Low Level Design

Customer Segmentation : Purchasing capabilities of a customer

Written By	Anwesha Das
Document Version	0.1
Last Revised Date	24-7-22

Document Control

Change Record:

Version	Date	Author	Comments
0.1	23– july - 2022	Anwesha Das	Introduction & Architecture defined
0.2	23– july - 2022		Architecture & Architecture Description appended and updated
0.3	24 –july-2022		Unit Test Cases defined and appended

Reviews:

Version	Date	Reviewer	Comments
0.1			

Approval Status:

Version Comments

Review Date	Reviewed By	Approved By

Contents

1. Introduction

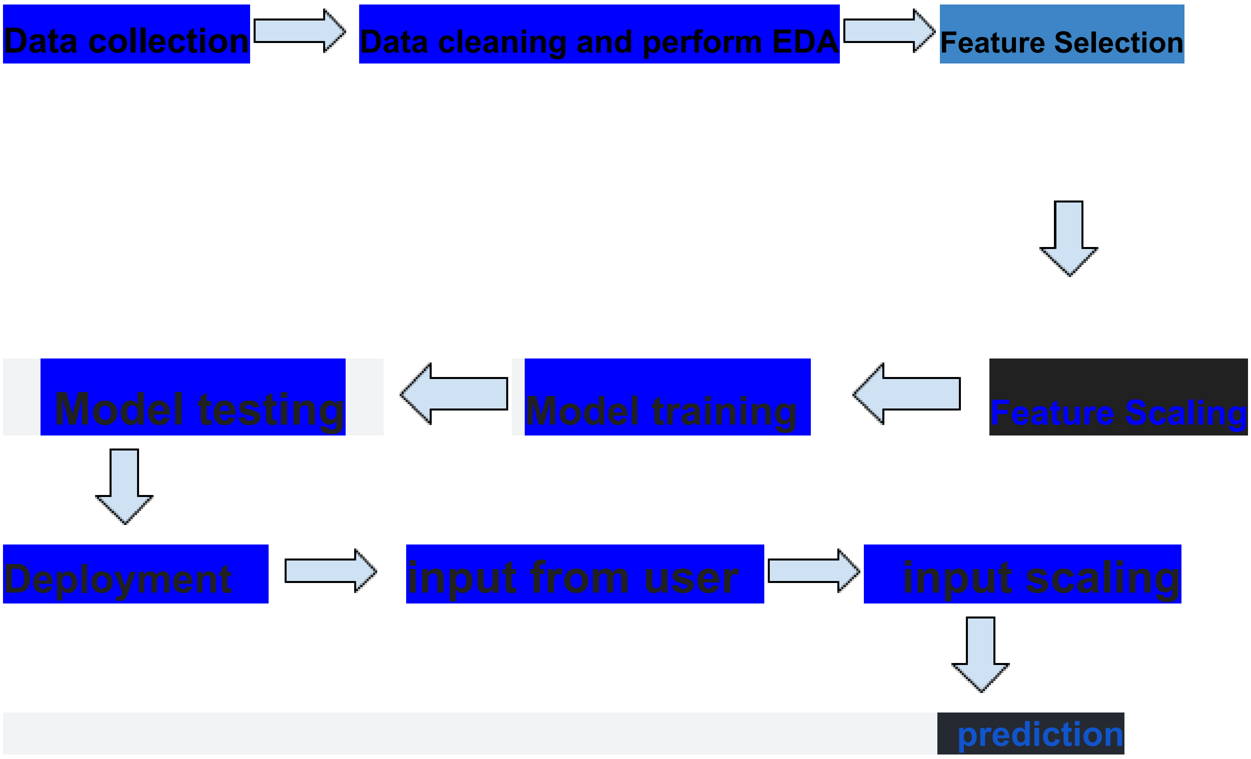
1.1. What is a Low-Level design document?

The goal of LLD or a low-level design document (LLDD) is to give the internal logical design of the actual program code for the purchasing capability of the customer. 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

2. Architecture



3. Architecture Description

3.1. Data Description

Custid - identification of Credit Card holder (Categorical)

BALANCE - Balance amount left in customers account to make purchases

BALANCE_FREQUENCY - How frequently the Balance is updated, score between 0 and 1

PURCHASES - Amount of purchases made from account

ONEOFF_PURCHASES - Maximum purchase amount done in one-go

INSTALLMENTS_PURCHASES - Amount of purchase done in installment

CASH_ADVANCE - Cash in advance given by the user

PURCHASES_FREQUENCY - How frequently the Purchases are being made, score between 0 and 1

ONE OFF PURCHASE FREQUENCY - How frequently Purchases are happening in one-go

PURCHASE INSTALMENTS FREQUENCY - How frequently purchases in installments are being done

CASH ADVANCE FREQUENCY - How frequently the cash in advance being paid

CASH ADVANCE TRX - Number of Transactions made with "Cash in Advance"

PURCHASES_TRX - Number of purchase transactions made

CREDIT_LIMIT - Limit of Credit Card for user

PAYMENTS - Amount of Payment done by user

MINIMUM_PAYMENTS - Minimum amount of payments made by user

PRC FULL PAYMENT - Percent of full payment paid by user

TENURE - Tenure of credit card service for user

3.2 **Data Pre-processing**

Unimportant variables were dropped and missing values were replaced with mean. Since we are building a clustering model, data cleaning is not required as the main goal of clustering is to capture the outliers and make them separate.

3.3 Model Building

Clustering Model: Here we have used K-means clustering for the segmentation of customers possessing similar characteristics with respect to spending behavior. clusters of customers have been created.

3.4 Data from user

Here we will collect data from users such as per description mentioned in Data Set Description. That data will be scaled down by Min Max scaler model, then this scaled data will be used as an input to our model

3.5 **Deployment**

We will be deploying the model to Heroku.

4. Unit Test Cases

Test Case Description	Prerequisite	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 to the application 3. User is logged in to the application	User should have options to filter the recommended results as well
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