Low Level Design

Adult Census Income Prediction

Written By	Sonu Kumar Pal
Document Version	0.1
Last Revised Date	02–November -2021

Document Control

Change Record:

Version	Date	Author	Comments
0.1	02–November-2021	Sonu kumar Pal	Introduction & Architecture defined

Contents

1. Introduction

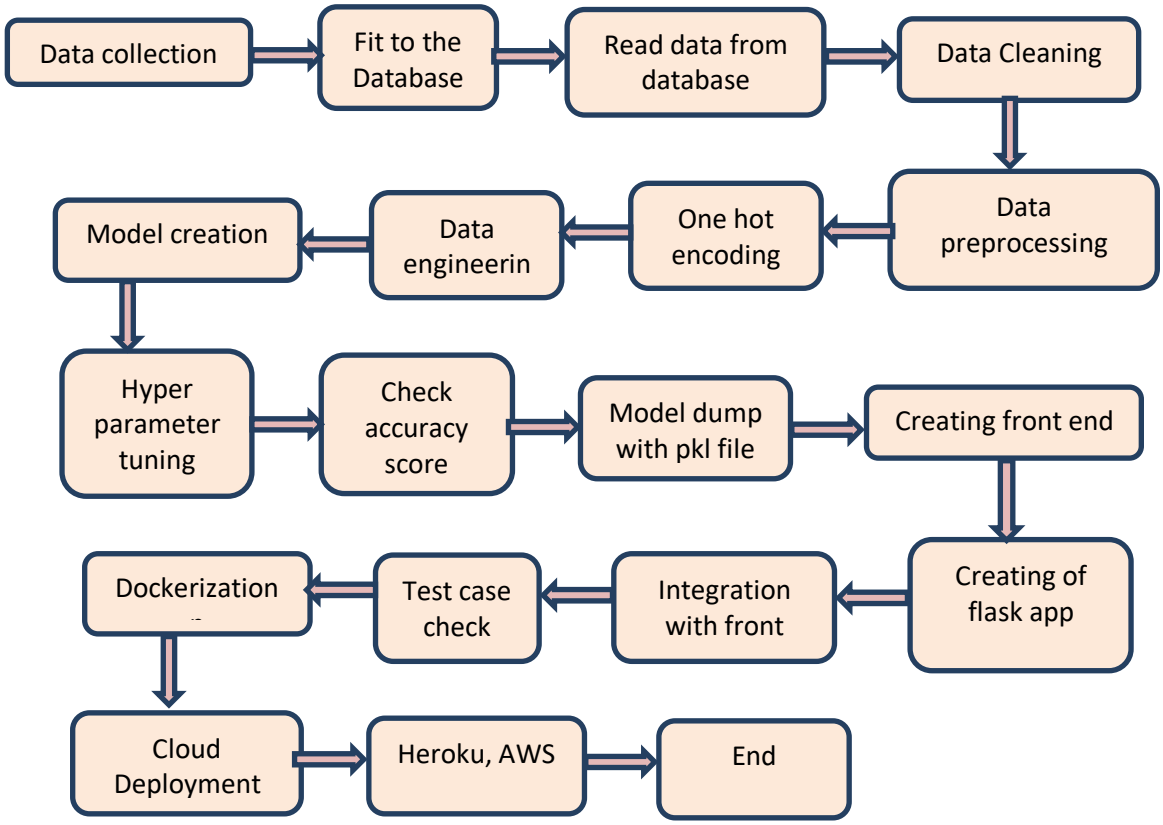
1.1 What is Low-Level design document?

LLD is a component-level design process that follows a step-by-step refinement process. It provides the details and definitions for the actual logic for every system component. It is based on HLD but digs deeper, going into the separate modules and features for every program in order to document their specifications.

1.2 Scope

LLD would more of follow a process flowchart of the module which may also be called as micro-level detailed design. It may consist of actions from designing data structures, required software architecture, source code and ultimately, performance algorithms which is used to convert high level solution to detailed solution.

2 Architecture



3 Architecture Description

3.1 Data Collection

I have dataset of 32562 rows which includes all the parameters required to predict the income. The data is available in a CSV file format and it's collected from the link as per provided in the project description.

3.2 Data Insertion into Database

a. Database Creation and connection - Create a database with Cassandra cloud version and connect to the database.
b. Insert the user input in data base and store it database is work as a hidden API in entire project.

3.3 Export Data from Database

Take the user input and store in Cassandra database.

3.4 Data Cleaning

In order to fit the data to the model I need to identify incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data.

3.5 Feature Engineering

In this section I try to take care of validating or replacing values with other values in the dataset. Additionally, I also take care of converting the categorical columns to numerical columns by one hot encoding and apply label encoding on label data.

3.6 Sampling

Here dataset is unbalanced so I try random oversampling to balance the data.

3.7 Model Building

After the completion of the above process I split the dataset into test and train. I use various classification algorithm like Decision trees, Random Forest, K-NN, Bagging, XgBoost, ExtraTree classifier and validate the accuracy. Finally I select the best algorithm and check the accuracy on train and test data. I used the metrics which validates the variance from model prediction to ground truth.

3.8 Hyperparameter Tuning

Here I have used Randomized Search CV for selection of the best parameters to reduce overfitting criteria. Along with this I have used k fold cross validation technique for training of the model and finally got 84% accuracy.

3.9 Model Dump with pkl file

I have saved the model using pickle.

3.10 Creation of front end

As per the required parameters for the user to predict the income I have created the front end in order to link with the Flask App. Here the styling and formatting of the html page is taken care with the CSS file.

3.11 Flask Web App

I have created a Flask Web App where I read the contents from the pickle file and made sure it linked to the html file to predict the income.

3.12 Test case check

Here I check for the cases where if a customer inputs wrong data the model should not give wrong results as predicted income. I have taken care of the all the conditions that might be possible.

3.13 Dockerize

I have used dockerize technique here which helps the application to run within a docker container. It is useful to take care of the environmental variables and helps the app to run in all the platforms.

3.14 Deployment

I have deployed the app on Heroku platform but I have prepared a deck to help user in order to deploy on other platforms as well.

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 signup In the application
Verify whether user is able to successfully use the application	1. Made sure to check for the test cases from backend.	User should be able to see successfully valid results.
Verify whether user is able to see input fields on logging	1. Application is accessible 2. User is able to log into the application.	User should be able to edit input fields on logging.
Verify whether user is able get the predicted results on the click of submit button	1. Application is accessible 2. User is logged into the application	User is presented with the predicted results on the screen.
Verify whether the results are as per the selection made by the user.	1. Application is accessible 2. User is logged in to the application	Recommendation is as per the selection made by the user.
Verify whether user is able to get the predicted flight price button once he wants to submit the inputs made.	1. Application is accessible . 2. User is logged in to the application	User is able to get the Submit button to submit input criteria's.