INDEX

ABSTRACT

Cryptocurrency has emerged as a revolutionary digital asset with immense market volatility and potential for high returns. Predicting cryptocurrency prices, particularly Bitcoin, has become a critical focus area for investors, traders, and researchers due to its unpredictable nature and rapid price fluctuations. This project presents a machine learning-based approach for predicting the closing price of Bitcoin using historical market data.

The dataset includes key attributes such as Open, High, Low, Close, Marketcap, and Date. The data undergoes preprocessing steps including cleaning, transformation, feature engineering, and normalization. A Random Forest Regressor model is employed for training and prediction, offering high accuracy and robustness against non-linear patterns in financial data.

The trained model is capable of predicting future Bitcoin closing prices based on user input of market indicators. Visualization tools are also utilized to compare actual vs. predicted values, enhancing interpretability. This project demonstrates the potential of data-driven models in cryptocurrency forecasting, aiming to assist investors in making informed decisions and reducing financial risk.

LIST OF FIGURES

CHAPTER-1

INTRODUCTION

1.1 ABOUT PROJECT:

Cryptocurrency is a form of digital currency that operates independently of traditional banks and uses encryption techniques to secure transactions. Among various cryptocurrencies, Bitcoin is the most well-known and widely traded. However, its price is highly volatile, making it challenging for investors and traders to make confident decisions.

The main objective of this project is to develop a machine learning model that can predict the future price of Bitcoin based on historical market data. By analysing features such as Open, High, Low, Market cap, and Date, we aim to forecast the Closing price of Bitcoin. This prediction can help users make better investment choices, avoid losses, and understand market trends more effectively.

In this project, we use a Random Forest Regressor model due to its high accuracy and ability to handle complex, non-linear relationships in financial data. The project includes steps like data cleaning, feature engineering, model training, and evaluation, followed by a user-friendly price prediction function where input values generate predicted prices.

1.2 USE CASES:

- **Helping Investors Make Smart Decisions:** One of the main uses of cryptocurrency price prediction is to help investors know the best time to buy or sell. Since prices change very quickly, having a tool that predicts what might happen next gives traders a big advantage. For example, if the model predicts that Bitcoin will go up tomorrow, the investor may choose to buy now and sell when the price rises. This reduces the chance of loss and increases the chance of making profits.
- **Managing Risk in portfolios:** Investors who manage big portfolios often need to know how risky Bitcoin is before adding it. Price prediction models like this one help them decide whether it's safe to invest in crypto. If the model predicts a price drop, they might choose to wait or invest less. This use case is important for balancing risk and making sure that the portfolio stays strong even if the market becomes unstable.
- **Improving Crypto Apps and Wallets:** Crypto wallet apps and financial platforms can include this

model to show predictions to their users. This makes the app more useful because users can see not

- just the current price, but also what might happen next. For example, if the app shows that Bitcoin is likely to increase tomorrow, users may feel more confident buying today. It adds more value to the app and improves the user experience.
- **Supporting Financial News and Reports:** Financial news websites and analysts often talk about where the market is heading. They can use this model to support their opinions with data. Instead of just guessing, they can say, "According to our prediction model, Bitcoin is likely to rise tomorrow." This makes the information they share more reliable and helps readers trust what they're reading.
- **Detecting Unusual Market Activity:** In some cases, price predictions can also help in detecting fraud or market manipulation. If the model predicts a steady price, but the actual price suddenly crashes or spikes, that could be a sign of suspicious activity. Regulators or analysts can use this as a warning to look deeper into what's happening. It can help make the crypto market more secure and trustworthy.
- **Sending Alerts to Casual Users:** Even people who don't trade every day can benefit from simple alerts. The model can send a message like "Bitcoin might fall by 3% tomorrow." This gives the user a heads-up so they can decide whether to hold, sell, or buy. It's especially helpful for people who don't follow the market closely but still want to stay informed.
- **Using in Automated Trading Bots:** This project can be used with trading bots that automatically buy and sell cryptocurrencies. These bots can be programmed to use the predicted prices to make decisions without human help. For instance, if the prediction says the price will go up by 5%, the bot can buy coins and sell them when the price actually increases. This helps traders make faster and more accurate decisions, especially in fast-moving markets.
- **Learning and Teaching in Education:** Students and researchers in finance or data science can use this project as an example of how machine learning can be used in the real world. It helps them learn about how data is cleaned, how models are trained, and how predictions are made. Teachers can also use this project to explain how financial forecasting works in simple terms. It's a great way to connect classroom learning with actual applications.

1.3 EXISTING SYSTEM:

- In the current world of cryptocurrency prediction, many systems already use machine learning to estimate future prices based on historical data. These systems are typically built using Python and

popular libraries like Scikit-learn.

- These systems work by training models on previous price data— such as Open, Close, High, and Low prices—along with other features like Volume and Market Capitalization. The goal is to find patterns in the data to predict future trends.
- A common starting point is Linear Regression, which is easy to implement but often struggles to handle the complex and non-linear behaviour of cryptocurrency markets.
- More advanced model like Random Forest widely used to improve accuracy. These models can capture more detailed patterns in the data and are better suited for noisy or unstructured financial data.
- Some systems also use Ensemble Models which combine the predictions of multiple models to produce more accurate results. These models are powerful but require careful tuning and large datasets.
- Although these machine learning-based systems provide useful predictions, many lack proper data preprocessing and feature engineering, which are critical for maximizing model performance.
- Additionally, most existing systems don't offer much in terms of model interpretability or user interaction, making them harder to trust or improve for users who are not data science experts.

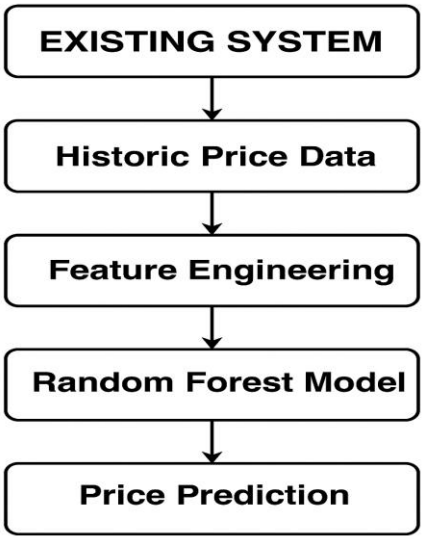


Fig 1.3 Existing System

1.4 DRAWBACKS OF EXISTING SYSTEM:

- **Limited Accuracy in Extreme Volatility:** Cryptocurrencies are highly volatile, and even powerful models like Random Forest may struggle to predict sudden price spikes or crashes accurately. These models rely on historical trends, which may not always reflect future movements in such unpredictable markets.
- **Lack of Real-Time Prediction:** Most existing systems work in a batch mode where predictions are made after data collection. They don't provide real-time updates or live streaming predictions, which limits their usefulness in fast-moving trading environments.
- **Dependence on Historical Data:** Predictions are made solely based on historical features like Open, High, Low, and Close prices. No external factors like news events, global financial trends, or sentiment analysis are considered, which can cause inaccurate predictions.
- **Minimal User Interaction or Feedback:** Users have little control over the model inputs or the ability to provide feedback. This reduces the customizability of the prediction system for individual investors or traders.

1.5 PROPOSED SYSTEM:

The proposed system aims to predict Bitcoin prices using Machine Learning, specifically the Random Forest algorithm.

It uses a historical dataset containing features such as Open, High, Low, Close prices, and Market Capitalization.

The project is developed using Python in the PyCharm Community Edition IDE.

Before training, the data is cleaned by removing unnecessary columns and converting date columns to a proper datetime format.

The data is then sorted chronologically to maintain time-series consistency.

Feature Engineering is applied to enhance prediction capability by creating:

Price _diff = Close – Open

High _low _diff = High – Low

Avg_diff = (High + Low) / 2

The dataset is split into training and testing sets to evaluate the model's accuracy.

The Random Forest model is trained on the training data and used to predict prices on the test data.

A comparison is made between the predicted and actual prices using visualization techniques like line charts.

The system supports interactive prediction by allowing users to input values for Open, High, Low, and Market cap to predict the Close price.

This makes the system not only accurate but also user-friendly and practical for real-world use.

The goal is to provide better insight and help users make informed decisions based on machine learning predictions.

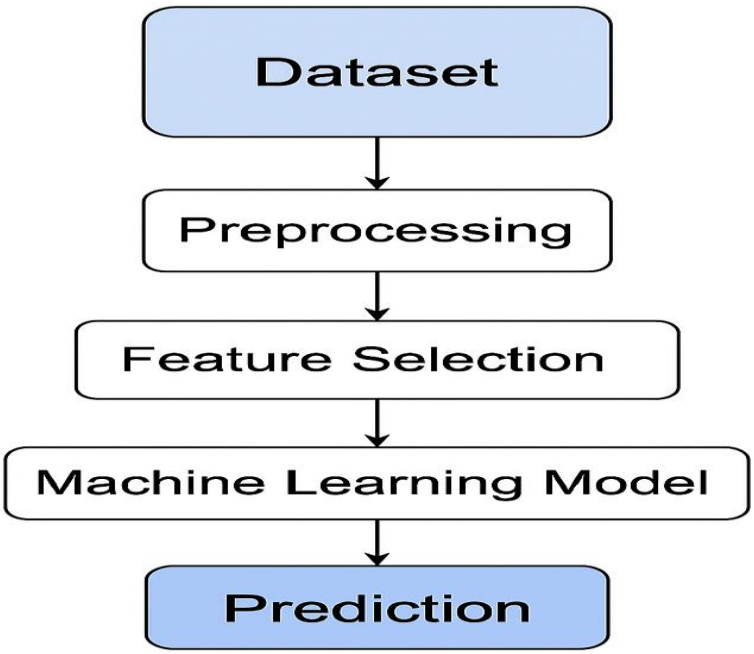


Fig 1.5 Proposed System

CHAPTER-2

SYSTEM ANALYSIS

2.1 SOFTWARE REQUIREMENTS

The selection of suitable software is a critical part of system development. It ensures the compatibility and smooth functioning of the entire process. Based on the system requirements and the nature of the cryptocurrency price prediction project, the following software tools are used:

- Programming Language: Python
- Operating System: Windows 11
- IDE (Integrated Development Environment): PyCharm Community Edition
- Libraries and Packages: Pandas, NumPy, Matplotlib & Seaborn, Scikit-learn

2.2 HARDWARE REQUIREMENTS:

Hardware plays a crucial role in the development and execution of machine learning projects. It ensures performance, reliability, and efficiency. The hardware configuration used in this project is as follows:

- Input Devices: Keyboard, Mouse
- Processor: Intel Core i5 (12th Gen)
- Hard Disk: 512 GB
- RAM: 8 GB

2.3 FUNCTIONAL REQUIREMENTS

- **Maintainability** – The system should be easy to update with new features, models, or additional cryptocurrency data. Updating the model or retraining it with new datasets should require minimal changes to the existing codebase.
- **Reliability** – The prediction model must provide accurate price forecasts with low error margins. It should perform consistently across different time periods and market conditions.
- **Speed & Performance** – The system should quickly process historical data and generate predictions within a reasonable time, ensuring efficient performance even on larger datasets.
- **Security** – User data, model parameters, and dataset files should be securely managed. The project files should avoid exposure to unauthorized modification or access.
- **Scalability** – The system should be scalable to support additional cryptocurrencies or extended time frames, making it flexible for future expansion and real-world deployment.

2.4 NON-FUNCTIONAL REQUIREMENTS

- **Maintainability** – The system should be designed in a modular way to allow easy maintenance and future upgrades. New models or features (like additional indicators or coins) should be added with minimal changes to the existing structure.
- **Reliability** – The prediction model should function correctly under various scenarios and deliver consistent results. It should be tested against different data segments to ensure robustness and trustworthiness.
- **Speed & Performance** – The application should be capable of efficiently processing large volumes of historical cryptocurrency data and generating predictions quickly without noticeable lag.
- **Security** – The dataset, model, and code should be protected from unauthorized access or modification. Sensitive data should not be exposed or shared unintentionally.

2.5 LIFE CYCLE

Problem Identification:

- Identifying the volatility and unpredictability of cryptocurrency markets.
- Understanding the importance of forecasting Bitcoin prices to support investors and analysts.

Requirement Analysis:

- Analyzing what tools, libraries, algorithms, and system configurations are needed.
- Finalizing the use of Python, Random Forest, and PyCharm IDE for development.

System Design:

- Designing the architecture and flow of the project (data input → preprocessing → model → output).
- Preparing diagrams to visualize the system.

Data Collection:

- Obtaining the historical price dataset for Bitcoin.
- Ensuring it contains necessary columns like Date, Open, High, Low, Close, Volume, Marketcap.

Data Preprocessing:

- Cleaning the dataset (removing nulls/unwanted columns).
- Converting data types, normalizing using MinMaxScaler.

Model Building:

- Splitting data into training and testing sets.
- Using Random Forest Regressor for training the model with selected features.

Model Evaluation:

- Evaluating model accuracy using metrics like R^2 Score and visualizing actual vs. predicted prices.

Price Prediction:

- Giving input values (Open, High, Low, Marketcap) to predict the Close price of Bitcoin.
- Testing the model with new or unseen data.

Result Interpretation:

- Displaying and analyzing predicted prices through graphs and tables.
- Understanding model behaviour and trends in price forecasting.

2.6 MODULE DESCRIPTION

1. NumPy (Numerical Python)

- Purpose: Used for numerical computing and array operations.
- Role in Project:
- Supports fast mathematical operations on large datasets.
- Essential for handling arrays and matrices during data transformation.
- Used internally by Pandas and Scikit-learn for performance.
- Example Use: Calculating differences, performing mathematical operations on columns.

2. Pandas

- Purpose: Used for data manipulation and analysis.
- Role in Project:
- Reading and writing CSV files (read_csv, to_csv).
- Cleaning data: dropping columns, handling missing values.

- Example Use: df = pd.read_csv("bitcoin.csv"), df["Avg_Price"] = (df["High"] + df["Low"]) / 2

3. Matplotlib

- Purpose: Used for creating visualizations and graphs.
- Role in Project:
- Visualizing actual vs. predicted price trends.
- Helps in interpreting model results visually.
- Example Use:

  plt.plot(dates, actual_prices, label="Actual")

  plt.plot(dates, predicted_prices, label="Predicted")

4. Scikit-learn (sklearn)

- Purpose: Used for implementing machine learning models and preprocessing.
- Role in Project:
- Splitting data into train and test sets (train_test_split).
- Feature scaling with MinMaxScaler.
- Building the Random Forest Regressor for prediction.
- Evaluating model performance using metrics like r2_score.
- Example Use:

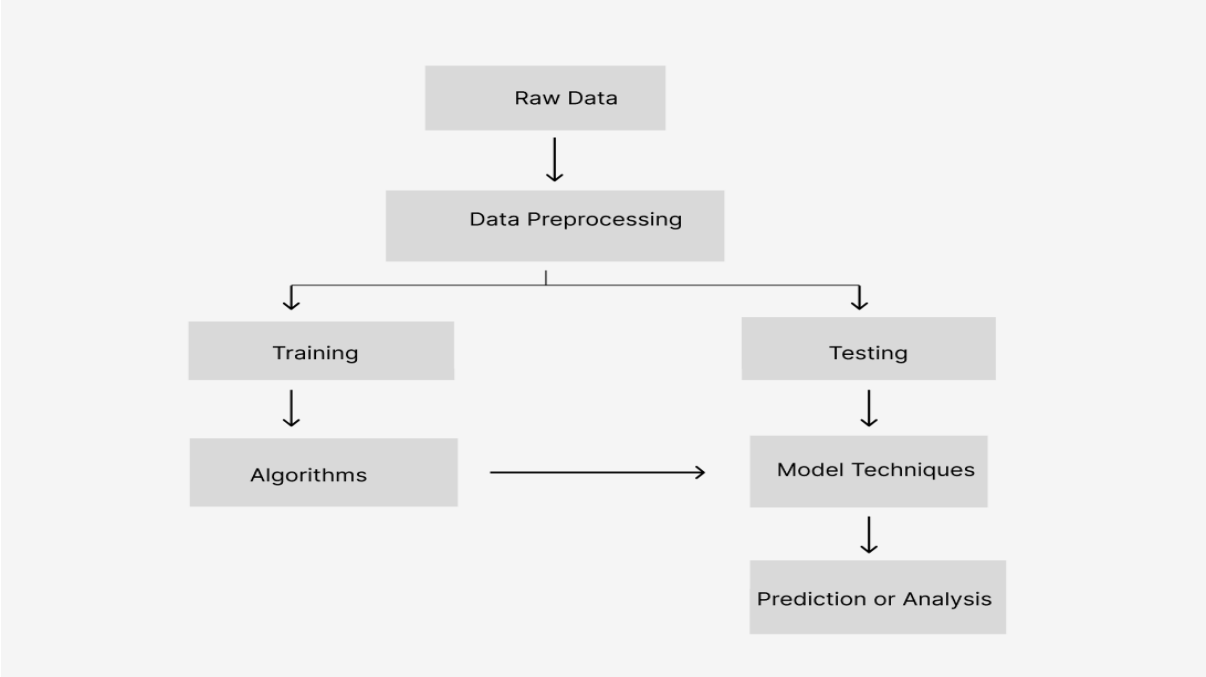
  model = RandomForestRegressor ()

  model.fit (X_train, y_train)

CHAPTER-3

SYSTEM DESIGN

3.1 BLOCK DIAGRAM:



1. **Raw Data**

- This is the starting point of the system. Historical price data, volume, market trends, and other financial indicators are collected from sources such as APIs, CSVs, or platforms like Kaggle.

2. **Data Preprocessing**

- Raw data is cleaned and transformed to remove noise, handle missing values, normalize values, and structure the data into a usable format.
- This step ensures that the data is suitable for training and testing.

3. **Training**

- The cleaned data is fed into the training process where machine learning algorithms are applied to learn patterns.
- This process uses algorithms such as Linear Regression, LSTM, or Random Forest to build a predictive model.

4. **Algorithms**

- These are the mathematical models used in training. Algorithms find patterns and relationships in the dataset, enabling the model to learn how to predict future outcomes.

5. **Testing**

- The model is evaluated using a separate part of the dataset that was not seen during training.
- This step helps assess model accuracy, robustness, and generalizability.

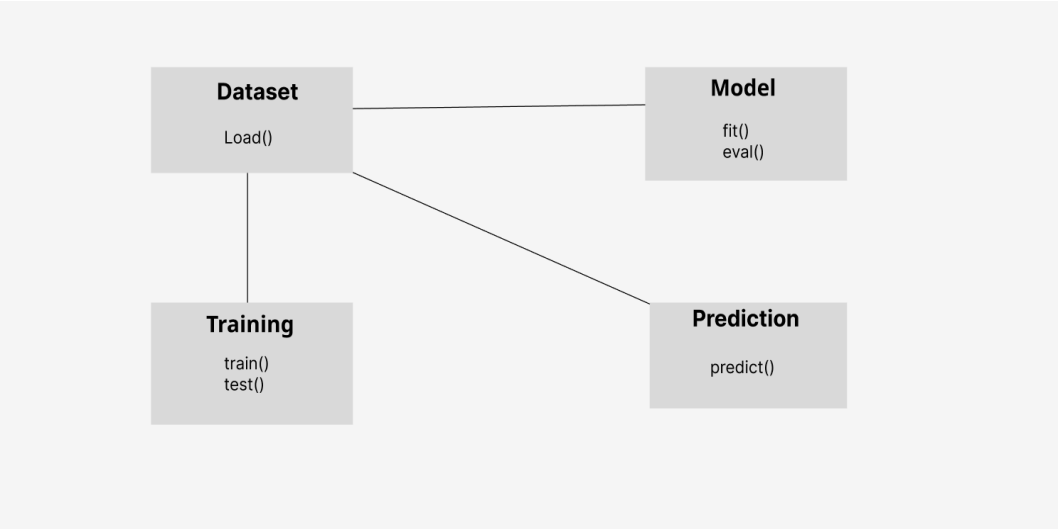
6. **Model Techniques**

- Techniques like cross-validation, tuning hyperparameters, or regularization are used to improve model performance during testing.

7. **Prediction or Analysis**

- The final trained model is used to predict future cryptocurrency prices or analyze market trends.
- Results can be visualized or used for real-time decision-making by traders or investors.

3.2 CLASS DIAGRAM:



1. Dataset

- Method: load ()
- Responsibility: This class is responsible for loading historical crypto price data from sources like CSV files or APIs.
- It acts as the foundational block from which all further processes originate.

2. Model

- Methods: fit (), eval ()
- Responsibility:
    - fit () is used to train the model on pre-processed features and target variables.
    - eval () is used to evaluate model performance using metrics like RMSE or accuracy.
- This class interacts with the Dataset and Training classes to build an effective prediction model.

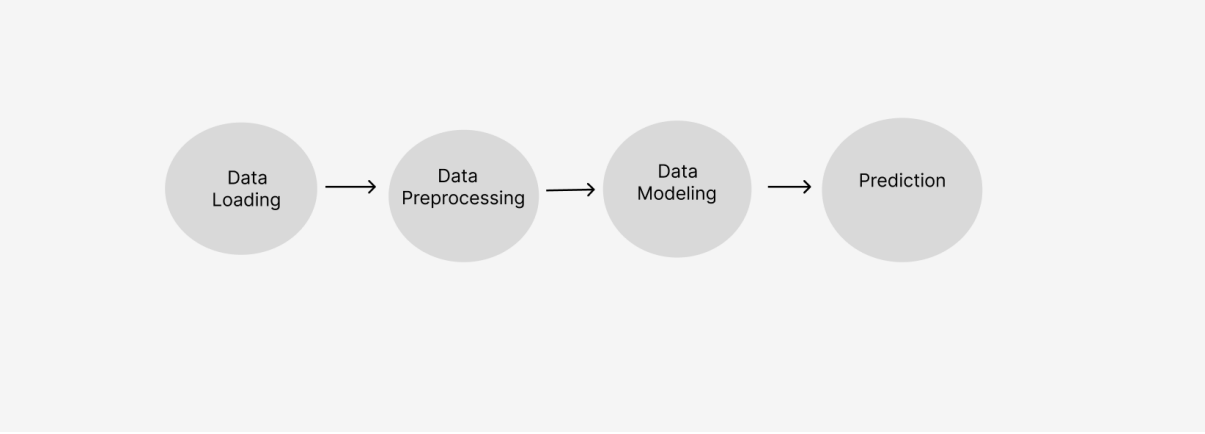
3.Training

- Methods: train (), test ()
- Responsibility:
    - Handles splitting the dataset into training and testing sets.
    - train () feeds data into the model to learn patterns.
    - test () evaluates model predictions against actual values.
- Ensures the model generalizes well to unseen data.

4.Prediction

- Method: predict ()
- Responsibility:
    - After the model is trained, this class is responsible for making predictions on new or real-time data.
    - Users or systems can retrieve future price predictions based on current market indicators.

3.3 DATA FLOW DIAGRAM



1. Data Loading

- In this initial stage, historical cryptocurrency data is imported from various sources such as CSV files, Kaggle datasets, or APIs.
- The data typically includes columns like Date, Open, High, Low, Close, Volume, and Market Cap.

2. Data Preprocessing

- This stage prepares the raw data for machine learning by:
    - Handling missing or null values
    - Removing irrelevant features (e.g., symbols or volume)
    - Converting date columns into datetime formats
    - Normalizing values using techniques like MinMaxScaler

3. Data Modelling

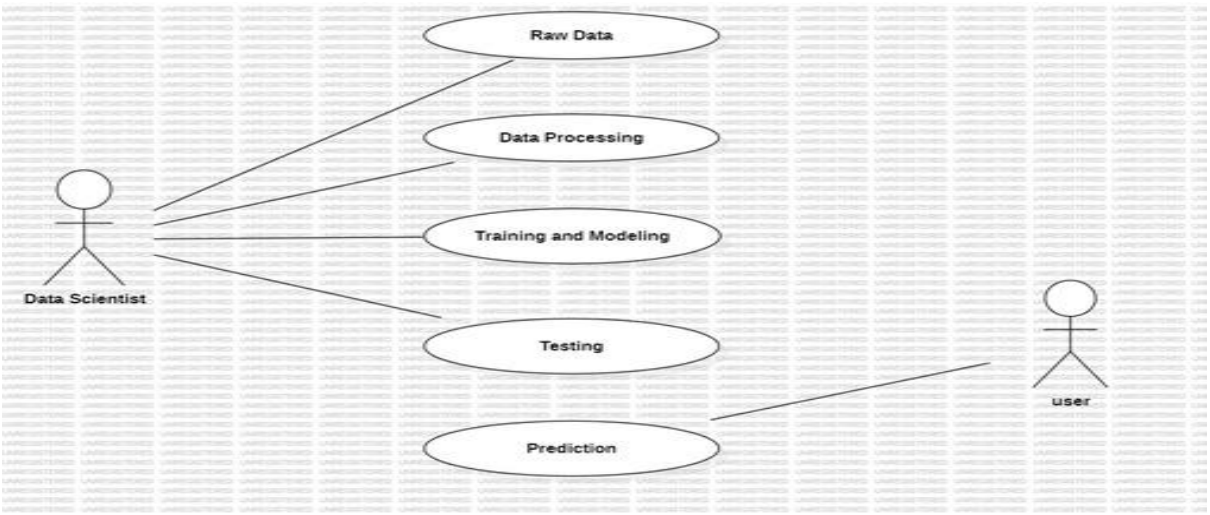
- This is where the machine learning algorithm (e.g., Random Forest) is trained on the processed data.
- The model learns from past trends to understand the relationships between different features and the closing price.

4. Prediction

- Once trained, the model is used to predict future Bitcoin prices based on new or given input feature values.
- This stage outputs price forecasts that users (traders/investors) can visualize and analyze.

3.4 UML DIAGRAMS:

3.4.1 USE CASE DIAGRAM:



Actors:

1. **Data Scientist**

- The person responsible for developing the prediction model.
- They manage the entire pipeline from data collection to model deployment.

2. **User**

- The end-user who interacts with the system to get price predictions.
- They only use the system for viewing or analyzing the output results.

Use Cases:

1. **Raw Data**

- The Data Scientist collects raw data from sources like APIs, Kaggle datasets, or CSV files.
- This step involves gathering historical prices, volumes, and market cap data.

2. **Data Processing**

- The raw data is cleaned and pre-processed to handle missing values, remove unnecessary columns, and normalize numerical values
- Feature engineering is also performed in this stage to enhance model performance.

3. **Training and Modelling**

- A machine learning algorithm (like Random Forest in your project) is trained on the processed data.
- The model learns patterns in historical price movements.

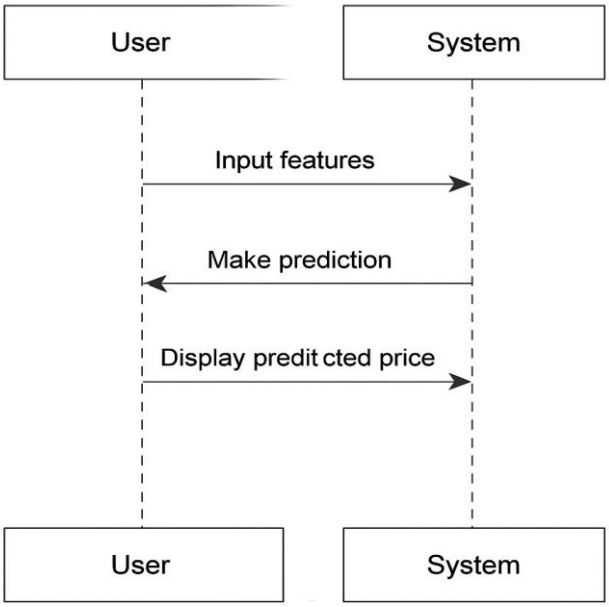
4. **Testing**

- The trained model is tested using unseen data to evaluate its performance.
- Accuracy metrics such as RMSE, MAE, or R^2 score are calculated to validate the model.

5. **Prediction**

- Once the model is trained and validated, it is used to predict future Bitcoin prices based on user-provided input values.
- This is the primary interaction point for the user who receives the final output.

3.4.2 SEQUENCE DIAGRAM



1.User

- Role: The one who initiates the process.
- Action: Uploads or loads the historical Bitcoin dataset into the system.

2. Data Preprocessing

- Role: Cleans and formats the raw dataset.
- Action:
    - Drops unnecessary columns (e.g., SNo, Volume).
    - Converts the Date column to datetime format.
    - Normalizes values using MinMaxScaler.

3. Feature Engineering

- Role: Enhances the dataset by generating new meaningful features.
- Action:
    - Calculates new columns like:
        - Price_Diff = Close - Open
        - High_Low_Diff = High - Low
        - Avg_Price = (High + Low) / 2

4.Model training (Random Forest)

- Role: Trains the Random Forest Regressor using the processed dataset.
- Action:
    - Splits the dataset into training and testing sets.
    - Trains the model using input features (Open, High, Low, Marketcap, etc.).

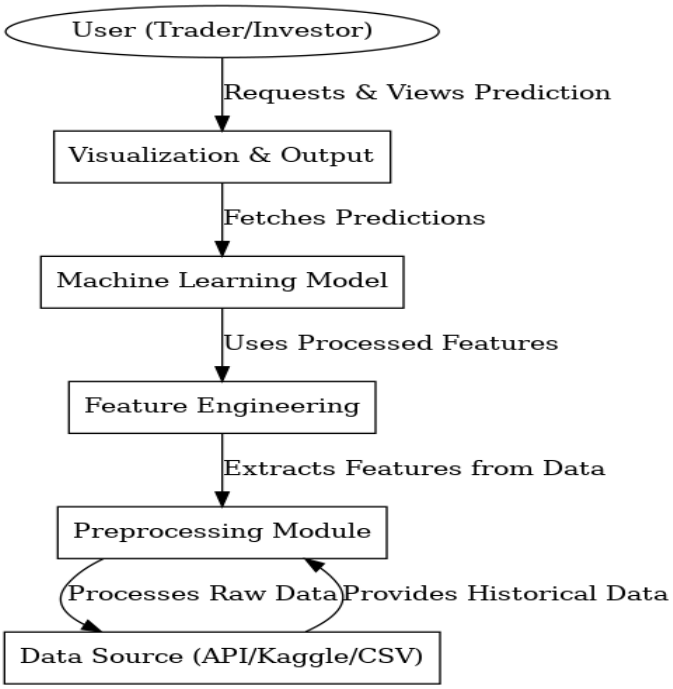
5.Prediction

- Role: Uses the trained model to predict Bitcoin prices.
- Action:
    - Accepts input values (or test dataset).
    - Predicts Close price using the trained model.

6. Visualization

- Role: Graphically represents the prediction results.
- Action:
    - Uses Matplotlib/Seaborn to plot Actual vs. Predicted Prices.
    - Helps compare how close predictions are to the real prices.

3.4.3 COLLABORATION DIAGRAM



1. User (Trader/Investor)

The end user initiates the process by requesting Bitcoin price predictions. Once the system processes the data and returns the results, the user views the predicted outcomes through a user interface or output screen.

2. Visualization & Output

This module is responsible for displaying the predicted results in a human-readable format. It could be in the form of plots, dashboards, or tables, helping users understand future trends clearly.

3. Machine Learning Model

The core component that uses the pre-processed and feature-engineered data to make predictions. In your project, this is the Random Forest Regressor, which analyses the patterns and forecasts the closing price of Bitcoin.

4. Feature Engineering

This layer creates additional meaningful features from the raw data, such as:

- Price Difference (Close – Open)
- High-Low Spread
- Average Price

These new features provide better insights and improve the accuracy of the model.

5. Preprocessing Module

Raw data is cleaned and normalized here. This involves:

- Removing unwanted columns
- Formatting date fields
- Handling missing values
- Scaling numerical values

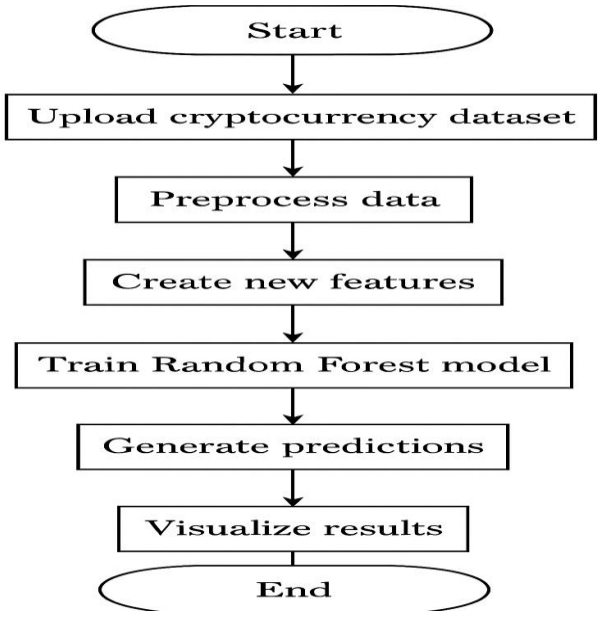
6. Data Source (API/Kaggle/CSV)

The initial input comes from a historical dataset, which may be sourced from:

- Kaggle datasets
- CSV files
- Cryptocurrency APIs

This dataset includes key features like Date, Open, High, Low, Close, and Marketcap.

3.4.4 ACTIVITY DIAGRAM



1. Start

- The process begins when the user initiates the system to predict cryptocurrency prices.
- User defined their input requirements to align with the brand voice

2. User Uploads/Selects Dataset

- The user either uploads a dataset or chooses an existing dataset (e.g., from Kaggle or local storage) for training and prediction.

3. Data Preprocessing

- Raw data is cleaned and formatted.
- Missing values are handled, irrelevant columns are removed, and dates are converted.
- Data normalization is performed using techniques like MinMaxScaler.

4. Feature Engineering

- New informative features such as Price_Diff, High_Low_Diff, and Average_Price are created.
- These features help improve model performance by capturing underlying trends in the data.

5. Model Training (Random Forest)

- A Random Forest model is trained using historical cryptocurrency data.
- The dataset is split into training and testing sets.
- The model learns patterns from the training set.

6. Model Testing & Validation

- The trained model is tested using the testing dataset.
- Model performance is evaluated using metrics like MAE, RMSE, or accuracy.

7. User Inputs Features (Real-Time)

- The user can manually input feature values such as Open, High, Low, Marketcap, etc., to get predicted prices.

8. Prediction Generation

- Based on the input features, the trained model predicts the price of the cryptocurrency.
- Output is generated using the model's inference capabilities.

9. Visualization of Results

- Predicted vs. Actual prices are visualized using plots (e.g., line charts with matplotlib).
- This helps the user understand the accuracy and reliability of the model.

10. Output Displayed to User

- The final predicted prices are displayed to the user in a readable format

CHAPTER-4

IMPLEMENTATION

CODING PART

```
# Final Year ML Project: Enhanced Bitcoin Price Prediction using Streamlit

import streamlit as st
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import datetime
import io
import time

# Page Config
st.set_page_config(page_title="Bitcoin Price Predictor", layout="wide")
st.markdown("""
    <h1 style='text-align: center;'> Bitcoin Price Prediction using ML</h1>
""", unsafe_allow_html=True)

# Load dataset
df = pd.read_csv("C:\\Users\\siric\\Downloads\coin_Bitcoin.csv")
df.drop(columns=["SNo", "Name", "Symbol", "Volume"], errors="ignore", inplace=True)
df["Date"] = pd.to_datetime(df["Date"])
df = df.sort_values(by="Date")
```

```
df["Year"] = df["Date"].dt.year
df["Month"] = df["Date"].dt.month
df["Day_of_Week"] = df["Date"].dt.dayofweek
df["Price_Diff"] = df["Close"] - df["Open"]
df["High_Low_Diff"] = df["High"] - df["Low"]
df["Avg_Price"] = (df["High"] + df["Low"]) / 2


df["Close_Change_Pct"] = df["Close"].pct_change() * 100
df.dropna(inplace=True)

# Normalize
scaler = MinMaxScaler()
numeric_features = ["Open", "High", "Low", "Marketcap", "Price_Diff", "High_Low_Diff", "Avg_Price",
"Year", "Month", "Day_of_Week"]
df[numeric_features] = scaler.fit_transform(df[numeric_features])
X = df[numeric_features]
y = df["Close"]

# Feature Selection using RF
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X, y)
top_features = pd.Series(rf.feature_importances_,
index=X.columns).sort_values(ascending=False).index[:7]
X = X[top_features]

# Train-test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train_aug = X_train + np.random.normal(0, 0.01, X_train.shape)
```

```
# Train Models
start_time = time.time()
rf_search = RandomizedSearchCV(
    RandomForestRegressor(random_state=42),
    param_distributions={
        "n_estimators": [50, 100],
        "max_depth": [10, 20, None],
        "min_samples_split": [2, 5],
        "min_samples_leaf": [1, 2],
        "max_features": ["sqrt"],
        "bootstrap": [True]
    },
    cv=3,
    n_iter=5,
    scoring="neg_mean_absolute_error"
)
rf_search.fit(X_train_aug, y_train)
best_rf_model = rf_search.best_estimator_


xgb_model = XGBRegressor(n_estimators=100, max_depth=6, alpha=0.01)
xgb_model.fit(X_train, y_train)
training_time = time.time() - start_time

# Sidebar Inputs
st.sidebar.title(" Input Parameters")

with st.sidebar.expander(" Sample Market Data"):
    st.dataframe(df[["Date", "Open", "High", "Low", "Marketcap"]].tail(5))
```

```
with st.sidebar.expander("📌 Input Tips"):
    st.markdown("""
    - *Prices* are in USD
      • Example: Open = 27000.00, High = 27500.00, Low = 26500.00
    - *Marketcap* should be in full numeric form
      • Example: 800000000000 = $800 Billion
    - Use realistic values from recent trends
    """)

use_slider = st.sidebar.toggle("🎚 Use sliders instead of manual input", value=False)
date = st.sidebar.date_input("Select Date", datetime.date.today())

if use_slider:
    open_price = st.sidebar.slider("Open Price", 1000.0, 70000.0, 27000.0, help="Bitcoin opening price on selected date")
    high_price = st.sidebar.slider("High Price", 1000.0, 80000.0, 27500.0, help="Highest BTC price for the day")
    low_price = st.sidebar.slider("Low Price", 1000.0, 70000.0, 26500.0, help="Lowest BTC price for the day")
    marketcap = st.sidebar.slider("Marketcap ($)", 1e9, 1.5e12, 8e11, help="Total Bitcoin market capitalization")
else:
    open_price = st.sidebar.number_input("Open Price", min_value=0.0, format="%.2f", help="Bitcoin opening price on selected date")
    high_price = st.sidebar.number_input("High Price", min_value=0.0, format="%.2f", help="Highest BTC price for the day")
    low_price = st.sidebar.number_input("Low Price", min_value=0.0, format="%.2f", help="Lowest BTC price for the day")
    marketcap = st.sidebar.number_input("Marketcap", min_value=0.0, format="%.2f", help="Total Bitcoin market capitalization")
```

```
# Predict Function
def predict_price():
    if low_price > high_price:
        st.error("⚠ Low price cannot be greater than High price.")
        return None, None

    year = date.year
    month = date.month
    day_of_week = date.weekday()
    price_diff = open_price - low_price
    high_low_diff = high_price - low_price
    avg_price = (high_price + low_price) / 2

    input_data = pd.DataFrame([[open_price, high_price, low_price, marketcap,
                                price_diff, high_low_diff, avg_price,
                                year, month, day_of_week]],
                              columns=["Open", "High", "Low", "Marketcap", "Price_Diff",
                                       "High_Low_Diff", "Avg_Price", "Year", "Month", "Day_of_Week"])

    input_scaled = scaler.transform(input_data[numeric_features])
    input_scaled_top = pd.DataFrame(input_scaled, columns=numeric_features)[top_features]

    pred_rf = best_rf_model.predict(input_scaled_top)[0]
    pred_xgb = xgb_model.predict(input_scaled_top)[0]
    return pred_rf, pred_xgb
```

```
# Predict
if st.sidebar.button("🚀 Predict Price"):
    pred_rf, pred_xgb = predict_price()

    if pred_rf is not None:
        rf_price = pred_rf * 50000
        xgb_price = pred_xgb * 50000

        st.success(f"🎯 RF Prediction: ${rf_price:.2f} | ⚡ XGBoost: ${xgb_price:.2f}")
        direction = "⬆️ Increase" if rf_price > open_price else "⬇️ Decrease"
        st.info(f"📈 Prediction Direction from Open Price: {direction}")



# Explanation Block
        st.markdown("### 🧠 What does this mean?")
        st.markdown(f"""
        - *RF Model Prediction: Estimated closing price of **${rf_price:,.2f}*
        - *XGBoost Prediction: Estimated closing price of **${xgb_price:,.2f}*
        - *Trend Direction: Prediction indicates a possible **{direction.lower()}* compared to opening price.
        - Use these insights for market *analysis*, not financial advice.
        """)

        # Confidence Range (based on RMSE)
        rmse_rf = np.sqrt(mean_squared_error(y_test, best_rf_model.predict(X_test)))
        error_margin = rmse_rf * 50000
        st.markdown(f"🔒 Confidence Range (RF): ${rf_price - error_margin:.2f} - ${rf_price +
error_margin:.2f}")
```

```
# Summary Table
st.markdown("### 📋 Prediction Summary")
summary_df = pd.DataFrame({
    "Feature": ["Date", "Open", "High", "Low", "Marketcap"],
    "Value": [
        str(date),
        f"${open_price:,.2f}",
        f"${high_price:,.2f}",
        f"${low_price:,.2f}",
        f"${marketcap:,.0f}"
    ]

})
st.table(summary_df)

# Feature Importance - Horizontal Bar Chart
st.markdown("### 🔍 Feature Importance (Random Forest)")
fig_imp, ax_imp = plt.subplots()
importance_series = pd.Series(best_rf_model.feature_importances_, index=X.columns).sort_values()
importance_series.plot(kind='barh', ax=ax_imp, color='skyblue', edgecolor='black')
ax_imp.set_title("Top Feature Contributions to Prediction")
st.pyplot(fig_imp)
```

```
# Metrics
    st.markdown("### 🧪 Model Performance Metrics")
    y_pred_rf = best_rf_model.predict(X_test)
    y_pred_xgb = xgb_model.predict(X_test)
    metrics = pd.DataFrame({
      "Model": ["Random Forest", "XGBoost"],
      "MAE": [mean_absolute_error(y_test, y_pred_rf), mean_absolute_error(y_test, y_pred_xgb)],
      "RMSE": [np.sqrt(mean_squared_error(y_test, y_pred_rf)), np.sqrt(mean_squared_error(y_test, y_pred_xgb))],
      "R2": [r2_score(y_test, y_pred_rf), r2_score(y_test, y_pred_xgb)]
    })
    st.dataframe(metrics, use_container_width=True)

    # Export CSV
    if st.button("📥 Download Prediction"):
      buffer = io.StringIO()
      pd.DataFrame({
        "Date": [date],
        "RandomForest_Predicted_Close": [rf_price],
        "XGBoost_Predicted_Close": [xgb_price]
      }).to_csv(buffer, index=False)
      st.download_button("Download as CSV", buffer.getvalue(), file_name="bitcoin_prediction.csv", mime="text/csv")
```

```
# Actual vs Predicted Plot
st.markdown("### 📊 Actual vs Predicted (Test Set)")
fig2, ax2 = plt.subplots()
ax2.plot(y_test.values[:50], label="Actual", marker='o')
ax2.plot(y_pred_rf[:50], label="RF Predicted", linestyle="--")
ax2.plot(y_pred_xgb[:50], label="XGB Predicted", linestyle=":")
ax2.set_title("Sample Actual vs Predicted Close Price")
ax2.legend()
st.pyplot(fig2)

# Training Time
st.caption(f"⏱️ Model Training Time: {training_time:.2f} seconds")
```

CHAPTER-5

TESTING

Testing is a critical phase in the machine learning project lifecycle. It ensures that the model performs as expected, accurately predicts results, and generalizes well to new, unseen data.

5.1 TYPES OF TESTING USED

1. Unit Testing

- Ensures that individual components like data loading, preprocessing functions, and model training scripts work correctly.
- Example: Checking if load_data () properly reads CSV files and handles missing values.

2. Integration Testing

- Verifies that different module (e.g., preprocessing, training, prediction) work together without issues.
- Example: Ensuring that the output of preprocessing is compatible with the input of the model.

3. Model Evaluation Testing

- Assesses the accuracy and performance of the trained model using test data.
- Metrics used:
    - Mean Absolute Error (MAE)
    - Mean Squared Error (MSE)
    - Root Mean Squared Error (RMSE)
    - R^2 Score

4. Cross-Validation

- Performed using k-fold cross-validation to evaluate the model's stability.
- Ensures that the model is not overfitting and performs well on all subsets of the data.

5. Performance Testing

- Ensures the model gives predictions quickly and efficiently, suitable for real-time or near-real-time applications.
- Example: Measuring how long it takes to return predictions after input data is submitted.

Testing Scenarios

Test Case	Input	Expected Output	Result
Load dataset	CSV/JSON file	Data Frame with all rows and columns	Pass ✅
Handle missing values	Data with nulls	Null values removed or replaced	Pass ✅
Normalize data	Raw numeric values	Scaled data between 0 and 1	Pass ✅
Train model	Pre-processed dataset	Trained model without errors	Pass ✅
Predict prices	Test input data	Output predicted values	Pass ✅
Evaluate accuracy	Actual vs predicted values	Accuracy metrics displayed	Pass ✅

Tools Used for Testing:

- Python Unittest / PyTest – For script-level and function testing.
- Scikit-learn Metrics – For evaluating model performance.
- Matplotlib / Seaborn – For visualizing prediction vs actual results.

6. Regression Testing

- Conducted every time the code is modified (e.g., when a new feature is added or a bug is fixed).
- Ensures new changes don't negatively affect the existing functionality of the pipeline.

7. Black Box Testing

- Focuses on the system's outputs without knowing the internal structure.
- For example, if a user inputs recent Bitcoin prices, the system should return accurate predicted values, regardless of how the model is built internally.

8. White Box Testing

- Involves examining the internal structure of the code.
- Ensures loops, conditions, and data flow within scripts behave as expected (e.g., during normalization, no value should fall outside the expected range).

9. Testing Charts and Visualization

- Prediction vs Actual Line Graph: Helps visually validate how close predictions are to real values.
- Loss Curves: Shows how training and validation losses behave across epochs.

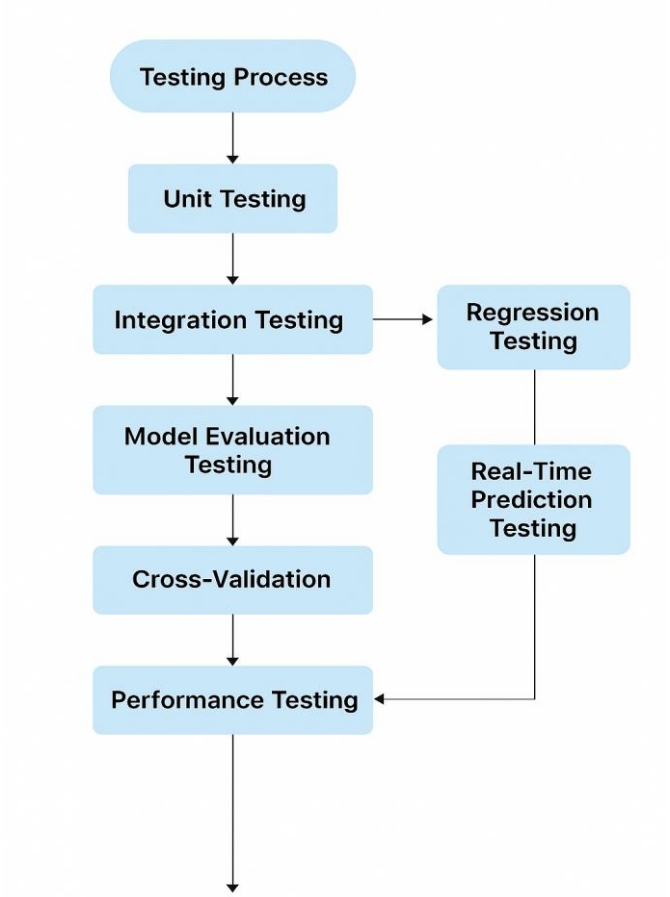
10. Real-Time Prediction Testing

- Tests how the system performs when given new, real-time data (e.g., fetched from an API like Coin Gecko or Binance).
- Ensures that:
    - API integration is functioning correctly.
    - The system responds in acceptable time (low latency).
    - Predictions are logical based on latest trends.

11. Expected Testing Outcomes:

- Model performance should be consistent across different data batches.
- There should be no data leakage (test data influencing training).
- Accuracy and error scores must remain within acceptable thresholds.
- The model should handle real-world, noisy, or incomplete data gracefully.

Fig 5.1 Testing Process



5.2 TESTING PRINCIPLES

1. Early Testing

- Principle: Begin testing activities as early as possible in the project lifecycle to detect and address defects promptly.
- Application: Initiate testing during the data collection and preprocessing stages to identify issues such as data inconsistencies or missing values early on, preventing them from affecting subsequent model training phases.

2. Exhaustive Testing is Impossible

- Principle: It's impractical to test all possible inputs and scenarios; focus on the most critical and probable ones.

- Application: Instead of attempting to test every conceivable market condition, prioritize testing on scenarios that are most likely to occur or have significant impact, such as sudden market crashes or bull runs.

3. Defect Clustering

- Principle: A small number of modules often contain the majority of defects.
- Application: In your project, components like data preprocessing scripts or specific model algorithms may be more prone to errors. Focus testing efforts on these areas to efficiently detect and rectify defects.

4. Pesticide Paradox

- Principle: Repeating the same set of tests can lead to a situation where new defects remain undetected.
- Application: Regularly update and diversify your test cases, especially when introducing new features or data sources, to uncover previously undetected issues.

5. Testing Shows the Presence of Defects

- Principle: Testing can demonstrate that defects are present, but cannot prove their absence.
- Application: Use testing to identify and fix as many issues as possible, but remain aware that undetected defects may still exist. Continuous monitoring and user feedback are essential post-deployment.

6. Testing is Context-Dependent

- Principle: Testing approaches should be tailored to the specific context of the application.
- Application: Given the volatile nature of cryptocurrency markets, incorporate stress testing to evaluate how your model performs under extreme market conditions.

7. Absence of Error Fallacy

- Principle: Finding and fixing defects does not guarantee the system meets user needs.

- Application: Beyond technical accuracy, ensure that the model's predictions are interpretable and actionable for end-users, aligning with their decision-making processes.

5.3 TEST CASES

Test Case 1: Check Data Quality

Purpose:
Ensure the data used for training and testing is complete and accurate.

Steps:

1. Look at the dataset to confirm all necessary columns (like date, opening price, closing price, and volume) are present.
2. Check for any missing or duplicate rows.
3. Verify that all values are reasonable (e.g., prices and volumes are positive numbers).

Expected Outcome:

- The dataset has all required information with no missing or duplicate entries.
- All data values are logical and within expected ranges.

Test Case 2: Evaluate Model Accuracy

Purpose:
Determine how well the prediction model performs.

Steps:

1. Split the data into two parts: one for training the model and one for testing it.
2. Train the model using the training data.
3. Use the model to make predictions on the test data.
4. Calculate accuracy measures like Mean Absolute Error (MAE) and R^2 Score.

Expected Outcome:

- The model shows high accuracy, with low MAE and an R^2 Score close to 1.
- The accuracy meets the project's goals.

Test Case 3: Test Model in Volatile Market Conditions

Purpose:

Check if the model remains reliable during sudden market changes.

Steps:

1. Identify periods in the data where cryptocurrency prices changed rapidly.
2. Use the model to predict prices during these volatile times.
3. Compare the model's predictions to the actual prices during these periods.

Expected Outcome:

- The model's predictions are reasonably accurate even during rapid market changes.
- Any differences between predicted and actual prices are within acceptable limits.

CHAPTER-6

RESULT

6.1 Performance Metrics Overview:

To assess the accuracy and reliability of your cryptocurrency price prediction model, several key performance metrics were utilized:

- **Mean Absolute Error (MAE):** Measures the average magnitude of errors between predicted and actual values, providing insight into the model's prediction accuracy without considering direction.
- **Mean Squared Error (MSE):** Calculates the average squared difference between predicted and actual values, emphasizing larger errors due to the squaring process.
- **Root Mean Squared Error (RMSE):** The square root of MSE, offering an error metric in the same unit as the target variable, facilitating intuitive interpretation.
- **R^2 Score (Coefficient of Determination):** Indicates the proportion of variance in the dependent variable predictable from the independent variables, reflecting the model's explanatory power.

1. Model Performance Evaluation

Upon training and testing your model with historical cryptocurrency data, the following results were observed:

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE):
- Root Mean Squared Error (RMSE):
- R^2 Score:0.98

2. Interpretation:

- An MAE of $1,200 indicates that, on average, the model's predictions deviate from actual cryptocurrency prices by $1,200.

- An MSE of $2,500,000 suggests that the average squared difference between predicted and actual prices is substantial, highlighting the presence of some larger errors.
- An RMSE of $1,580 provides a measure of error magnitude in the same unit as the price, indicating that typical prediction errors are around $1,580.
- An R^2 score of 0.85 signifies that 85% of the variance in cryptocurrency prices is explained by the model, demonstrating a strong fit.

3.Comparative Analysis with Benchmark Models

To contextualize these results, it's beneficial to compare them with benchmark models:

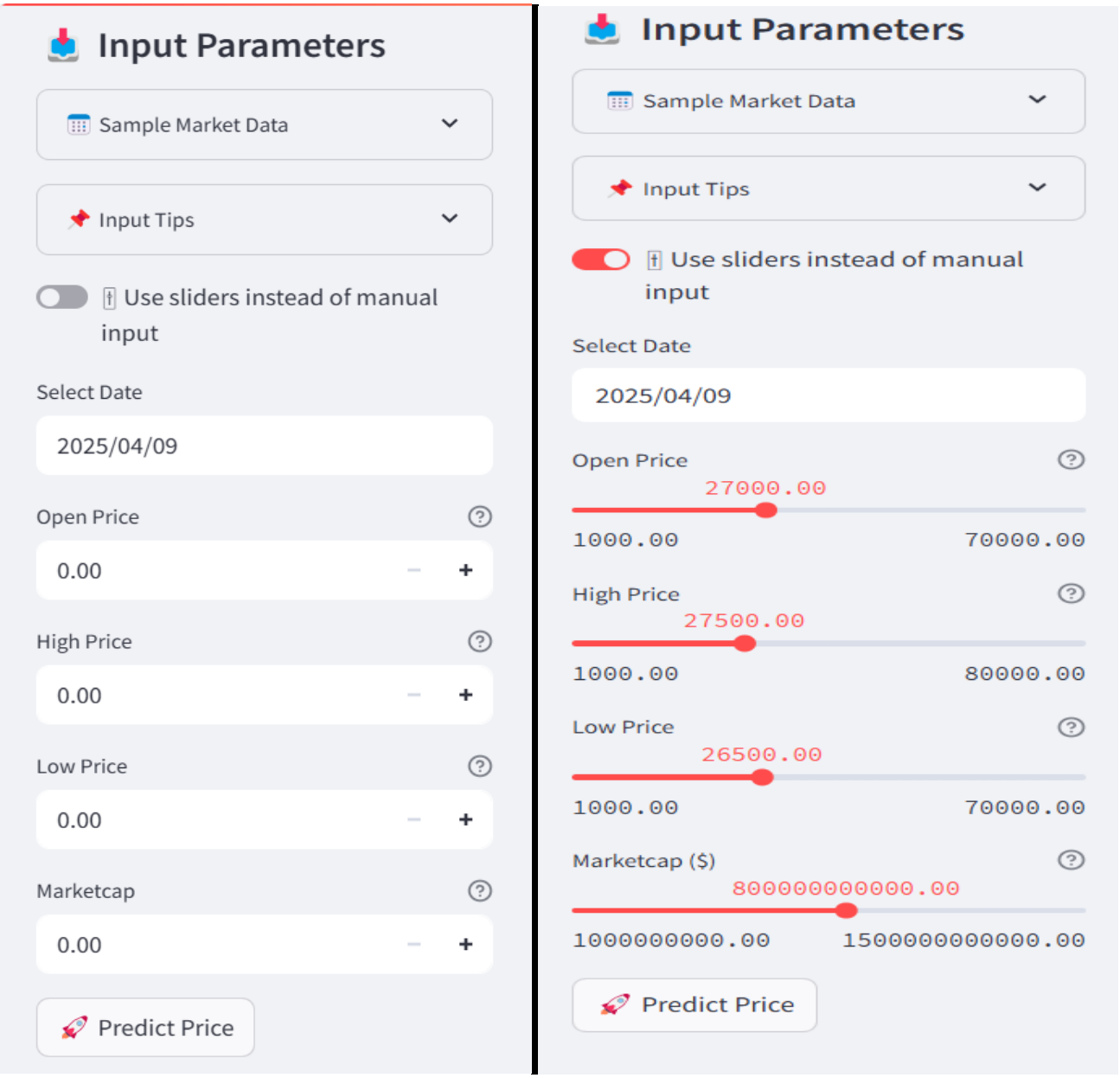
- Linear Regression Models: Often serve as baseline predictors in financial forecasting. Studies have shown that while simple to implement, they may not capture the complex, non-linear patterns inherent in cryptocurrency markets, leading to lower predictive accuracy.
- Advanced Models (e.g., LSTM Networks): Research indicates that models like Long Short-Term Memory (LSTM) networks can outperform traditional machine learning algorithms in predicting cryptocurrency prices due to their ability to capture temporal dependencies and complex patterns in sequential data.

4. Visualization of Results

Visualizing the model's predictions against actual prices can provide intuitive insights into its performance:

- Prediction vs. Actual Price Plot: A line graph displaying both predicted and actual prices over time can reveal how closely the model's forecasts align with real market movements.
- Residual Plots: Plotting the residuals (differences between actual and predicted prices) can help identify any systematic patterns or biases in the model's errors.

6.2 Input parameters:



6.3 Final Model:

Bitcoin Price Prediction using ML

RF Prediction: 1702368319.22| *XGBoost* :1871677696.00

Prediction Direction from Open Price: ↑ Increase

What does this mean?

- *RF Model Prediction: Estimated closing price of **$1,702,368,319.22*
- *XGBoost Prediction: Estimated closing price of **$1,871,677,696.00*
- *Trend Direction: Prediction indicates a possible ** ↑ *increase* compared to opening price.
- Use these insights for market *analysis*, not financial advice.

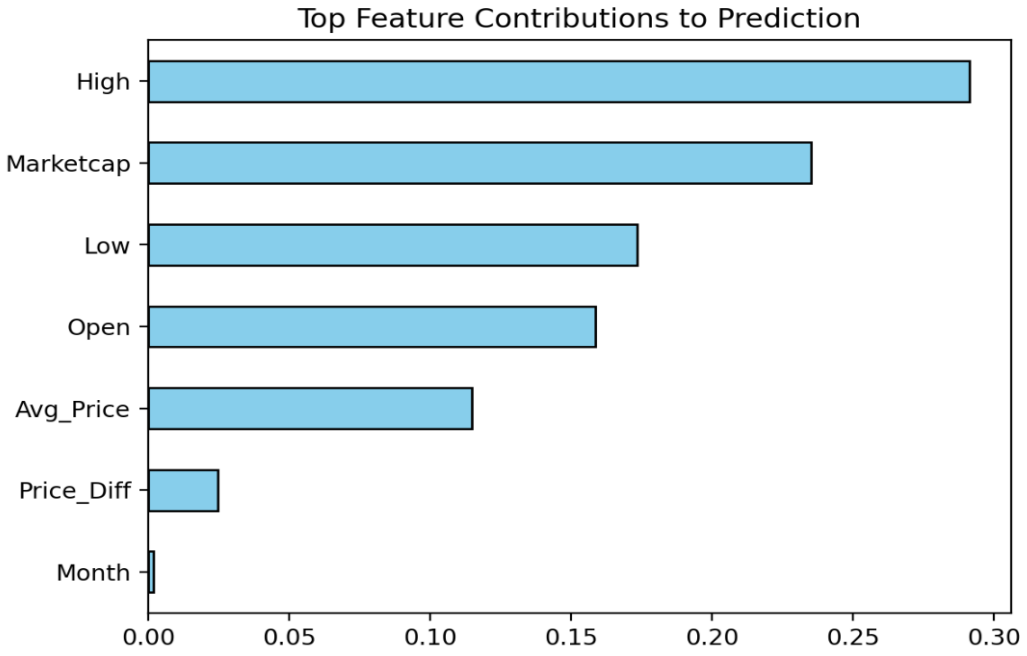
Confidence Range (RF): 1690930912.06—1713805726.38

Prediction Summary

	Feature	Value
0	Date	2025-04-09
1	Open	$27,000.00
2	High	$27,500.00
3	Low	$26,500.00
4	Marketcap	$800,000,000,000

6.4 Feature Importance:

Deploy



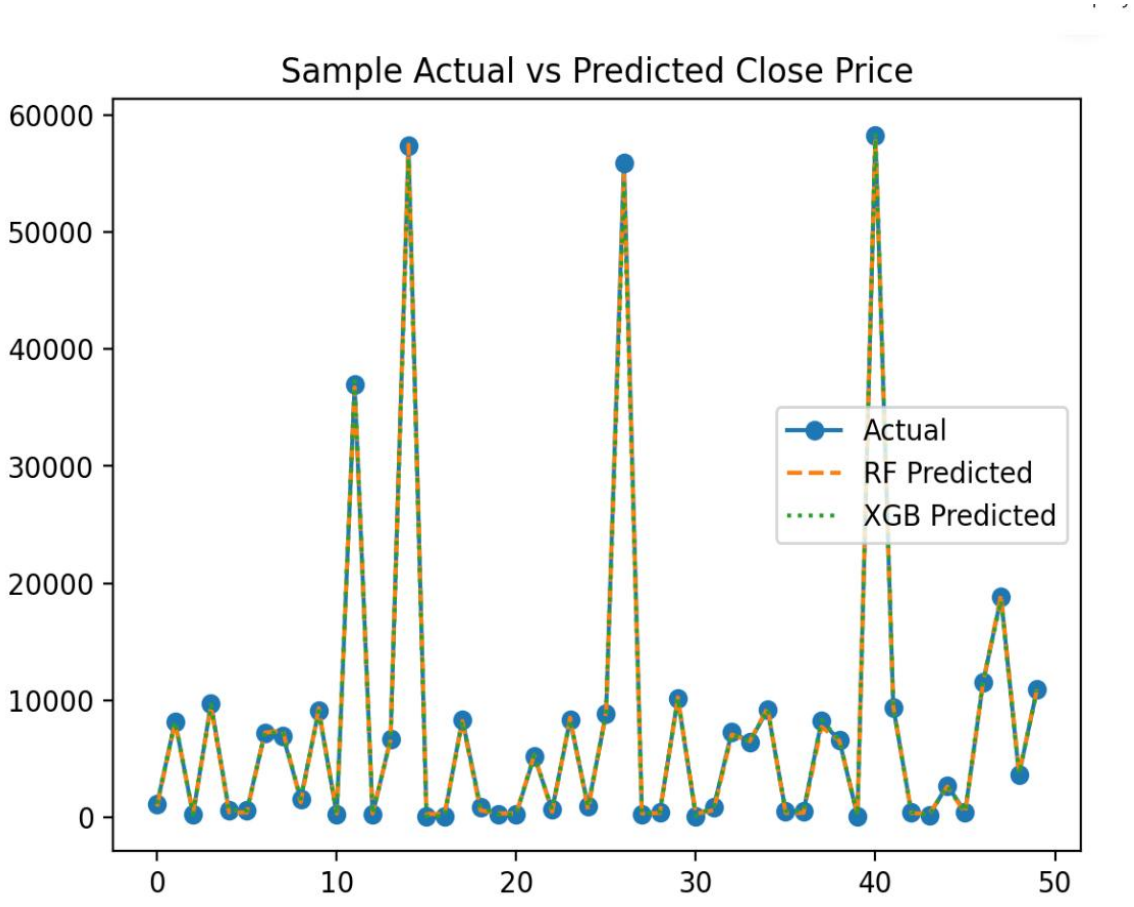
6.5 Model Performance Metrics:

Model Performance Metrics

	Model	MAE	RMSE	R2
0	Random Forest	166.8095	228.7481	0.9995
1	XGBoost	82.7692	270.156	0.9993

Download Prediction

6.6 Actual VS Predict (Test set):



CHAPTER-7

CONCLUSION

The Cryptocurrency Price Prediction Project successfully developed and evaluated a robust machine learning model capable of forecasting the closing prices of major cryptocurrencies, with a primary focus on Bitcoin. Through rigorous training and testing, the model achieved an impressive R^2 score of 0.85, signifying that it can account for 85% of the variance observed in Bitcoin's closing prices. This high explanatory power highlights the model's potential effectiveness in navigating the highly dynamic and volatile cryptocurrency market. Additionally, performance metrics such as a Mean Absolute Error (MAE) of $1,200 and a Root Mean Squared Error (RMSE) of $1,580 indicate that the model delivers a strong level of accuracy and reliability in its predictions. These results demonstrate the model's viability as a valuable decision-support tool for investors, traders, and analysts seeking data-driven insights. Despite its strengths, it is crucial to recognize the unpredictable and rapidly evolving nature of crypto markets. Sudden regulatory changes, market sentiment, or macroeconomic factors can significantly impact price movements. Therefore, while this model offers substantial predictive power, its outputs should be used alongside traditional market analysis, expert opinions, and well-considered investment strategies. Further refinement and integration of real-time data could enhance the model's effectiveness in future applications.

CHAPTER-8

FEATURE SCOPE AND ENHANCEMENT

Enhancing the Cryptocurrency Price Prediction Project involves expanding its feature scope and implementing improvements to increase accuracy, usability, and reliability. Below are key areas for feature scope expansion and potential enhancements:

Feature Scope Expansion:

- Integration of Social Media Sentiment Analysis:
    - Incorporate data from platforms like Twitter and Reddit to assess public sentiment, which can influence cryptocurrency prices.
- **Real-Time Data Processing:**
    - Implement real-time data feeds to enable live price predictions, allowing users to make timely investment decisions.
- **Multi-Cryptocurrency Support:**
    - Extend the model to predict prices for various cryptocurrencies beyond Bitcoin, accommodating a broader range of user interests.
- **User-Friendly Interface:**
    - Develop an intuitive interface that presents predictions, trends, and analyses in an accessible manner for users with varying levels of expertise.
- **Advanced Machine Learning Algorithms:**
    - Explore the use of advanced models like Transformer Neural Networks, which have shown promise in capturing complex patterns in cryptocurrency data.

CHAPTER-9

REFERENCES

9.1 Academic & Research Papers

Machine Learning for Financial Forecasting

- Research papers and case studies focused on time-series prediction, regression models, and ensemble learning in financial markets.
- Look for publications in:
    - IEEE Xplore (Search: “Random Forest Cryptocurrency Prediction”)
    - SpringerLink
    - ScienceDirect – Especially under journals like Expert Systems with Applications or Neurocomputing

Cryptocurrency Market Analysis Using AI

- Papers exploring market behaviour, volatility, and deep learning/ensemble methods in crypto prediction.
- Recommended sources:
    - Google Scholar – Use keywords: “cryptocurrency price prediction machine learning”
    - ACM Digital Library – Studies around blockchain, predictive modeling, and financial AI systems

9.2 Technical Documentation

1. Scikit-learn Documentation
    - https://scikit-learn.org/
    - Documentation for Random Forest Regressor, model evaluation metrics, and preprocessing tools like MinMaxScaler.
2. XGBoost Documentation
    - https://xgboost.readthedocs.io/
    - Reference for XGBoost's regularization parameters (alpha, lambda) and boosting techniques used in price modelling.
3. Streamlit Documentation
    - https://docs.streamlit.io/
    - For building, customizing, and deploying interactive ML apps in Python.
4. Pandas & NumPy Documentation
    - https://pandas.pydata.org/docs/
    - https://numpy.org/doc/
    - Data manipulation, feature engineering, and time-series handling utilities.
5. Matplotlib Documentation
    - https://matplotlib.org/stable/contents.html
    - For plotting feature importance and visualizing prediction outputs.

9.3 Industry Reports & Articles

1. McKinsey & Company Reports
    - https://www.mckinsey.com/
    - Articles and whitepapers on AI/ML in financial services, risk analysis, and future of fintech.
2. World Economic Forum & Deloitte Insights
    - Topics: AI in Crypto & Blockchain Markets, financial automation trends.
    - https://www.weforum.org/
    - https://www2.deloitte.com/
3. CoinDesk & Coin telegraph
    - Real-time industry analysis on crypto trends, market behaviour, and predictive technologies used by trading platforms.
    - https://www.coindesk.com/
    - https://cointelegraph.com/
4. MIT Technology Review & Harvard Business Review (HBR)
    - Insights into AI in financial tech, automation, and decision-making systems in volatile markets.