Low-Level Design

Stores Sales Prediction

|  |  |
| --- | --- |
| Written By | Ganesh Sayaji Thorat |
| Version | 1.0 |
| Date | 10-09-2021 |

**Document Change Control Record**

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

### Reviews:

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

### Approval Status:

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

**Index**

|  |  |
| --- | --- |
| **Content** | **Page No** |
| 1. Introduction | 4 |
| 1.1 What is Low-Level Design Document | 4 |
| 1.2 Scope | 4 |
| Architecture | 5 |
| 2. Architecture Description | 5 |
| 2.1 Data Description | 5 |
| 2.2 Data Gathering | 6 |
| 2.3 Raw Data Validation | 6 |
| 2.4 Data Transformation | 6 |
| 2.5 New Feature Generation | 7 |
| 2.6 Data Pre-processing | 7 |
| 2.7 Feature Engineering | 7 |
| 2.8 Parameter Tuning | 7 |
| 2.9 Model Building | 7 |
| 2.10 Model Saving | 7 |
| 2.11 FastAPI and Flask setup | 8 |
| 2.12 GitHub | 8 |
| 2.13 Deployment | 8 |
| 3. Unit Test Cases | 9 |

**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 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**

**2. Architecture Description**

**2.1 Data Description**

Given is the variable name, variable type, the measurement unit, and a brief description. 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 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. |

**2.2 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.

**2.3 Raw Data Validation**

After data is loaded, various types of validation are required before we proceed further with any operation. Validations like checking for zero standard deviation for all the columns, checking for complete missing values in any columns, etc. These are required because The attributes which contain these are of no use. It will not play role in contributing to the sales of an item from respective outlets.

Like if any attribute is having zero standard deviation, it means that’s all the values are the same, its mean is zero. This indicates that either the sale is increasing or decrease that attribute will remain the same. Similarly, if any attribute is having full missing values, then there is no use in taking that attribute into an account for operation. It’s unnecessary increasing the chances of dimensionality curse.

**2.4 Data Transformation**

Before sending the data into the database, data transformation is required so that data are converted into such form with which it can easily insert into the database. Here, the ‘Item Weight’ and “Outlet Type’ attributes contain the missing values. So they are filled in both the train set as well as the test set with supported appropriate data types.

**2.5 New feature generations**

We can derive new item category from item type and Outlet\_Years.

**2.6 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 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. 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’.

**2.7 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.

**Model Selection:**

For model selection we had used EvalML library that will perform multiple operations and give algorithm best for data. EvalML had given Random Forest regressor as best for our data.

**2.8 Parameter Tuning**

Parameters are tuned using Randomized searchCV. The parameters are tuned on Random Forest model.

**2.9 Model Building**

After doing all kinds of pre-processing operations mention above and performing scaling and hyperparameter tuning data is passed to Random Forest model It was found that it performs best with the smallest RMSE value i.e. 1052.2633636510775

and the highest R2 score equals 0.5937 on test data. So Random forest performed well in this problem.

**2.10 Model Saving**

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

**2.11 FastAPI and Flask Setup for Data Extraction**

After saving the model, the API building process started using FastAPI. Web application creation was created using Flask for testing purpose. 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.

**2.13 GitHub**

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

**2.14 Deployment**

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

API link- <https://stores-sales-api.herokuapp.com>

WebApp link - <https://stores-sale-prediction.herokuapp.com/>

**3. 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 |