Low Level Design

Insurance Premium Prediction

Written By	Ankith Patil
Document Version	0.3
Last Revised Date	12-August-22

Document Control

Change Record:

Change Record: Version	Date	Author	Comments
0.1	08 - August -2022	Ankith Patil	Introduction & Architecture defined
0.2	09 - August -2022	Ankith Patil	Architecture & Architecture Description appended and updated
0.3	12 - August-2021	Ankith Patil	Unit Test Cases defined and appended

Reviews: Version	Date	Reviewer	Comments

Approval Status:	Review Date	Reviewed By	Approved By	Comments
Version				

Contents

1. Introduction

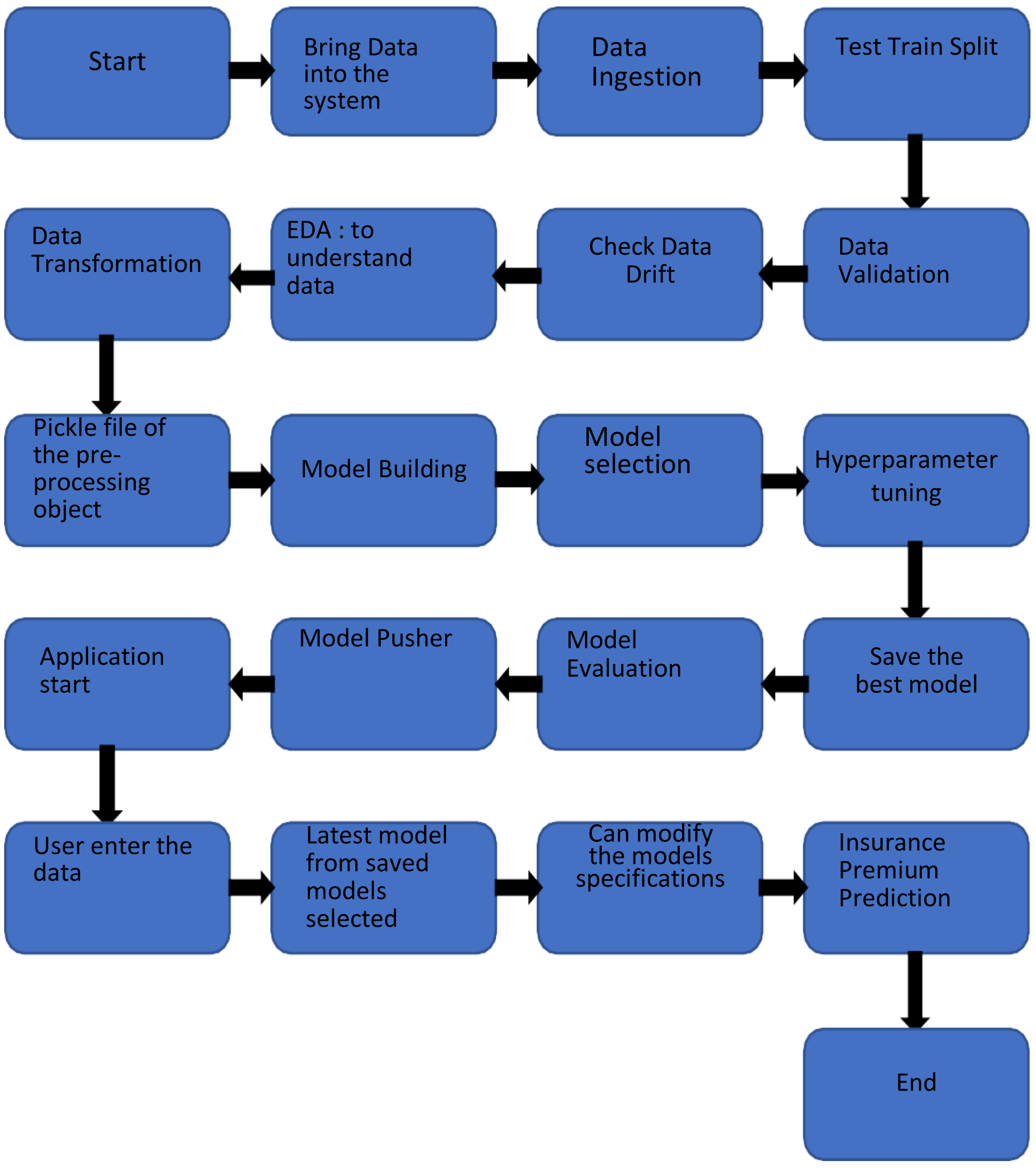
1.1. What is Low-Level design document?

The goal of LLD or a low-level design document (LLD) is to give the internal logical design of the actual program code for Food Recommendation 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-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

1.1. Data Description

Data of more than 1300+ people wherein based on multiple factors like individual's gender, age, BMI, number of children one has, whether one is smoker or not, to which region does one belong on which the insurance premium one needs to pay depends. Thus, this expense on insurance premium prediction is in the csv format.

1.2. Data Ingestion

The dataset is given in the csv format which we can upload on the database and then extract from there for use or we can download to the local system.

After saving it in the local system, we divide the dataset into train and test dataset wherein we work on train data on further steps of the pipeline and leave the test data untouched.

1.3. Data Validation

In the Validation Process, we cross check the data ingested with the schema file provided by the client. And ensure that the features and their datatypes are checked and number of columns are checked and checking the names of the test and train dataset location/file names are checked will convert our original dataset which is in JSON format to CSVFormat. And will merge it with the Scrapped dataset.

1.4. Data Transformation

We after learning the data and doing the feature engineering on the jupyter notebook we segregate the features/columns into numerical and categorical data types and do the fit transform on the train dataset and consequently save this preprocessing object to the pickle file.

This pickle file could be used later to transform the test dataset to the requirements of the model for getting the result.

1.5. Data Pre-processing

Data Pre-processing steps we could use are **Null value handling**, **domain value check** , any **anomalies** , **Imbalanced data set handling**, **Handling columns** with standard deviation zero or below a threshold, duplicate data ,data range etc.

1.6. Model Training

Model training in machine learning is the process in which a machine learning (ML) algorithm is fed with sufficient training data to learn from.

Using the train dataset available and the model factory function we look for the different models given in the **json file** along with the different parameters that need to be checked for cross validation which the **GridSearch** the parameters will be passed with the best parameters derived from Grid-Search.

We will calculate the accuracy scores for each of the models and compare it with threshold and also check the difference between its train and test accuracy to check whether it faces any over-fitting or under-fitting issue. Thus it also compares different models of different algorithms. And selects the model with largest accuracy or the least **rmse**(root mean squared error)

1.7. Model Evaluation

In this model evaluation, we check whether the newly trained model is better than the previously best models. Only if the model is better, only then the models are saved, else the model is rejected.

1.8. Model Pusher

The latest model if better than the previous best model, then the model is pushed to a different file directory, **saved model**.

1.9. Data from User

Here we will collect physiological data from user such as user **BMI**, **Gender**, **Age**; as well as information directly provided by the user such as whether he/she is a **smoker**, **number of children** one has, the **region** of the person.

1.10. Deployment

We will be deploying the model to **Heroku**.

As we have built a **CI-CD pipeline**, we tend to create a new version of the application with every git hub repository changes we commit. We also define the **trigger** for the deployment of whatever changes we commit on the git hub repository in the **github/workflows** directory.

This trigger deploys the application on cloud even when we perform any small changes to the code base. Thus, keeping it updated of every change. And **flexible to new adjustments.**

2. 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