Low-Level Design

Stores Sales Prediction

|  |  |
| --- | --- |
| Written By | Anubhav Tewari |
| Version | 1.0 |
| Date | 28-09-2021 |

**Document Change Control Record**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comments** |
| 1 | 28/9/21 | Anubhav Tewari | Created and documented a ML model for store sales prediction with r2 score of 59.8 and RMSE of 1008. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### **Reviews:**

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

### **Approval Status:**

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

**Index**

|  |  |
| --- | --- |
| **Content** | **Page No** |
| 1. Introduction | 2 |
| 1.1 What is Low-Level Design Document | 2 |
| 1.2 Scope | 2 |
| 2. Architecture | 3 |
| 3. Architecture Description | 4 |
| 3.1 Data Description | 4 |
| 3.2 Data Gathering | 4 |
| 3.3 Data Transformation | 5 |
| 3.4 Data Pre-processing | 5 |
| 3.5 Feature Engineering | 5 |
| 3.6 Parameter Tuning | 5 |
| 3.7 Model Building | 5 |
| 3.8 Model Saving | 6 |
| 3.9 Flask Setup for Data Deployment | 6 |
| 3.10 GitHub | 6 |
| 3.11 Deployment | 6 |
| 4. Unit Test Cases | 6 |

**1. Introduction**

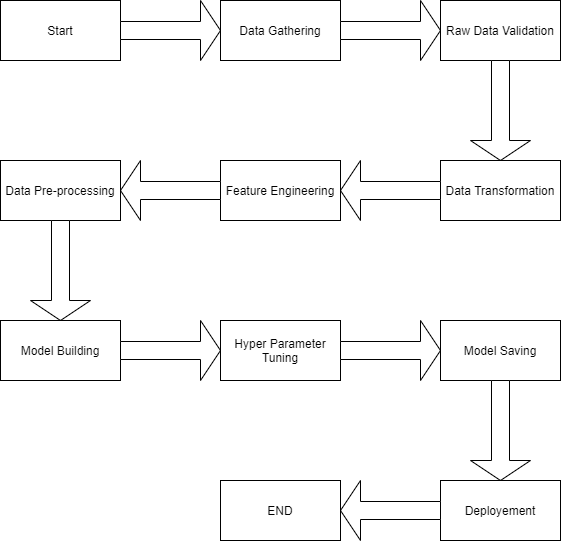
**1.1 What is Low-Level Design Document.**

The goal of LLD or a low-level design document (LLDD) is to give the internal logical design of the actual program code for **‘Stores 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](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.

**2.Architecture**



**3. Architecture Description**

**3.1 Data Description**

The data scientists at BigMart have collected 2013 sales data for 1559 products across 10 stores in different cities. Also, certain attributes of each product and store have been defined. The aim is to build a predictive model and find out the sales of each product at a particular store.

Using this model, BigMart will try to understand the properties of products and stores which play a key role in increasing sales.

|  |  |  |
| --- | --- | --- |
| 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. |

**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 Transformation**

Before sending the data into the dataframe, data transformation is required so that data are converted into such form with which it can be easily inserted into the dataframe. ‘Item Weight’ and “Outlet Type’ attributes contain the missing values. Since, “Item Weight” is a numerical value it is replaced by Mean whereas “Outlet Type” is replaced by Mode.

**3.4Data Preprocessing**

In data preprocessing all the processes required before sending the data for model building are performed. Processes like label encoding for all categorical data is done and also then all attributes are Standardized to a value near 1 so that there is consistency throughout the dataset.

**3.5 Feature Engineering**

In Feature Engineering we remove all unnecessary attributes and add value adding attributes to the data frame. Attributes removed had no correlation with the store sales directly. These attributes were

**3.6 Parameter Tuning**

Parameters are tuned using Randomized searchCV. Four algorithms are used in this problem, Linear Regression, Random Forest, Gradient boost, Random Forest, and Lasso regressor. The parameters of Gradient Boost is tuned and passed .

**3.7 Model Building**

After doing all kinds of preprocessing operations mention above and performing scaling and hyperparameter tuning, the data set is passed into all four models, Linear Regression, Gradient boost, Random Forest, and Lasso regressor. It was found that Gradient boost performs best with the smallest RMSE value i.e. 1046 and the highest R2 score equals 0.598. So ‘Gradient boost’ performed well in this problem.

**3.8 Model Saving**

Model is saved using joblib library in `.sav` format.

**3.9 Flask Setup for Data Extraction**

After saving the model, the API building process started using Flask. 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, this is performed in this stage.

**3.10 GitHub**

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

**3.11 Deployment**

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

App link- https://storesalespredictions.herokuapp.com/

**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 | Application URL should be  accessible to the user |
| Verify whether the Application loads completely for the user when the URL is accessed. | Application URL is accessible  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 | Application is accessible  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. | Application is accessible  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 | Application is accessible  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 | Application is  accessible  The user is able to see the input fields.  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 | Application is accessible  The user is able to see the input fields.  The user is able to see the submit button | The result should be in accordance with the input that the user has entered |