Low Level Design

Food Recommendation System

|  |  |
| --- | --- |
| Written By | Aadil Hussain Teeli |
| Document Version | 0.1 |
| Last Revised Date | 12-02-2024 |

**Document Control**

**Change Record:**

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

**Reviews:**

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

**Approval Status:**

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

Contents

# 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 Travel Package Purchase System . 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-bystep [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

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

# 3. Architecture Description

## 3.1.Data Description

The description of our data is defined using the following table where we have feature name, its type and its description.

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Description |
| Customer\_id | int | Unique customer ID |
| Prod\_Taken | int | Whether the customer has purchased a package or not (0: No, 1: Yes) |
| Age | float | Age of Customer |
| TypeofContact | object | How customer was contacted (Company Invited or Self Inquiry) |
| CityTier | int | City tier depends on the development of a city, population, facilities, and living standards. The categories are ordered i.e. Tier 1 > Tier 2 > Tier 3 |
| Occupation | object | Occupation of customer |
| Gender | object | Gender of customer |
| NumberOfPersonVisiting | Int | Total number of persons planning to take the trip with the customer |
| PreferredPropertyStar | float | Preferred hotel property rating by customer |
| MaritalStatus | object | Marital status of customer |
| NumberOfTrips | float | Average number of trips in a year by customer |
| Passport | int | The customer has a passport or not (0: No, 1: Yes) |
| OwnCar | int | Whether the customers own a car or not (0: No, 1: Yes) |
| NumberOfChildrenVisiting | float | Total number of children with age less than 5 planning to take the trip with the customer |
| MonthlyIncome | float | Gross monthly income of the customer |
| Designation | object | Designation of the customer in the current organization |

## 3.2.Data Validation

Following the loading of data, it is essential to conduct various validations before proceeding with any further operations. These validations include checking for zero standard deviation across all columns and identifying columns with complete missing values. These checks are imperative because attributes exhibiting these characteristics are deemed useless and do not contribute to the sales of items from respective outlets.

For instance, if an attribute displays zero standard deviation, it implies that all values are the same, with a mean of zero. This suggests that regardless of whether the sales are increasing or decreasing, that attribute will remain constant. Similarly, if an attribute contains entirely missing values, including it in operations serves no purpose and unnecessarily increases the risk of the curse of dimensionality.

## 

## 3.3.Data Transformation

## Prior to sending the data into the database, it is necessary to undergo data transformation to convert it into a format suitable for easy insertion into the database. Notably, the 'Age,' 'Duration of Pitch,' and 'Monthly Income' attributes exhibit missing values. Therefore, in both the train set and the test set, these missing values are filled with appropriate data types to ensure completeness and compatibility with the database structure.

## 3.4.Data Pre-processing

Before constructing the model, extensive pre-processing was conducted on the customer data. The handling of missing values took into account data type and distribution; for instance, numerical features were imputed with the mean, categorical features were filled with the mode, and potentially removed if deemed significant. Correction or removal of invalid values depended on their severity and impact. Outliers were identified, and their influence on the analysis led to their addressing and removal. Additionally, feature scaling and normalization were implemented to guarantee uniform scales for all features, thereby enhancing the efficiency of the model building process.

## 3.5.Feature Engineering

Feature Engineering was conducted after the pre-processing stage, revealing that certain attributes were deemed unimportant for the specific outlet. Consequently, these irrelevant attributes were removed from the dataset. Additionally, one-hot encoding was implemented to transform categorical features into numerical features, enhancing their compatibility for subsequent analysis and modeling.

## 3.6.Pipelining

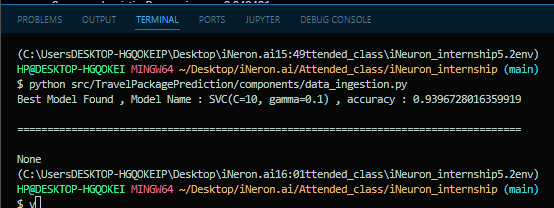
During the pre-processing phase of my project, distinct pipelines were set up for handling numerical and categorical features separately. The numerical pipeline is designed to address tasks such as imputation and scaling, while the categorical pipeline employs methods such as one-hot encoding. This customized approach ensures that each type of feature undergoes suitable processing, thereby optimizing both the performance and interpretability of the model. The implementation of these pipelines streamlines the pre-processing procedures, facilitating the efficient transformation of the dataset and ultimately improving the accuracy of our predictive models.

## 3.7.Parameter Tunning

The tuning of parameters was carried out using GridSearchCV, where various algorithms including Logistic Regression, Decision Tree SVM, Random Forest, Naive\_Bayes\_Classifier, KNeighborsClassifier and many more algorithms were employed for solving the problem. The parameters of these algorithms were fine-tuned and incorporated into the models. Remarkably, the SVC emerged as the most effective, yielding a training accuracy of 99% and a testing accuracy of 93.9%.

## 3.8.Model building

After doing all kinds of pre-processing operations mention above and performing scaling and hyper-parameter tuning, the data set is passed into these models, we got the best accuracy from SVC Classifier with training accuracy of 99% and the testing accuracy of 93.9%. So SVC Classifier’ performed well in this problem.



## 3.9.Model Saving

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

## 3.10.Django Setup for Data Extraction

Following the model's preservation, the API building process commenced utilizing Flask, leading to the creation of a web application. At this stage, the functionality was developed to receive user-input data through the web interface. The entered data is then extracted by the model to generate predictions regarding sales, providing a seamless interaction between the user and the predictive model within the web application.

## 3.4.Github

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

## 3.4.Deployment

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

App link- https://vxh7bvmiaw.eu-west-3.awsapprunner.com/predictdata

# 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 the User is able to sign up in the application | 1. Application is accessible | The User should be able to sign up in the application |
| Verify whether user is able to successfully login to the application | 1. Application is accessible 2. User is signed up to the application | User should be able to successfully login to the application |
| Verify whether user is able to see input fields on logging in | 1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application | User should be able to see input fields on logging in |
| Verify whether user is able to edit all input fields | 1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application | User should be able to edit all input fields |
| Verify whether user gets Submit button to submit the inputs | 1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application | User should get Submit button to submit the inputs |
| Verify whether user is presented with recommended results on clicking submit | 1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application | User should be presented with recommended results on clicking submit |
| Verify whether the recommended results are in accordance to the selections user made | 1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application | The recommended results should be in accordance to the selections user made |
| Verify whether user has options to filter the recommended results as well | 1. Application is accessible 2. User is signed up | User should have options to filter the recommended results as well |
|  | to the application 3. User is logged in to the application |  |
| Verify whether KPIs modify as per the user inputs for the user's health | 1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application | KPIs should modify as per the user inputs for the user's health |
| Verify whether the KPIs indicate details of the suggested recipe | 1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application | The KPIs should indicate details of the suggested recipe |