**Time Series Analysis For Bitcoin Price Prediction Using Prophet**

|  |
| --- |
| CONTENT |
| Abstract |
| 1. INTRODUCTION |
| 1.1 Motivation |
| 1.2 Problem Statement |
| 1.3 Objective of the Project |
| 1.4 Scope |
| 1.5 Project Introduction |
| 2. LITERATURE REVIEW |
| 2.1 Previous Studies |
| 3. SYSTEM ANALYSIS |
| 3.1 Existing system |
| 3.2 Limitations |
| 3.3 Proposed System Overview |
| 3.4 Benefits |
| 3.5 Proposed System Workflow |
| 4. REQUIREMENT ANALYSIS |
| 4.1 Functional and Non-functional Requirements |
| 4.2 Hardware Specifications |
| 4.3 Software Specifications |
| 4.4 System Architecture |
| 4.5. Methodology |
| 4.5.1. XGBoost |
| 4.5.2. Random Forest |
| 4.5.3. Gradient boost |
| 4.5.4. Stacking Classifier |
| 4.5.5. KNN |
| 4.5.6. CNN |
| 5. SYSTEM DESIGN |
| 5.1 Input Design Overview |
| 5.2 UML Diagrams (Class, Use Case, Sequence, Collaboration, Deployment, Activity, ER, and Component) |
| 5.3 Data Flow Diagram |
| 6. IMPLEMENTATION AND RESULTS |
| 6.1 Module Breakdown |
| 6.2 Output Screens Overview |
| 7. Testing Methods and Test Cases |
| 8. CONCLUSION |
| 9.Future enhancements |
| 10. REFERENCES |

**Abstract:**

This study focuses on time series analysis for predicting Bitcoin prices using various methodologies, including Recurrent Neural Networks (RNN), Long Short-Term Memory networks (LSTM), Auto Regressive Integrated Moving Average (ARIMA), and Facebook's Prophet. We utilize a dataset consisting of timestamps and closing prices to train and evaluate the performance of these models. The objective is to identify the most effective forecasting technique for Bitcoin price movements, addressing the inherent volatility of cryptocurrency markets. By leveraging historical price data, we aim to enhance prediction accuracy, contributing to more informed trading decisions. Our findings will provide valuable insights into the applicability of different predictive models in the context of cryptocurrency, ultimately aiming to assist investors in navigating the complexities of Bitcoin trading. The results underscore the strengths and weaknesses of each method, paving the way for future research in financial time series analysis.

Keywords: RNN, LSTM, ARIMA and Prophet, Kaggle dataset.

**INTRODUCTION**

The rapid rise of cryptocurrencies, particularly Bitcoin, has transformed financial markets and sparked immense interest among investors, analysts, and researchers. Bitcoin, as the first decentralized digital currency, has garnered significant attention due to its price volatility and potential for high returns. This volatility presents both opportunities and challenges, necessitating robust predictive models to assist stakeholders in making informed decisions. Time series analysis, a statistical technique focused on analyzing time-ordered data points, offers valuable insights into price trends and movements, making it an essential tool in the realm of cryptocurrency forecasting. Various methodologies exist for time series prediction, each with unique strengths and weaknesses. Traditional approaches, such as the Auto Regressive Integrated Moving Average (ARIMA), have been widely employed due to their simplicity and interpretability. However, the increasing complexity of financial data and non-linear relationships often necessitate the use of more sophisticated models. Recurrent Neural Networks (RNN) and Long Short-Term Memory networks (LSTM) represent advanced techniques capable of capturing complex temporal patterns, making them suitable for financial time series analysis.

Additionally, Facebook's Prophet has emerged as a powerful tool for forecasting time series data, particularly in scenarios involving seasonality and trends. Its user-friendly interface and ability to handle missing data make it an attractive option for practitioners in various fields, including finance. This study aims to evaluate the effectiveness of these methodologies in predicting Bitcoin prices by leveraging a comprehensive dataset of historical price movements. By comparing the predictive capabilities of RNN, LSTM, ARIMA, and Prophet, we seek to identify the most effective forecasting technique, ultimately aiding investors in navigating the volatile cryptocurrency landscape. Through rigorous analysis and evaluation, this research will contribute to the broader discourse on financial time series analysis, highlighting the applicability of different models in the cryptocurrency market.

1. **INTRODUCTION**

**1.1 Motivation:**

The motivation for this study arises from the increasing significance of Bitcoin as a leading cryptocurrency and the challenges posed by its price volatility. Accurate price prediction is crucial for investors seeking to make informed decisions in this dynamic market. By exploring various forecasting methodologies, including RNN, LSTM, ARIMA, and Facebook's Prophet, this research aims to identify the most effective techniques for predicting Bitcoin price movements. Enhancing prediction accuracy can significantly improve trading strategies, ultimately contributing to more stable investment outcomes. This study not only addresses the practical need for reliable forecasts but also enriches the understanding of financial time series analysis in cryptocurrency.

**1.2 Problem Statement:**

The cryptocurrency market, particularly Bitcoin, is characterized by high volatility, making accurate price prediction a significant challenge for investors. Traditional forecasting methods often struggle to adapt to the dynamic nature of these markets. This study aims to explore the effectiveness of various time series analysis techniques, including RNN, LSTM, ARIMA, and Facebook's Prophet, to predict Bitcoin prices. By employing a dataset of historical timestamps and closing prices, we seek to determine which methodology yields the most accurate forecasts, ultimately providing insights that can aid investors in making informed trading decisions amidst market fluctuations.

**1.3 Objective of the Project**:

The primary objective of this project is to conduct a comprehensive time series analysis for predicting Bitcoin prices by comparing various forecasting methodologies, including RNN, LSTM, ARIMA, and Facebook's Prophet. We aim to utilize historical closing price data to develop and evaluate these models, with a focus on enhancing prediction accuracy amidst the inherent volatility of the cryptocurrency market. By identifying the most effective predictive technique, this study seeks to provide insights that will assist investors in making informed trading decisions, thereby contributing to a deeper understanding of financial time series analysis in the context of cryptocurrency.

**1.4 Scope:**

This study explores the application of time series analysis techniques for predicting Bitcoin prices, focusing on RNN, LSTM, ARIMA, and Prophet models. By utilizing a comprehensive dataset of timestamps and closing prices, we aim to evaluate each model's performance in forecasting Bitcoin price movements amidst market volatility. The research will examine the strengths and weaknesses of these methodologies, offering insights into their effectiveness in cryptocurrency trading. Additionally, the findings will serve as a foundation for future studies in financial time series analysis, enabling investors to make data-driven decisions in an increasingly complex trading environment.

**2. LITERATURE SURVEY**

**1. L. Serena, S. Ferretti, and G. D’Angelo, “Cryptocurrencies activity as a complex network: Analysis of transactions graphs,” Peer-Peer Netw. Appl., vol. 15, no. 2, pp. 839–853, Mar. 2022.**

The paper by L. Serena, S. Ferretti, and G. D’Angelo explores cryptocurrency transaction networks using complex network theory. Analyzing Bitcoin, DogeCoin, Ethereum, and Ripple, the study focuses on transaction patterns over time to uncover insights into user behavior on these Distributed Ledger Technologies (DLTs). The authors introduce the Distributed Ledger Network Analyzer (DiLeNA), a tool designed to investigate transaction networks. Their findings reveal that transaction graphs across all studied DLTs exhibit small-world properties, highlighting the importance of network analysis for understanding user interactions and the dynamics of cryptocurrency ecosystems.

**2. B. Tao, I. W. Ho, and H.-N. Dai, “Complex network analysis of the Bitcoin blockchain network,” in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), May 2021, pp. 1–5.**

The paper by Tao, Ho, and Dai presents a comprehensive analysis of the Bitcoin blockchain network through a complex network approach. They introduce the BABD-13 dataset, which includes detailed Bitcoin transaction data from July 2019 to May 2021, featuring 13 types of addresses and 148 attributes. Using machine learning models like k-nearest neighbors, decision tree, and XGBoost, they achieve classification accuracies between 93.24% and 97.13%. Additionally, the study proposes a k-hop subgraph generation algorithm for in-depth network analysis and examines Bitcoin address behavior patterns, contributing to better understanding and tracking of blockchain transactions.

**3. B. Tao, H.-N. Dai, J. Wu, I. W. Ho, Z. Zheng, and C. F. Cheang, “Complex network analysis of the Bitcoin transaction network,” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 69, no. 3, pp. 1009–1013, Mar. 2022.**

This paper presents a comprehensive framework for analyzing Bitcoin transactions to identify illicit activities within the cryptocurrency ecosystem. The study introduces the BABD-13 dataset, the largest publicly available labeled dataset of Bitcoin addresses, covering transactions from July 2019 to May 2021. It features 13 address types, 5 indicator categories, and 148 features, with 544,462 labeled entries. A novel subgraph generation algorithm, BTC-SubGen, extracts k-hop subgraphs from the Bitcoin transaction network. Classification using various machine learning models achieved accuracy rates between 93.24% and 97.13%. The study also explores feature importance and behavior patterns of Bitcoin addresses.

**4. N. Tovanich, N. Soulié, N. Heulot, and P. Isenberg, “An empirical analysis of pool hopping behavior in the Bitcoin blockchain,” in Proc. IEEE Int. Conf. Blockchain Cryptocurrency, May 2021, pp. 1–9.**

In their 2021 paper, N. Tovanich et al. investigate pool-hopping behavior in Bitcoin mining, where miners switch pools to maximize rewards. They propose a new detection methodology using time window analysis of mining rewards. Their approach includes algorithms for miner identification and revenue tracking, tested on the top five mining pools over two periods in 2020 and 2021. The study finds that while pool-hopping remains beneficial, the newer reward systems have improved fairness, reducing the disparity between pool-hoppers and static miners. Despite this, pool-hoppers still achieve a 33% higher median cumulative gain compared to static miners.

**5. I. Alqassem, I. Rahwan, and D. Svetinovic, “The anti-social system properties: Bitcoin network data analysis,” IEEE Trans. Syst., Man, Cybern., Syst., vol. 50, no. 1, pp. 21–31, Jan. 2020.**

The paper by Alqassem, Rahwan, and Svetinovic, titled “The Anti-Social System Properties: Bitcoin Network Data Analysis,” published in IEEE Transactions on Systems, Man, and Cybernetics: Systems (2020), examines the Bitcoin network through a lens of anti-social system properties. This study is part of a broader index covering technical items published in 2020, including papers, correspondence, and reviews. The Author Index details the primary entries for each item, organized by the first author’s name, and includes coauthors, paper titles, and publication specifics. The Subject Index categorizes entries by relevant subject headings, providing comprehensive bibliographic information. The focus of the paper is on analyzing Bitcoin’s network dynamics and its implications within an anti-social system framework.

**6. P. Nerurkar, D. Patel, Y. Busnel, R. Ludinard, S. Kumari, and M. K. Khan, “Dissecting Bitcoin blockchain: Empirical analysis of Bit- coin network (2009–2020),” J. Netw. Comput. Appl., vol. 177, Mar. 2021, Art. no. 102940.**

The paper by P. Nerurkar et al. explores the Bitcoin network's evolution from 2009 to 2020, examining its dual nature as both a social and antisocial entity. Bitcoin, a decentralized payment system using cryptocurrency, facilitates value exchange without intermediaries, fostering community trust. However, its anonymity also impedes law enforcement efforts to track illicit transactions. The study delves into the network’s local topology and geometry to understand how these social and antisocial characteristics influence Bitcoin's development. By analyzing transaction data from Bitcoin’s initial decade, the authors provide insights into the network's structural dynamics and how user behaviors shape its growth and challenges.

**3. SYSTEM ANALYSIS & FEASIBILITY STUDY**

**Existing System:**

The existing system for Bitcoin forecasting primarily relies on traditional time series analysis models such as ARIMA and PROPHET, implemented using the R analytics platform. ARIMA focuses on capturing linear trends and seasonality in historical Bitcoin price data, while PROPHET is adept at handling non-linear trends and holidays or special events. Both methods are evaluated on the same dataset from May 2016 to March 2018, with pre-processing techniques like timestamp conversion and feature selection applied. The system is designed to improve accuracy by incorporating additional variables derived from correlations between cryptocurrencies and real currencies.

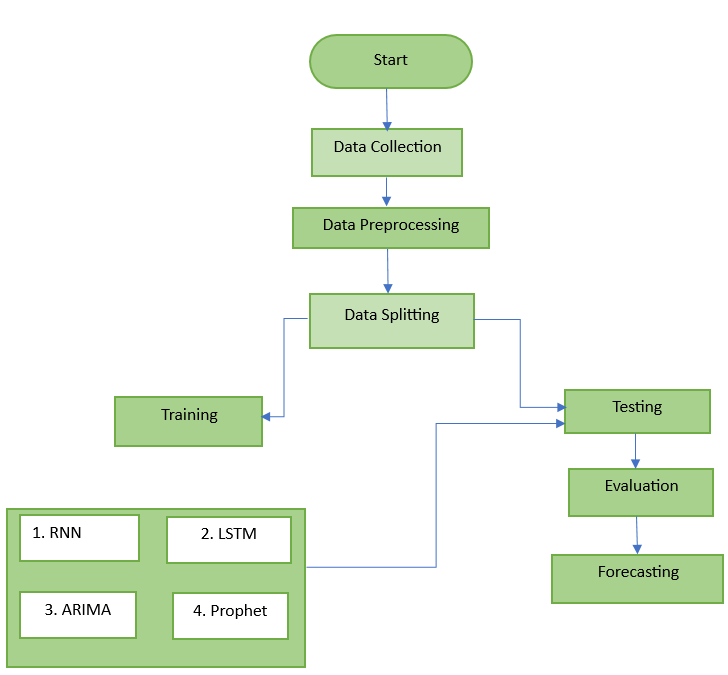
**Disadvantages of Existing System:**

1. **ARIMA:** Struggles to model non-linear patterns, limiting its forecasting accuracy for volatile Bitcoin data.
2. **PROPHET:** Handling of special events may not capture unforeseen market shocks in cryptocurrency prices.
3. **Data Sensitivity:** Both ARIMA and PROPHET models are sensitive to noisy and incomplete data, impacting forecast reliability.
4. **R Analytics Platform:** Has a steep learning curve, which can hinder accessibility for users unfamiliar with it.
5. **Lack of Machine Learning:** Limited incorporation of advanced machine learning techniques may restrict improvements in forecast precision.

**Proposed System:**

This study proposes a comprehensive system for predicting Bitcoin prices through time series analysis. By implementing various forecasting models—RNN, LSTM, ARIMA, and Facebook's Prophet—we will analyse historical price data to assess each model's effectiveness. The system will utilize a Kaggle dataset containing timestamps and closing prices, allowing us to train and evaluate the performance of the different methodologies. Our approach aims to enhance prediction accuracy amidst Bitcoin's volatility, offering insights into the strengths and limitations of each technique. Ultimately, the system seeks to aid investors in making informed trading decisions within the cryptocurrency market.

**Block Diagram**



**Advantages of Proposed System:**

**Comprehensive model comparison:** The system evaluates the strengths of RNN, LSTM, ARIMA, and Prophet for Bitcoin forecasting.

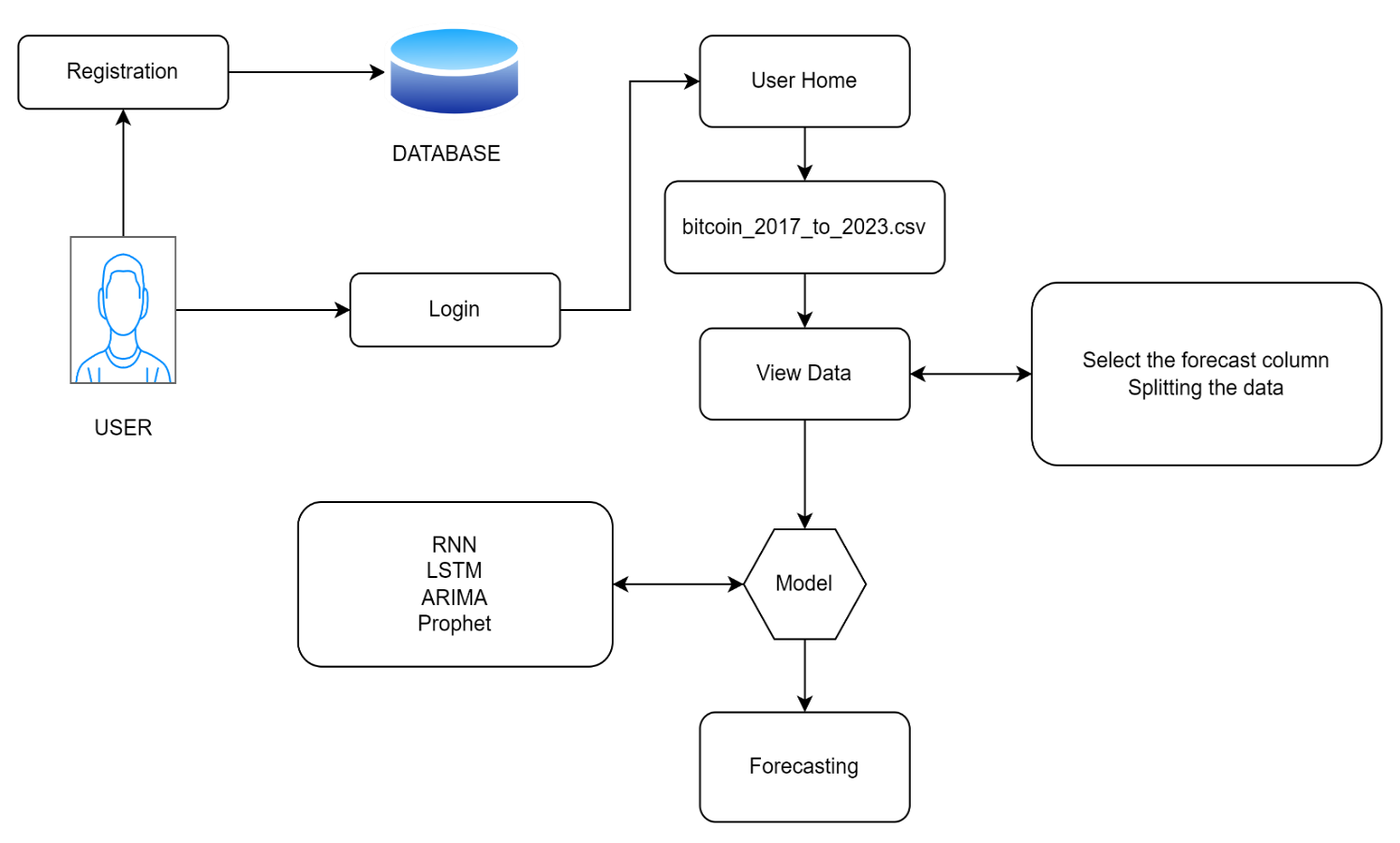
**Enhanced prediction accuracy:** By leveraging multiple models, it improves the precision of Bitcoin price predictions.

**Data-driven insights:** Historical price data analysis provides valuable insights for understanding Bitcoin's volatility.

**Investor decision support:** The system educates investors on model-driven forecasts to improve trading strategies.

**Educational focus:** The project offers students and researchers a deeper understanding of time series forecasting techniques.

**Architecture:**

****

**4. SOFTWARE DEVELOPMENT LIFE CYCLE – SDLC:**

In our project we use waterfall model as our software development cycle because of its step-by-step procedure while implementing.



**Fig1**: Waterfall Model

* **Requirement Gathering and analysis** − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
* **System Design** − the requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
* **Implementation** − with inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
* **Integration and Testing** − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
* **Deployment of system** − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
* **Maintenance** − There are some issues which come up in the client environment. To fix those issues, patches are released. Also, to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

**FEASIBILITY STUDY**

The feasibility of the project is analysed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

* ECONOMICAL FEASIBILITY
* TECHNICAL FEASIBILITY
* SOCIAL FEASIBILITY

**Economic feasibility:**

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus, the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

### Technical feasibility:

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

**Social feasibility:**

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

**SYSTEM REQUIREMENTS SPECIFICATION**

**Functional and non-functional requirements:**

Requirement’s analysis is very critical process that enables the success of a system or software project to be assessed. Requirements are generally split into two types: Functional and non-functional requirements.

**Functional Requirements**: These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

Examples of functional requirements:

1. Authentication of user whenever he/she logs into the system
2. System shutdown in case of a cyber-attack
3. A verification email is sent to user whenever he/she register for the first time on some software system.

**Non-functional requirements**: These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements.  
They basically deal with issues like:

* Portability
* Security
* Maintainability
* Reliability
* Scalability
* Performance
* Reusability
* Flexibility

Examples of non-functional requirements:

1. Emails should be sent with a latency of no greater than 12 hours from such an activity.
2. The processing of each request should be done within 10 seconds
3. The site should load in 3 seconds whenever of simultaneous users are > 10000

**SOFTWARE AND HARDWARE REQUIREMENTS:**

**Hardware:**

Operating system : Windows 7 or 7+

RAM : 8 GB

Hard disc or SSD : More than 500 GB

Processor : Intel 3rd generation or high or Ryzen with 8 GB Ram

**Software:**

Software’s : Python 3.6 or high version

IDE : VSCode.

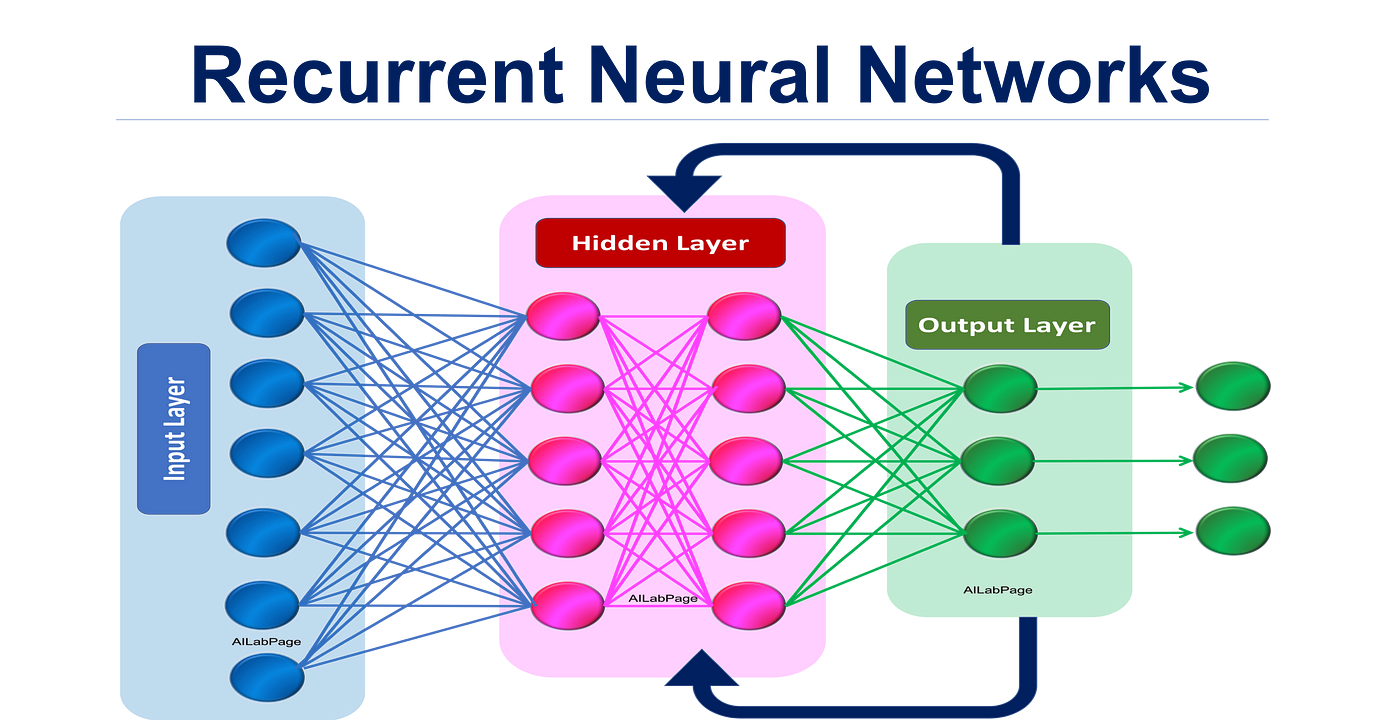
Framework : Flask

**METHODOLOGY AND ALGORITHMS:**

**Recurrent Neural Networks (RNNs):**

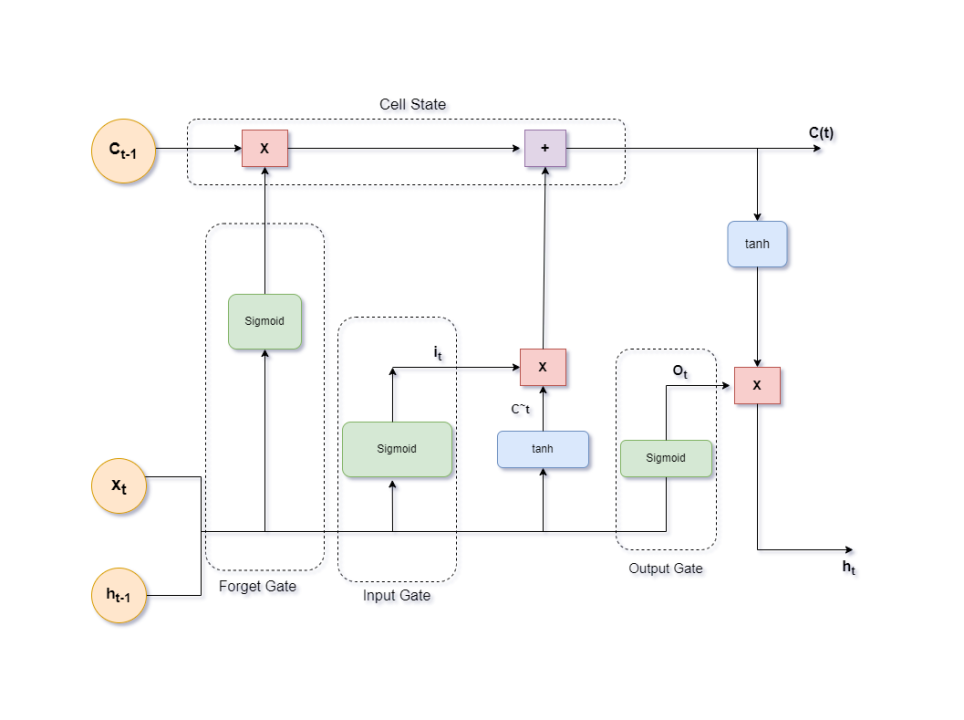
Recurrent Neural Networks (RNNs) are a powerful class of neural networks designed for sequential data, making them suitable for time series analysis, including Bitcoin price prediction. Unlike traditional feedforward networks, RNNs have connections that loop back on themselves, allowing them to maintain a memory of previous inputs. This feature enables RNNs to capture temporal dependencies in the data, which is crucial for predicting future price movements based on historical trends.

In the context of Bitcoin price prediction, RNNs can learn complex patterns from historical price data, trading volumes, and other relevant features. By processing the data sequentially, RNNs can adjust their predictions based on the sequence of past values, providing more accurate forecasts. Variants of RNNs, such as Long Short-Term Memory (LSTM) networks, enhance this capability by mitigating issues like vanishing gradients, enabling the model to retain information over longer periods. Overall, RNNs offer a promising approach for analysing and predicting Bitcoin prices effectively.



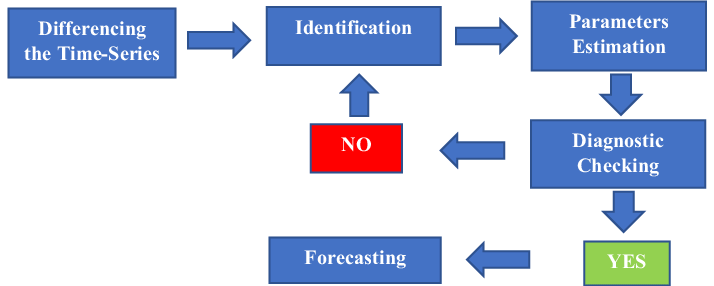
**LSTM:**

Long Short-Term Memory (LSTM) networks are a specialized type of recurrent neural network (RNN) designed to capture temporal dependencies in sequential data. In the context of Bitcoin price prediction, LSTM models excel due to their ability to retain information over extended periods, effectively handling the intricacies of price movements influenced by various factors. The architecture includes memory cells that allow the network to learn from historical data while mitigating the vanishing gradient problem commonly encountered in traditional RNNs. By training on past Bitcoin prices and relevant market indicators, LSTM can discern patterns and trends, enabling more accurate forecasts. The model's capacity to adapt to non-linear relationships makes it particularly suitable for financial time series, where price dynamics can be erratic. Evaluating the performance of LSTM against other forecasting models can provide insights into its effectiveness in predicting Bitcoin prices, contributing to enhanced investment strategies and decision-making in the cryptocurrency market.



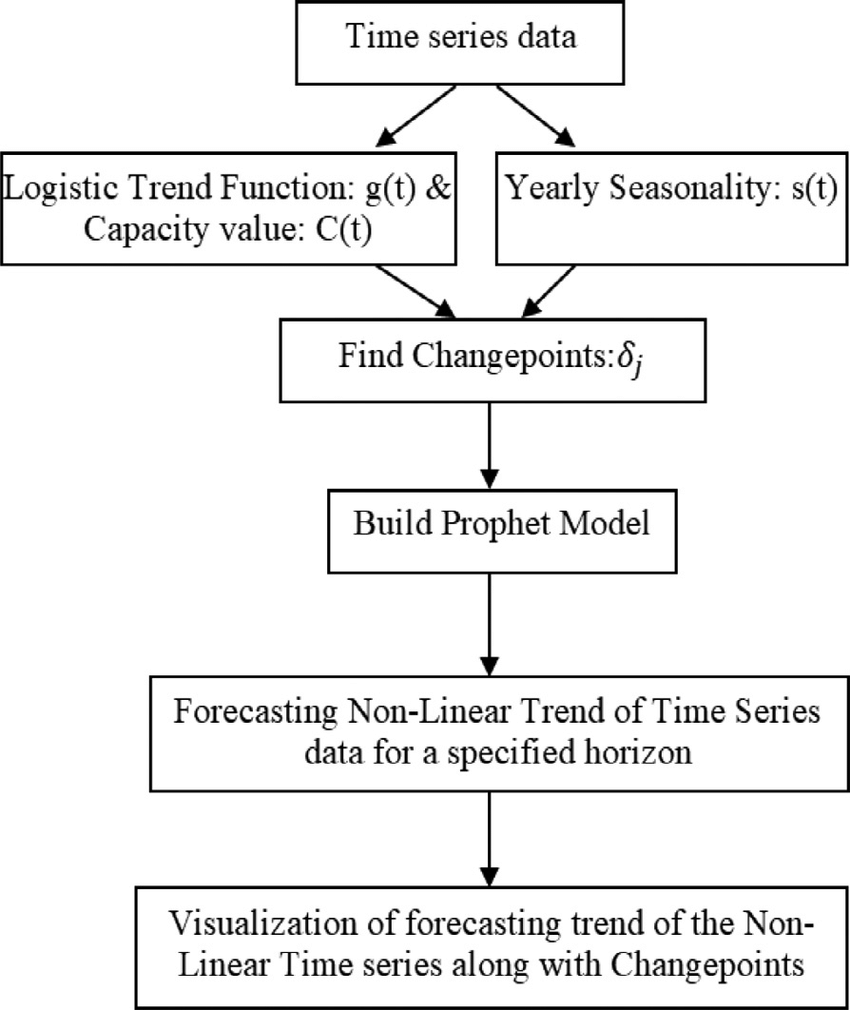
**ARIMA:**

ARIMA (AutoRegressive Integrated Moving Average) is a widely used statistical method for time series forecasting, particularly effective for analyzing and predicting Bitcoin prices. The ARIMA model combines three components: auto regression (AR), differencing (I), and moving average (MA). The autoregressive part captures the relationship between an observation and a number of lagged observations, while the moving average component accounts for the dependency between an observation and a residual error from a moving average model applied to lagged observations. Differencing is used to make the time series stationary, ensuring constant mean and variance over time. This method is advantageous for Bitcoin price prediction due to its ability to model trends and seasonality in volatile financial markets. By optimizing the model parameters (p, d, q), analysts can create robust forecasts, making ARIMA a valuable tool for investors and researchers aiming to understand Bitcoin price dynamics and trends.



**Prophet:**

Prophet is an open-source forecasting tool developed by Facebook, designed to handle time series data that exhibits trends and seasonality. It is particularly effective for data with strong seasonal effects and several seasons of historical data, making it a suitable choice for predicting Bitcoin prices. Prophet utilizes an additive model, which means it decomposes the time series into three main components: trend, seasonality, and holidays. The trend component captures the overall direction of the data, while the seasonal component accounts for periodic fluctuations. One of the standout features of Prophet is its ability to accommodate missing data and outliers, which are common in financial time series. With a user-friendly interface and flexibility in handling various forecasting scenarios, Prophet allows analysts and researchers to generate reliable predictions for Bitcoin price movements, helping investors make informed decisions based on historical trends and patterns.



**5. SYSTEM DESIGN:**

## **Input Design:**

In an information system, input is the raw data that is processed to produce output. During the input design, the developers must consider the input devices such as PC, MICR, OMR, etc.

Therefore, the quality of system input determines the quality of system output. Well-designed input forms and screens have following properties −

* It should serve specific purpose effectively such as storing, recording, and retrieving the information.
* It ensures proper completion with accuracy.
* It should be easy to fill and straightforward.
* It should focus on user’s attention, consistency, and simplicity.
* All these objectives are obtained using the knowledge of basic design principles regarding −
  + What are the inputs needed for the system?
  + How end users respond to different elements of forms and screens.

### Objectives for Input Design:

The objectives of input design are −

* To design data entry and input procedures
* To reduce input volume
* To design source documents for data capture or devise other data capture methods
* To design input data records, data entry screens, user interface screens, etc.
* To use validation checks and develop effective input controls.

**Output Design:**

The design of output is the most important task of any system. During output design, developers identify the type of outputs needed, and consider the necessary output controls and prototype report layouts.

### Objectives of Output Design:

The objectives of input design are:

* To develop output design that serves the intended purpose and eliminates the production of unwanted output.
* To develop the output design that meets the end user’s requirements.
* To deliver the appropriate quantity of output.
* To form the output in appropriate format and direct it to the right person.
* To make the output available on time for making good decisions.

**UML DIAGRAMS**

UML stands for Unified Modelling Language. UML is a standardized generalpurpose modelling language in the field of objectoriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of objectoriented computer software. In its current form UML is comprised of two major components: a Metamodel and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modelling Language is a standard language for specifying, Visualization, Constructing and documenting the artefacts of software system, as well as for business modelling and other nonsoftware systems.

The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems.

The UML is a very important part of developing objectsoriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

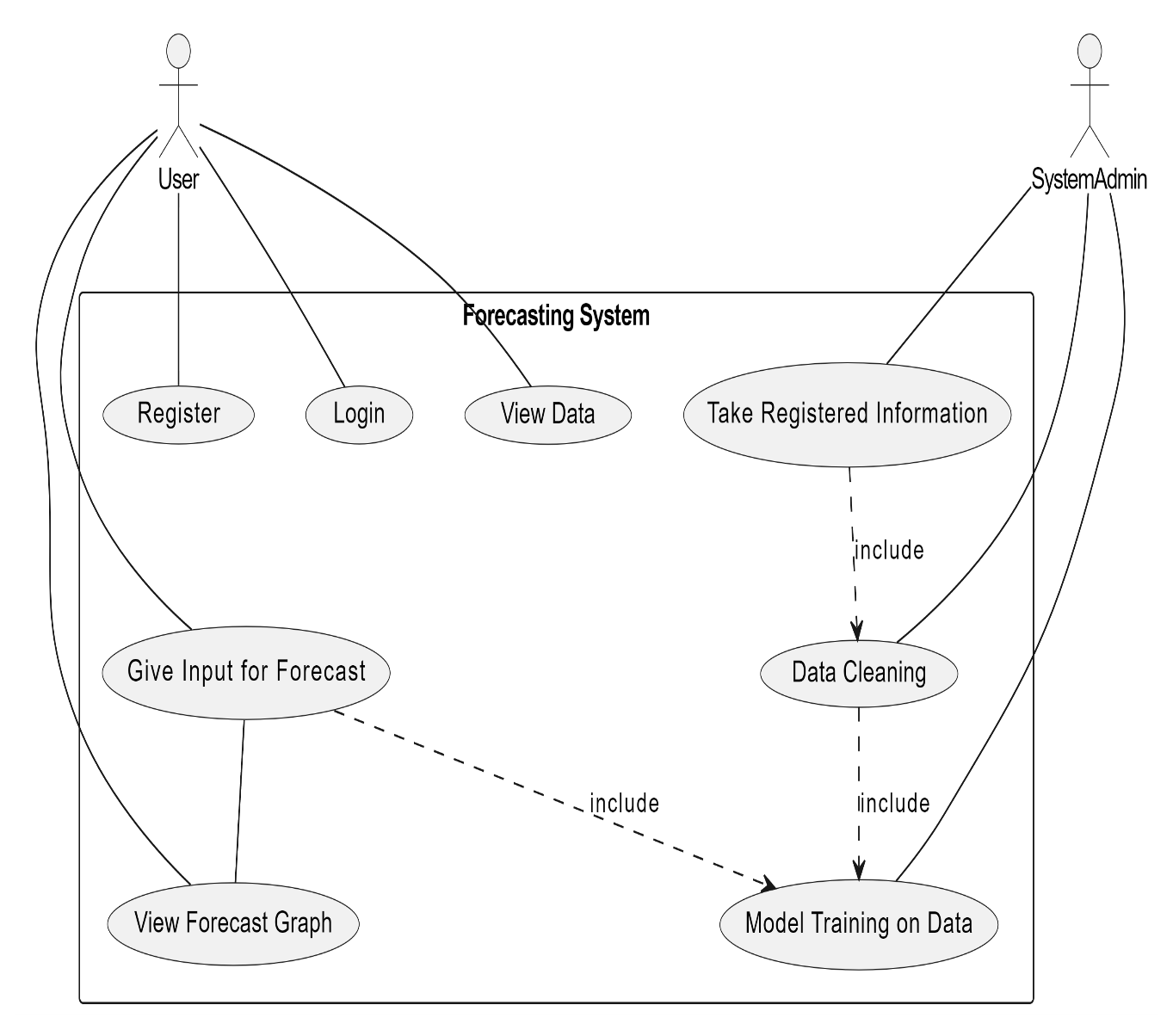
**GOALS:**

The Primary goals in the design of the UML are as follows:

1. Provide users a readytouse, expressive visual modelling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modelling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

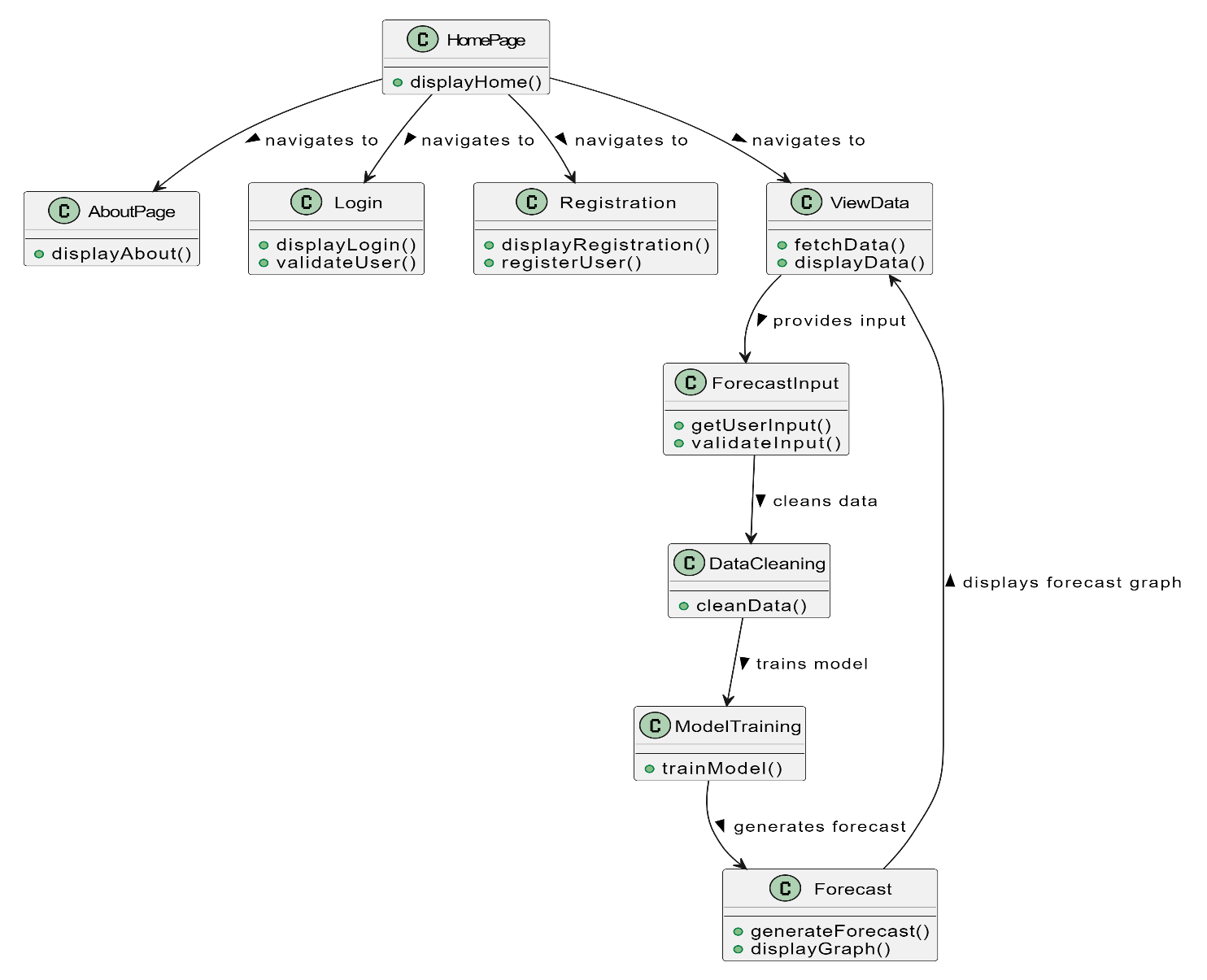
**USE CASE DIAGRAM**

* A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Usecase analysis.
* Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases.
* The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.



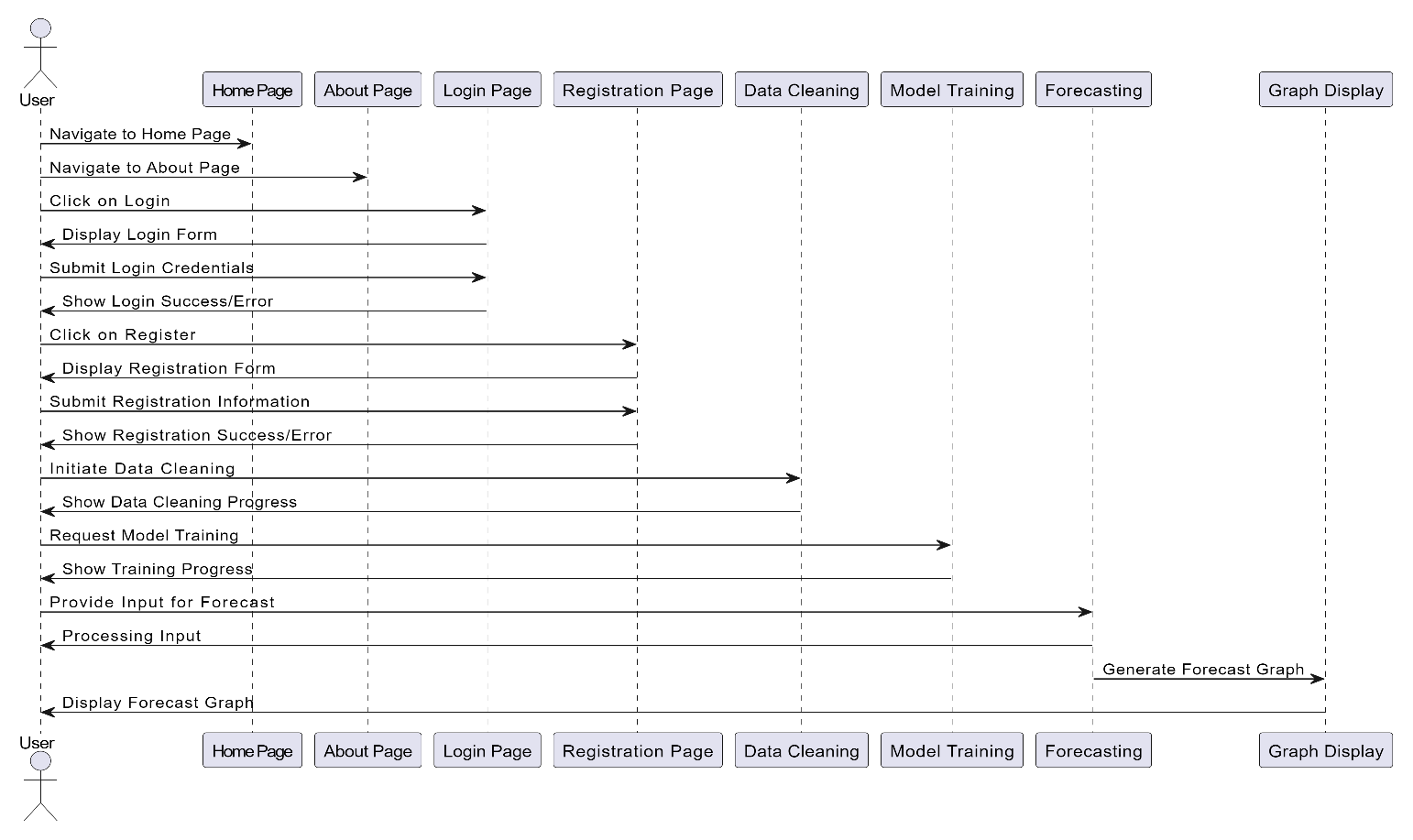
**CLASS DIAGRAM**

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information



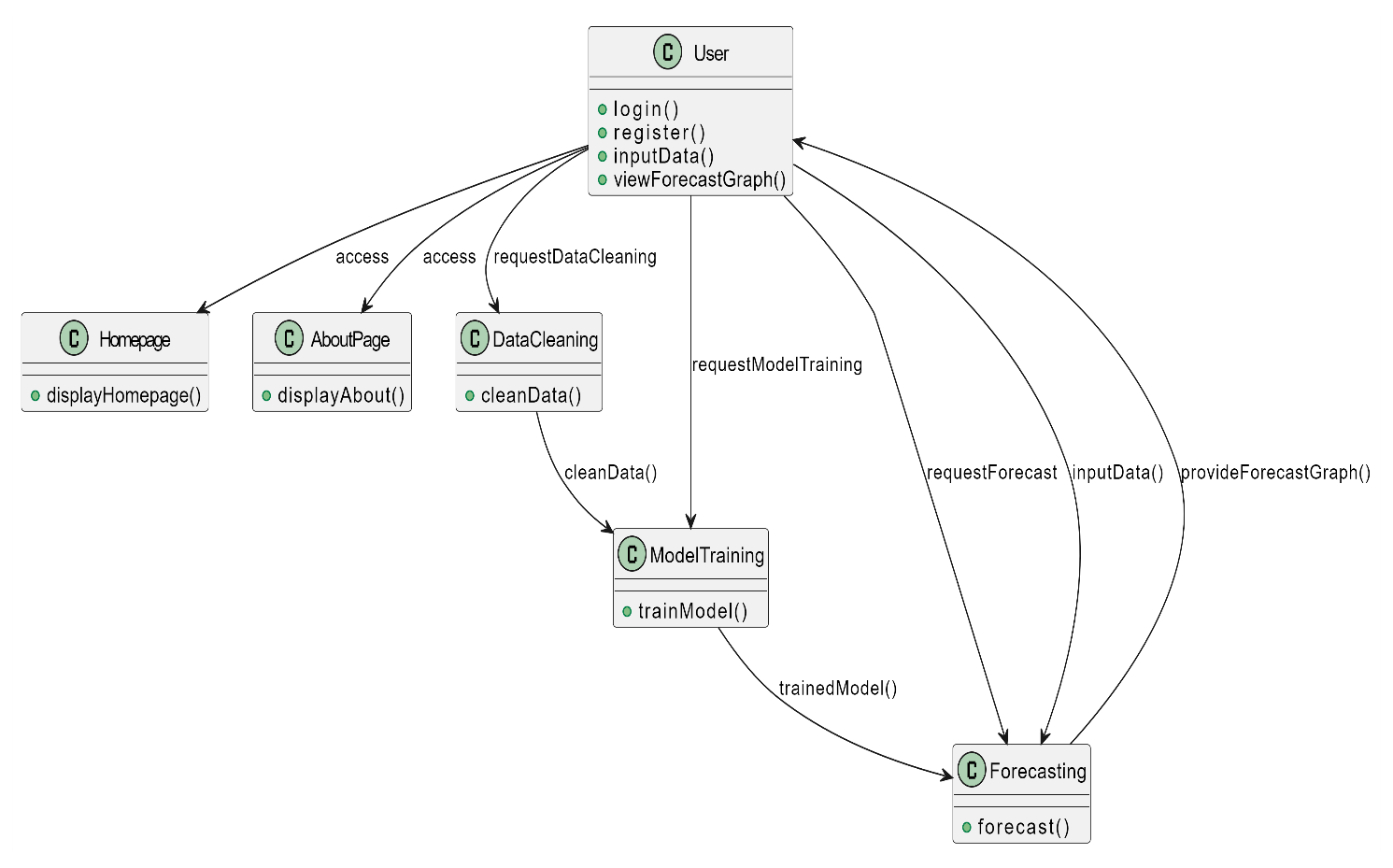
**SEQUENCE DIAGRAM**

* A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order.
* It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams



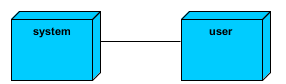
**COLLABORATION DIAGRAM:**

In collaboration diagram the method call sequence is indicated by some numbering technique as shown below. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram. The method calls are similar to that of a sequence diagram. But the difference is that the sequence diagram does not describe the object organization whereas the collaboration diagram shows the object organization.



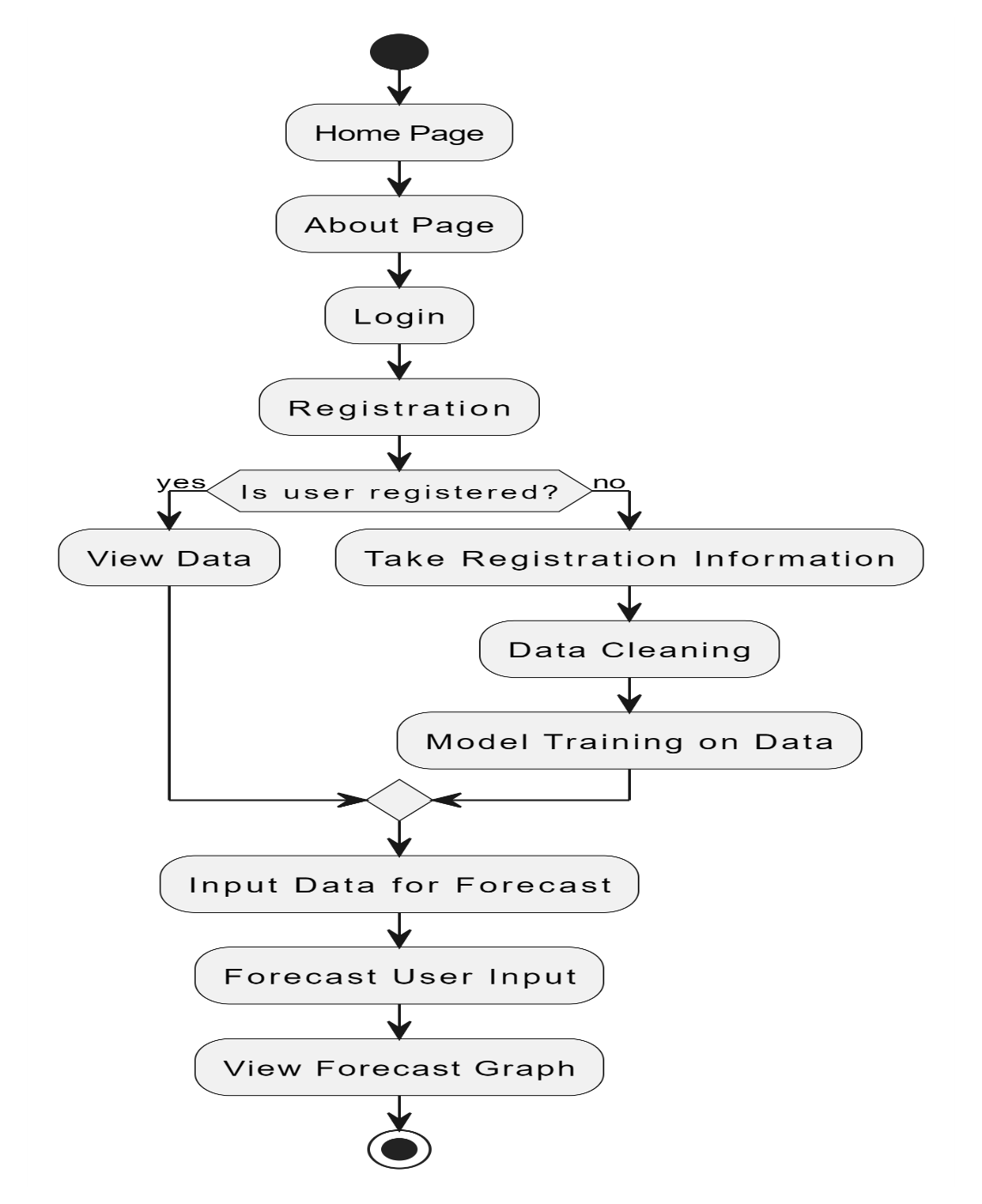
**DEPLOYMENT DIAGRAM**

Deployment diagram represents the deployment view of a system. It is related to the component diagram. Because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical hardware’s used to deploy the application.



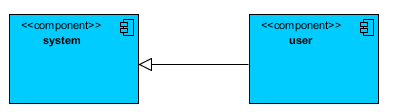
**ACTIVITY DIAGRAM:**

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling Language, activity diagrams can be used to describe the business and operational stepbystep workflows of components in a system. An activity diagram shows the overall flow of control.



**COMPONENT DIAGRAM**:

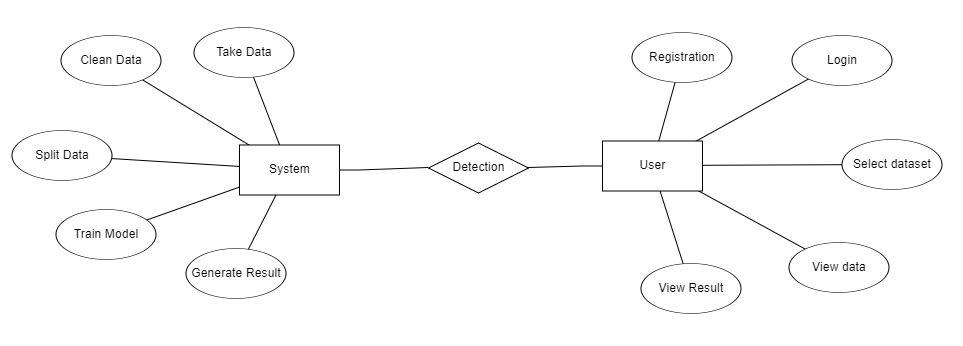
A component diagram, also known as a UML component diagram, describes the organization and wiring of the physical **c**omponents in a system. Component diagrams are often drawn to help model implementation details and doublecheck that every aspect of the system's required function is covered by planned development.



**ER DIAGRAM:**

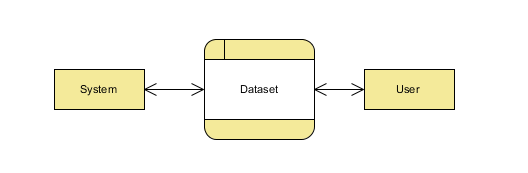
An Entity–relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of ER model are: entity set and relationship set.

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database. Let’s have a look at a simple ER diagram to understand this concept.

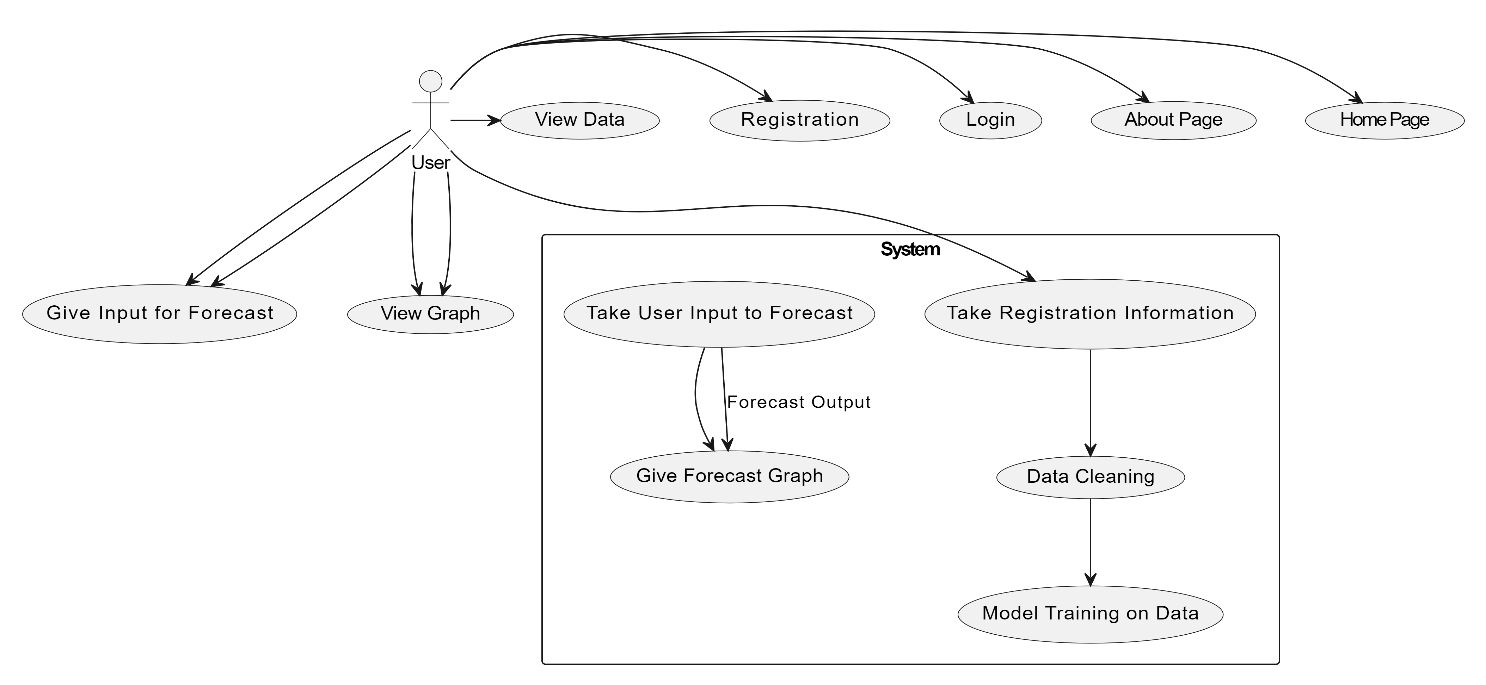


**DFD DIAGRAM:**

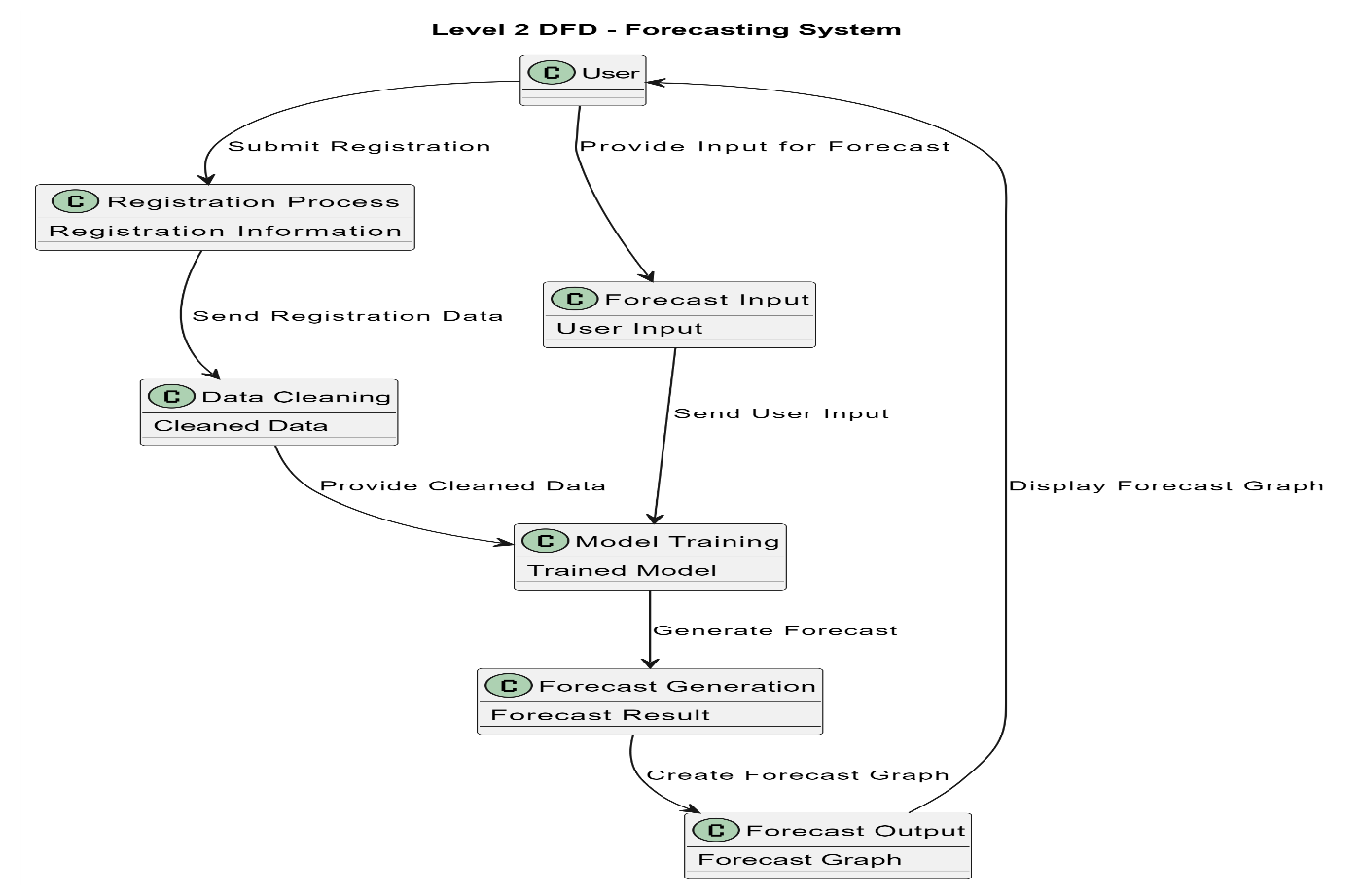
A Data Flow Diagram (DFD) is a traditional way to visualize the information flows within a system. A neat and clear DFD can depict a good amount of the system requirements graphically. It can be manual, automated, or a combination of both. It shows how information enters and leaves the system, what changes the information and where information is stored. The purpose of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communications tool between a systems analyst and any person who plays a part in the system that acts as the starting point for redesigning a system.



**Level 1:**



**Level 2:**



**6. IMPLEMENTATION AND RESULTS**

**MODULES:**

1. **User:**
   1. **Registration:**

User have to register by provide the credential.

* 1. **Login:**

Here user have Login using their register credential

* 1. **Input Model:**

The user must provide input values as a Weeks in order to get results as Graph.

* 1. **View Results:**

User view’s the generated Forecast graph.

1. **System**
   1. **Working on dataset:**

System checks for data whether it is available or not and load the data in csv files.

* 1. **Pre-processing:**

Data need to be pre-processed according the models it helps to increase the accuracy of the model and better information about the data.

* 1. **Splitting the data:**

After pre-processing the data will split into two parts as train and test data before training with the given algorithms.

* 1. **Model Training**

Train the model on Dataset to forecasting the future result.

* 1. **Generate the Result:**

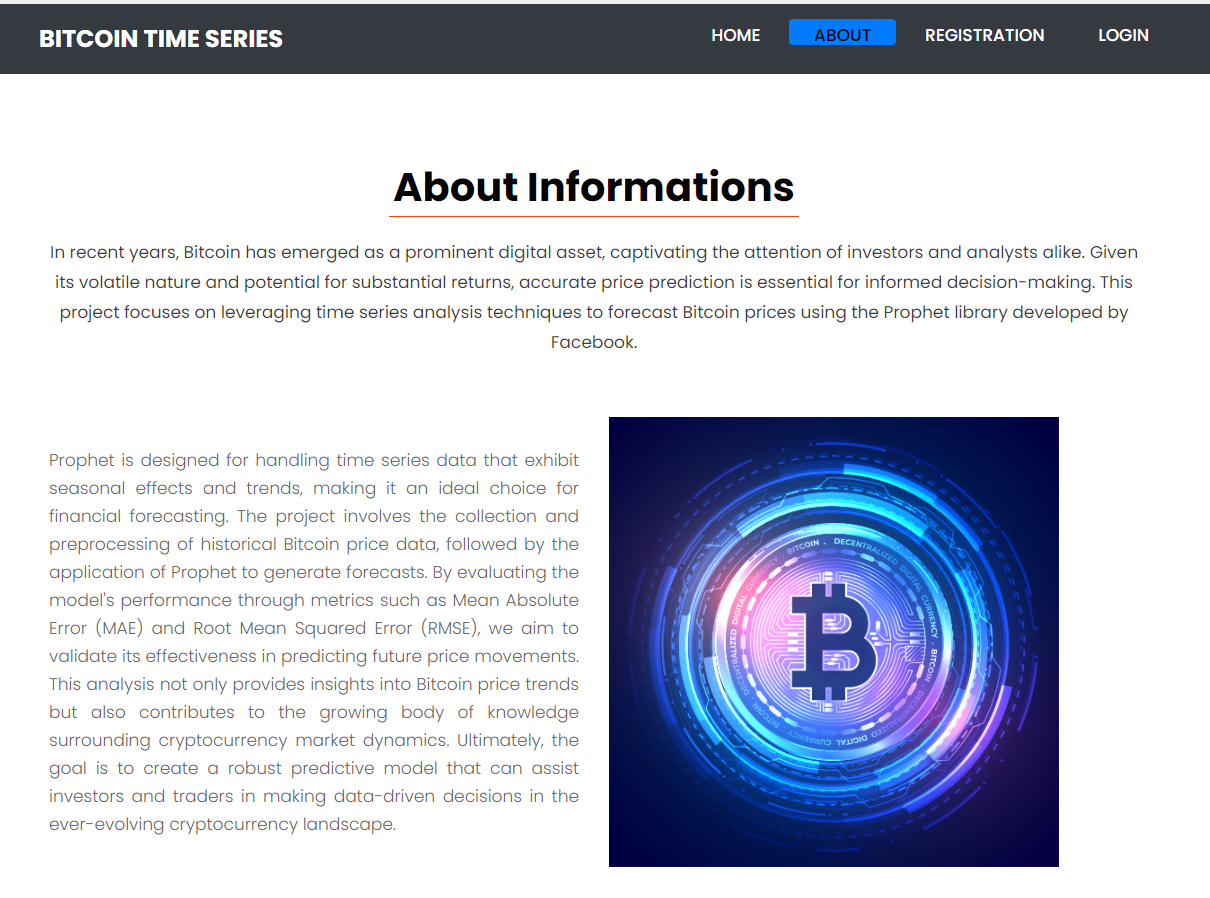
It will give the Forecast graph for the future time Bitcoin price is increases or decreases.

**OUTPUT SCREENS OVERVIEW**

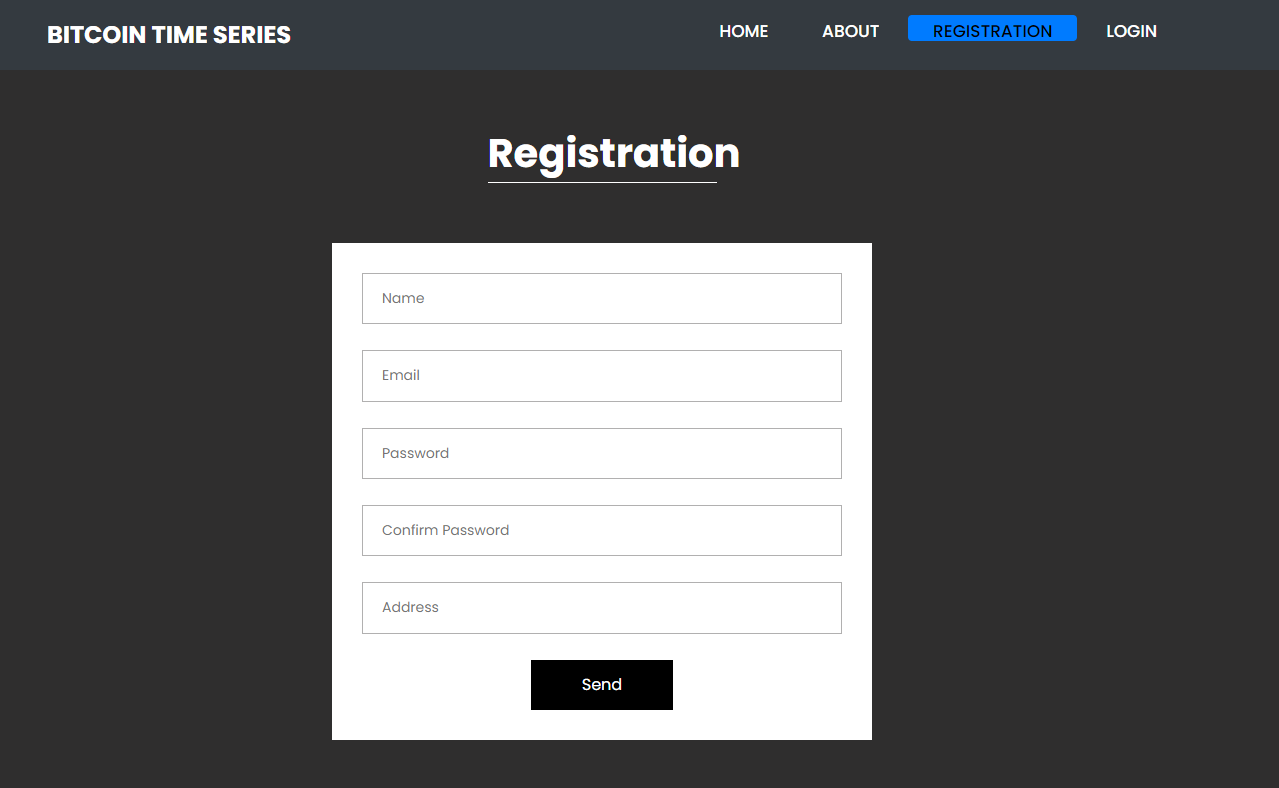
**Home:** Welcome to Time Series Analysis For Bitcoin Price Prediction Using Prophet.

****

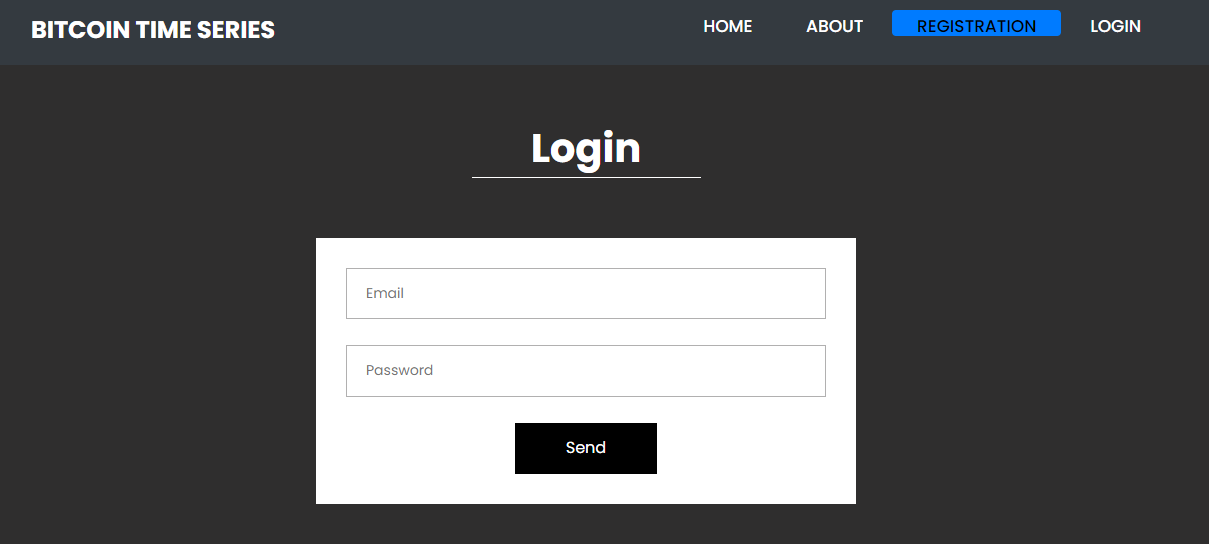
**About:** Here is the about information regarding to your project.

****

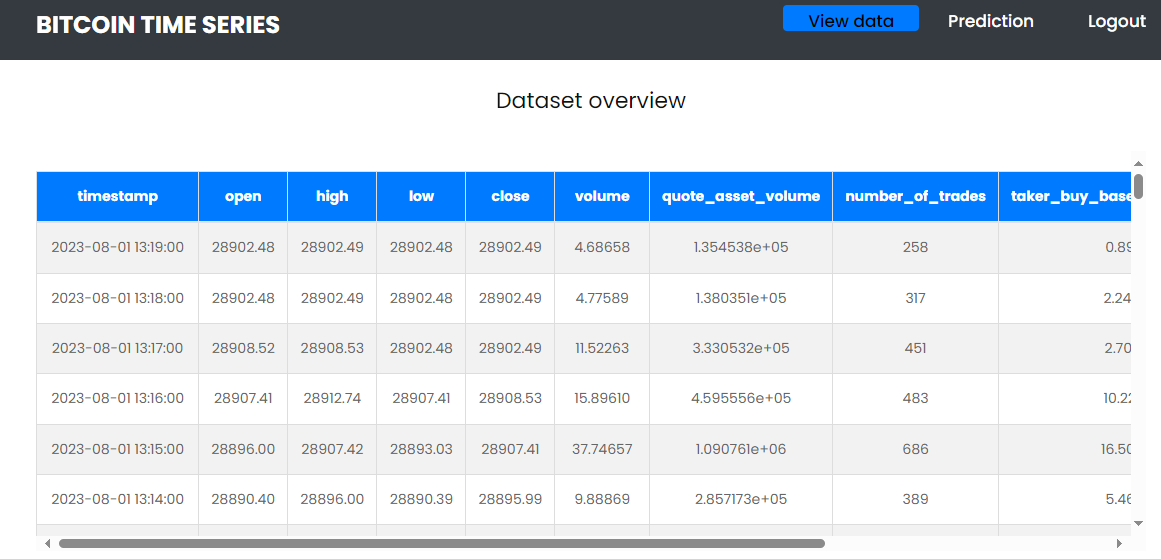
**Register:** The registration page enables new users to create an account so user have to provide name, email, password, confirm password and Address to register.

****

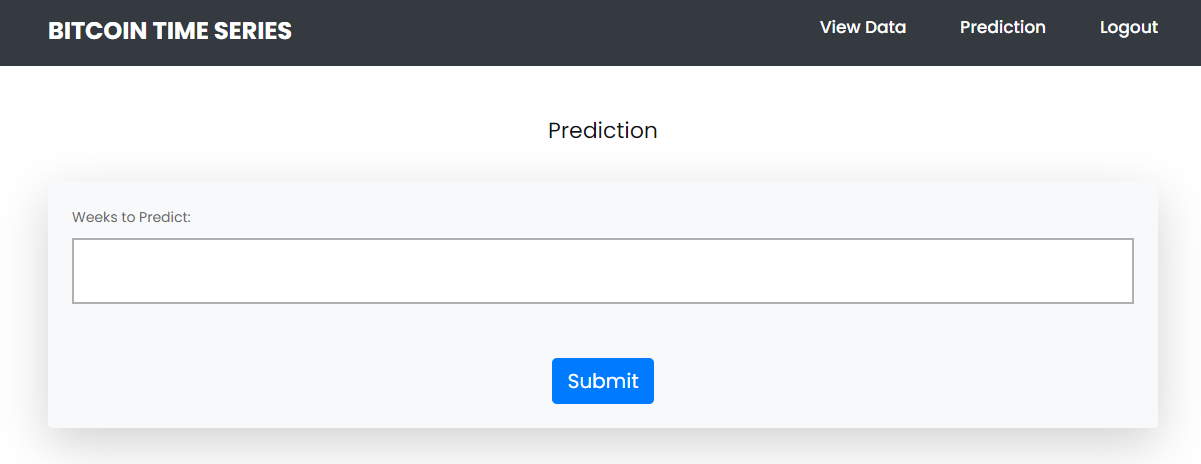
**Login:** The login page allows registered users to securely access the application. User have to provide register email and password.

****

**View:** The view data page provides users with access to the data used for Time Series Analysis Bitcoin.

****

**Prediction:** This is the prediction page here user have to provide values in week to forecast the bitcoin value.

****

**7. TEST CASES:**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Result** |
| Input | Tested for different model given by user on the different model. | Success |
| Prophet | Tested for different input given by the user on different models are created using the different algorithm and data. | Success |
| Prediction | Prediction will be performed using to build from the algorithm. | Success |

**Test cases Model building:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.NO** | **Test cases** | **I/O** | **Expected O/T** | **Actual O/T** | **P/F** |
| 1 | Read the datasets. | Dataset’s path. | Datasets need to read successfully. | Datasets fetched successfully. | It produced P. If this not F will come in case the data is not in the form of .csv |
| 2 | Feature engineering | Need to check the dataset null values/categorical values | Dataset Pre processed successfully | Data wrangled successfully | It produced P. If this not F will come |
| 3 | Modelling | Input with algorithms to get metrics | Algorithm accuracy will be in the form of percentage | We can get the accuracy of each and every model one by one | It produced P. If this is not, it will undergo F |
| 4 | Prediction | Need to enter the input values | Need to predict the output based on the user input | Result successfully predicted with particular algorithm | It produced P. If this is not, it will undergo F |

**8. CONCLUSION**:

In conclusion, this study demonstrates the effectiveness of various time series methodologies, including RNN, LSTM, ARIMA, and Facebook's Prophet, in predicting Bitcoin prices. By analysing historical price data, we identified that each model has distinct advantages and limitations, particularly in handling the volatility characteristic of cryptocurrency markets. Our findings suggest that while traditional models like ARIMA provide a solid baseline, advanced neural network approaches like LSTM and RNN may yield more accurate forecasts in certain scenarios. Facebook's Prophet also proved to be a valuable tool for capturing seasonality and trends. Ultimately, this research not only contributes to understanding Bitcoin price movements but also offers insights that can aid investors in making informed trading decisions amidst market fluctuations.

**9. FUTURE ENHANCEMENTS:**

Future research can explore the integration of additional predictive models, such as Support Vector Regression and XGBoost, to further diversify the forecasting landscape for Bitcoin prices. Incorporating more granular data, including social media sentiment analysis and on-chain metrics, could enhance prediction accuracy by capturing broader market influences. Additionally, applying advanced ensemble techniques, which combine multiple models to improve robustness and reliability, presents a promising direction. Implementing real-time forecasting systems that adapt to market changes could also provide significant benefits for traders. Finally, expanding the study to include other cryptocurrencies would offer comparative insights, enriching the understanding of price dynamics across the digital asset spectrum and paving the way for more comprehensive financial analyses.

**10. REFERENCES:**

[1] I. Alqassem, I. Rahwan, and D. Svetinovic, “The anti-social system properties: Bitcoin network data analysis,” IEEE Trans. Syst., Man, Cybern., Syst., vol. 50, no. 1, pp. 21–31, Jan. 2020. [11] J. Kim, J. Kim, H. Kim, M. Shim, and E. Choi, ‘‘CNN-based network intrusion detection against denial-of-service attacks,’’ Electronics, vol. 9, no. 6, p. 916, Jun. 2020.

[2] P. Nerurkar, D. Patel, Y. Busnel, R. Ludinard, S. Kumari, and M. K. Khan, “Dissecting Bitcoin blockchain: Empirical analysis of Bit- coin network (2009–2020),” J. Netw. Comput. Appl., vol. 177, Mar. 2021, Art. no. 102940.

[3] M. K. Popuri and M. H. Gunes, Empirical Analysis of Crypto Curren- cies. Berlin, Germany: Springer, 2016, pp. 281–292.

[4] D. Ron and A. Shamir, “Quantitative analysis of the full Bitcoin transaction graph,” in Proc. Int. Conf. Financial Cryptography Data Secur. Cham, Switzerland: Springer, 2013, pp. 6–24.

[5] L. Serena, S. Ferretti, and G. D’Angelo, “Cryptocurrencies activity as a complex network: Analysis of transactions graphs,” Peer-Peer Netw. Appl., vol. 15, no. 2, pp. 839–853, Mar. 2022.

[6] B. Tao, I. W. Ho, and H.-N. Dai, “Complex network analysis of the Bitcoin blockchain network,” in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), May 2021, pp. 1–5.

[7] B. Tao, H.-N. Dai, J. Wu, I. W. Ho, Z. Zheng, and C. F. Cheang, “Complex network analysis of the Bitcoin transaction network,” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 69, no. 3, pp. 1009–1013, Mar. 2022.

[8] S. Ranshous et al., “Exchange pattern mining in the Bitcoin transaction directed hypergraph,” in Financial Cryptography and Data Security. Sliema, Malta: Springer, 2017, pp. 248–263.

[9] M. Romiti, A. Judmayer, A. Zamyatin, and B. Haslhofer, “A deep dive into Bitcoin mining pools: An empirical analysis of mining shares,” 2019, arXiv:1905.05999.

[10] N. Tovanich, N. Soulié, N. Heulot, and P. Isenberg, “An empirical analysis of pool hopping behavior in the Bitcoin blockchain,” in Proc. IEEE Int. Conf. Blockchain Cryptocurrency, May 2021, pp. 1–9.

**BIBLIOGRAPHY:**

**SOFTWARE INSTALLATION FOR MACHINE LEARNING PROJECTS:**

**Installing Python:**

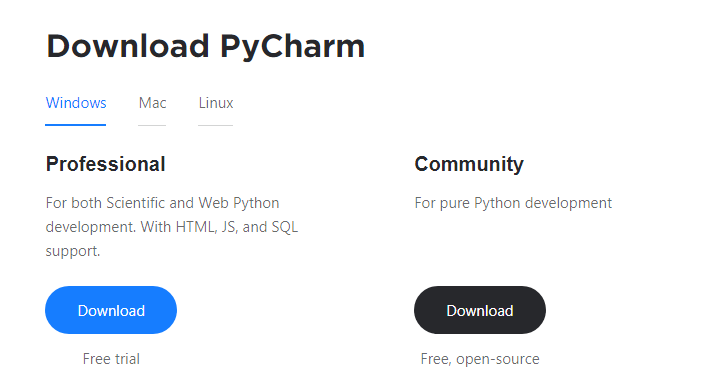
1. To download and install Python visit the official website of Python <https://www.python.org/downloads/> and choose your version.



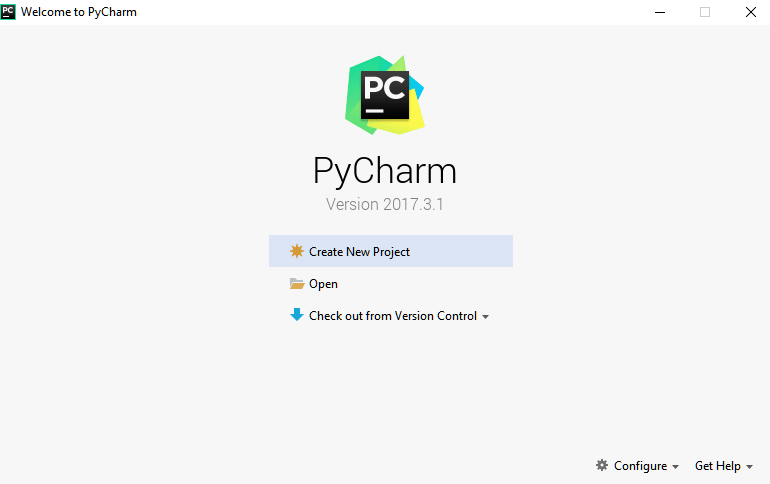
1. Once the download is complete, run the exe for install Python. Now click on Install Now.
2. You can see Python installing at this point.
3. When it finishes, you can see a screen that says the Setup was successful. Now click on "Close".

**Installing PyCharm:**

1. To download PyCharm visit the website <https://www.jetbrains.com/pycharm/download/> and click the "DOWNLOAD" link under the Community Section.



1. Once the download is complete, run the exe for install PyCharm. The setup wizard should have started. Click “Next”.
2. On the next screen, Change the installation path if required. Click “Next”.
3. On the next screen, you can create a desktop shortcut if you want and click on “Next”.
4. Choose the start menu folder. Keep selected JetBrains and click on “Install”.
5. Wait for the installation to finish.
6. Once installation finished, you should receive a message screen that PyCharm is installed. If you want to go ahead and run it, click the “Run PyCharm Community Edition” box first and click “Finish”.
7. After you click on "Finish," the Following screen will appear.

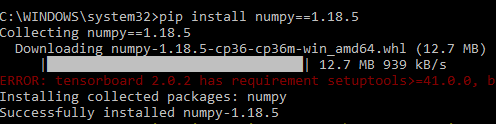


9. You need to install some packages to execute your project in a proper way.

10. Open the command prompt/ anaconda prompt or terminal as administrator.

11. The prompt will get open, with specified path, type “pip install package name” which you want to install (like NumPy, pandas, sea born, scikitlearn, Matplotlib, Pyplot)

Ex: Pip install NumPy



# **INTRODUCTION TO PYTHON**

* Python

### What Is a Script?

Up to this point, I have concentrated on the interactive programming capability of Python.  This is a very useful capability that allows you to type in a program and to have it executed immediately in an interactive mode

Scripts are reusable

Basically, a script is a text file containing the statements that comprise a Python program.  Once you have created the script, you can execute it over and over without having to retype it each time.

Scripts are editable

Perhaps, more importantly, you can make different versions of the script by modifying the statements from one file to the next using a text editor.  Then you can execute each of the individual versions.  In this way, it is easy to create different programs with a minimum amount of typing.

You will need a text editor

Just about any text editor will suffice for creating Python script files.

You can use *Microsoft Notepad, Microsoft WordPad, Microsoft Word,*or just about any word processor if you want to.

Difference between a script and a program

Script:

Scripts are distinct from the core code of the application, which is usually written in a different language, and are often created or at least modified by the enduser. Scripts are often interpreted from source code or byte code, whereas the applications they control are traditionally compiled to native machine code.

Program:

The program has an executable form that the computer can use directly to execute the instructions.

The same program in its humanreadable source code form, from which executable programs are derived (e.g., compiled)

Python

What is Python? Chances you are asking yourself this. You may have found this book because you want to learn to program but don’t know anything about programming languages. Or you may have heard of programming languages like C, C++, C#, or Java and want to know what Python is and how it compares to “big name” languages. Hopefully I can explain it for you.

Python concepts

If you’re not interested in the how’s and whys of Python, feel free to skip to the next chapter. In this chapter I will try to explain to the reader why I think Python is one of the best languages available and why it’s a great one to start programming with.

• Opensource generalpurpose language.

• Object Oriented, Procedural, Functional

• Easy to interface with C/ObjC/Java/Fortran

• Easyis to interface with C++ (via SWIG)

• Great interactive environment

• Great interactive environment

Python is a highlevel, interpreted, interactive and objectoriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

* Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
* Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
* Python is ObjectOriented − Python supports ObjectOriented style or technique of programming that encapsulates code within objects.
* Python is a Beginner's Language − Python is a great language for the beginnerlevel programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

History of Python

Python was developed by Guido van Possum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula3, C, C++, Algol68, Smalltalk, and UNIX shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Possum still holds a vital role in directing its progress.

Python Features

Python's features include −

* Easytolearn − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
* Easytoread − Python code is more clearly defined and visible to the eyes.
* Easytomaintain − Python's source code is fairly easytomaintained.
* A broad standard library − Python's bulk of the library is very portable and crossplatform compatible on UNIX, Windows, and Macintosh.
* Interactive Mode − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
* Portable − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
* Extendable − you can add lowlevel modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
* Databases − Python provides interfaces to all major commercial databases.
* GUI Programming − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
* Scalable − Python provides a better structure and support for large programs than shell scripting.

Apart from the abovementioned features, Python has a big list of good features, few are listed below −

* It supports functional and structured programming methods as well as OOP.
* It can be used as a scripting language or can be compiled to bytecode for building large applications.
* It provides very highlevel dynamic data types and supports dynamic type checking.
* IT supports automatic garbage collection.
* It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Dynamic vs. Static

Types Python is a dynamictyped language. Many other languages are static typed, such as C/C++ and Java. A static typed language requires the programmer to explicitly tell the computer what type of “thing” each data value is.

For example, in C if you had a variable that was to contain the price of something, you would have to declare the variable as a “float” type.

This tells the compiler that the only data that can be used for that variable must be a floating point number, i.e. a number with a decimal point.

If any other data value was assigned to that variable, the compiler would give an error when trying to compile the program.

Python, however, doesn’t require this. You simply give your variables names and assign values to them. The interpreter takes care of keeping track of what kinds of objects your program is using. This also means that you can change the size of the values as you develop the program. Say you have another decimal number (a.k.a. a floating point number) you need in your program.

With a static typed language, you have to decide the memory size the variable can take when you first initialize that variable. A double is a floating point value that can handle a much larger number than a normal float (the actual memory sizes depend on the operating environment).

If you declare a variable to be a float but later on assign a value that is too big to it, your program will fail; you will have to go back and change that variable to be a double.

With Python, it doesn’t matter. You simply give it whatever number you want and Python will take care of manipulating it as needed. It even works for derived values.

For example, say you are dividing two numbers. One is a floating point number and one is an integer. Python realizes that it’s more accurate to keep track of decimals so it automatically calculates the result as a floating point number

Variables

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Standard Data Types

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types −

* Numbers
* String
* List
* Tuple
* Dictionary

## Python Numbers

Number data types store numeric values. Number objects are created when you assign a value to them

## Python Strings

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from 1 at the end.

## Python Lists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end 1. The plus (+) sign is the list concatenation operator, and the asterisk () is the repetition operator.

## Python Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as readonly lists.

## Python Dictionary

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of keyvalue pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

Different modes in python

Python has two basic modes: normal and interactive.

The normal mode is the mode where the scripted and finished .pie files are run in the Python interpreter.

Interactive mode is a command line shell which gives immediate feedback for each statement, while running previously fed statements in active memory. As new lines are fed into the interpreter, the fed program is evaluated both in part and in whole

# 20 Python libraries

1. Requests. The most famous http library written by Kenneth remits. It’s a must have for every python developer.

2. Scrappy. If you are involved in web scraping then this is a must have library for you. After using this library you won’t use any other.

3. Python. A guy toolkit for python. I have primarily used it in place of tinder. You will really love it.

4. Pillow. A friendly fork of PIL (Python Imaging Library). It is more user friendly than PIL and is a must have for anyone who works with images.

5. SQL Alchemy. A database library. Many love it and many hate it. The choice is yours.

6. Beautiful Soup. I know it’s slow but this xml and html parsing library is very useful for beginners.

7. Twisted. The most important tool for any network application developer. It has a very beautiful ape and is used by a lot of famous python developers.

8. Numbly. How can we leave this very important library? It provides some advance math functionalities to python.

9. Skippy. When we talk about numbly then we have to talk about spicy. It is a library of algorithms and mathematical tools for python and has caused many scientists to switch from ruby to python.

10. Matplotlib. A numerical plotting library. It is very useful for any data scientist or any data analyser.

11. Pygmy. Which developer does not like to play games and develop them? This library will help you achieve your goal of 2d game development.

12. Piglet. A 3d animation and game creation engine. This is the engine in which the famous [python port](https://github.com/fogleman/Minecraft) of mine craft was made

13. Pit. A GUI toolkit for python. It is my second choice after python for developing GUI’s for my python scripts.

14. Pit. Another python GUI library. It is the same library in which the famous Bit torrent client is created.

15. Scaly. A packet sniffer and analyser for python made in python.

16. Pywin32. A python library which provides some useful methods and classes for interacting with windows.

17. Notch. Natural Language Toolkit – I realize most people won’t be using this one, but it’s generic enough. It is a very useful library if you want to manipulate strings. But its capacity is beyond that. Do check it out.

18. Nose. A testing framework for python. It is used by millions of python developers. It is a must have if you do test driven development.

19. Simply. Simply cando algebraic evaluation, differentiation, expansion, complex numbers, etc. It is contained in a pure Python distribution.

20. I Python. I just can’t stress enough how useful this tool is. It is a python prompt on steroids. It has completion, history, shell capabilities, and a lot more. Make sure that you take a look at it.

NumPy

Humpy’s main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In numbly dimensions are called axes. The number of axes is rank.

• Offers Matlabish capabilities within Python

• Fast array operations

• 2D arrays, multiD arrays, linear algebra etc.

Matplotlib

• High quality plotting library.

Python class and objects

These are the building blocks of OOP. Class creates a new object. This object can be anything, whether an abstract data concept or a model of a physical object, e.g. a chair. Each class has individual characteristics unique to that class, including variables and methods. Classes are very powerful and currently “the big thing” in most programming languages. Hence, there are several chapters dedicated to OOP later in the book.

The class is the most basic component of objectoriented programming. Previously, you learned how to use functions to make your program do something.

Now will move into the big, scary world of ObjectOriented Programming (OOP). To be honest, it took me several months to get a handle on objects.

When I first learned C and C++, I did great; functions just made sense for me.

Having messed around with BASIC in the early ’90s, I realized functions were just like subroutines so there wasn’t much new to learn.

However, when my C++ course started talking about objects, classes, and all the new features of OOP, my grades definitely suffered.

Once you learn OOP, you’ll realize that it’s actually a pretty powerful tool. Plus many Python libraries and APIs use classes, so you should at least be able to understand what the code is doing.

One thing to note about Python and OOP: it’s not mandatory to use objects in your code in a way that works best; maybe you don’t need to have a fullblown class with initialization code and methods to just return a calculation. With Python, you can get as technical as you want.

As you’ve already seen, Python can do just fine with functions. Unlike languages such as Java, you aren’t tied down to a single way of doing things; you can mix functions and classes as necessary in the same program. This lets you build the code

Objects are an encapsulation of variables and functions into a single entity. Objects get their variables and functions from classes. Classes are essentially a template to create your objects.

Here’s a brief list of Python OOP ideas:

• The class statement creates a class object and gives it a name. This creates a new namespace.

• Assignments within the class create class attributes. These attributes are accessed by qualifying the name using dot syntax: ClassName.Attribute.

• Class attributes export the state of an object and its associated behaviour. These attributes are shared by all instances of a class.

• Calling a class (just like a function) creates a new instance of the class.

This is where the multiple copy’s part comes in.

• Each instance gets ("inherits") the default class attributes and gets its own namespace. This prevents instance objects from overlapping and confusing the program.

• Using the term self identifies a particular instance, allowing for perinstance attributes. This allows items such as variables to be associated with a particular instance.

Inheritance

First off, classes allow you to modify a program without really making changes to it.

To elaborate, by sub classing a class, you can change the behaviour of the program by simply adding new components to it rather than rewriting the existing components.

As we’ve seen, an instance of a class inherits the attributes of that class.

However, classes can also inherit attributes from other classes. Hence, a subclass inherits from a superclass allowing you to make a generic superclass that is specialized via subclasses.

The subclasses can override the logic in a superclass, allowing you to change the behaviour of your classes without changing the superclass at all.

Operator Overloads

Operator overloading simply means that objects that you create from classes can respond to actions (operations) that are already defined within Python, such as addition, slicing, printing, etc.

Even though these actions can be implemented via class methods, using overloading ties the behavior closer to Python’s object model and the object interfaces are more consistent to Python’s builtin objects, hence overloading is easier to learn and use.

Usermade classes can override nearly all of Python’s builtin operation methods

Exceptions

I’ve talked about exceptions before but now I will talk about them in depth. Essentially, exceptions are events that modify program’s flow, either intentionally or due to errors.

They are special events that can occur due to an error, e.g. trying to open a file that doesn’t exist, or when the program reaches a marker, such as the completion of a loop.

Exceptions, by definition, don’t occur very often; hence, they are the "exception to the rule" and a special class has been created for them. Exceptions are everywhere in Python.

Virtually every module in the standard Python library uses them, and Python itself will raise them in a lot of different circumstances.

Here are just a few examples:

• Accessing a non−existent dictionary key will raise a Key Error exception.

• Searching a list for a non−existent value will raise a Value Error exception

. • Calling a non−existent method will raise an Attribute Error exception.

• Referencing a non−existent variable will raise a Name Error exception.

• Mixing data types without coercion will raise a Type Error exception.

One use of exceptions is to catch a fault and allow the program to continue working; we have seen this before when we talked about files.

This is the most common way to use exceptions. When programming with the Python command line interpreter, you don’t need to worry about catching exceptions.

Your program is usually short enough to not be hurt too much if an exception occurs.

Plus, having the exception occur at the command line is a quick and easy way to tell if your code logic has a problem.

However, if the same error occurred in your real program, it will fail and stop working. Exceptions can be created manually in the code by raising an exception.

It operates exactly as a systemcaused exceptions, except that the programmer is doing it on purpose. This can be for a number of reasons. One of the benefits of using exceptions is that, by their nature, they don’t put any overhead on the code processing.

Because exceptions aren’t supposed to happen very often, they aren’t processed until they occur.

Exceptions can be thought of as a special form of the if/elf statements. You can realistically do the same thing with if blocks as you can with exceptions.

However, as already mentioned, exceptions aren’t processed until they occur; if blocks are processed all the time.

Proper use of exceptions can help the performance of your program.

The more infrequent the error might occur, the better off you are to use exceptions; using if blocks requires Python to always test extra conditions before continuing.

Exceptions also make code management easier: if your programming logic is mixed in with errorhandling if statements, it can be difficult to read, modify, and debug your program.

UserDefined Exceptions

I won’t spend too much time talking about this, but Python does allow for a programmer to create his own exceptions.

You probably won’t have to do this very often but it’s nice to have the option when necessary.

However, before making your own exceptions, make sure there isn’t one of the builtin exceptions that will work for you.

They have been "tested by fire" over the years and not only work effectively, they have been optimized for performance and are bugfree.

Making your own exceptions involves objectoriented programming, which will be covered in the next chapter

. To make a custom exception, the programmer determines which base exception to use as the class to inherit from, e.g. making an exception for negative numbers or one for imaginary numbers would probably fall under the Arithmetic Error exception class.

To make a custom exception, simply inherit the base exception and define what it will do.

Python modules

Python allows us to store our code in files (also called modules). This is very useful for more serious programming, where we do not want to retype a long function definition from the very beginning just to change one mistake. In doing this, we are essentially defining our own modules, just like the modules defined already in the Python library.

To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a module; definitions from a module can be imported into other modules or into the main module.

Testing code

As indicated above, code is usually developed in a file using an editor.

To test the code, import it into a Python session and try to run it.

Usually there is an error, so you go back to the file, make a correction, and test again.

This process is repeated until you are satisfied that the code works. T

His entire process is known as the development cycle.

There are two types of errors that you will encounter. Syntax errors occur when the form of some command is invalid.

This happens when you make typing errors such as misspellings, or call something by the wrong name, and for many other reasons. Python will always give an error message for a syntax error.

Functions in Python

It is possible, and very useful, to define our own functions in Python. Generally speaking, if you need to do a calculation only once, then use the interpreter. But when you or others have need to perform a certain type of calculation many times, then define a function.

You use functions in programming to bundle a set of instructions that you want to use repeatedly or that, because of their complexity, are better selfcontained in a subprogram and called when needed. That means that a function is a piece of code written to carry out a specified task.

## To carry out that specific task, the function might or might not need multiple inputs. When the task is carved out, the function can or cannot return one or more values.

## There are three types of functions in python:

## Help (), min (), print ().

Namespaces in Python are implemented as Python dictionaries, this means it is a mapping from names (keys) to objects (values). The user doesn't have to know this to write a Python program and when using namespaces.

Some namespaces in Python:

* global names of a module
* local names in a function or method invocation
* builtin names: this namespace contains builtin functions (e.g. abs(), camp(), ...) and builtin exception names

Garbage Collection

Garbage Collector exposes the underlying memory management mechanism of Python, the automatic garbage collector. The module includes functions for controlling how the collector operates and to examine the objects known to the system, either pending collection or stuck in reference cycles and unable to be freed.

Python XML Parser

XML is a portable, open source language that allows programmers to develop applications that can be read by other applications, regardless of operating system and/or developmental language.

What is XML? The Extensible Markup Language XML is a markup language much like HTML or SGML.

This is recommended by the World Wide Web Consortium and available as an open standard.

XML is extremely useful for keeping track of small to medium amounts of data without requiring a SQLbased backbone.

XML Parser Architectures and APIs the Python standard library provides a minimal but useful set of interfaces to work with XML.

The two most basic and broadly used APIs to XML data are the SAX and DOM interfaces.

Simple API for XML SAX: Here, you register callbacks for events of interest and then let the parser proceed through the document.

This is useful when your documents are large or you have memory limitations, it parses the file as it reads it from disk and the entire file is never stored in memory.

Document Object Model DOM API : This is a World Wide Web Consortium recommendation wherein the entire file is read into memory and stored in a hierarchical tree − based form to represent all the features of an XML document.

SAX obviously cannot process information as fast as DOM can when working with large files. On the other hand, using DOM exclusively can really kill your resources, especially if used on a lot of small files.

SAX is readonly, while DOM allows changes to the XML file. Since these two different APIs literally complement each other, there is no reason why you cannot use them both for large projects.

Python Web Frameworks

A web framework is a code library that makes a developer's life easier when building reliable, scalable and maintainable web applications.

## Why are web frameworks useful?

Web frameworks encapsulate what developers have learned over the past twenty years while programming sites and applications for the web. Frameworks make it easier to reuse code for common HTTP operations and to structure projects so other developers with knowledge of the framework can quickly build and maintain the application.

Common web framework functionality

Frameworks provide functionality in their code or through extensions to perform common operations required to run web applications. These common operations include:

1. URL routing
2. HTML, XML, JSON, and other output format tinplating
3. Database manipulation
4. Security against Crosssite request forgery (CSRF) and other attacks
5. Session storage and retrieval

Not all web frameworks include code for all of the above functionality. Frameworks fall on the spectrum from executing a single use case to providing every known web framework feature to every developer. Some frameworks take the "batteriesincluded" approach where everything possible comes bundled with the framework while others have a minimal core package that is amenable to extensions provided by other packages.

## Comparing web frameworks

There is also a repository called [comparepythonwebframeworks](https://github.com/mattmakai/compare-python-web-frameworks) where the same web application is being coded with varying Python web frameworks, tinplating engines and object.

## Web framework resources

* When you are learning how to use one or more web frameworks it's helpful to have an idea of what the code under the covers is doing.
* Frameworks is a really well done short video that explains how to choose between web frameworks. The author has some particular opinions about what should be in a framework. For the most part I agree although I've found sessions and database ORMs to be a helpful part of a framework when done well.
* What is a web framework? Is an indepth explanation of what web frameworks are and their relation to web servers?
* Jingo vs. Flash vs. Pyramid: Choosing a Python web framework contains background information and code comparisons for similar web applications built in these three big Python frameworks.
* This fascinating blog post takes a look at the code complexity of several Python web frameworks by providing visualizations based on their code bases.
* Python’s web frameworks benchmarks  is a test of the responsiveness of a framework with encoding an object to JSON and returning it as a response as well as retrieving data from the database and rendering it in a template. There were no conclusive results but the output is fun to read about nonetheless.
* What web frameworks do you use and why are they awesome? Is a language agnostic Reedit discussion on web frameworks? It's interesting to see what programmers in other languages like and dislike about their suite of web frameworks compared to the main Python frameworks.
* This uservoted question & answer site asked "What are the best general purpose Python web frameworks usable in production?” The votes aren't as important as the list of the many frameworks that are available to Python developers.

## Web frameworks learning checklist

1. Choose a major Python web framework (Jingo or Flask are recommended) and stick with it. When you're just starting it's best to learn one framework first instead of bouncing around trying to understand every framework.
2. Work through a detailed tutorial found within the resources links on the framework's page.
3. Study open source examples built with your framework of choice so you can take parts of those projects and reuse the code in your application.
4. Build the first simple iteration of your web application then go to the [deployment](https://www.fullstackpython.com/deployment.html) section to make it accessible on the web.

2. SYSTEM STUDY

### SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

TYPES OF TESTS

Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

SYSTEM TEST

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configurationoriented system integration test. System testing is based on process descriptions and flows, emphasizing predriven process links and integration points.

White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box. you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

6.1 Unit Testing:

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

* All field entries must work properly.
* Pages must be activated from the identified link.
* The entry screen, messages and responses must not be delayed.

Features to be tested

1. Verify that the entries are of the correct format
2. No duplicate entries should be allowed
3. All links should take the user to the correct page.

# 6.2 Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

6.3 Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.