LOW LEVEL DESIGN

Insurance Premium Prediction

Written By	Ranjit Singh
Document Version	1.0
Last Revised Date	

Document Version Control

Change Record:

VERSION	DATE	AUTHOR	COMMENT
1.0	24-04-2023	RANJIT SINGH	INTRODUCTION & ARCHITECTURE DEFINED

REVIEWS:

VERSION	DATE	REVIEWER	COMMENT

APPROVAL STATUS:

VERSION	REVIEW DATE	REVIEWED BY	APPROVED BY	COMMENTS

CONTENTS

1. Introduction

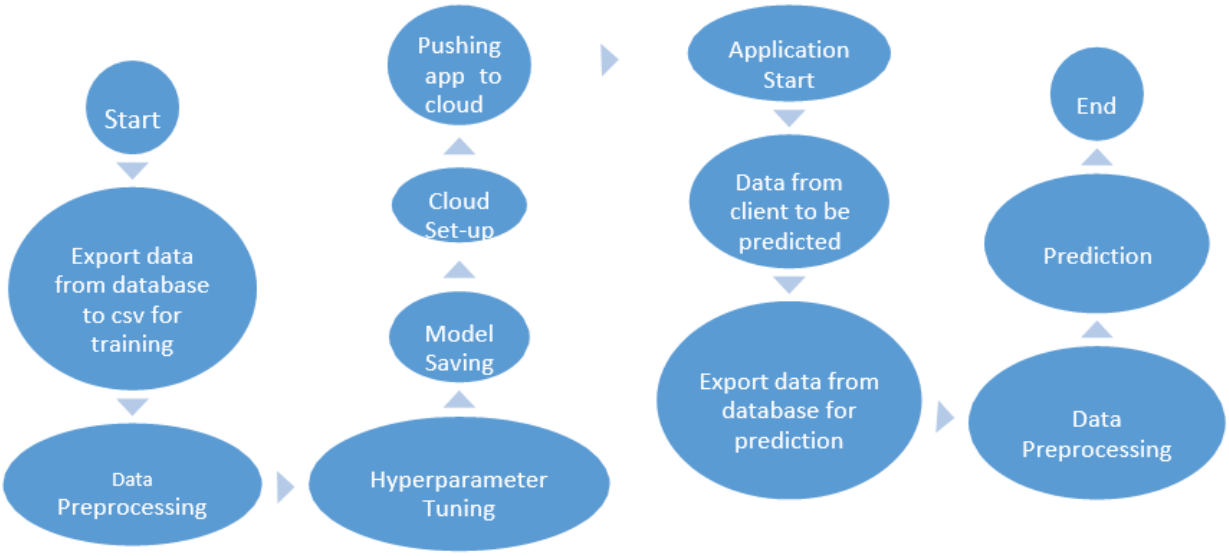
1.1 What is Low-Level design document?

The goal of LLD or a low-level design document (LLD) is to give the internal logical design of the actual program code for Insurance premium prediction. LLD describe the class diagrams with the methods and relations between classes and program specs. It describe 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 stepby-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

We will be using Insurance Premium Prediction Data Set present in Kaggle. This Data set is satisfying our data requirement. Total 1338 instances present in different batches of data.

3.2 Export Data from database to CSV for Training

Here we will be exporting all batches of data from database into one csv file for training.

3.3 Data Preprocessing

We will be exploring our data set here and do EDA if required and perform data preprocessing depending on the data set. We first explore our data set in Jupyter Notebook and decide what pre-processing and Validation we have to do such as imputation of null values, dropping some column, etc and then we have to write separate modules according to our analysis, so that we can implement that for training as well as prediction data.

3.4 Hyperparameter Tuning

Now, we will do hyperparameter tuning for all the models and try to increase the performance of the model.

3.5 Model Saving

After performing hyperparameter tuning for models, we will save our models so that we can use them for prediction purpose.

3.6 Cloud Setup

Here we will do cloud setup for model deployment. Here we will also create our flask app and user interface and integrate our model with flask app and UI.

3.7 Push app to cloud

After doing cloud setup and checking app locally, we will push our app to cloud to start the application.

3.8 Data from client side for prediction purpos

Now our application on cloud is ready for doing prediction. The prediction data which we receive from client side will be exported from DB and further will do same data cleansing process as we have done for training data using modules we will write for training data. Client data will also go along the same process of Exporting data from DB, Data pre-processing, Data clustering and according to each cluster number we will use our saved model for prediction on that cluster.

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