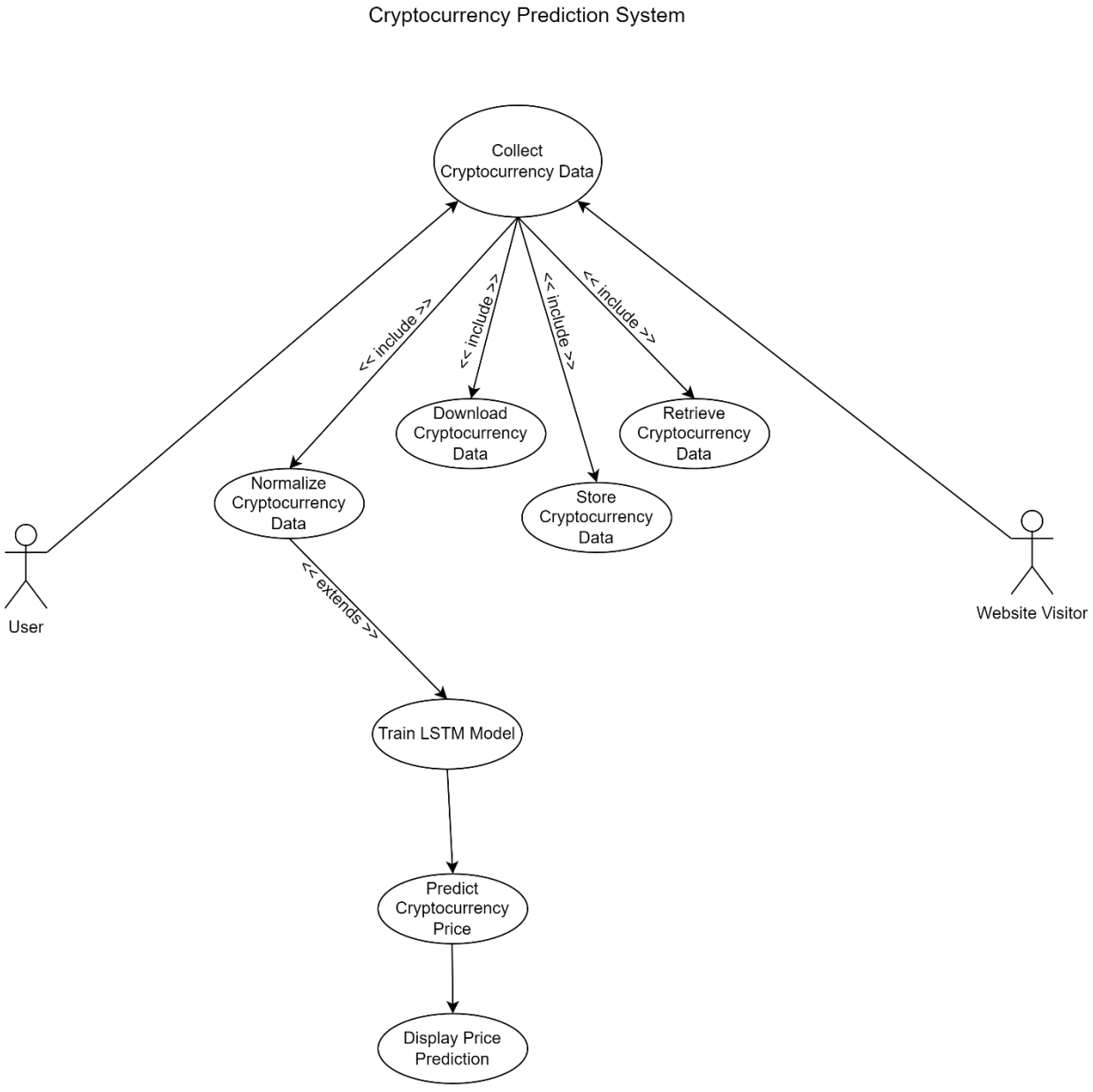
Predict Cryptocurrency Price: Use case for predicting cryptocurrency prices using the selected model.

Display Prediction Results: Use case for displaying the prediction results to the user.

Relationships:

Includes: Indicates that the "Collect Cryptocurrency Data" use case includes sub-use cases for retrieving, downloading, storing, and normalizing data.

Extends: Indicates that the "Normalize Cryptocurrency Data" use case extends to training the LSTM, allowing for additional functionality when needed.



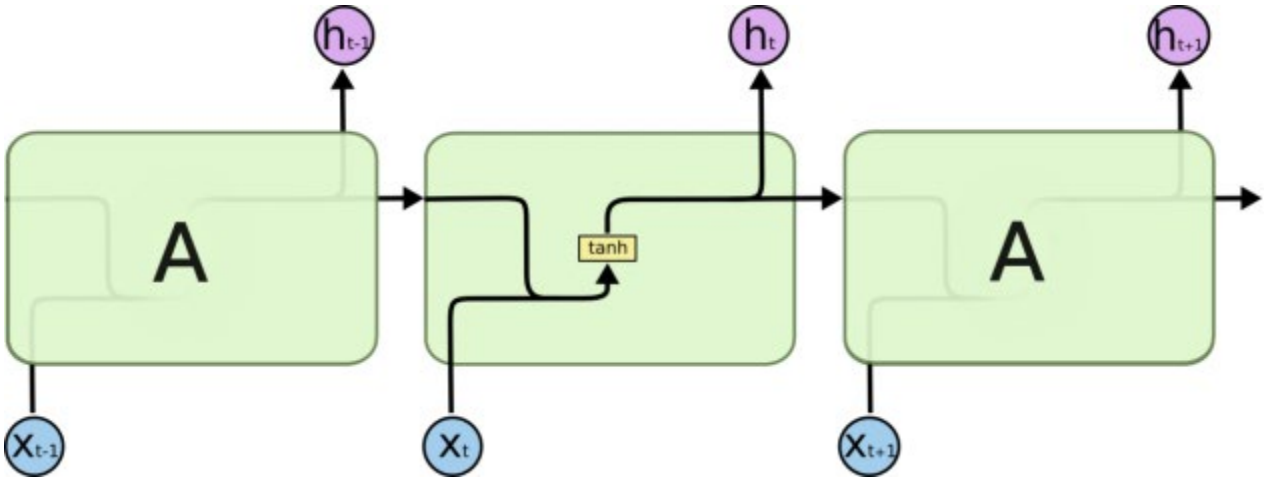
Model Description

Long Short-Term Memory (LSTM)

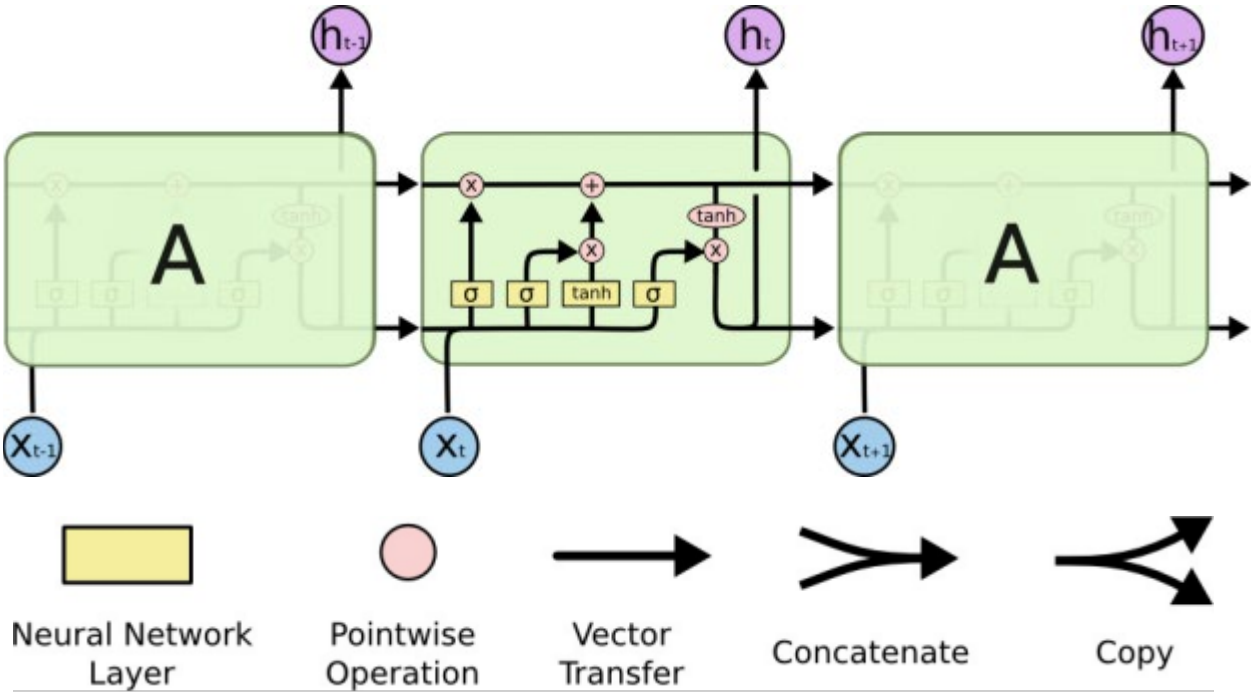
Long Short-Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work. They work tremendously well on a large variety of problems, and are now widely used.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behaviour, not something they struggle to learn.

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



LSTMs also have this chain-like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denotes its content being copied and the copies going to different locations.

LSTMs, or Long Short-Term Memory networks, are a powerful type of recurrent neural network (RNN) architected specifically to tackle the challenges of sequential data. Unlike standard RNNs, LSTMs excel at capturing long-term dependencies within sequences, making them a go-to choice for applications like machine translation, speech recognition, and time series forecasting.

Understanding RNNs: The Foundation of LSTMs

LSTMs build upon the core concept of Recurrent Neural Networks (RNNs). RNNs are designed to process sequential data, where the order of elements matters. They achieve this by having a loop in their structure, allowing them to consider the output from previous data points when processing the current one. This loop enables them to learn relationships between elements within a sequence.

However, a major limitation of traditional RNNs is the vanishing gradient problem. As information travels through the network over long sequences, gradients (signals used for learning) can become vanishingly small or explode. This makes it difficult for RNNs to learn long-term dependencies, where information from far back in the sequence needs to be remembered.

The LSTM Architecture: Overcoming the Vanishing Gradient Problem

LSTMs address this limitation with a unique architecture that incorporates a cell state and gating mechanisms. Here's a closer look at these key components:

Cell State: Imagine the cell state as a conveyor belt running through the LSTM. It acts as a long-term memory, allowing the network to retain crucial information across the entire sequence. Unlike the fading gradients in RNNs, the cell state can hold onto information for extended periods.

Gating Mechanisms: LSTMs have three special gates that regulate the flow of information within the cell state:

- **Forget Gate:** This gate decides what information to discard from the previous cell state. It analyzes the current input and the past state, assigning a value between 0 and 1 to each element. A value close to 1 signifies retaining the information, while 0 indicates discarding it.
- **Input Gate:** This gate controls what new information gets added to the cell state. It considers the current input and the previous state, determining how much of the new information is relevant and should be kept.
- **Output Gate:** This gate decides what information from the cell state gets passed on to the next step in the network. It analyzes the current cell state and the previous state, selecting the most relevant information to be used for the next prediction.

By selectively remembering and forgetting information through these gates, LSTMs can effectively learn long-term dependencies within sequences.

The Training Process

LSTMs are trained using a technique called backpropagation through time (BPTT). This algorithm allows the network to adjust its internal parameters based on the difference between the predicted output and the actual output for a given sequence. Over many training iterations, the LSTM learns to optimize its gates and cell state to capture the underlying patterns in the sequential data.

Model Evaluation

Developing a web app for cryptocurrency price prediction using an LSTM model requires careful evaluation to ensure its effectiveness and reliability.

Metrics for Performance Evaluation:

Mean Squared Error (MSE): This metric measures the average squared difference between the predicted prices and the actual prices. Lower MSE indicates better model performance.

Root Mean Squared Error (RMSE): The square root of MSE, representing the standard deviation of the prediction errors. Lower RMSE signifies better predictions on a more interpretable scale (same units as the data).

Mean Absolute Error (MAE): This metric calculates the average absolute difference between predicted and actual prices. It's less sensitive to outliers compared to MSE.

Mean Absolute Percentage Error (MAPE): This metric expresses the error as a percentage of the actual price. It allows for easier comparison across cryptocurrencies with different price scales.

R-squared (R^2): This metric indicates the proportion of variance in the actual prices explained by the model's predictions. A value closer to 1 signifies a better fit.

Walk-Forward Validation: Cryptocurrency markets are highly dynamic. To assess the model's performance on unseen data, a technique called walk-forward validation can be employed. Here, the model is trained on a historical dataset and then evaluated on a more recent timeframe not included in the training data. This helps simulate real-world usage and ensures the model generalizes well.

In this work, we have implemented a model to predict real time bitcoin prices using various parameters such as High, low, open, volume etc. Thus, drawing a comparison between the actual prices and the predicted prices by taking into consideration various time series data to ultimately reduce the

difference between the predicted and actual prices, aiming to get the highest accuracy out of it using appropriate machine learning algorithms.

```
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Mean Absolute Error (MAE)
mae = mean_absolute_error(actual_prices, prediction_prices)

# Mean Squared Error (MSE)
mse = mean_squared_error(actual_prices, prediction_prices)

# Root Mean Squared Error (RMSE)
rmse = mean_squared_error(actual_prices, prediction_prices, squared=False)

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}')
```

```
Mean Absolute Error (MAE): 1837.65
Mean Squared Error (MSE): 13236713.34
Root Mean Squared Error (RMSE): 3638.23
```

Code

Model Building file (Bitcoin_Final.py)

```
import numpy as np
import pandas as pd
import yfinance as yf
from datetime import datetime
import plotly.express as px

from sklearn.preprocessing import MinMaxScaler
from keras.layers import Dense, Dropout, LSTM
from keras.models import Sequential
import plotly.graph_objects as go

import warnings
warnings.filterwarnings('ignore')
start = datetime(2014, 9, 17)
end = datetime.now().date().isoformat()
symbol = 'BTC-USD'

data = yf.download(symbol, start = start, end = end)

data.info()

data.head()

data.tail()

data.describe()

#Null Values Check
data.isnull().sum()

data.shape

data.columns

fig = px.line(data.Close)
fig.show()

data['ma100'] = data['Close'].rolling(window=100).mean()
fig2 = px.line(data, x=data.index, y=['Close', 'ma100'],
color_discrete_map={'Close': 'blue', 'ma100': 'red'})
```

```
fig2.show()



data['ma200'] = data['Close'].rolling(window=200).mean()
fig3 = px.line(data, x=data.index, y=['Close', 'ma200'],
color_discrete_map={'Close': 'blue', 'ma200': 'red'})
fig3.show()

fig4 = px.line(data, x=data.index, y=['Close', 'ma100', 'ma200'],
color_discrete_map={'Close': 'blue', 'ma100':'yellow', 'ma200':
'red'})
fig4.show()

# Prepare Data
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(data['Close'].values.reshape(-
1,1))

prediction_days = 80
future_day = 15

x_train, y_train = [], []

for x in range(prediction_days, len(scaled_data)-future_day):
    x_train.append(scaled_data[x-prediction_days:x, 0])
    y_train.append(scaled_data[x+future_day, 0])

x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0],
x_train.shape[1], 1))

# Create Neural Network
model = Sequential()

# Layer 1
model.add(LSTM(units=50, return_sequences=True,
input_shape=(x_train.shape[1], 1)))
model.add(Dropout(0.2))

# Layer 2
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))

# Layer 3
```

```
model.add(LSTM(units=50))
model.add(Dropout(0.2))

model.add(Dense(units=1))

model.summary()

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, y_train, epochs=25, batch_size=32)

model.save('model.h5')

start = datetime(2020, 1, 1)
end = datetime.now().date().isoformat()
symbol = 'BTC-USD'
test_data = yf.download(symbol, start = start, end = end)

test_data.head()

test_data.tail()

actual_prices = test_data['Close'].values

total_dataset = pd.concat((data['Close'], test_data['Close']),
axis=0)

model_inputs = total_dataset[len(total_dataset)-len(test_data)-
prediction_days:].values
model_inputs = model_inputs.reshape(-1,1)
model_inputs = scaler.fit_transform(model_inputs)

x_test = []

for x in range(prediction_days, len(model_inputs)):
    x_test.append(model_inputs[x-prediction_days:x, 0])

x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1],
1))

prediction_prices = model.predict(x_test)
prediction_prices = scaler.inverse_transform(prediction_prices)

actual_prices.shape
```

```
prediction_prices.shape

# Creating a Plotly figure
fig = go.Figure()

# Trace for actual prices
fig.add_trace(go.Scatter(x=test_data.index,
y=actual_prices.flatten(), mode='lines', name='Actual Prices',
line=dict(color='blue')))

# Trace for predicted prices
fig.add_trace(go.Scatter(x=test_data.index,
y=prediction_prices.flatten(), mode='lines', name='Predicted
Prices', line=dict(color='red')))

# Update layout
fig.update_layout(
                  xaxis_title='Date',
                  yaxis_title='Price',
                  legend=dict(x=0, y=1.1, orientation='h'),
                  margin=dict(l=0, r=0, t=50, b=0))

# Displaying the plot
fig.show()

# Predict Next Day
real_data = [model_inputs[len(model_inputs)+1-
prediction_days:len(model_inputs)+1, 0]]
real_data = np.array(real_data)
real_data = np.reshape(real_data, (real_data.shape[0],
real_data.shape[1], 1))

prediction = model.predict(real_data)
prediction = scaler.inverse_transform(prediction)

print(end)

print(prediction)

test_data['Close'].mean()

from sklearn.metrics import mean_absolute_error,
mean_squared_error
```

```
# Mean Absolute Error (MAE)
mae = mean_absolute_error(actual_prices, prediction_prices)

# Mean Squared Error (MSE)
mse = mean_squared_error(actual_prices, prediction_prices)

# Root Mean Squared Error (RMSE)
rmse = mean_squared_error(actual_prices, prediction_prices,
squared=False)

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}')
```

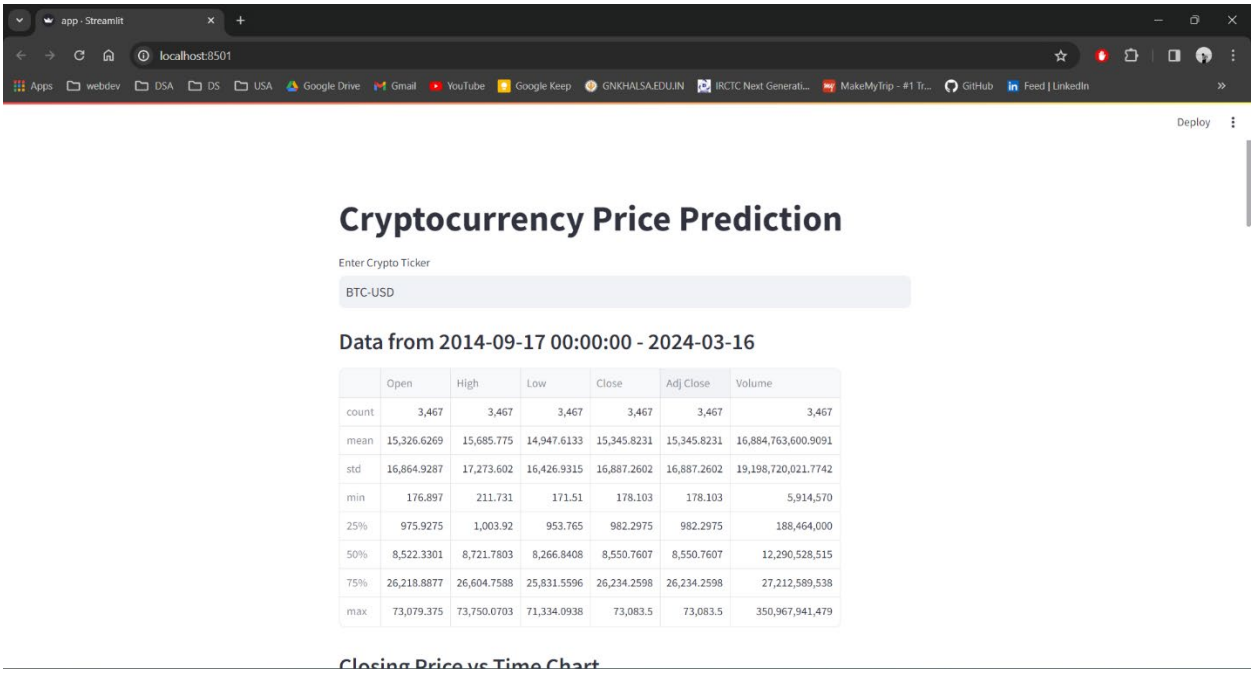
Web app file(app.py)

```
import numpy as np
import pandas as pd
import yfinance as yf
from datetime import datetime
import plotly.express as px
import plotly.graph_objects as go
from sklearn.preprocessing import MinMaxScaler
from keras.models import load_model
import streamlit as st

start = datetime(2014, 9, 17)
end = datetime.now().date().isoformat()

st.title('Cryptocurrency Price Prediction')

user_input = st.text_input('Enter Crypto Ticker', 'BTC-USD')

data = yf.download(user_input, start = start, end = end)

# Describing Data
st.subheader(f'Data from {start} - {end}')
st.write(data.describe())

# Visulizations
st.subheader('Closing Price vs Time Chart')
fig1 = px.line(data.Close, color_discrete_map={'Close':'blue'})
st.plotly_chart(fig1)
```

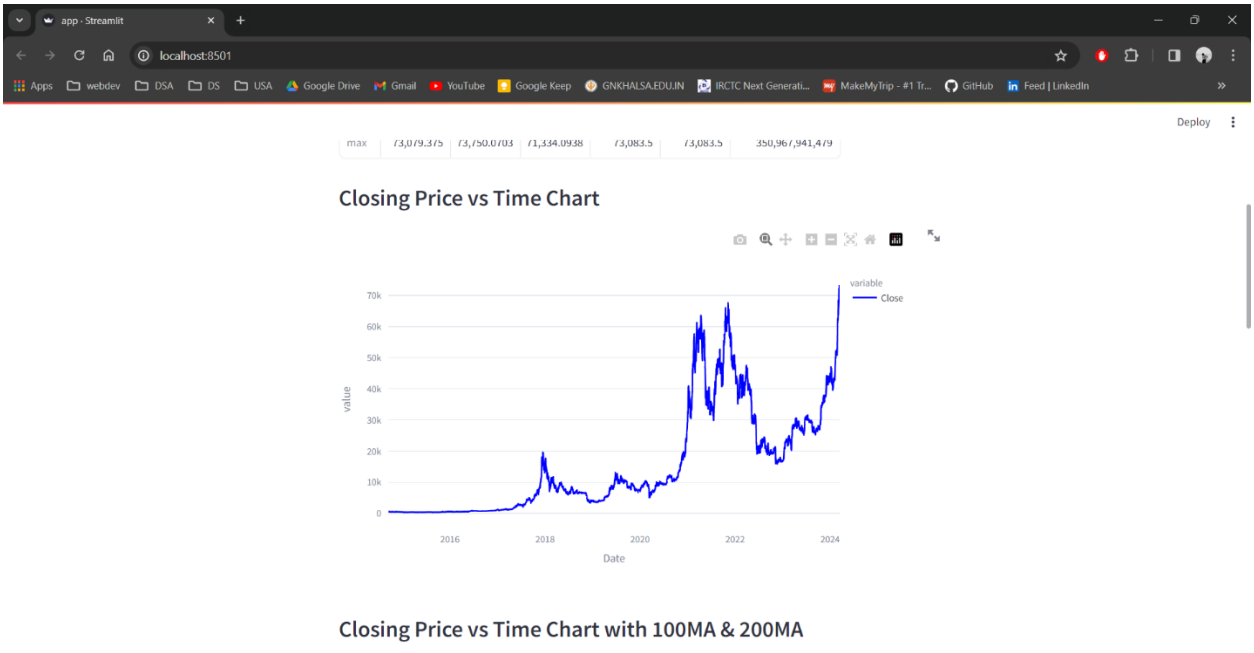
```
data['ma100'] = data['Close'].rolling(window=100).mean()
data['ma200'] = data['Close'].rolling(window=200).mean()
st.subheader('Closing Price vs Time Chart with 100MA & 200MA')
fig4 = px.line(data, x=data.index, y=['Close', 'ma100', 'ma200'],
color_discrete_map={'Close': 'blue', 'ma100':'yellow', 'ma200':
'red'})
st.plotly_chart(fig4)

prediction_days = 80
future_day = 15

scaler = MinMaxScaler(feature_range=(0,1))

# Load Model
model = load_model('model.h5')

#Testing part
start = datetime(2020, 1, 1)
end = datetime.now().date().isoformat()
test_data = yf.download(user_input, start=start, end=end)
st.subheader('Test Data for Prediction')
fig_test_data = px.line(test_data.Close,
color_discrete_map={'Close':'blue'})
st.plotly_chart(fig_test_data)

actual_prices = test_data['Close'].values
total_dataset = pd.concat((data['Close'], test_data['Close']),
axis=0)
model_inputs = total_dataset[len(total_dataset)-len(test_data)-
prediction_days:].values
model_inputs = model_inputs.reshape(-1,1)
model_inputs = scaler.fit_transform(model_inputs)

x_test = []
for x in range(prediction_days, len(model_inputs)):
    x_test.append(model_inputs[x-prediction_days:x, 0])
x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1],
1))

prediction_prices = model.predict(x_test)
prediction_prices = scaler.inverse_transform(prediction_prices)
```

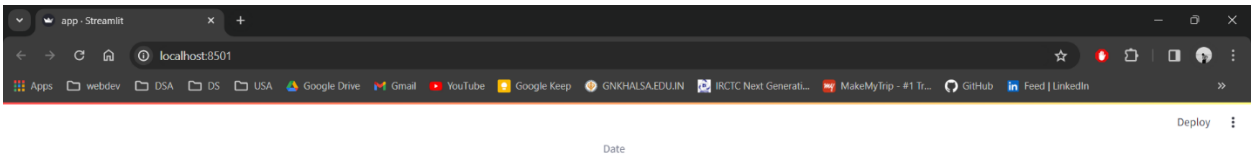
```
st.subheader(f'{user_input} Predicted Price Vs Actual Price Using
LSTM Model (Future {future_day} days)')
# Create a Plotly figure
fig_lstm = go.Figure()
# Add trace for actual prices
fig_lstm.add_trace(go.Scatter(x=test_data.index,
y=actual_prices.flatten(), mode='lines', name='Actual Prices',
line=dict(color='blue')))
# Add trace for predicted prices
fig_lstm.add_trace(go.Scatter(x=test_data.index,
y=prediction_prices.flatten(), mode='lines', name='Predicted
Prices', line=dict(color='red')))
fig_lstm.update_layout(xaxis_title='Date',
                  yaxis_title='Price',
                  legend=dict(x=0, y=1.1, orientation='h'),
                  margin=dict(l=0, r=0, t=50, b=0))
# Display the plot
st.plotly_chart(fig_lstm)

# Predicting Next Day Price
real_data = [model_inputs[len(model_inputs)+1-
prediction_days:len(model_inputs)+1, 0]]
real_data = np.array(real_data)
real_data = np.reshape(real_data, (real_data.shape[0],
real_data.shape[1], 1))

prediction = model.predict(real_data)
prediction = scaler.inverse_transform(prediction)

st.subheader('Next Day Price Prediction')
st.write(end)
st.write(prediction)
```

Implementation (UI)



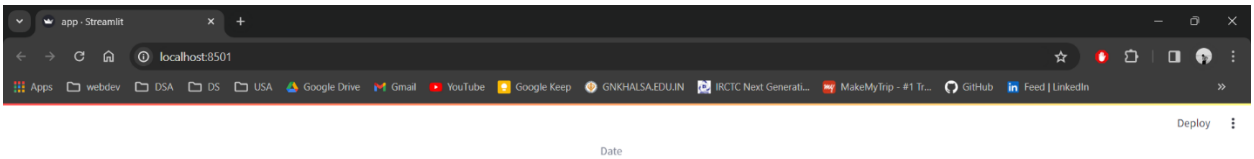
	Open	High	Low	Close	Adj Close	Volume
count	3,467	3,467	3,467	3,467	3,467	3,467
mean	15,326.6269	15,685.775	14,947.6133	15,345.8231	15,345.8231	16,884,763,600.9091
std	16,864.9287	17,273.602	16,426.9315	16,887.2602	16,887.2602	19,198,720,021.7742
min	176.897	211.731	171.51	178.103	178.103	5,914,570
25%	975.9275	1,003.92	953.765	982.2975	982.2975	188,464,000
50%	8,522.3301	8,721.7803	8,266.8408	8,550.7607	8,550.7607	12,290,528,515
75%	26,218.8877	26,604.7588	25,831.5596	26,234.2598	26,234.2598	27,212,589,538
max	73,079.375	73,750.0703	71,334.0938	73,083.5	73,083.5	350,967,941,479





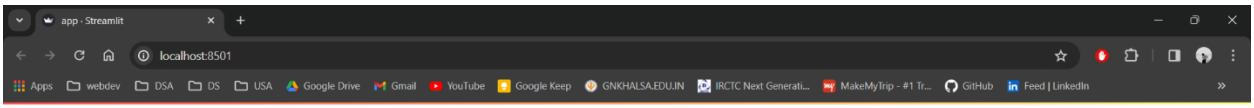
Closing Price vs Time Chart with 100MA & 200MA

Test Data for Prediction



Test Data for Prediction

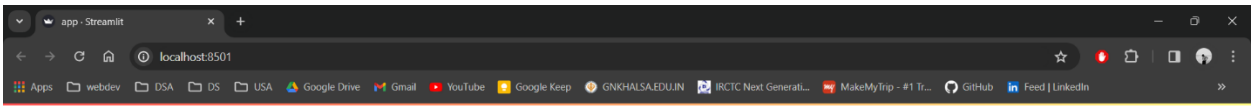
BTC-USD Predicted Price Vs Actual Price Using LSTM Model



Deploy

Date

BTC-USD Predicted Price Vs Actual Price Using LSTM Model (Future 15 days)



Deploy

Next Day Price Prediction

2024-03-16

0
69,798.8672

Future Enhancement

Ensemble Methods:

Integrate ensemble methods such as bagging, boosting, or stacking to combine predictions from multiple LSTM models or other types of models.

Ensemble methods often lead to improved prediction accuracy by leveraging the diversity of individual models.

Feature Engineering:

Explore additional features beyond historical price data, such as market sentiment analysis from news articles, social media feeds, or blockchain transaction data.

Experiment with technical indicators (e.g., moving averages, RSI, MACD) or fundamental factors (e.g., trading volume, market capitalization) to enrich the input data.

Hyperparameter Optimization:

Utilize techniques like grid search, random search, or Bayesian optimization to tune the hyperparameters of the LSTM model for better performance.

Hyperparameters include the number of LSTM layers, hidden units, learning rate, dropout rate, batch size, and sequence length.

Explainability and Interpretability:

Enhance the interpretability of the model's predictions by integrating techniques such as SHAP (SHapley Additive exPlanations) values or attention visualization.

Providing insights into which features or time periods are most influential in driving the model's predictions can increase user trust and understanding.

Real-Time Prediction:

Implement real-time prediction capabilities to enable users to receive up-to-date cryptocurrency price forecasts as new data becomes available.

Utilize streaming data sources and optimize the model's inference pipeline for low latency.

User Customization:

Allow users to customize their prediction horizons, choosing between short-term and long-term forecasts based on their investment strategies or risk preferences.

Enable users to adjust the model's parameters or select different cryptocurrencies for prediction.

Model Monitoring and Maintenance:

Implement monitoring tools to track the performance of the LSTM model over time and detect any drift or degradation in prediction accuracy.

Regularly retrain the model with updated data to adapt to changing market conditions and maintain its effectiveness.

Mobile Optimization:

Optimize the web app for mobile devices to enhance accessibility and usability for users who prefer to access the platform on smartphones or tablets.

Consider developing native mobile apps for iOS and Android platforms for a seamless user experience.