

## Table of Contents

Abstract	i
Acknowledgement	ii
List of tables	iii
List of figures	iv
<b>Chapter 1</b>	<b>4</b>
PREAMBLE	4
1.1 Introduction	4
1.2 Existing System	4
1.2.1 Top-down approach	5
1.2.2 Bottom-up approach	5
1.3 Problem Statement	6
1.4 Objective of the project	6
1.5 Proposed System	6
1.5.1 Dataset/Features	6
1.5.2 System Flow	7
1.5.3 Advantages of proposed system	7
1.6 Phase Description	8
Table 1.6: Phase Description	8
1.7 Organization of the project report	8
<b>Chapter 2</b>	<b>10</b>
LITERATURE SURVEY	10
<b>Chapter 3</b>	<b>12</b>
THEORETICAL BACKGROUND	12
3.1 Introduction to Gradient boosting decision tree	12
3.2 Introduction to Random Forest	13
<b>Chapter 4</b>	<b>16</b>
SYSTEM REQUIREMENT SPECIFICATION	16
4.1 Functional Requirement	16
4.2 Non-Functional Requirement	17
4.2.1 Product Requirements	18
4.2.2 Organizational Requirements	19
4.2.3 User Requirements	19
4.2.4 Basic Operational Requirements	19

□ Operational life cycle - It defines the system lifetime .....	20
4.3 Hardware Requirements .....	20
4.4 Software Requirements .....	21
<b>Chapter 5</b> .....	22
SYSTEM ANALYSIS .....	22
5.1 Feasibility Study.....	22
5.1.1 Performance Analysis .....	22
5.1.2 Technical Analysis.....	22
5.1.3 Economical Analysis .....	23
<b>Chapter 6</b> .....	24
SYSTEM DESIGN .....	24
6.1 System development methodology .....	24
6.1.1 Software Development Life Cycle.....	24
6.1.2 Agile Software Development.....	27
6.2 Design Using UML .....	28
6.2.1 Architectural Design .....	30
6.2.2 Data Flow Diagram.....	30
6.2.3 Flow Chart .....	32
6.2.4 Use Case Diagram.....	33
6.2.5 Activity Diagram .....	33
6.2.6 Sequence Diagram .....	35
<b>Chapter 7</b> .....	37
IMPLEMENTATION.....	37
7.1 Introduction to Python.....	37
7.1.1 Features of Python Programming .....	37
7.2 Modules in implementation.....	38
7.2.1 Functions used for input.....	39
7.2.2 Functions used for processing data using machine learning algorithm .....	39
7.2.3 Functions used for output.....	40
<b>Chapter 8</b> .....	42
TESTING.....	42
8.1 Testing Methodologies .....	42
8.1.1 White box testing .....	42
8.1.2 Black box testing.....	43
8.2 Unit Testing.....	43
8.3 Integration Testing .....	45
<b>Chapter 9</b> .....	51

RESULTS AND EXECUTION.....	51
9.1    Snapshots.....	51
<b>Chapter 10</b> .....	54
CONCLUSION.....	54
References	
Appendix	

## Chapter 1

### PREAMBLE

#### 1.1 Introduction

Forecasting is the use of various applicable analytical methods to project a variable(revenue) into the future. Financial forecasting is a crucial tool for any business because it enables business owners to anticipate profits. The ability to accurately predict fluctuations in revenue allows business owners plan strategy, production and distribution for coming years, overcome cash flow issues and budget accordingly. Forecasts range in sophistication from educated guesses by business owners to complex mathematical studies.

New start-ups incur huge time and capital investments to establish. When the new outlet fails to break even, the site closes within a short time and operating losses are incurred. Finding an algorithmic model to increase the return on investments in new start-up sites would facilitate businesses to direct their investments in other important business areas, like innovation, and training for new employees.

#### 1.2 Existing System

Forecasting involves the following steps-

1. **Define the Problem-** What issues affect the forecast and presentation?
2. **Gather Information-** Obtain statistical data, along with accumulated judgment and expertise, to support forecasting.
3. **Conduct a Preliminary/Exploratory Analysis-** Examine data to identify major drivers and important trends. This establishes basic familiarity with the revenue being forecast.
4. **Select Methods-** Determine the most appropriate quantitative and qualitative methods.
5. **Implement Methods-** Use the selected methods to make the long-range forecast.

Forecasts range in sophistication from educated guesses by business owners to complex mathematical studies. While there are many methodologies for preparing a financial forecast, two of the most common are top-down and bottom-up analysis.

### **1.2.1 Top-down approach**

A top-down analysis starts with a business assessing the market as a whole. First, it determines the current market size available for the business and factors in relevant sales trends. Then, how much of the market will buy the products or services can be estimated. In the context of these trends, then examine the company's strengths and weaknesses and, ideally, how to amplify strengths and remedy weaknesses in the business. In simple terms, top-down models start with the entire market and works down.

#### **Limitations-**

- With top-down forecasting, profits from various products and regions are averaged together rather than considered on an item-by-item basis. The resulting average doesn't properly represent any individual component.
- It is very inaccurate with respect to timing. Therefore, it offers little value for revenue and cash-flow planning for shorter periods of time and for sale management, staffing and supply chain decisions.
- Top-down forecasting predicts sales poorly when the markets and advertising of individual items are different. Top-down forecasting is thus often less successful at the item level.
- Forecasts are grounded on assumptions rather than facts and provides unrealistic expectations about potential growth.

### **1.2.2 Bottom-up approach**

A bottom-up analysis is grounded in the product or service itself, from which a projection is made based on what is needed to get the offering to the market (i.e. number of employees, how many factories can be opened or number of clients). Also known as an operating expense plan, bottom-up forecasts examine factors such as production capacity, department-specific expenses, and addressable market in order to create a more accurate sales projection. In simple terms, bottom-up forecasts begin with the individual business and expand out.

## **Limitations-**

- Bottom-Up forecasting begins with the micro view and builds to a macro view. Errors at the micro level are amplified as they approach the macro level. They are compounded during the expansion process.
- Need up-to-the-minute Point Of Sale (POS) data to forecast results.
- Slower when compared to other methods.
- Bottom-up forecasting may not be simple because of complications with the accuracy of the data submitted. The usefulness of the data is contingent upon honest and complete answers from customers, and on the importance and priority given to a survey by the sales staff.

## **1.3 Problem Statement**

The problem can be defined as- Design an automated approach to decide the task environment for new start-up by applying concepts of Gradient boosting decision tree and Random Forest on certain parameters, it will predict the annual revenue of a new start-up outlet which would help to determine its feasibility.

## **1.4 Objective of the project**

The primary objective is to help start-ups make a more informed and optimal decision about opening new outlets. It aims to find an algorithmic model to increase the effectiveness of investments in new start-up sites.

## **1.5 Proposed System**

### **1.5.1 Dataset/Features**

The data set consists of a training and test set with 137 and 100,000 samples respectively. The 137 training samples provide actual revenue while the test set does not and expects the user to submit their predictions on the 100,000 testing examples. The 43 features provided are listed below-

- P-Variables (parameters from P1 to P37)- Obfuscated variables from three categories- demographic data, which includes population, age, gender; real estate data which includes car parking availability and number of front facade; commercial data denotes points of interest like

banks, schools, other public places etc. Each variable may contain a combination of the three categories or may be mutually exclusive

- ID- start-up ID
- Open Date- Date that the start-up opened in the format MM/DD/YYYY
- City- name of the city in which the start-up is located.
- City Group- city group can be either metro, big city or other
- Revenue- Annual revenue of a start-up in a given year and is the target.

Majority of the data fields are obfuscated variables without giving the statistician any prior knowledge of each one.

### **1.5.2 System Flow**

The proposed system takes in the value of features from the user. The data fields such as opening date, city name, city type, number of front sides, car parking and other points of interest are taken and approximated revenue depending upon inputs provided is generated. Thus, output value indicates the annual revenue of the proposed restaurant site. The output is given in US dollars.

### **1.5.3 Advantages of proposed system**

- One of the biggest features of the proposed application is that it aims to predict the revenue of new outlets of existing start-ups.
- Analytical prediction of data has proven more effective than by human judgement.
- Further, it can allow analysis and comparison of multiple new sites.
- Human errors can be avoided and operations can be performed faster than previous methods.
- Revenue prediction system will compute an accurate forecast of a start-up's future revenues.

## 1.6 Phase Description

Phase	Task	Description
Phase 1	Analysis	Analyzing the core of the IEEE paper and provide Literature review based on analysis.
Phase 2	Literature survey	Collect raw data and elaborate on literature surveys.
Phase 3	System analysis	Analyses the requirements of the project and lists the specific requirements needed.
Phase 4	Design	Object designing and Functional description
Phase 5	Implementation	Implement the code based on the object specification
Phase 6	Testing	Test the project according to Test Specification
Phase 7	Documentation	Prepare the document for this project with conclusion and future enhancement.

**Table 1.6: Phase Description**

## 1.7 Organization of the project report

The project report is organized as follows-

**Chapter 2 Literature Review** - Gives a brief overview of the survey papers and the research sources that have been studied to establish a thorough understanding of the project under consideration.

**Chapter 3 Theoretical Background** - Establishes groundwork for the proposed project by giving a detailed analysis of the project topic, existing research relevant to the project, arguments in favor and against the existing solutions and finally explores the motivation behind the proposed solution.

**Chapter 4 System Requirement Specification** - Discusses in details about the different kinds of requirements needed to successfully complete the project.



- Chapter 5      System Analysis** - gives details about several analysis that are performed to facilitate taking decision of whether the project is feasible enough or not.
- Chapter 6      System Design** - Gives the design description of the project, conceptual and detailed design well supported with design diagrams.
- Chapter 7      Implementation** - Discusses the implementation details of the project and reasons the use of the programming language and development environment.
- Chapter 8      Testing** - Briefs the testing methods used for testing the different modules in the project.
- Chapter 9      Results and Performance Analysis** - Gives the snapshots and graphs of the proposed protocols.
- Chapter 10    Conclusion and Future Scope** - Gives the concluding remarks of the project, throwing light on its future aspects.
- References**    Lists the websites and references referred during the project work.

## Chapter 2

### LITERATURE SURVEY

Literature survey is mainly carried out in order to analyze the background of the current project which helps to uncover the problems and flaws in the existing system which motivated to propose solutions and work on this project.

Multiple machine learning algorithms were studied for predicting the annual revenue of start-ups. Research papers on Gradient boosting, Random Forest, SVM were referred.

***[1] “PhDc. Dilmurat Zakirov, PhDc. Aleksey Bondarev and Prof. Dr. Nodar Momtselidze - A Comparison of Data Mining Techniques in Evaluating Retail Credit Scoring Using R Programming” [6]***

Retail credit scoring has become more efficient in recent years because of the use of data mining techniques to better estimate their customers credibility. Support vector machines (SVMs), gradient boosted model (GBM), Random Forest (RF), Naïve Bayes classification and other algorithms are used as analytical methods for customer credit scoring estimation and evaluation, using real dataset. At the end of the study it is found that Random Forest model has better accuracy rate when compared to other models.

***[2] “Badre Labiad, Abdelaziz Berrado, Loubna Benabbou - Machine Learning Techniques for Short Term Stock Movements Classification for Moroccan Stock Exchange” [5]***

Accurate stock price forecasting is important for investors and traders to make an informed trading decision. However, prices have a complex behavior due to their non-linearity and non-stationarity. Three Machine learning techniques are implemented to predict a very short-term variation of the Moroccan stock market- Random Forest (RF), Gradient Boosted Trees (GBT) and Support Vector Machine (SVM). The experimental results have shown that RF and GBT are superior to SVM for the stock dataset.

***[3] “Magdalena Graczyk, Tadeusz Lasota, Bogdan Trawiński, Krzysztof Trawiński - Comparison of Bagging, Boosting and Stacking Ensembles Applied to Real Estate Appraisal” [4]***

The experiments aimed to compare three methods to create ensemble models. Six common algorithms were used to generate individual committees(ensemble) for actual data sets of residential premises sales/purchase real estate transactions. The experimental results show that there is no single algorithm which produces the best ensembles and it is worth to seek an optimal hybrid multi-model solution using greater number of different data sets.

## Chapter 3

### THEORETICAL BACKGROUND

Theoretical background highlighting some topics related to project work. The description contains several topics which are worth to discuss and also highlight some of their limitation that encourage going on finding solution as well as highlights some of their advantages for which reason these topics and their features are used in this project.

#### 3.1 Introduction to Gradient boosting decision tree

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function. Thus, gradient boosting combines weak "learners" into a single strong learner, in an iterative fashion.

Gradient boosting involves the following three elements -

1. A loss function to be optimized.
2. A weak learner to make predictions.
3. An additive model to add weak learners to minimize the loss function.

##### 1. Loss Function

The loss function used depends on the type of problem being solved. It must be differentiable, but many standard loss functions are supported. For example, regression may use a squared error and classification may use logarithmic loss.

A benefit of the gradient boosting framework is that a new boosting algorithm does not have to be derived for each loss function that may want to be used, instead, it is a generic enough framework that any differentiable loss function can be used.

##### 2. Weak Learner

Decision trees are used as the weak learner in gradient boosting. Specifically, regression trees are used that output real values for splits and whose output can be added together, allowing subsequent models outputs to be added and "correct" the residuals in

the predictions. Trees are constructed in a greedy manner, choosing the best split points based on purity scores like Gini or to minimize the loss.

Initially, such as in the case of AdaBoost, very short decision trees were used that only had a single split, called a decision stump. Larger trees can be used generally with 4-to-8 levels. It is common to constrain the weak learners in specific ways, such as a maximum number of layers, nodes, splits or leaf nodes. This is to ensure that the learners remain weak, but can still be constructed in a greedy manner.

### **3. Additive Model**

Trees are added one at a time, and existing trees in the model are not changed. A gradient descent procedure is used to minimize the loss when adding trees.

Traditionally, gradient descent is used to minimize a set of parameters, such as the coefficients in a regression equation or weights in a neural network. After calculating error or loss, the weights are updated to minimize that error.

Instead of parameters, weak learner sub-models or more specifically decision trees are used. After calculating the loss, to perform the gradient descent procedure, add a tree to the model that reduces the loss (i.e. follow the gradient). This is done by parameterizing the tree, then modifying the parameters of the tree and moving in the right direction by reducing the residual loss.

The output for the new tree is then added to the output of the existing sequence of trees in an effort to correct or improve the final output of the model. A fixed number of trees are added or training stops once loss reaches an acceptable level or no longer improves on an external validation dataset.

## **3.2 Introduction to Random Forest**

A random forest is an ensemble of decision trees created using random variable selection and bootstrap aggregating (bagging). First a group of decision trees are created. For each individual tree, a random sample with replacement of the training data is used for training. Also, at each node of the tree, the split is created by only looking at a random subset of the variables. A commonly used number for each split is the square root of the

number of variables. The prediction is made by averaging the predictions of all the individual trees.

### **Bootstrap Aggregation (Bagging)**

Bootstrap Aggregation (or Bagging for short), is a simple and very powerful ensemble method. An ensemble method is a technique that combines the predictions from multiple machine learning algorithms together to make more accurate predictions than any individual model.

Bootstrap Aggregation is a general procedure that can be used to reduce the variance for those algorithm that have high variance. An algorithm that has high variance are decision trees, like classification and regression trees (CART).

Decision trees are sensitive to the specific data on which they are trained. If the training data is changed (e.g. a tree is trained on a subset of the training data) the resulting decision tree can be quite different and in turn the predictions can be quite different.

Bagging is the application of the Bootstrap procedure to a high-variance machine learning algorithm, typically decision trees. Bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

When bagging with decision trees, individual trees overfitting the training data is not a concern. For this reason and for efficiency, the individual decision trees are grown deep and the trees are not pruned. These trees will have both high variance and low bias.

The only parameters when bagging decision trees is the number of samples and hence the number of trees to include. This can be chosen by increasing the number of trees on run after run until the accuracy begins to stop showing improvement.

## Random Forest

Random Forests are an improvement over bagged decision trees.

A problem with decision trees like CART is that they are greedy. They choose which variable to split on using a greedy algorithm that minimizes error. As such, even with Bagging, the decision trees can have a lot of structural similarities and in turn have high correlation in their predictions.

Combining predictions from multiple models in ensembles works better if the predictions from the sub-models are uncorrelated or at best weakly correlated. Random forest changes the algorithm for the way that the sub-trees are learned so that the resulting predictions from all of the subtrees have less correlation.

In CART, when selecting a split point, the learning algorithm is allowed to look through all variables and all variable values in order to select the most optimal split-point. The random forest algorithm changes this procedure so that the learning algorithm is limited to a random sample of features of which to search. The number of features that can be searched at each split point ( $m$ ) must be specified as a parameter to the algorithm.

For each bootstrap sample taken from the training data, there will be samples left behind that were not included. These samples are called Out-Of-Bag samples or OOB.

The performance of each model on its left-out samples when averaged can provide an estimated accuracy of the bagged models. This estimated performance is often called the OOB estimate of performance. These performance measures are reliable test error estimate and correlate well with cross validation estimates.

## Summary

This chapter mainly concentrates on the basic theoretical background related to the topic of focus. It gives information about the platform on which this application has been developed. Machine learning algorithms-Gradient boosting decision tree and Random Forest are also discussed in this chapter.

## Chapter 4

### SYSTEM REQUIREMENT SPECIFICATION

Software requirement Specification is a fundamental document, which forms the foundation of the software development process. It not only lists the requirements of a system but also has a description of its major feature. An SRS is basically an organization's understanding (in writing) of a customer or potential client's system requirements and dependencies at a particular point in time (usually) prior to any actual design or development work. It's a two-way insurance policy that assures that both the client and the organization understand the other's requirements from that perspective at a given point in time.

The SRS also functions as a blueprint for completing a project with as little cost growth as possible. The SRS is often referred to as the "parent" document because all subsequent project management documents, such as design specifications, statements of work, software architecture specifications, testing and validation plans, and documentation plans, are related to it. It is important to note that an SRS contains functional and nonfunctional requirements only; it doesn't offer design suggestions, possible solutions to technology or business issues, or any other information other than what the development team understands the customer's system requirements to be.

#### 4.1 Functional Requirement

Functional Requirement defines a function of a software system and how the system must behave when presented with specific inputs or conditions. These may include calculations, data manipulation and processing and other specific functionality. In this system following are the functional requirements-

- The application must not stop working when kept running for even a long time.
- The application must function as expected for every set of test cases provided.
- The application should generate the output for given input test case and input parameters.
- The application should generate on-demand services.



- Input test case must not have compilation and runtime errors.

## 4.2 Non-Functional Requirement

Non-functional requirements are the requirements which are not directly concerned with the specific function delivered by the system. They specify the criteria that can be used to judge the operation of a system rather than specific behaviors. They may relate to emergent system properties such as reliability, response time and store occupancy. Non-functional requirements arise through the user needs, because of budget constraints, organizational policies, the need for interoperability with other software and hardware systems or because of external factors such as-

- Product Requirements
- Organizational Requirements
- User Requirements
- Basic Operational Requirements

In systems engineering and requirements engineering, a non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviours. This should be contrasted with functional requirements that define specific behaviour or functions. The plan for implementing non-functional requirements is detailed in the system architecture. Broadly, functional requirements define what a system is supposed to do and non-functional requirements define how a system is supposed to be. Functional requirements are usually in the form of “system shall do <requirement>”, an individual action of part of the system, perhaps explicitly in the sense of a mathematical function, a black box description input, output, process and control functional model or IPO Model. In contrast, non-functional requirements are in the form of “system shall be <requirement>”, an overall property of the system as a whole or of a particular aspect and not a specific function. The systems' overall properties commonly mark the difference between whether the development project has succeeded or failed.

Non-functional requirements of our project include-

- **Response time** - The time the system takes to load and the time for responses on any action the user does.

- **Processing time** - How long is acceptable to perform key functions or export / import data?
- **Throughput** - The number of transactions the system needs to handle must be kept in mind.
- **Storage** - The amount of data to be stored for the system to function.
- **Growth Requirements** - As the system grows it will need more storage space to keep up with the efficiency.
- **Locations of operation** - Geographic location, connection requirements and the restrictions of a local network prevail.
- **Architectural Standards** - The standards needed for the system to work and sustain.

#### 4.2.1 Product Requirements

- **Portability** - Since Python and PHP are used, the system is portable.
- **Correctness** - It follows a well-defined set of procedures and rules to compute and also rigorous testing is performed to confirm the correctness of the data.
- **Ease of Use** - The front end is designed in such a way that it provides an interface which allows the user to interact in an easy manner.
- **Modularity** - The complete product is broken up into many modules and well-defined interfaces are developed to explore the benefit of flexibility of the product.
- **Robustness** - This software is being developed in such a way that the overall performance is optimized and the user can expect the results within a limited time with utmost relevancy and correctness.

Non-functional requirements are also called the qualities of a system. These qualities can be divided into execution quality & evolution quality. Execution qualities are security & usability of the system which are observed during run time, whereas evolution quality involves testability, maintainability, extensibility or scalability.

## 4.2.2 Organizational Requirements

**Process Standards** - IEEE standards are used to develop the application which is the standard used by the most of the standard software developers all over the world.

**Design Methods** - Design is one of the important stages in the software engineering process. This stage is the first step in moving from problem to the solution domain. In other words, starting with what is needed design takes us to work how to satisfy the needs.

## 4.2.3 User Requirements

The user requirements document (URD) or user requirements specification is a document usually used to software engineering that specifies the requirements the user expects from software to be constructed in a software project. Once the required information is completely gathered it is documented in a URD, which is meant to spell out exactly what the software must do and becomes part of the contractual agreement. A customer cannot demand feature not in the URD, whilst the developer cannot claim the product is ready if it does not meet an item of the URD.

The URD can be used as a guide to planning cost, timetables, milestones, testing etc. The explicit nature of the URD allows customers to show it to various stakeholders to make sure all necessary features are described.

Formulating a URD requires negotiation to determine what is technically and economically feasible. Preparing a URD is one of those skills that lies between a science and economically feasible. Preparing a URD is one of those skills that lies between a science and an art, requiring both software technical skills and interpersonal skills.

## 4.2.4 Basic Operational Requirements

Operational requirement is the process of linking strategic goals and objectives to tactic goals and objectives. It describes milestones, conditions for success and explains how, or what portion of, a strategic plan will be put into operation during a given operational period, in the case of, a strategic plan will be put into operation during a given operational period, in the case of commercial application, a fiscal year or another given budgetary term. An operational plan is the basis for, and justification of an annual

operating budget request. Therefore, a five-year strategic plan would typically require five operational plans funded by five operating budgets.

Operational plans should establish the activities and budgets for each part of the organization for the next 1-3 years. They link the strategic plan with the activities the organization will deliver and the resources required to deliver them.

An operational plan draws directly from agency and program strategic plans to describe agency and program missions and goals, program objectives, and program activities. Like a strategic plan, an operational plan addresses four questions-

- Where are we now?
- Where do we want to be?
- How do we get there?

The customers are those that perform the eight primary functions of systems engineering, with special emphasis on the operator as the key customer. Operational requirements will define the basic need and, at a minimum, will be related to these following points-

- **Mission profile or scenario** - It describes about the procedures used to accomplish mission objective. It also finds out the effectiveness or efficiency of the system.
- **Performance and related parameters** - It points out the critical system parameters to accomplish the mission
- **Utilization environments** - It gives a brief outline of system usage. Finds out appropriate environments for effective system operation.
- **Operational life cycle** - It defines the system lifetime

### 4.3 Hardware Requirements

- Processors : Pentium IV and above
- Processor Speed : 3.00 GHZ
- RAM : 2 GB
- Storage : 20 GB

## 4.4 Software Requirements

- Operating system : Windows 7 and above
- Coding Language : Python
- Scripting language : PHP
- Tools : Pycharm
- Library : Numpy, Pandas, Sklearn, D3, Mapbox

## Summary

This chapter gives details of the functional requirements, non-functional requirements, resource requirements, hardware requirements, software requirements etc. Again, the non-functional requirements in turn contain product requirements, organizational requirements, user requirements, basic operational requirements etc.

## Chapter 5

### SYSTEM ANALYSIS

#### Overview

Analysis is the process of finding the best solution to the problem. System analysis is the process by which we learn about the existing problems, define objects and requirements and evaluates the solutions. It is the way of thinking about the organization and the problem it involves, a set of technologies that helps in solving these problems. Feasibility study plays an important role in system analysis which gives the target for design and development.

#### 5.1 Feasibility Study

All systems are feasible when provided with unlimited resource and infinite time. But unfortunately, this condition does not prevail in practical world. So it is both necessary and prudent to evaluate the feasibility of the system at the earliest possible time. Months or years of effort, thousands of rupees and untold professional embarrassment can be averted if an ill-conceived system is recognized early in the definition phase. Feasibility & risk analysis are related in many ways. If project risk is great, the feasibility of producing quality software is reduced. In this case there are three primary areas of interest-

##### 5.1.1 Performance Analysis

For the complete functionality of the project work, the project is run with the help of healthy networking environment. Performance analysis is done to find out whether the proposed system is time efficient and accurate. It is essential that the process of performance analysis and definition must be conducted in parallel.

##### 5.1.2 Technical Analysis

System is only beneficial only if it can be turned into information systems that will meet the organization's technical requirement. Simply stated, this test of feasibility asks whether the system will work or not when developed & installed, whether there are any major barriers to implementation. Regarding all these issues in technical analysis there are several points to focus on-

- **Changes to bring in the system** - All changes should be in positive direction, there will be increased level of efficiency and better customer service.
- **Required skills** - Platforms & tools used in this project are widely used. So, the skilled manpower is readily available in the industry.
- **Acceptability** - The structure of the system is kept feasible enough so that there should not be any problem from the user's point of view.

### 5.1.3 Economical Analysis

Economical analysis is performed to evaluate the development cost weighed against the ultimate income or benefits derived from the developed system. For running this system, we simply need a computer. All the features in this system run even on the other Operating Systems. So, the system is economically feasible enough.

## Summary

The main aim of this chapter is to find out whether the system is feasible enough or not. For these reasons different kinds of analysis, such as performance analysis, technical analysis, economical analysis etc. is performed.

## Chapter 6

# SYSTEM DESIGN

## Overview

Design is a meaningful engineering representation of something that is to be built. It is the most crucial phase in the developments of a system. Software design is a process through which the requirements are translated into a representation of software. Design is a place where design is fostered in software Engineering. Based on the user requirements and the detailed analysis of the existing system, the new system must be designed. This is the phase of system designing. Design is the perfect way to accurately translate a customer's requirement in the finished software product. Design creates a representation or model, provides details about software data structure, architecture, interfaces and components that are necessary to implement a system. The logical system design arrived at as a result of systems analysis is converted into physical system design.

### 6.1 System development methodology

System development method is a process through which a product will get completed or a product gets rid from any problem. Software development process is described as a number of phases, procedures and steps that gives the complete software. It follows series of steps which is used for product progress. The development method followed in this project is **Agile Development**.

#### 6.1.1 Software Development Life Cycle

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

#### Planning and Requirement Analysis-

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.



Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

### **Defining requirements-**

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an **SRS (Software Requirement Specification)** document which consists of all the product requirements to be designed and developed during the project life cycle.

### **Designing the product architecture-**

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third-party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

### **Developing the product-**

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the

code. Different high-level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

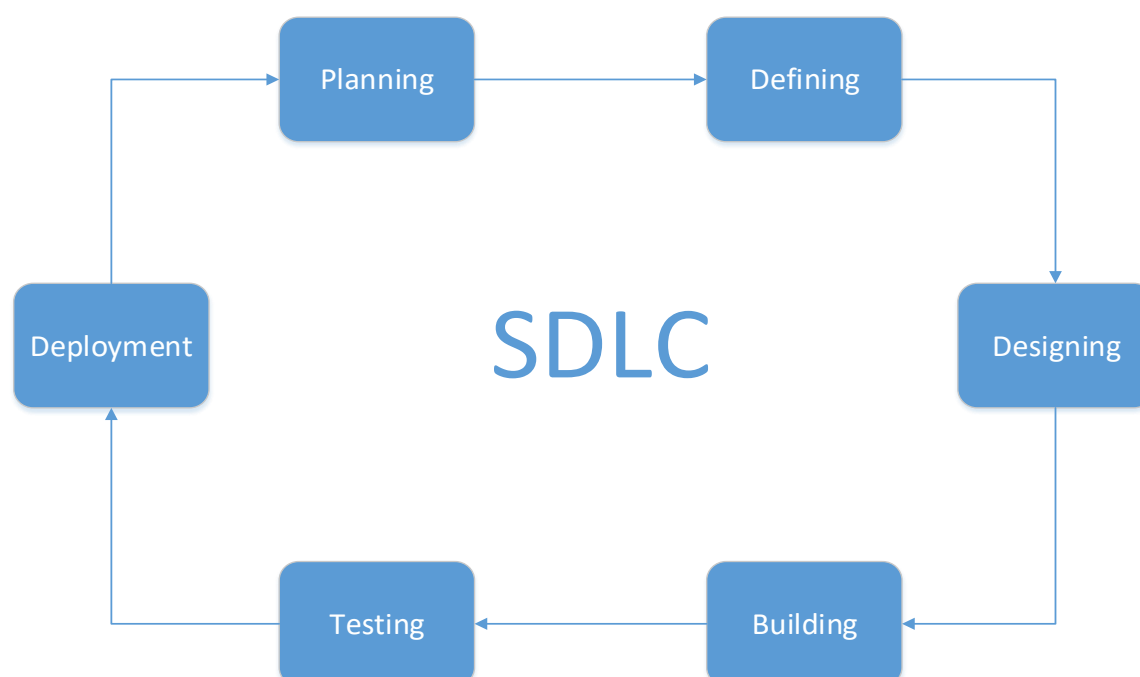
### Testing the product-

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

### Deployment and Maintenance-

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.



**Figure 6.1.1:** Software development life cycle

## 6.1.2 Agile Software Development

Agile Development is an umbrella term for several iterative and incremental software development methodologies. The most popular agile methodologies include Extreme Programming (XP), Scrum, Crystal, Dynamic Systems Development Method (DSDM), Lean Development, and Feature-Driven Development (FDD).

Most agile methods attempt to minimize risk by developing software in short timeboxes, called iterations, which typically last one to four weeks. Each iteration is like a miniature software project of its own, and includes all the tasks necessary to release the mini-increment of new functionality: planning, requirements analysis, design, coding, testing, and documentation. While iteration may not add enough functionality to warrant releasing the product, an agile software project intends to be capable of releasing new software at the end of every iteration. At the end of each iteration, the team reevaluates project priorities.

Agile methods emphasize real-time communication, preferably face-to-face, over written documents. Most agile teams are located in a bullpen and include all the people necessary to finish the software. At a minimum, this includes programmers and the people who define the product such as product managers, business analysts, or actual customers. The bullpen may also include testers, interface designers, technical writers, and management.

While each of the agile methodologies is unique in its specific approach, they all share a common vision and core values. They all fundamentally incorporate iteration and the continuous feedback that it provides to successively refine and deliver a software system. They all involve continuous planning, continuous testing, continuous integration, and other forms of continuous evolution of both the project and the software. They are all lightweight, especially compared to traditional waterfall-style processes, and inherently adaptable. What is more important about agile methods is that they all focus on empowering people to collaborate and make decisions together quickly and effectively.

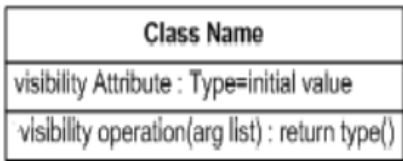

The Manifesto for Agile Software Development is based on twelve principles-











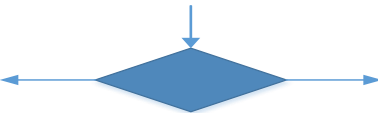
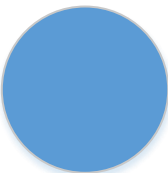
1. Customer satisfaction by early and continuous delivery of valuable software
2. Welcome changing requirements, even in late development



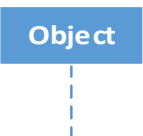
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the principal measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity-the art of maximizing the amount of work not done is essential
11. Best architectures, requirements, and designs emerge from self-organizing teams
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly.

## 6.2 Design Using UML

Designing UML diagram specifies, how the process within the system communicates along with how the objects within the process collaborate using both static as well as dynamic UML diagrams since in this ever-changing world of Object Oriented application development, it has been getting harder and harder to develop and manage high quality applications in reasonable amount of time. As a result of this challenge and the need for a universal object modeling language every one could use, the Unified Modeling Language (UML) is the Information industries version of blue print. It is a method for describing the systems architecture in detail. Easier to build or maintains system, and to ensure that the system will hold up to the requirement changes.

Sl. No	Symbol Name	Symbol	Description
1	Class		Classes represent a collection of similar entities grouped together.
2	Association		Association represents a static relation between classes.

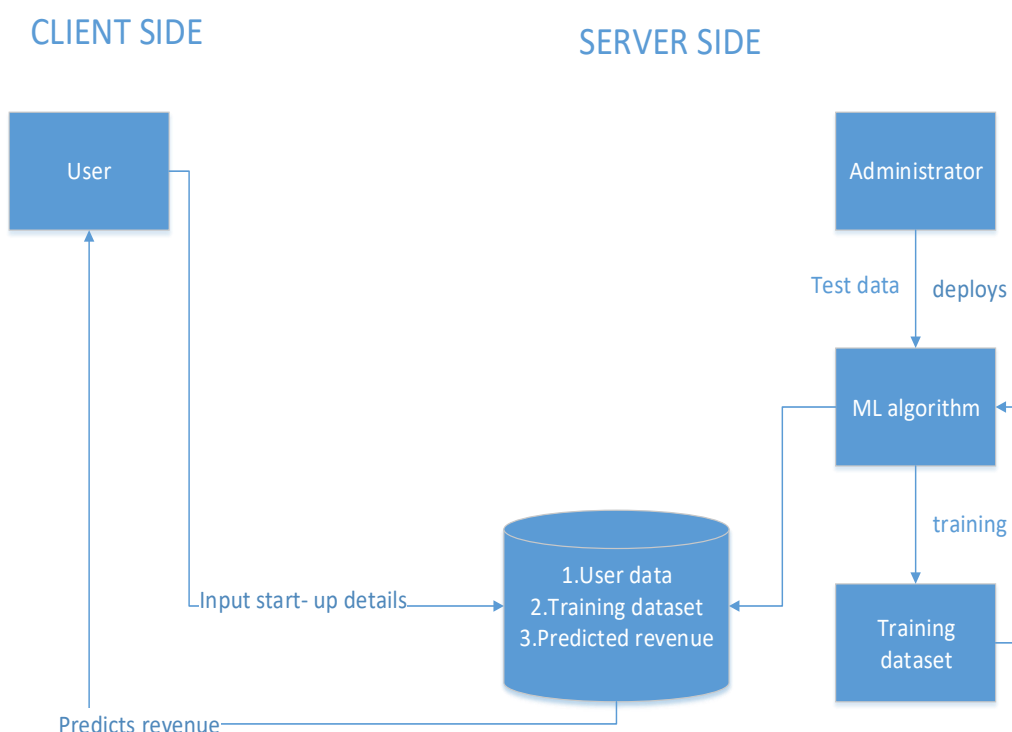
3	Aggregation		Aggregation is a form of association. It aggregates several classes into a single class.
4	Composition		Composition is a special type of aggregation that denotes a strong ownership between classes.
5	Actor		Actor is the user of the system and other external entity that react with the system.
6	Use Case		A use case is an interaction between system and the external environment.
7	Relation (Uses)		It is used for additional purpose communication.
8	Communication		It is the communication between use cases.
9	State		It represents the state of process. Each state goes through various flows.
10	Initial State		It represents initial state of object.
11	Final State		It represents final state of object.
12	Control Flow		It represents decision making process for object.
13	Decision Box		It represents the decision-making process from a constraint.
14	Data Process/ State		A circle in a DFD represents a state or process which has been triggered due to some other event or action.

15	Message		It represents messages exchanged.
16	Transition		It represents any communication that occurs between processes.
17	Object Lifeline		Object lifeline represents the vertical dimension that object communicates.

**Table 6.2: Symbols used in UML**

### 6.2.1 Architectural Design

The overall logical structure of the project is divided into processing modules and a conceptual data structure is defined as Architectural Design.



**Figure 6.2.1: System Architecture**

### 6.2.2 Data Flow Diagram

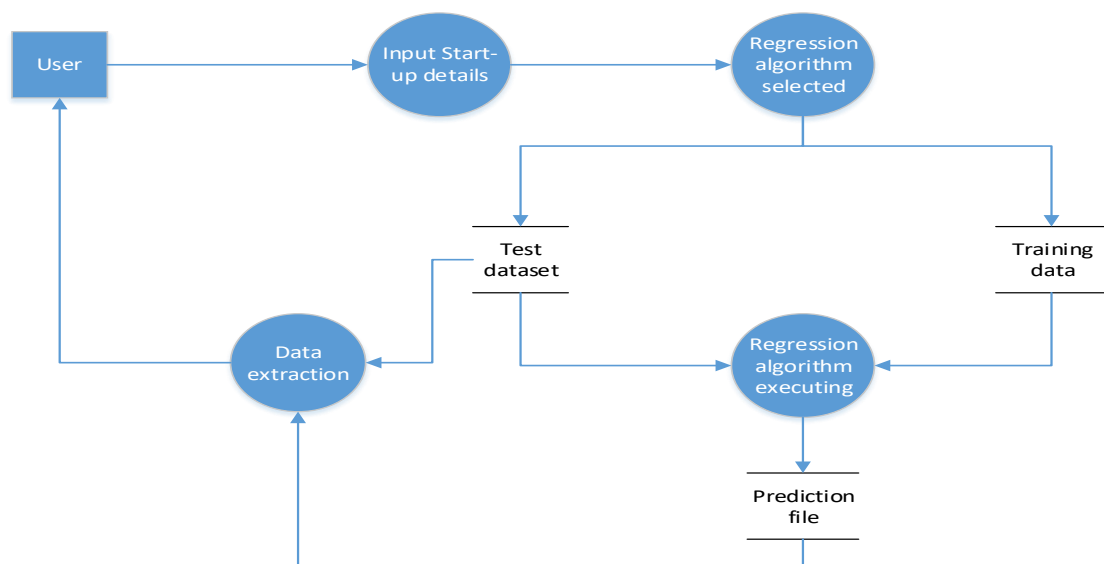
A data flow diagram (DFD) is graphic representation of the "flow" of data through an information system. A data flow diagram can also be used for the visualization of data

processing (structured design). It is common practice for a designer to draw a context-level DFD first which shows the interaction between the system and outside entities. DFD's show the flow of data from external entities into the system, how the data moves from one process to another, as well as its logical storage. There are only four symbols-

1. Squares representing **external entities**, which are sources and destinations of information entering and leaving the system.
2. Rounded rectangles representing **processes**, in other methodologies, may be called 'Activities', 'Actions', 'Procedures', 'Subsystems' etc. which take data as input, do processing to it, and output it.
3. Arrows representing the **data flows**, which can either, be electronic data or physical items. It is impossible for data to flow from data store to data store except via a process, and external entities are not allowed to access data stores directly.
4. The flat three-sided rectangle is representing **data stores** should both receive information for storing and provide it for further processing.



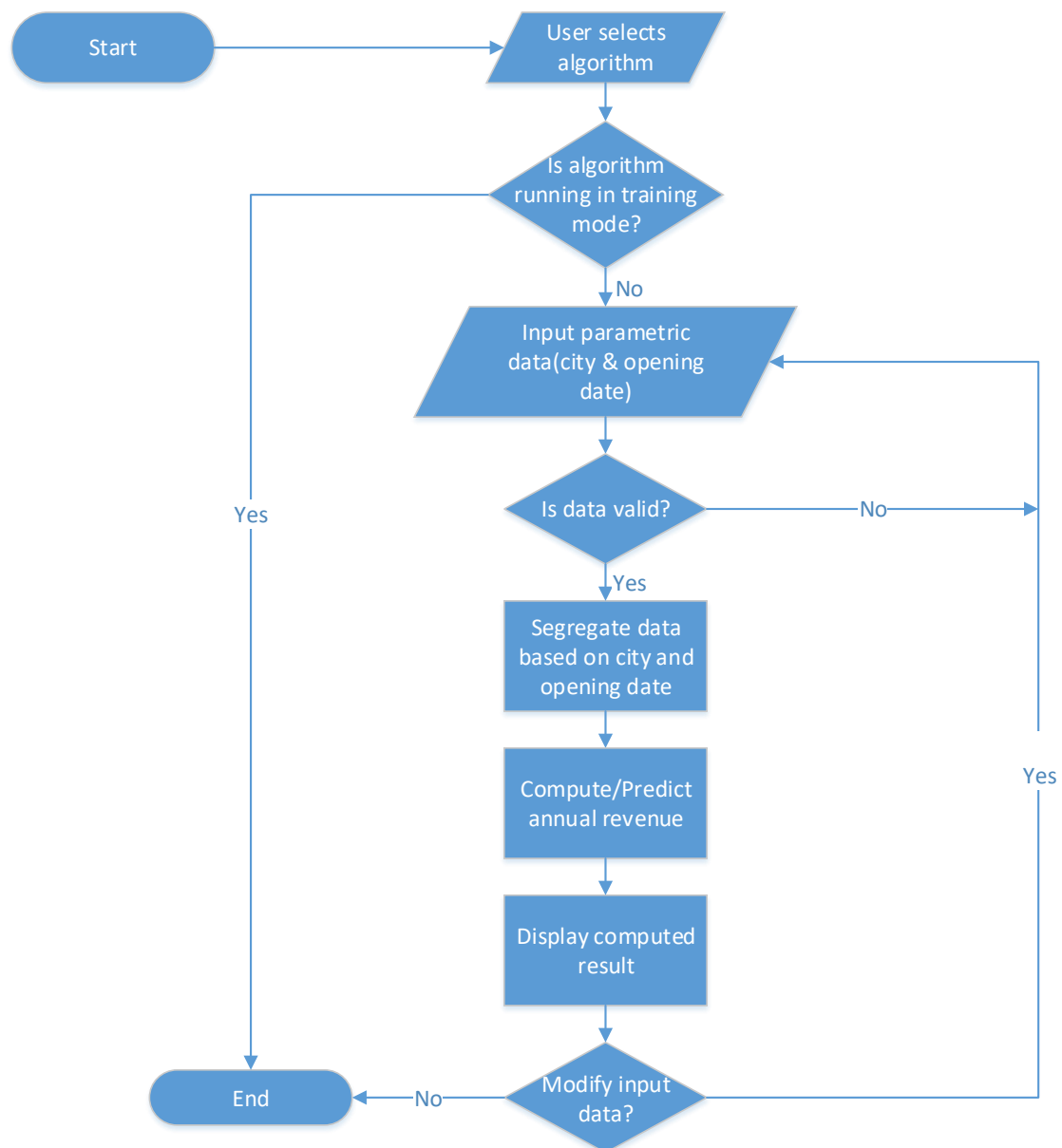
**Figure 6.2.2.1: Level 0 Data Flow Diagram**



**Figure 6.2.2.2: Level 1 Data flow diagram**

### 6.2.3 Flow Chart

A **flowchart** is a type of diagram that represents an algorithm, workflow or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows. This diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

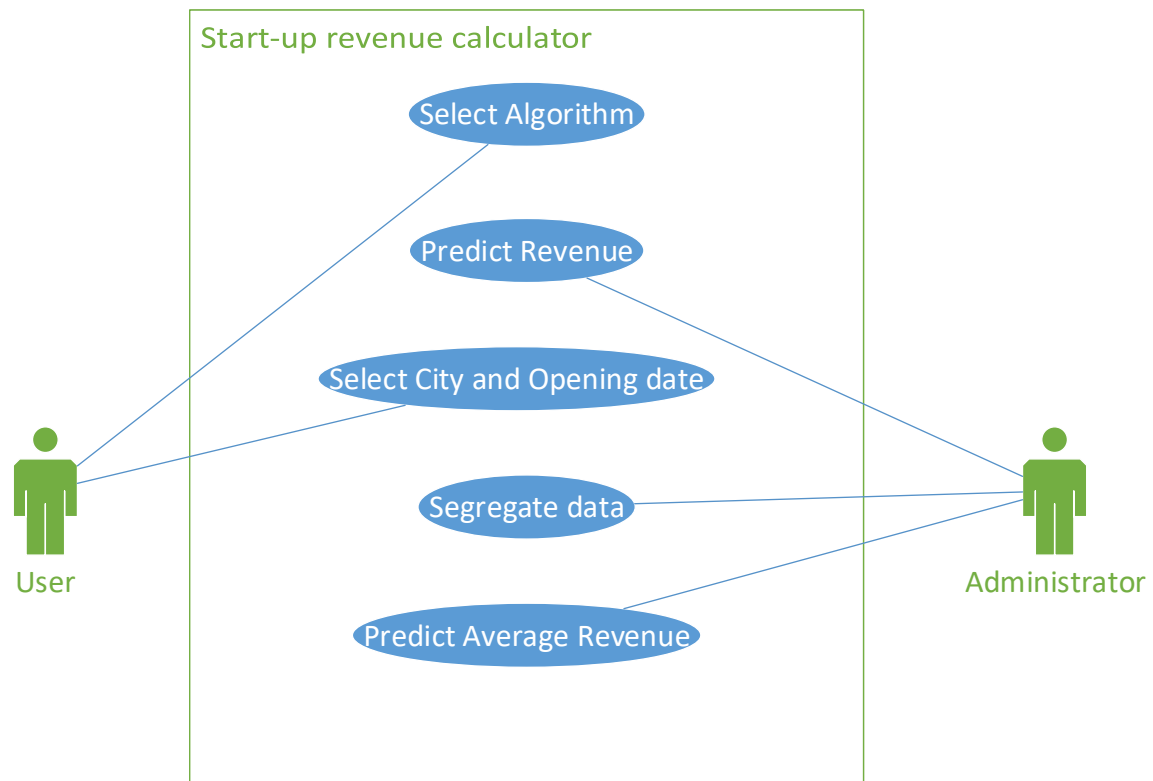


**Figure 6.2.3:** Flowchart



## 6.2.4 Use Case Diagram

A use case defines a goal-oriented set of interactions between external entities and the system under consideration. The external entities which interact with the system are its actors. A set of use cases describe the complete functionality of the system at a particular level of detail and it can be graphically denoted by the use case diagram.



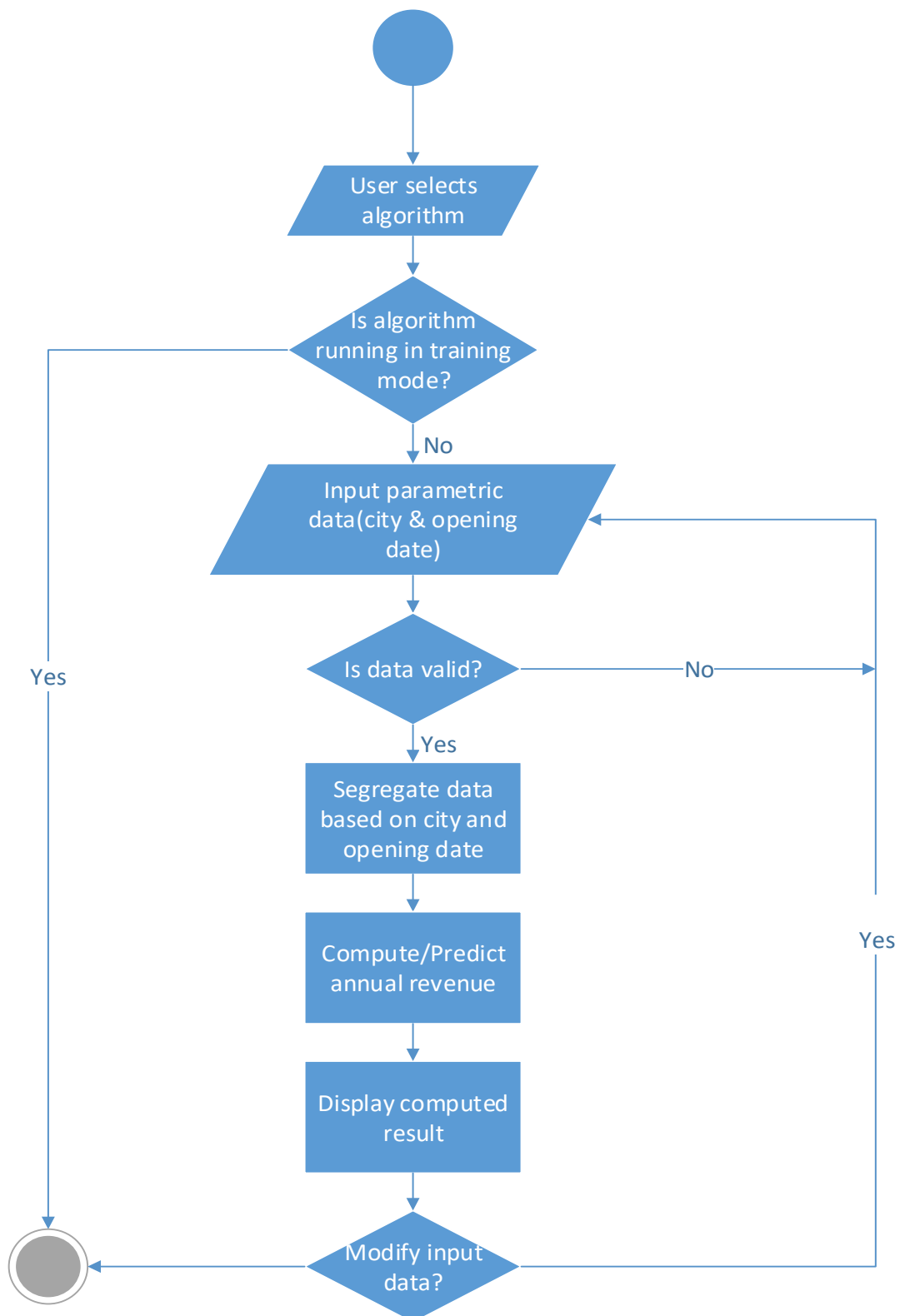
**Figure 6.2.4:** Use Case diagram

## 6.2.5 Activity Diagram

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e. workflows). Activity diagrams show the overall flow of control.

Rounded rectangles represent actions, diamonds represent decisions, bars represent the start or end of concurrent activities, a solid circle represents the start (initial node) of the workflow and an encircled solid circle represents end (final node).

Activity diagrams are constructed from a limited number of shapes, connected with arrows.



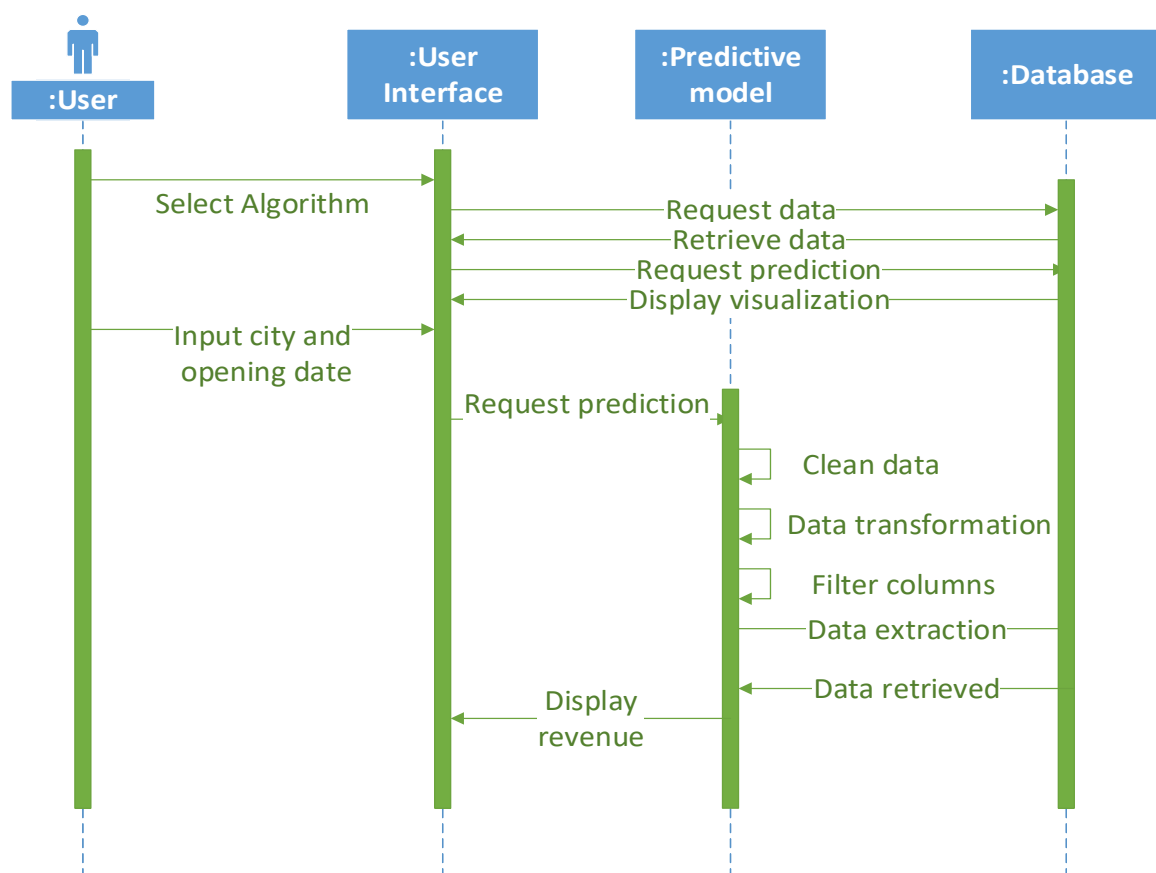
**Figure 6.2.5:** Activity Diagram

## 6.2.6 Sequence Diagram

Sequence diagram are an easy and intuitive way of describing the behavior of a system by viewing the interaction between the system and the environment. A sequence diagram shows an interaction arranged in a time sequence. A sequence diagram has two dimensions- vertical dimension represents time, the horizontal dimension represents the objects existence during the interaction.

### Basic elements-

- Vertical rectangle- represent the object is active (method is being performed).
- Vertical dashed line- represent the life of the object.
- "X"- represent the life end of an object. (Being destroyed from memory)
- Horizontal line with arrows- messages from one object to another.



**Figure 6.2.6:** Sequence Diagram

### 6.2.7 Class Diagram

UML class diagram shows the static structure of the model. The class diagram is a collection of static modeling elements, such as classes and their relationships', connected as a graph to each other and to their contents.

The class diagram is the main building block of object oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects and or interactions in the application and the objects to be programmed.

**Fig 6.2.7:** Class diagram

## Summary

This chapter mainly concentrates on few fundamental design concepts such as system development methodology, system architecture, class diagram, flowchart, sequence diagram, use-case diagram, activity diagram, data flow diagram etc.

## Chapter 7

### IMPLEMENTATION

The implementation phase of the project is where the detailed design is actually transformed into working code. Aim of the phase is to translate the design into a best possible solution in a suitable programming language. This chapter covers the implementation aspects of the project, giving details of the programming language and development environment used. It also gives an overview of the core modules of the project with their step by step flow.

The implementation stage requires the following tasks.

- Careful planning.
- Investigation of system and constraints.
- Design of methods to achieve the changeover.
- Evaluation of the changeover method.
- Correct decisions regarding selection of the platform
- Appropriate selection of the language for application development

#### 7.1 Introduction to Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, makes it attractive for rapid application development as well as for use as a scripting language. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code re-use. The python interpreter and extensive standard library are available in source or binary form without charge for all major platforms, can be freely distributed.

##### 7.1.1 Features of Python Programming

- **A simple language which is easier to learn**-Python has a very simple and elegant syntax. It's much easier to read and write Python programs compared to other languages like C++, Java, C#. Python makes programming fun and allows you to focus on the solution rather than syntax.

- **Free and open-source**-It can be freely used and distributed, even for commercial use. Python has a large community constantly improving it in each iteration.
- **Portability**-Python programs can be moved from one platform to another, and can be executed without any changes. It runs seamlessly on almost all platforms including Windows, Mac OS X and Linux.
- **Extensible and Embeddable**-Suppose an application requires high performance. Python code can be easily combined with pieces of C/C++ or other languages. This will give the application high performance as well as scripting capabilities which other languages may not provide out of the box.
- **A high-level, interpreted language**-Unlike C/C++, daunting tasks like memory management, garbage collection and so on don't have to be taken care of. Likewise, when Python code is run, it automatically converts the code to the language the computer understands.
- **Large standard libraries to solve common tasks**-Python has a number of standard libraries which makes life of a programmer much easier. Standard libraries in Python are well tested and used by hundreds of people.
- **Object-oriented**-Everything in Python is an object. Object oriented programming (OOP) helps to solve a complex problem intuitively. With OOP, complex problems are divided into smaller sets by creating objects.

## 7.2 Modules in implementation

This project mainly implements three main modules (i.e.) Input, Processing data using Machine Learning Algorithm and Output. In this section, functions used for implementing each of the modules are discussed. Therefore, the functions used to implement these modules are described as follows:

- Functions used for Input
- Functions used for Processing data using Machine Learning Algorithm
- Functions used for Output

### 7.2.1 Functions used for input

- `Pandas.read_csv ( filepath, sep, dtype, converters)`

**File path-** The string could be URL. Valid URL schemes include http, ftp and file. For file URLs, a host is expected.

**Sep-** Delimiter to use.

**Dtype-** data types for data or columns.

**Converters-** Dictionary of functions for converting values in certain columns. Keys can be either integers or column labels.

- `Open ( filepath, mode)`

**File path-** The string could be URL. Valid URL schemes include http, ftp and file. For file URLs, a host is expected.

**Mode-** The file is opened in write mode.

### 7.2.2 Functions used for processing data, training and testing the model(machine learning algorithm)

- `Sklearn.preprocessing.StandardScalar()-` Standardize features by removing the mean and scaling to unit variance.

Methods-

**Fit\_transform()-** Fit to data, then transform it.

- `GradientBoostingRegressor (n_estimators, max_depth, random_state, loss, learning_rate)`

**N\_estimators-** The number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance.

**Max\_depth-** maximum depth of the individual regression estimators. The maximum depth limits the number of nodes in the tree. Tune this parameter for best performance; the best value depends on the interaction of the input variables.

**Random\_state-** If int, random\_state is the seed used by the random number generator.

**Loss-** loss function to be optimized. Huber is the combination of least squares regression and least absolute deviation.

**Learning\_rate-** learning rate shrinks the contribution of each tree by learning\_rate. There is a trade-off between learning\_rate and n\_estimators.

- RandomForestRegressor (n\_estimators, max\_depth, random\_state, oob\_score, learning\_rate)

**N\_estimators-** The number of trees in the forest.

**Max\_depth-** Maximum depth of the tree. The maximum depth limits the number of nodes in the tree.

**Random\_state-** If int, random\_state is the seed used by the random number generator.

**Learning\_rate-** learning rate shrinks the contribution of each tree by learning\_rate. There is a trade-off between learning\_rate and n\_estimators.

**Oob\_score-** Boolean value, optional (default=False). It is used to specify whether to use out-of-bag samples to estimate the  $R^2$  on unseen data.

- Map(value)- map the categorical values in the dataset to numerical values
- Model.fit(training, test)- fit the gradient boosting and random forest regressor model.

### 7.2.3 Functions used for output

- Out=open (filepath, mode)

**File path-** The string could be URL. Valid URL schemes include http, ftp and file. For file URLs, a host is expected.

**Mode-** The file is opened in write mode.

- Out.write()- To the output file, prediction calculated by regression algorithm is written



## Summary

The chapter discusses the implementation details of the different modules of the system and gives the step by step flow of each of them. Along with these, this chapter also highlights some of the important features of the platform and language used for implementation purpose.

## Chapter 8

### TESTING

System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that all the system elements have been properly integrated and perform allocated functions. The testing process is actually carried out to make sure that the product exactly does the same thing what is supposed to do. In the testing stage following goals are tried to achieve-

- To affirm the quality of the project.
- To find and eliminate any residual errors from previous stages.
- To validate the software as a solution to the original problem.
- To provide operational reliability of the system.

#### 8.1 Testing Methodologies

There are many different types of testing methods or techniques used as part of the software testing methodology. Some of the important testing methodologies are-

##### 8.1.1 White box testing

White box testing (clear box testing, glass box testing, and transparent box testing or structural testing) uses an internal perspective of the system to design test cases based on internal structure. It requires programming skills to identify all paths through the software. The tester chooses test case inputs to exercise paths through the code and determines the appropriate outputs. While white box testing is applicable at the unit, integration and system levels of the software testing process, it is typically applied to the unit. While it normally tests paths within a unit, it can also test paths between units during integration, and between subsystems during a system level test.

Though this method of test design can uncover an overwhelming number of test cases, it might not detect unimplemented parts of the specification or missing requirements, but one can be sure that all paths through the test object are executed. Using white box testing we can derive test cases that-

- Guarantee that all independent paths within a module have been exercised at least once.
- Exercise all logical decisions on their true and false sides.

- Execute all loops at their boundaries and within their operational bounds.
- Execute internal data structure to assure their validity

### 8.1.2 Black box testing

Black box testing focuses on the functional requirements of the software. It is also known as functional testing. It is a software testing technique whereby the internal workings of the item being tested are not known by the tester. For example, in a black box test on software design the tester only knows the inputs and what the expected outcomes should be and not how the program arrives at those outputs.

The tester does not ever examine the programming code and does not need any further knowledge of the program other than its specifications. It enables us to derive sets of input conditions that will fully exercise all functional requirements for a program. Black box testing is an alternative to white box technique. Rather it is a complementary approach that is likely to uncover a different class of errors in the following categories-

- Incorrect or missing function.
- Interface errors.
- Performance errors.
- Initialization and termination errors.
- Errors in objects.

#### Advantages

- The test is unbiased as the designer and the tester are independent of each other.
- The tester does not need knowledge of any specific programming languages.
- The test is done from the point of view of the user, not the designer.
- Test cases can be designed as soon as the specifications are complete.

## 8.2 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.

### 8.2.1 Functions for Data Processing and Revenue Prediction

**Table 8.2.1: Functions for Data Processing and Revenue Prediction**

Function	Tests done	Remarks
Map(value)	Tested to check if categorical values in the dataset are mapped to numerical values	Success
Sklern.preprocessing. StandardScalar.fit_transform()	Tested to check if features are standardized and if data is fitted and transformed.	Success
GradientBoostingRegressor (n_estimators, max_depth, random_state, loss, learning_rate)	Tested to check if a regression tree is fit on the negative gradient of the given loss function in each stage	Success
RandomForestRegressor (n_estimators, max_depth, random_state, oob_score, learning_rate)	Tested to check if a number of classifying decision trees on various sub-samples of the dataset is fit.	Success
Model.fit (training, test)	Tested to check if gradient boosting and random forest regressor models are fitted.	Success

## 8.2.2 Functions for Input

**Table 8.2.2: Functions for Input**

Function	Tests done	Remarks
Open (filepath, mode)	Tested to check if specified csv file opens in the specified mode.	Success
Pandas.read_csv (filepath, sep, dtype, converters)	Tested to check if values are read from the csv file as well as if the values in certain columns are converted	Success

## 8.2.3 Functions for Output

**Table 8.2.3: Functions for Output**

Function	Tests done	Remarks
open (filepath, mode)	Tested to check if specified csv file opens in the specified mode.	Success
Out.write()	Tested to check if prediction is written to the specified csv file.	Success

## 8.3 Integration Testing

Upon completion of unit testing, integration testing begins. Individual modules are combined and tested as a group. Integration testing is black box testing. The purpose of integration testing is to ensure distinct components of the application still work in accordance to user requirements. Integration testing is considered complete, when actual results and expected results are either in line or differences are explainable based on client input. It concentrates on data transfer between modules. Integration testing is a logical extension of unit testing. Two units that have already been tested are combined into a component and the interface between them is tested. Integration testing identifies problems that occur when units are combined. The errors that arise can be attributed to those occurring due to the combination of modules, resulting from errors across interface.

The Integration Testing shows the functions that are combined into different classes and the module as a whole tested for its functionality. Finally, all the modules are integrated and tested. This is important to check for error-free interaction between various classes and its modules.

**Table 8.3: Integration Testing**

Modules	Functions integrated	Tests done
Input Module	<ul style="list-style-type: none"> <li>Function used to open csv file</li> <li>Function used to read csv file</li> </ul>	Tested the functions of Input Module.
Data Processing and Prediction Module	<ul style="list-style-type: none"> <li>Function used to map columns in the dataset to numerical values</li> <li>Function used to fit data</li> <li>Functions used to train the model.</li> <li>Functions used to test the model.</li> </ul>	Tested the functions of Data Processing, and Prediction Module.
Output Module	<ul style="list-style-type: none"> <li>Function used to open csv file</li> <li>Function used in write to csv file</li> </ul>	Tested the functions of Output Module.

## 8.4 System Testing

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

As a rule, system testing takes, as its input, all of the "integrated" software components that have passed integration testing and also the software system itself integrated with any applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called assemblages) or between any of the assemblages and the hardware. System testing is a more limited type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

System testing is performed on the entire system in the context of a Functional Requirement Specification(s) (FRS) and/or a System Requirement Specification (SRS).

System testing tests not only the design, but also the behavior and even the believed expectations of the customer. It is also intended to test up to and beyond the bounds defined in the software/hardware requirements specification(s).

The following examples are different types of testing that should be considered during System testing:

- Graphical user interface testing
- Usability testing
- Software performance testing
- Compatibility testing
- Exception handling
- Load testing
- Volume testing

Although different testing organizations may prescribe different tests as part of System testing, this list serves as a general framework or foundation to begin with.

## 8.5 Quality Assurance

Quality assurance consists of the auditing and reporting functions of management. The goal of quality assurance is to provide management with the data necessary to be informed about product quality, thereby gaining insight and confident that the product quality is meeting its goals. This is an —umbrella activityl that is applied throughout the engineering process. Software quality assurance encompasses-

- Analysis, design, coding and testing methods and tools
- Formal technical reviews that are applied during each software engineering
- Multitier testing strategy
- Control of software documentation and the change made to it
- A procedure to ensure compliance with software development standards.
- Measurement and reporting mechanisms.

Quality Assurance (QA) is a way of preventing mistakes or defects in manufactured products and avoiding problems when delivering solutions or services to customers. QA is applied to physical products in pre-production to verify what will be made meets specifications and requirements, and during manufacturing production runs by validating lot samples meet specified quality controls. QA is also applied to software to verify that features and functionality meet business objectives, and that code is relatively bug free prior to shipping or releasing new software products and versions. Quality Assurance refers to administrative and procedural activities implemented in a quality system so that requirements and goals for a product, service or activity will be fulfilled. It is the systematic measurement, comparison with a standard, monitoring of processes and an associated feedback loop that confers error prevention. This can be contrasted with quality control, which is focused on process output.

Two principles included in Quality Assurance are- "Fit for purpose", the product should be suitable for the intended purpose; and "Right first time", mistakes should be eliminated. QA includes management of the quality of raw materials, assemblies, products and components, services related to production, and management, production and inspection processes. Suitable quality is determined by product users, clients or customers, not by society in general. It is not related to cost, and adjectives or descriptors such as "high" and "poor" are not applicable. For example, a low priced product may be viewed as having high quality because it is disposable, where another may be viewed as having poor quality because it is not disposable.

Software quality assurance (SQA) consists of a means of monitoring the software engineering processes and methods used to ensure quality. The methods by which this is accomplished are many and varied, and may include ensuring conformance to one or more standards, such as ISO 9000 or a model such as CMMI. SQA encompasses the entire software development process, which includes processes such as requirements definition, software design, coding, source code control, code reviews, software configuration management, testing, release management, and product integration.

### **8.5.1 Quality Factors**



An important objective of quality assurance is to track the software quality and assess the impact of methodological and procedural changes on improved software quality. The factors that affect the quality can be categorized into two broad groups-

- Factors that can be directly measured.
- Factors that can be indirectly measured

These factors focus on three important aspects of a software product

- Its operational characteristics
- Its ability to undergo changes
- Its adaptability to a new environment.
- Effectiveness or efficiency in performing its mission
- Duration of its use by its customer.

In the context of software engineering, software quality refers to two related but distinct notions that exist wherever quality is defined in a business context- Software functional quality reflects how well it complies with or conforms to a given design, based on functional requirements or specifications. That attribute can also be described as the fitness for purpose of a piece of software or how it compares to competitors in the marketplace as a worthwhile product; Software structural quality refers to how it meets non-functional requirements that support the delivery of the functional requirements, such as robustness or maintainability, the degree to which the software was produced correctly.

Structural quality is evaluated through the analysis of the software inner structure, its source code, at the unit level, the technology level and the system level, which is in effect how its architecture adheres to sound principles of software architecture outlined in a paper on the topic by OMG. In contrast, functional quality is typically enforced and measured through software testing. Historically, the structure, classification and terminology of attributes and metrics applicable to software quality management have been derived or extracted from the ISO 9126-3 and the subsequent ISO 25000:2005 quality model, also known as SQuaRE. Based on these models, the Consortium for IT Software Quality (CISQ) has defined five major desirable structural characteristics needed for a piece of software to provide business value- Reliability, Efficiency, Security, Maintainability and (adequate) Size.

Software quality measurement quantifies to what extent a software or system rates along each of these five dimensions. An aggregated measure of software quality can be computed through a qualitative or a quantitative scoring scheme or a mix of both and then a weighting system reflecting the priorities. This view of software quality being positioned on a linear continuum is supplemented by the analysis of "critical programming errors" that under specific circumstances can lead to catastrophic outages or performance degradations that make a given system unsuitable for use regardless of rating based on aggregated measurements.

Such programming errors found at the system level represent up to 90% of production issues, whilst at the unit-level, even if far more numerous, programming errors account for less than 10% of production issues. As a consequence, code quality without the context of the whole system, as W. Edwards Deming described it, has limited value.

To view, explore, analyze, and communicate software quality measurements, concepts and techniques of information visualization provide visual, interactive means useful, in particular, if several software quality measures have to be related to each other or to components of a software or system.

## Chapter 9

### RESULTS AND EXECUTION

The following snapshots define the results or outputs that are obtained after step by step execution of the system for different inputs.

#### 9.1 Snapshots

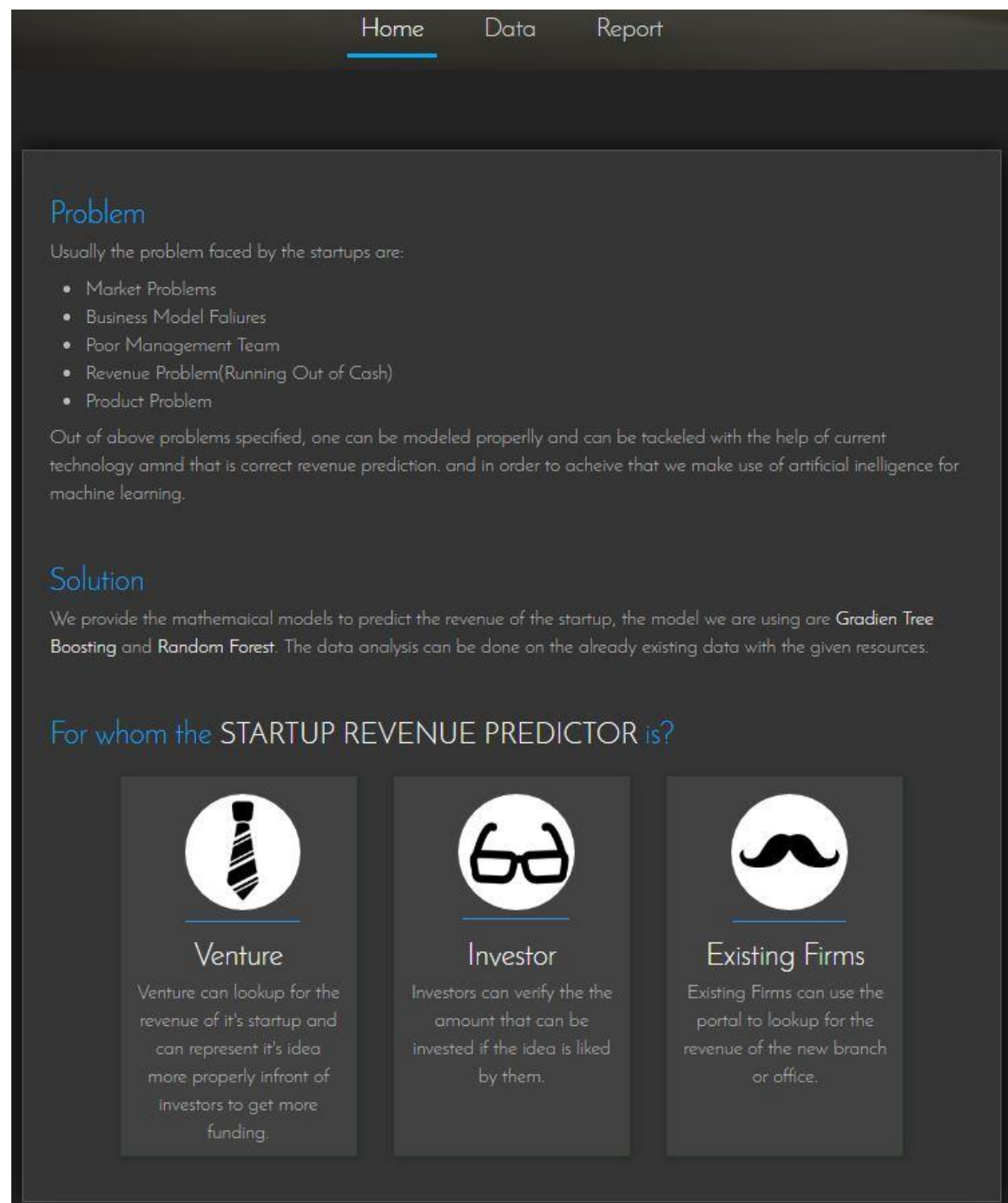
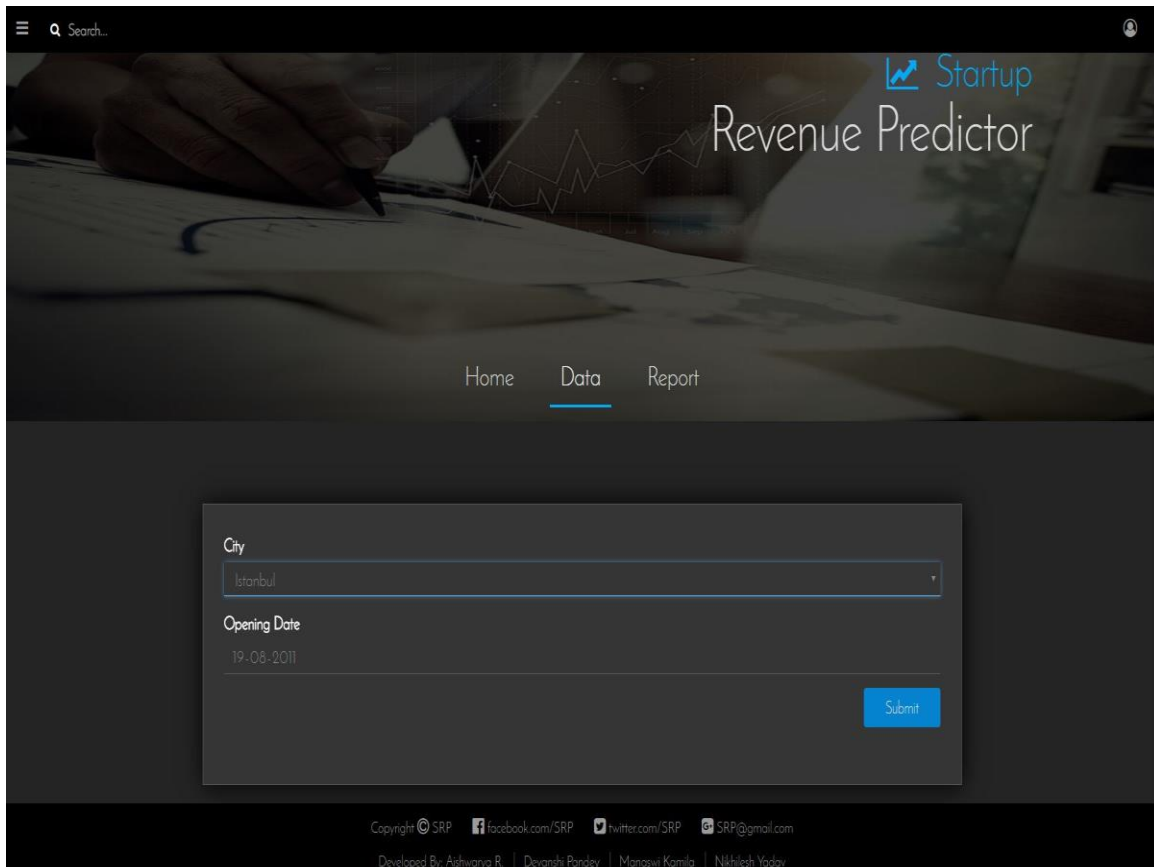
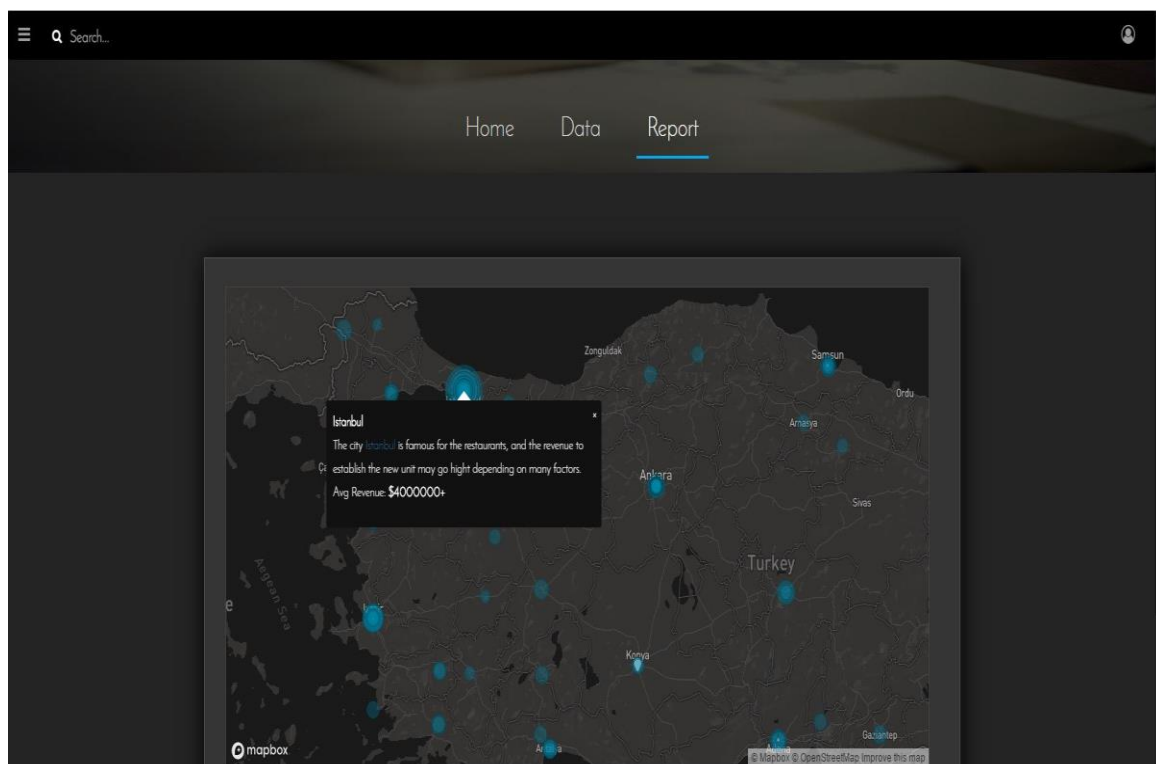


Figure 9.1.1: Home page

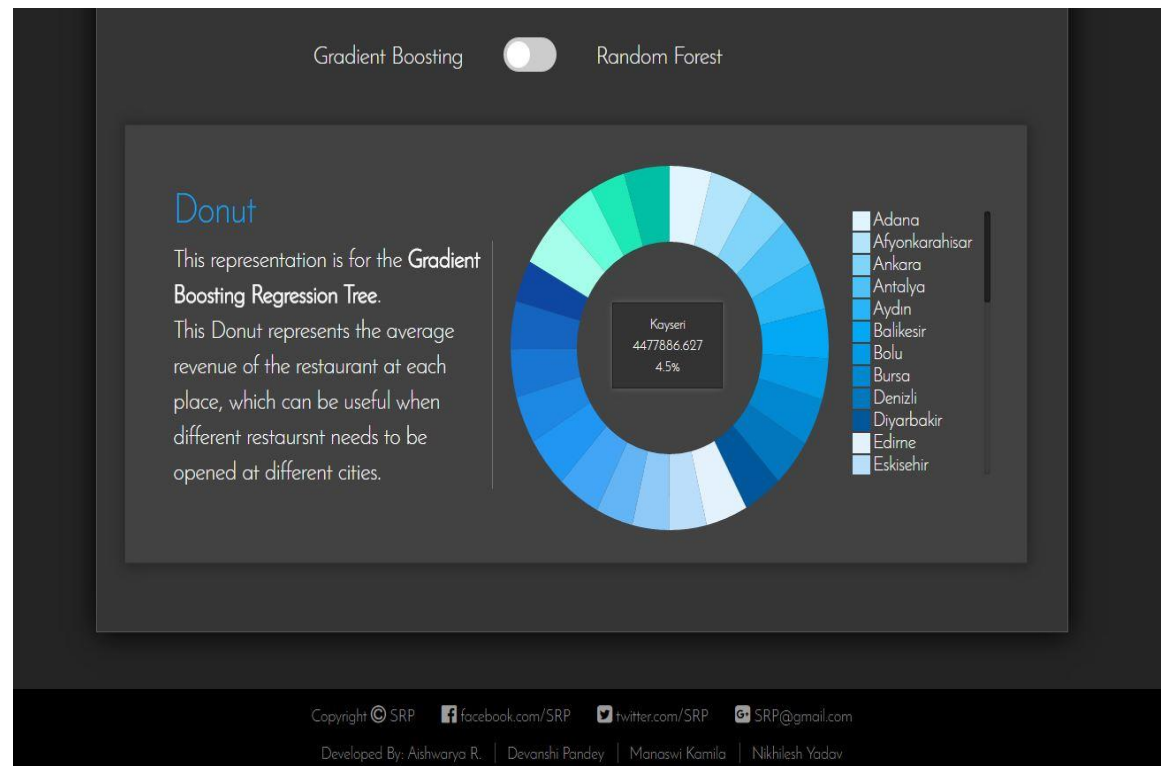


The screenshot shows the 'Data' tab of the 'Startup Revenue Predictor' application. The form contains two input fields: 'City' with 'Istanbul' selected and 'Opening Date' with '19-08-2011' entered. A blue 'Submit' button is located at the bottom right of the form. The footer includes copyright information for SRP, social media links for Facebook, Twitter, and Email, and a list of developers: Ashwanya R., Devanshi Pandey, Manasvi Kamila, and Nikhlesh Yadav.

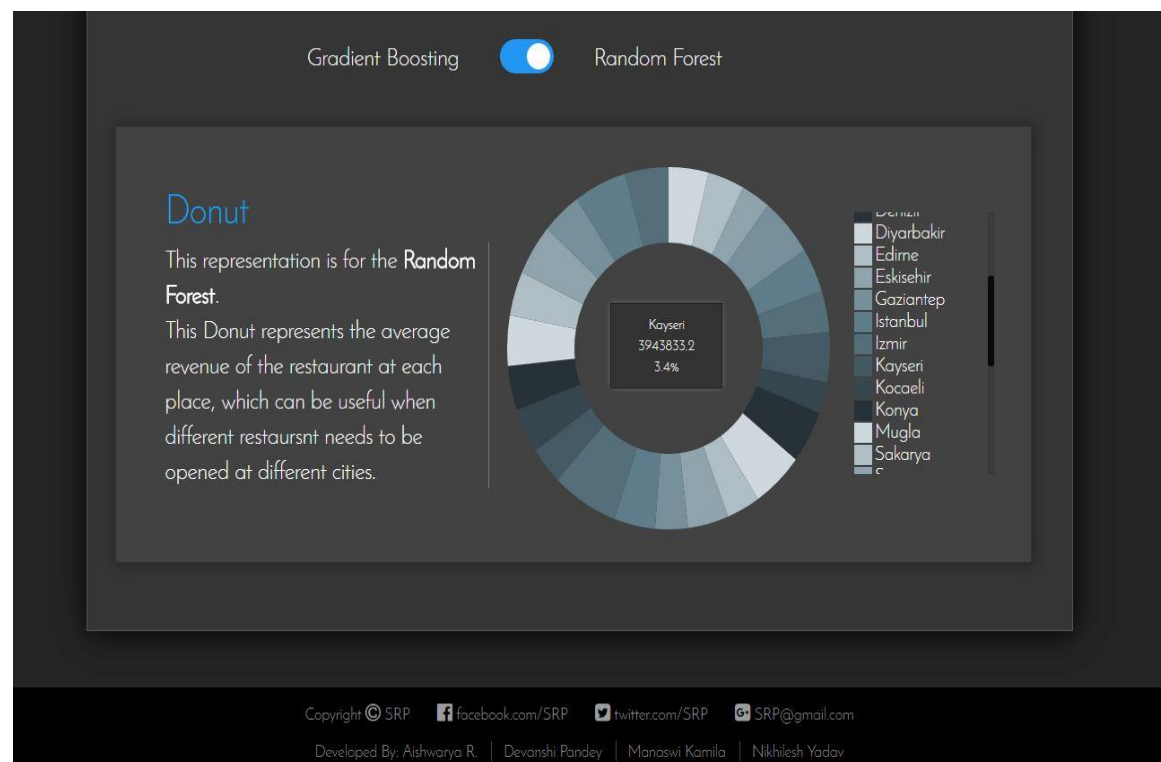
**Figure 9.1.2:** Input start-up details



**Figure 9.1.3:** Map representation of revenues based on city



**Figure 9.1.4:** Donut representation for average revenue generated using gradient boosting regressor



**Figure 9.1.5:** Donut representation for average revenue generated using random forest regressor

## Chapter 10

### CONCLUSION

Thus, the concept of prediction system for future revenues of new start-ups can be developed. Random forest and Gradient boosting decision tree were used to predict the annual revenue. Using this, a reference can be provided to aid human judgement and operational losses can be minimized for start-ups.