**Low-Level Design (LLD)**

**Store Sales Prediction**

|  |  |
| --- | --- |
| **Written By** | Rakesh Ranjan |
| **Document Version** | LLD - V.1.1 |
| **Last Revised Date** | 15-Apr-2023 |

**Document Version Control**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comments** |
| **LLD - V.1.0** | 10-Apr-2023 | Rakesh Ranjan | Architecture data creation |
| **LLD - V.1.1** | 15-Apr-2023 | Rakesh Ranjan | Document updated |
|  |  |  |  |

# **Contents**

1. Introduction.................................................................................................................... 1
   1. What is a Low-Level design document
   2. Scope
2. Architecture.................................................................................................................... 2
3. Architecture Description................................................................................................. 3
   1. Data Description
   2. Data Gathering
   3. Data Pre-processing
   4. Feature Engineering
   5. Parameter Tuning
   6. Model Building
   7. Deployment
4. Unit Test Cases .............................................................................................................. 5

1. **Introduction**

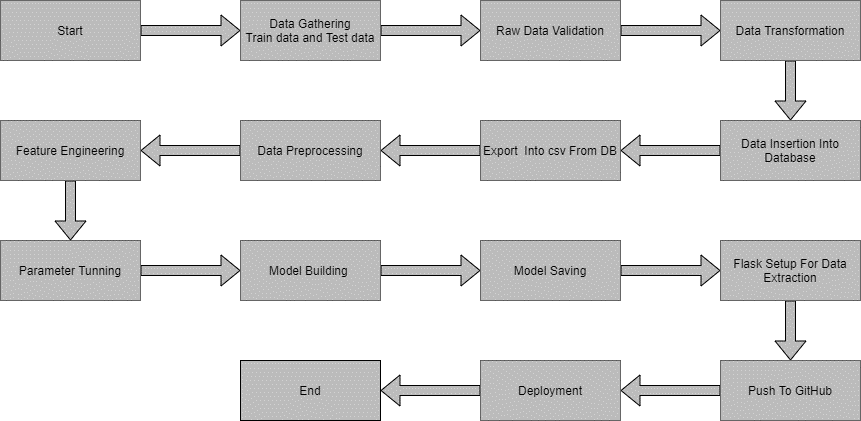
1.1. What is a Low-Level design document?

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.

1.2. Scope

Low-level design (LLD) is a component-level design process that follows a step-by-step refinement process. This process can be used to design data structures, required software architecture, source code, and performance algorithms. Overall, the data organization may be defined during requirement analysis and then refined during data design work.

2. **Architecture**



3. **Architecture Description**

3.1. Data Description

[Big Mart Sales Data](https://www.kaggle.com/brijbhushannanda1979/bigmart-sales-data) is the biggest publicly available recipe dataset. We have the train (8523) and test (5681) datasets, and the train dataset has both input and output variable(s). We need to predict the sales for the test data set.

The variable name, variable type, measurement unit, and a brief description are given. The concrete compressive strength is the regression problem. The order of this listing corresponds to the order of numerals along 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 the 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. |

3.2. Data Gathering

Data source: [**https://www.kaggle.com/brijbhushannanda1979/bigmart-sales-data**](https://www.kaggle.com/brijbhushannanda1979/bigmart-sales-data)

Train and Test data are stored in .csv format.

3.3. Data Pre-processing

In data pre-processing 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 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. A new “Item Type” attribute was added which just takes the first two characters of the Item Identifier which indicates the types of the items. Then mapping of “Fat content” is done based on ‘Low’, ‘Reg’ and ‘Non-edible’

3.4. Feature Engineering

After pre-processing it was found that some of the attributes are not important to the item sales for the particular outlet. So those attributes are removed. Even one hot encoding is also performed to convert the categorical features into numerical features.

3.5. Parameter Tuning

Parameters are tuned using Randomized SearchCV. Four algorithms are used in this problem, Linear Regression, Gradient boost, Random Forest, and XG Boost regressor. The parameters of all these 4 algorithms are tuned and passed into the model.

3.6. Model Building

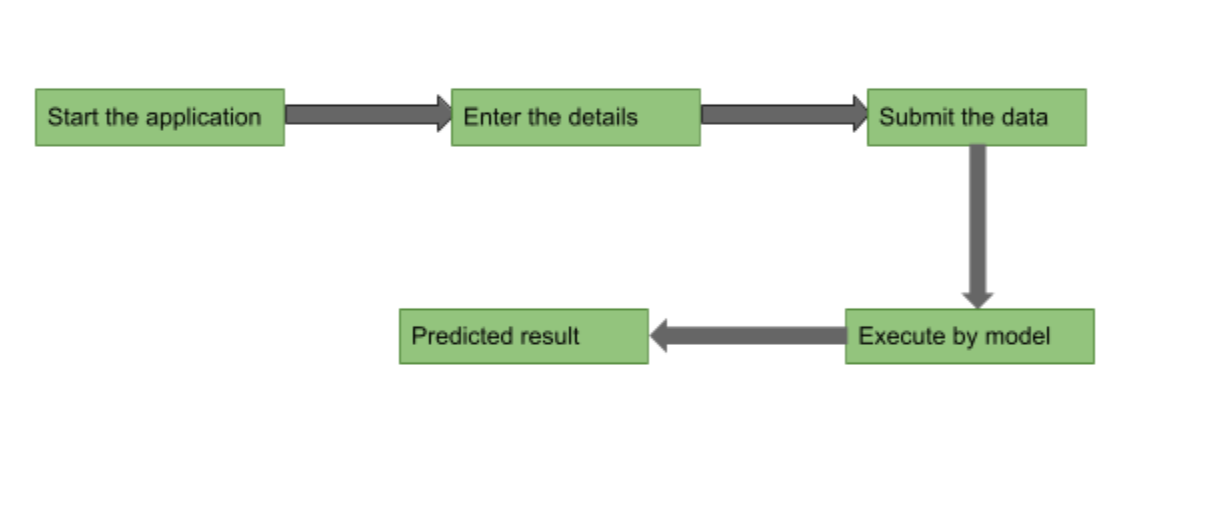
After doing all kinds of pre-processing operations mentioned above and performing scaling and hyperparameter tuning, the data set is passed into all four models, Linear Regression, Gradient boost, Random Forest, and XG Boost regressor. It was found that Gradient boost performs best with the smallest RMSE value i.e. 587.0 and the highest R2 score equals 0.55. So ‘Gradient boost’ performed well in this problem.

3.7. Deployment

The cloud environment was set up and the project was deployed from GitHub into the google cloud platform.

App link- Will appear on Youtube.

App demo link - <https://youtu.be/qo_S9TkWylE>



4. **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 | The 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 the user is able to see input fields | 1. Application is  accessible | The user should be able to see the input  fields |
| Verify whether the user gets Submit button to submit the inputs | 1. Application is  accessible | The user should get Submit button to  submit the inputs |
| Verify whether the predicted  results are in accordance with the selections user made | 1. Application is  accessible | The predicted results should  be in accordance with the selections  user made |