

**SALES PRICE PREDICTION**

Low Level Design

Domain: Machine Learning

Creator: Aravind S

Date: 23.07.2023

|  |  |  |  |
| --- | --- | --- | --- |
| Date issued | Version | Description | Author |
| July 23, 2023 | 1.1 | First Draft | Aravind S |

**Document Version Control**

Contents

[Introduction 3](#_Toc141028563)

[What is Low-Level Design Document? 3](#_Toc141028564)

[Scope 3](#_Toc141028565)

[Architecture 4](#_Toc141028566)

[Process Flow 4](#_Toc141028567)

[Deployment Process 4](#_Toc141028568)

[Data Description 5](#_Toc141028569)

[Data Gathering 5](#_Toc141028570)

[Exception and logging file 6](#_Toc141028571)

[Data Transformation 6](#_Toc141028572)

[Data preprocessing 6](#_Toc141028573)

[Feature Engineering 7](#_Toc141028574)

[Hyperparameter Tuning 7](#_Toc141028575)

[Model Building 7](#_Toc141028576)

[Deployment using Streamlit 8](#_Toc141028577)

[GitHub repository 8](#_Toc141028578)

[Deployment 8](#_Toc141028579)

[Unit cases 9](#_Toc141028580)

# Introduction

## What is Low-Level Design Document?

The objective of the Low-Level Design Document (LLDD) is to provide an internal logical design of the 'Stores Sales Prediction' program code. The LLDD encompasses detailed class diagrams, outlining methods and relationships between classes, along with program specifications. Additionally, it describes the various modules to enable programmers to directly code the program based on the information provided in the document.

## Scope

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

## Process Flow

## Deployment Process

Architecture Description

## Data Description

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

## 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.

## Exception and logging file

A custom exception is a user-defined exception that extends the base Exception class, enabling developers to raise specific errors for unique situations. By creating custom exceptions, code can be organized and managed more effectively. Additionally, logging files record and store important information about an application's execution, providing insights into errors and events, facilitating debugging and system monitoring.

## Data Transformation

Data transformation is necessary to ensure that it is in a suitable format for easy for machine to learn. In this process, the 'Item Weight' and 'Outlet Type' attributes, which contain missing values, are filled with appropriate data in both the train set and the test set, ensuring they have the correct data types for while predicting.

## Data preprocessing

Data preprocessing involves essential tasks performed before using the data for model building. In this context, the 'Item Visibility' attribute had some values equal to 0, which is inappropriate since an item in the market should not have zero visibility. To address this, these zero values were replaced with the average visibility of items within the respective 'Item Identifier' category.

Additionally, two new attributes were introduced: 'Outlet years', which calculates the number of years since the establishment year by subtracting it from the current year; and 'Item Type', which extracts the first two characters from the 'Item Identifier', indicating the type of items.

Moreover, the 'Fat content' attribute was mapped to categorical values ('Low', 'Regular') for better representation and analysis.

By performing these preprocessing steps, the data is now ready for effective model building and analysis.

## Feature Engineering

After preprocessing the data, it was observed that certain attributes do not significantly impact the item sales for the particular outlet. Consequently, these non-important attributes were removed from the dataset to streamline and optimize the modeling process.

Furthermore, to handle categorical features, the technique of one-hot encoding was applied. This conversion process transformed categorical attributes into numerical features, making them compatible with machine learning algorithms that typically work with numerical data.

By removing irrelevant attributes and converting categorical features into numerical representations, the dataset is now better prepared for model building and analysis, focusing on the most relevant information to predict item sales for each outlet accurately.

## Hyperparameter Tuning

Hyperparameter tuning with GridSearchCV involves systematically searching through a predefined hyperparameter grid, evaluating model performance via cross-validation, and selecting the best combination of hyperparameters to optimize model accuracy or performance. It automates the process and ensures the most effective hyperparameters are chosen.

## Model Building

After conducting all the preprocessing steps, including scaling and hyperparameter tuning, the dataset is split and fed into various models such as Linear Regression, Random Forest Regressor, XGBoost, Catboost, Gradient Boosting, Ada boosting. The evaluation results reveal that the Catboost regressor performs the best among all, achieving the highest R-squared (R2) score of 0.62. Thus, the Catboost model is the preferred model for this specific problem, demonstrating superior performance.

## Deployment using Streamlit

After saving the trained model, the process of building a Web UI using streamlit begins. This involves creating a web application where users can input data. The web application extracts the user-entered data, which is then passed to the saved model for sales prediction. The model processes the data and returns the sales prediction to the user through the web application. This stage enables real-time sales predictions based on user input.

## GitHub repository

Pushing a project to a GitHub repository allows for version control, collaboration, and easy access to project files. It enables teams to work together, track changes, and maintain a history of project updates. Additionally, it serves as a secure backup and facilitates seamless deployment to production environments.

## Deployment

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

App link-

# Unit 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 |