Low Level Design

Insurance Premium Prediction

Prepared By: Mangesh Devkate

Data Science Intern

iNeuron.ai

Document Version Control

Version	Date	Author	Comments
1	1-12-2022	Mangesh Devkate	

Contents

Document Version Control 2

1.0 Introduction 4

  1.1 What is Low-Level Design Document? 4

  1.2 Scope 4

2.0 Architecture 4

3.0 Architecture Description 5

  3.1 Data Gathering 5

  3.2 Raw Data Validation 5

  3.3 Exploratory Data Analysis 5

  3.4 Feature Engineering 5

  3.5 Feature Selection 5

  3.6 Train-Test-Split 5

  3.7 Model Building 5

  3.8 Model Evaluation 5

  3.8 Model Saving 6

  3.9 Flask Setup for Web Application 6

  3.10 GitHub 6

  3.11 Deployment 6

4.0 Unit Test Cases 6

1.0 Introduction

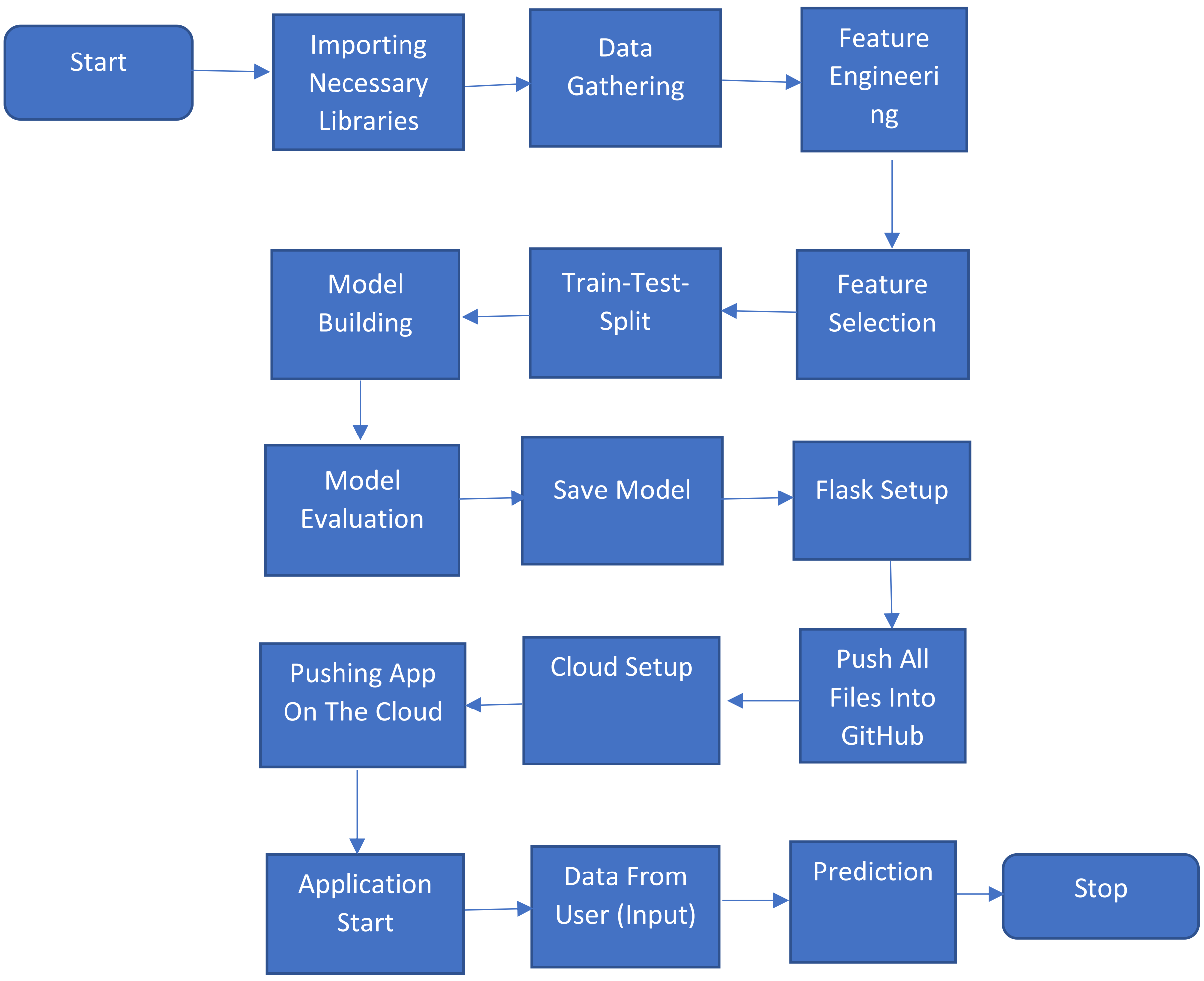
1.1 What is Low-Level Design Document?

The goal of LLD or Low-Level design document (LLDD) is to give the internal logical design of the actual program code. Low-Level design is created based on the High-Level design. 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 is a detailed description of every module of software. It describes every module in detail by incorporating the logic behind every component in the system. It delves deep into every specification of every system, providing a micro-level design.

2.0 Architecture



3.0 Architecture Description

3.1 Data Gathering

Data Source : https://www.kaggle.com/datasets/noordeen/insurance-premium-prediction

3.2 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 estimating cost of the premium.

3.3 Exploratory Data Analysis

In EDA we have done analysis of dependent and independent variables. We have done univariate analysis and bivariate analysis to get insights of the data.

3.4 Feature Engineering

Feature Engineering consists various steps like removing duplicate rows in the dataset, filling NULL/Missing values, outlier detection and handling outliers, transformation of categorical columns into numerical columns using encoding techniques like label encoding, one hot encoding. Adding to these feature engineering consists other steps like feature scaling, feature binning.

3.5 Feature Selection

Feature Selection is the method of reducing the input variable to your model by using only relevant data and getting rid of noise in data. It is the process of automatically choosing relevant features for your machine learning model based on the type of problem you are trying to solve. Feature selection techniques are wrapper method, filter method, embedded method.

3.6 Train-Test-Split

The train_test_split() method is used to split our data into train and test sets. First, we need to divide our data into features (X) and labels (y). The dataframe gets divided into X_train, X_test , y_train and y_test. X_train and y_train sets are used for training and fitting the model.

3.7 Model Building

Building an ML Model requires splitting of data into two sets, such as ‘training set’ and ‘testing set’ in the ratio of 80:20 or 70:30. A set of supervised (for labelled data) and unsupervised (for unlabeled data) algorithms are available to choose from depending on the nature of input data and business outcome to predict.

3.8 Model Evaluation

Model evaluation is the process of using different evaluation metrics to understand a machine learning model's performance.

3.8 Model Saving
Model is saved using pickle library in pickle` format.

3.9 Flask Setup for Web Application
After saving the model, the API building process started using Flask. Web application creation was created in Flask for testing purpose. Data entered by user extracted by the model to estimate the premium of insurance.

3.10 GitHub
The whole project directory pushed into the GitHub repository.

3.11 Deployment
The project was deployed from GitHub into the Heroku platform.

4.0 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 URL is deployed	Application URL should load completely for the user when URL is accessed
Verify whether user can see input field after opening URL	1. Application is accessible	User should be able to see input fields after opening URL
Verify whether user has options to filter the inputs fields	1. Application is accessible	User should filter the options of input fields
Verify whether user gets submit button to submit the inputs	1. Application is accessible	User should get submit button to submit the inputs
Verify whether user can see the output after submitting the inputs	1. Application is accessible	User should get outputs after submitting the inputs