LOW LEVEL DESIGN (LLD)

ADULT CENSUS INCOME PREDICTION

Revision Number: 1.0

Last date of revision: 24/09/2023

Document Version Control

Date Issued	Version	Description	Author
24/07/2023	1	Initial LLD - VI .0	C Sindhuja
24/09/2023	2	Updated LLD - VI .0	C Sindhuja

Table of Contents

1 Introduction

1.1 What is Low-Level design document?

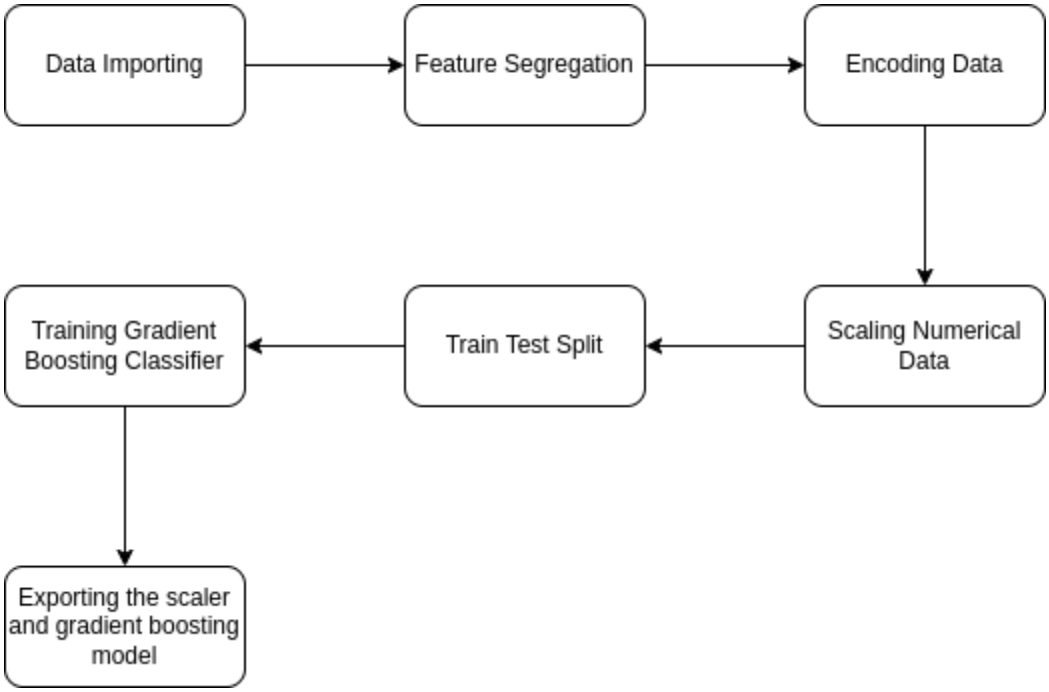
The goal of LLD or a low-level design document (LDD) is to give the internal logical design of the actual program code for flight fare estimation System. LLD describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer can directly code the program from the document.

1.2 Scope

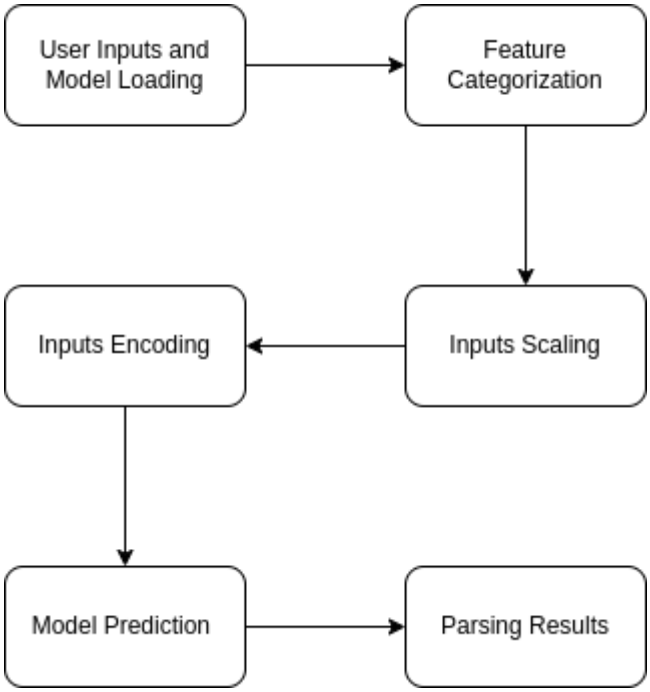
Low-level design (LLD) is a component-level design process that follows a step-by-step refinement process. This process can be used for designing data structures, required software architecture, source code and ultimately, performance algorithms. Overall, the data organization may be defined during requirement analysis and then refined during data design work.

2 Architecture

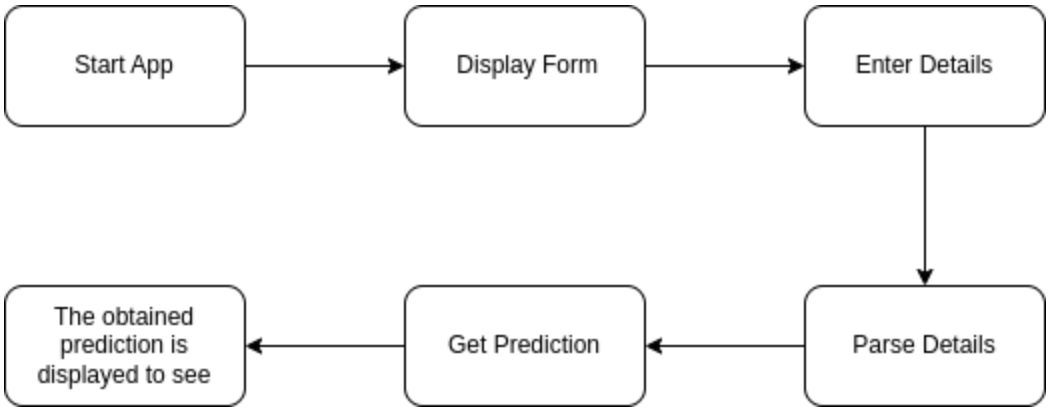
2.1 Training Flow



2.2 Prediction Flow



2.3 Application Flow



3 Architecture Description

3.1 Training Flow

1. **Data Importing**

   Data importing is the initial step in a data analysis or machine learning project, where the pandas library's read_csv function is used to read a dataset from a CSV file. This function allows data to be loaded into a panda Data Frame, making it accessible for further processing.

2. **Feature Segregation**

   Feature segregation involves categorizing the dataset's features into two main types: categorical and numerical. Categorical features are those with discrete values like colors or categories, while numerical features are those with continuous values like age or price. This step helps in choosing appropriate data preprocessing techniques.

3. **Encoding Data**

   Encoding data is necessary when dealing with categorical features. Nominal features (without a natural order) and ordinal features (with a specific order) are encoded separately using techniques like one-hot encoding (get_dummies) to convert them into a numerical format suitable for machine learning algorithms.

4. **Scaling Numerical Data**

   Scaling is a preprocessing step for numerical features that ensures they have similar scales. The Standard Scaler from the scikit-learn Python package is applied to standardize numerical features, making them have a mean of zero and a standard deviation of one. This ensures that features with larger scales do not dominate the learning process.

5. Train Test Split

After encoding and scaling, the data is combined and then split into two subsets: a training set and a test set. This is done to assess the model's performance on unseen data. The common split ratio is 70:30, with 70% of the data used for training and 30% for testing.

6. Training Gradient Boosting Classifier

In this step, a Gradient Boosting Classifier is trained using the training data. This ensemble machine learning algorithm builds multiple decision trees sequentially, with each tree correcting the errors of the previous one. The model's accuracy is evaluated using the test data to assess its predictive performance.

7. Exporting the Scaler and Model

To use the trained model and scaler in the future without retraining, they are exported in the '.pkl' format using the pickle package. This allows for easy reuse of the preprocessing steps and the trained machine learning model for making predictions on new, unseen data.

3.2 Prediction Flow

1. User Inputs and Model Loading

In this step, the function takes user inputs as input data and also loads the necessary models. These models can include the trained Gradient Boosting Classifier model for making predictions and the scaler model for preprocessing numerical features.

2. Feature Categorization

The function categorizes the input features into two main types: numerical and categorical. Numerical features typically represent continuous values, while categorical features represent discrete categories or labels.

3. Inputs Scaling

Numerical values within the input data are scaled using the previously loaded scaler model. This ensures that the input data is processed in the same way as during the model training phase, maintaining consistency.

4. Inputs Encoding

Categorical values within the input data are encoded following the same encoding techniques used in the training pipeline. This ensures that categorical data is transformed into a format suitable for the model's predictions.

5. Model Prediction

After preprocessing the input data, it is passed to the pre-trained Gradient Boosting Classifier model. The model then generates predictions based on the provided input data, which may correspond to class labels or other relevant output.

6. Parsing Results

The results produced by the model are parsed, extracting the relevant information. Typically, this involves returning the predicted label or output, which can be used for decision-making or reporting purposes.

3.3 Application Flow

1. Start App

This step involves initiating a Flask application, which serves as the foundation for a web-based application. The application is started, and the web page associated with it is loaded, making it accessible to users.

2. Display Form

Upon accessing the web application, a user is presented with a form interface. This form is designed to collect specific details or input from the user. It typically includes fields and input elements for data entry.

3. Enter Details

Users are prompted to interact with the displayed form by entering the required information. This step involves user input, where individuals provide data or respond to prompts within the form. The information entered is typically related to the specific task or purpose of the application.

4. Parse Details

After the user submits the form or input data, the entered details are sent to the application's backend. In the backend, these details are processed and parsed, ensuring they are in the correct format and ready for further processing or analysis.

5. Get Predictions

The parsed input data is then passed to a prediction function or algorithm. This function uses the provided data to make predictions, calculations, or perform specific tasks, depending on the application's functionality. For example, it might predict an outcome based on the user's input.

6. Display Results

Finally, the results generated by the prediction function are fetched and displayed to the user. This can involve presenting the predicted outcome, relevant information, or computed results in a user-friendly format on the web page. The displayed results serve to inform the user about the application's response or findings based on their input.