

# CAMPUS PLACEMENT PREDICTION

## Architecture

Written By	Bhavika Pathak
Document Version	1.0
Last Revised Date	24-Feb-2024

## Document Control

### Change Records:

Version	Date	Author	Comments
1.0	24-Mar-2024	Bhavika Pathak	Introduction & Architecture defined

### Reviews:

Version	Date	Reviewer	Comments

### Approval Status:

Version	Review Date	Reviewed By	Approved By	Comments

# Contents

- 1. Introduction
  - 1.1. What is Low-Level design document?
  - 1.2. Scope
- 2. Architecture
- 3. Architecture Description
  - 3.1. Database (Kaggle)
  - 3.2. Data Collection
  - 3.3. Exploratory Data Analysis (EDA)
  - 3.4. Feature Engineering
  - 3.5. Feature Selection
  - 3.6. Model Creation
  - 3.7. Model Performance
  - 3.8. Model Saving
  - 3.9. User Interface Designing
  - 3.10. Flask API Development
  - 3.11. User Data Validation
  - 3.12. Cloud Setup
  - 3.13. Model Deployment
- 4. Unit Test Cases

# **1. Introduction**

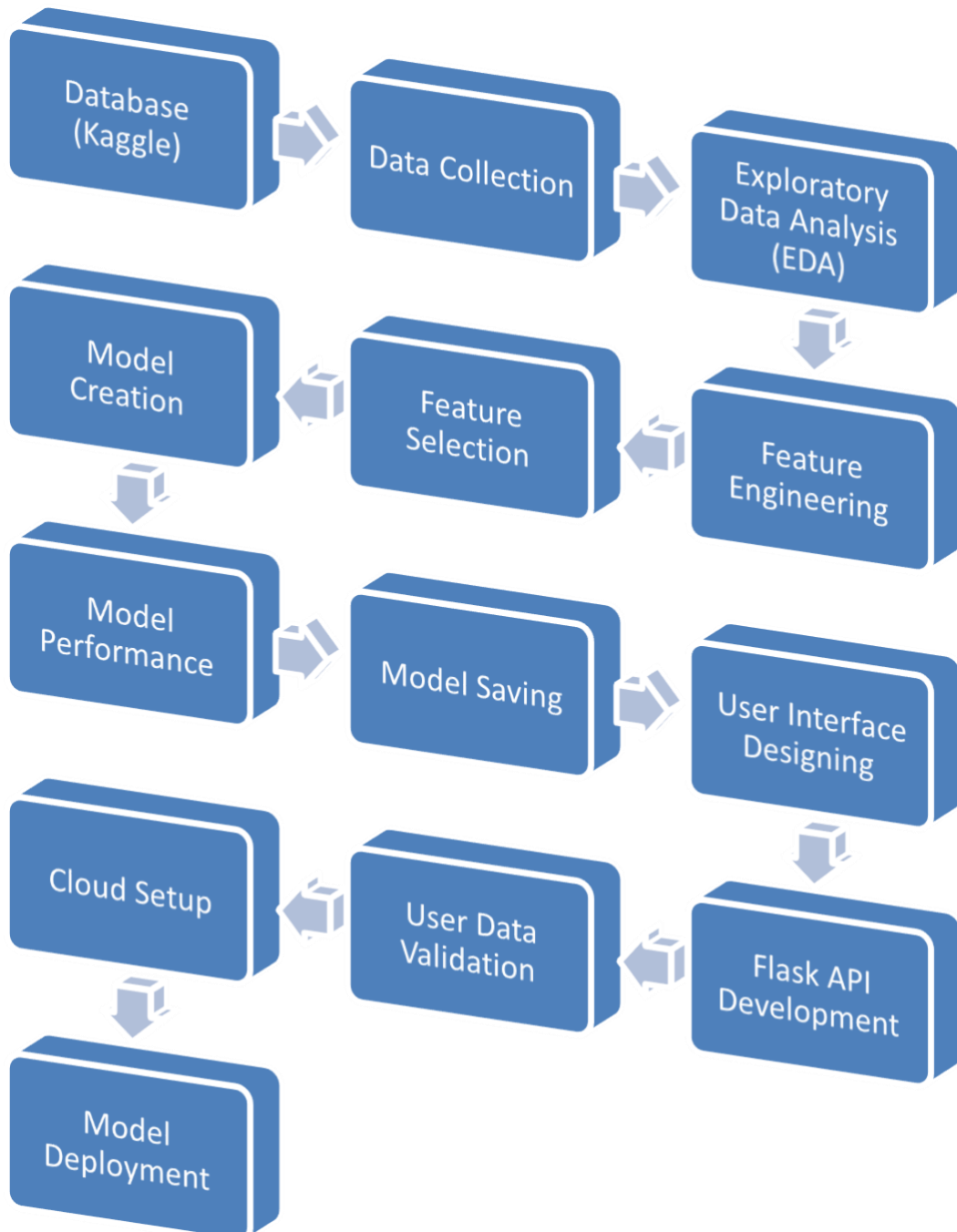
## **1.1. What is Low-Level design document?**

The goal of LLD or a low-level design document (LLD) is to give the internal logical design of the actual program code for Campus Placement Prediction System. LLD describe 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**

### **3.1 Database (Kaggle)**

There are a total of 250 students in the training set

Below is the link to access the datasets present in Kaggle :

<https://www.kaggle.com/c/ml-with-python-course-project/data>

Data fields

- gender - Of the student
- secondary education percentage-marks obtained in secondary education
- higher secondary percentage-marks obtained in higher secondary education
- degree percentage-marks obtained in degree
- Under-graduation(Degree-type)-Field of degree education
- Work-experience
- Employability-test-package
- specialisation-field of study

### **3.2 Data Collection**

Obtain relevant datasets containing historical Campus Placement data, including information of Person.

### **3.3 Exploratory Data Analysis (EDA)**

In the Cleaning process, we cleaned up all the data which is not present in format which can be recognized by machine learning algos. Also, dropped the features which are not relevant. We handled the Null values and replaced with median value. We visualized the data using different libraries like seaborn, matplotlib, etc.

## Architecture

### **3.4 Feature Engineering**

We replaced the outlier with median which was present in the age feature in the dataset. Also performed One hot and label encoding to convert the categorical features to numerical Features. We transformed the data using different transformations like 'log' and 'square\_root' transformations to remove the skewness of the data present in different features.

### **3.5 Feature Selection**

We dropped the columns which were not required after transforming the data. We also checked the multicollinearity between the independent features using correlation and VIF techniques. Highly correlated features were removed after this process.

### **3.6 Model Creation**

After completing the feature selection process, data was split between train and test data. Dataset was imbalanced having high no. of negative classes in comparison to other classes therefore used SMOTE technique to handle it. Different machine learning algorithms like Logistic Regression, DecisionTree, SVM, RandomForest, K-NN were applied on the training data and testing data.

### **3.7 Model Performance**

After fitting the data on different models, we checked the performance using different metrics like accuracy score, precision, recall. We got accuracy 89.74 %.

### **3.8 Model Saving**

After comparing all accuracies, we have chosen SVM as our best model and so we have dumped this model in a pickle file format.

### **3.9 User Interface Designing**

## Architecture

We created a user interactive page where user can enter their input values to different fields of our application. In these front-end page, we designed a form which is taking values using text-boxes. These html user input data is transferred in json format to backend. After filling the fields values, user can see the result by clicking on the predict button.

### **3.10 Test Case Validation**

Data validation is done here which was entered by user.

### **3.11 Cloud Setup**

We are using heroku cloud to deploy our application.

### **3.12 Model Deployment**

We deployed our model to heroku which generate one URL. Using that URL, user can access the User Interface of the application.



## 4. Unit Test Cases

<u>Test Case Description</u>	<u>Pre-requisite</u>	<u>Expected Result</u>
Verify whether the Application URL is accessible to the user.	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	Application URL is accessible. Application is deployed.	The Application should load completely for the user when the URL is accessed.
<u>Verify whether user is giving standard input.</u>	Handled test cases at backends.	User should be able to see successfully valid results.
Verify whether user is able to edit all input fields	Application is accessible.	User should be able to edit all input fields.
Verify whether user gets Predict button to submit the inputs	Application is accessible.	User should get Submit button to submit the inputs.
Verify Predicted result is displayed	Application is accessible.	User should be able to see the predicted result