Low Level Design

Store Sales Prediction

|  |  |
| --- | --- |
| Written By | Surya Prakash S |
| Document Version | 1.0 |
| Last Revised Date | 29-05-2023 |

**Document Control**

### Change Record:

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comments** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### Reviews:

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Reviewer** | **Comments** |
|  |  |  |  |

### Approval Status:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Review**  **Date** | **Reviewed By** | **Approved By** | **Comments** |
|  |  |  |  |  |

Contents

[1. Introduction 1](#_Toc136246698)

[1.1. What is Low-Level design document? 1](#_Toc136246699)

[1.2. Scope 1](#_Toc136246700)

[2. Architecture 2](#_Toc136246701)

[3. Architecture Description 3](#_Toc136246702)

[3.1. Data Description 3](#_Toc136246703)

[3.2. Data Gathering 3](#_Toc136246704)

[3.3. Importing CSV 3](#_Toc136246705)

[3.4. Data Preprocessing 4](#_Toc136246706)

[3.5. Feature Engineering 4](#_Toc136246707)

[3.6. Parameter Tuning 4](#_Toc136246708)

[3.7. Model Building 5](#_Toc136246709)

[3.8. Model Saving 5](#_Toc136246710)

[3.9. Django Setup for Data Extraction 5](#_Toc136246711)

[3.10. Push to Github 5](#_Toc136246712)

[3.11. Deployment 5](#_Toc136246713)

[4. Unit Test Cases 6](#_Toc136246714)

# Introduction

## What is Low-Level design document?

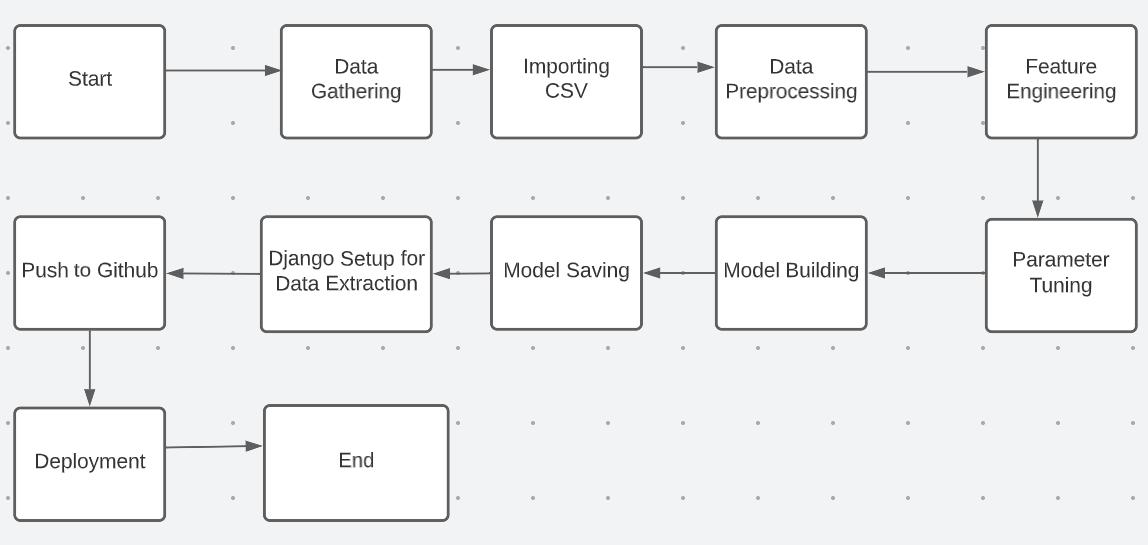
The goal of LLD or a low-level design document (LLDD) is to give the internal logical design of the actual program code for Store Sales Prediction. LLD describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer can directly code the program from the document.

## Scope

Low-level design (LLD) is a component-level design process that follows a step-by-

step [refinement](https://en.wikipedia.org/wiki/Refinement_(computing)) process. This process can be used for designing data structures, required software architecture, source code and ultimately, performance algorithms. Overall, the data organization may be defined during requirement analysis and then refined during data design work.

# Architecture



# Architecture Description

## Data Description

The provided information includes the name of the variable, its type, the unit of measurement, and a brief description. The concrete compressive strength is the main focus of the regression problem. The sequence of the listed items aligns with the numerical order found in the rows of the database.

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Measurement** |
| Item\_Identifier | String | Unique product ID |
| Item\_Weight | Float | Weight of product |
| Item\_Fat\_Content | String | Whether the product is low fat or not |
| Item\_Visibility | Float | The % of a total display area of all products in a store allocated to the particular product |
| Item\_Type | String | The category to which the product belongs |
| Item\_MRP | Float | Maximum Retail Price (list price) of the product |
| Outlet\_Identifier | String | Unique store ID |
| Outlet\_Establishment\_Year | Integer | The year in which the store was established |
| Outlet\_Size | String | The size of the store in terms of ground area covered |
| Outlet\_Location\_Type | String | The type of city in which the store is located |
| Outlet\_Type | String | Whether the outlet is just a grocery store or some sort of supermarket |
| Item\_Outlet\_Sales | Float | Sales of the product in the particular store. This is the outcome variable to be predicted. |
| Outlet\_Size | String | The size of the store in terms of ground area covered |
| Outlet\_Location\_Type | String | The type of city in which the store is located |
| Outlet\_Type | String | Whether the outlet is just a grocery store or some sort of supermarket |
| Item\_Outlet\_Sales | Float | Sales of the product in the particular store. This is the outcome variable to be predicted. |

## Data Gathering

Data source: <https://www.kaggle.com/brijbhushannanda1979/bigmart-sales-data>

Train and Test data which are stored in .csv format.

## Importing CSV

Once we gather data from the data source, we will import the csv files with the help of pandas.

## Data Preprocessing

In data preprocessing all the processes required before sending the data for model building are performed. Like, here the ‘Item Visibility’ attributes are having some values equal to 0, which is not appropriate because if an item is present in the market, then how its visibility can be 0. So, it has been replaced with the average value of the item visibility of the respective ‘Item Identifier’ category. New attributes were added named ‘’Outlet years”, where the given establishment year is subtracted from the current year. Mapping of “Item\_Type” is done based on ‘Drinks’,’Food’,’Non-Consumable’. And mapping of “Fat content” is done based on ‘Low Fat’, ‘Regular’ and ‘Non-edible’.

## Feature Engineering

After preprocessing, it was discovered that certain attributes do not significantly impact the sales of the specific outlet. Therefore, those attributes were eliminated. Additionally, one-hot encoding was utilized to transform the categorical features into numerical features.

## Parameter Tuning

Parameters are tuned using RandomizedSearchCV. The algorithm used in this problem is Random Forest Regressor. The parameters of the algorithm are tunned and passed into the model.

## Model Building

After completing various preprocessing tasks mentioned earlier and conducting scaling and hyperparameter tuning, the dataset will be fed into the Random Forest regressor. It was observed that the Random Forest regressor exhibited the best performance with an RMSE value of 1120.40. Therefore, the 'Random Forest regressor' proved to be effective in solving this problem.

## Model Saving

Model is saved using pickle library in ‘rf.pkl’ format.

## Django Setup for Data Extraction

After saving the model, the API building process started using Django. Web application creation was created here. Whatever the data user will enter and then that data will be extracted by the model to predict the prediction of sales.

## Push to Github

The whole project directory will be pushed into the GitHub repository.

## Deployment

We will be deploying the model to AWS.

# Unit Test Cases

|  |  |  |
| --- | --- | --- |
| **Test Case Description** | **Pre-Requisite** | **Expected Result** |
| Verify whether the Application URL is  accessible to the user | 1. Application URL  should be defined | Application URL should be  accessible to the user |
| Verify whether the Application loads completely for the user when the URL is accessed | 1. Application URL is accessible. 2. Application is deployed | The Application should load completely for the user when the URL is accessed |
| Verify whether a user is able to see input fields while opening the application | 1. Application is accessible. 2. The user is able to see the input fields | Users should be able to see input fields on logging in |
| Verify whether a user is able to enter the input values. | 1. Application is accessible. 2. The user is able to see the input fields | The user should be able to fill the input field. |
| Verify whether a user gets predict button to submit the inputs | 1. Application is accessible. 2. The user is able to see the input fields | Users should get Submit button to submit the inputs. |
| Verify whether a user is presented with recommended results on clicking submit | 1. Application is   accessible   1. The user is able to see the input fields. 2. The user is able to see the submit button | Users should be presented with recommended results on clicking submit. |
| Verify whether a result is in accordance with the input that the user has entered | 1. Application is accessible. 2. The user is able to see the input fields. 3. The user is able to see the submit button | The result should be in accordance with the input that the user has entered. |