Low Level Design

Stores Sales Prediction

|  |  |
| --- | --- |
| Written By | Ayesha Khan |
| Version | 0.1 |
| Date | 03 – Oct - 2021 |

**Document Control**

### Change Record:

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

### Reviews:

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

### Approval Status:

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

Contents

|  |  |
| --- | --- |
| **Content** | **Page No** |
| 1. Introduction | 1 |
| 1.1 What is Low-Level Design Document | 1 |
| 1.2 Scope | 1 |
| 2 Architecture | 2 |
| 3.1 Architecture Description | 3 |
| 3.2 Data Description | 3 |
| 3.3 Data Gathering | 3 |
| 3.4 Data Cleaning | 3 |
| 3.5 Handling Missing Data | 3 |
| 3.6 New Feature Generation | 3 |
| 3.7 Feature Selection | 4 |
| 3.8 Encoding Categorical Data | 4 |
| 3.9 Parameter Tuning | 4 |
| 3.10 Model Building | 4 |
| 3.11 Model Saving | 4 |
| 3.12 Flask setup | 4 |
| 3.13 GitHub | 4 |
| 3.14 Deployment | 4 |
| 4 Unit Test Cases | 5 |

# 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

Start

Data gathering

Data Cleaning

Handling Missing Data

Parameter tuning

Model building

Model saving

End

Feature Generation

Deployment

Export into csv

Push to GitHub

Flask setup

Encoding Categorical Data

New feature creation

# Architecture Description

## Data Description

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

## Data Gathering

Dataset link :- [Link](https://www.kaggle.com/brijbhushannanda1979/bigmart-sales-data)

We got two csv files train and test. We combined these two files for the data preparation purpose and EDA.

## Data Cleaning

We observed some inconsistency in the Item\_Fat\_content feature such as for low fat category we had LF, low fat and Low Fat so we clubbed them all as LF. The other category had Regular and reg we clubbed them Low Fat and Regular.

## Handling Missing Data

There were missing values in two columns i.e., Outlet\_Size and Item\_Weight with 28.3% and 17.2% missing values respectively. We used EDA to identify the appropriate value to be filled for Outlet\_Size column. So, we replaced the missing value in Outlet\_Size with ‘small’. For Item\_Weight, we replace missing values according to Item\_Identifier feature. Each Item\_Identifier has its own weight. We easily replaced missing value by mapping Item\_Weight with their respective Item\_Identifier.

## Feature Generation

For Item\_Identifier feature, we have around 1559 unique values. This column would not be important in sales prediction. We created new column Item\_type\_Combined with three categories using Item\_Identifier namely ‘Food’, ‘Non-Consumable’ and ‘Drinks’. We also created Years\_Established column from Outlet\_Established\_Year by subtracting Outlet\_Established\_Year from current year.

## Feature Selection

We included all the features for model training except for Outlet\_Identifier and of course Outlet\_Establishment\_Year and Item\_Identifier

## Encoding Categorical Data

Label Encoding was used for ordinal columns Outlet\_Size & Outlet\_Location\_Type. We used dummy encoding for Item\_Fat\_Content, Outlet\_Type, Item\_Type\_Combined and Item\_Type.

## Model Selection

For model selection we had used Pycaret library that will perform multiple operations and give algorithm best for data. Pycaret had given Gradient Boosting regressor as best for our data.

## Parameter Tuning

Parameters are tuned using Grid searchCV. The parameters are tuned on Gradient Boost model.

## 3.10 Model Building

After doing all kinds of pre-processing operations mention above and hyperparameter tuning data is passed to Gradient Regressor model It was found that it performs best with the smallest RMSE value i.e. 1157.2381611038

## 3.11 Model Saving

Model is saved using pickle library in `.pkl` format.

## 3.12 Flask Setup

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

## 3.13 Github

The whole project directory will be pushed into the GitHub repository

## Deployment

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

.

**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 | 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 |