# LOW- LEVEL DESIGN (LLD)

## Insurance Premium Prediction

Written By- Dhananjay K. Gurav

## Document Version Control:

| Date Issued | Version | Description | Author |
|---|---|---|---|
| 11/12/2023 | 1.0 | Initial LLD- V1.0 | Dhananjay |
| | | | |
| | | | |

# 1. Introduction

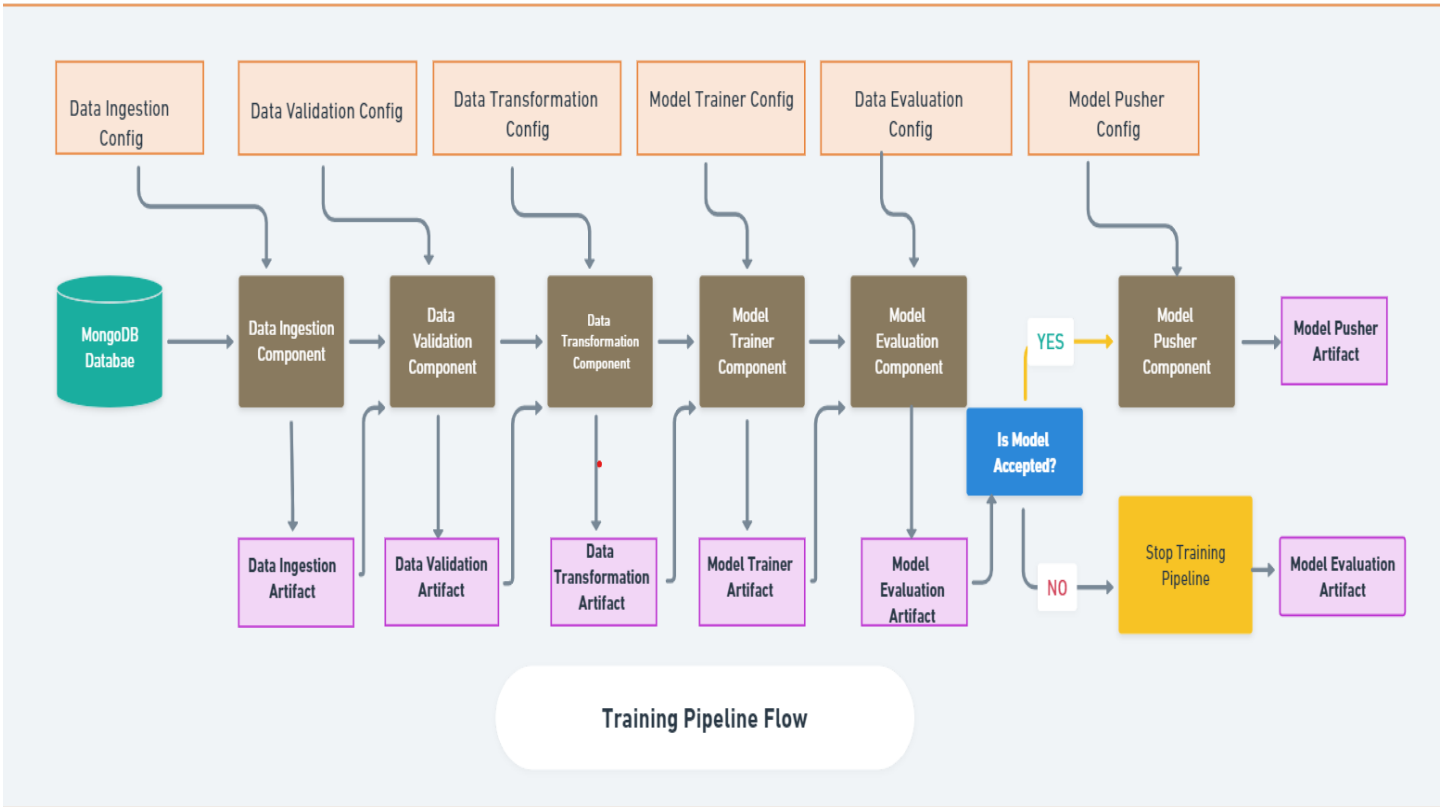
## 1.1. What is Low-Level design document?

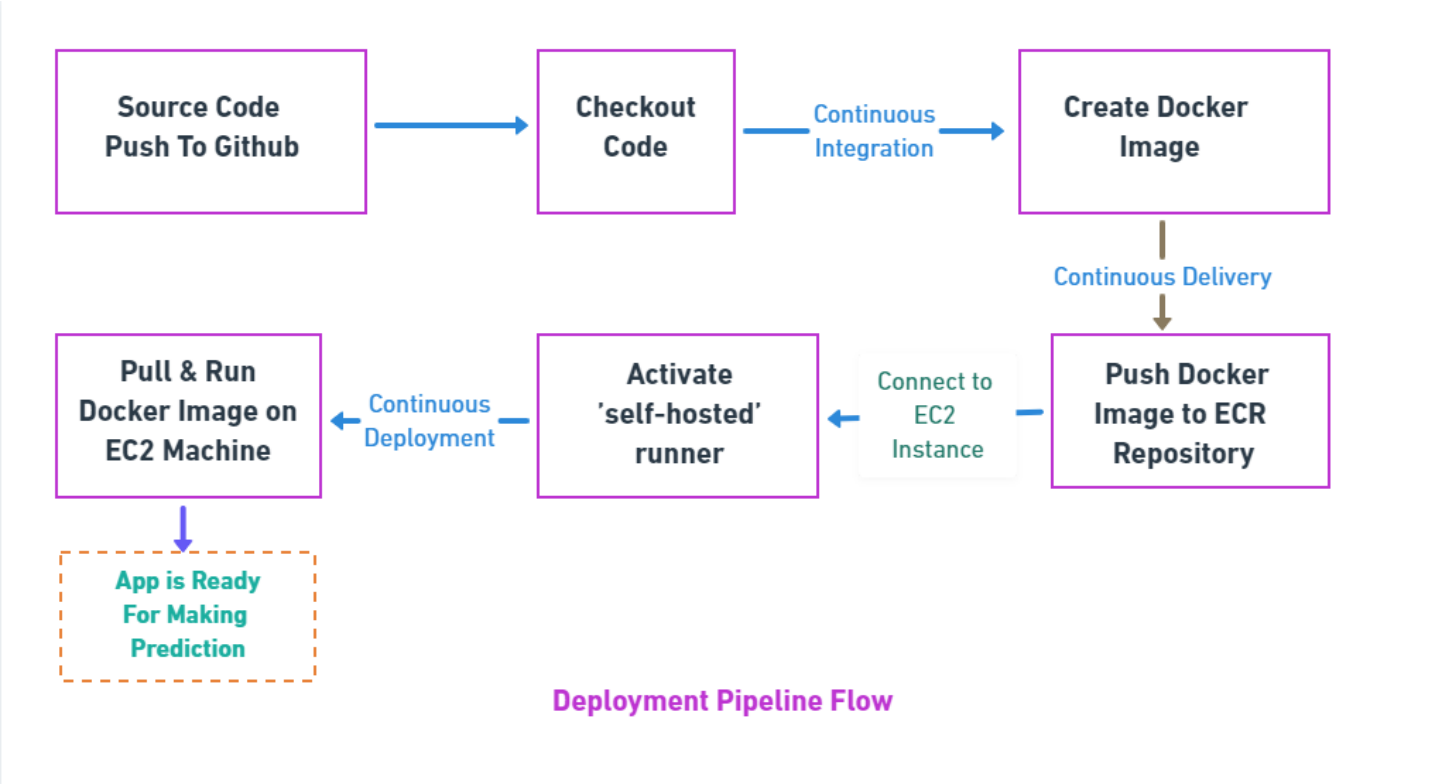
The goal of LLD or a low-level design document (LLD) is to give the internal logical design of the actual program code for the Insurance premium prediction System. LLD describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer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

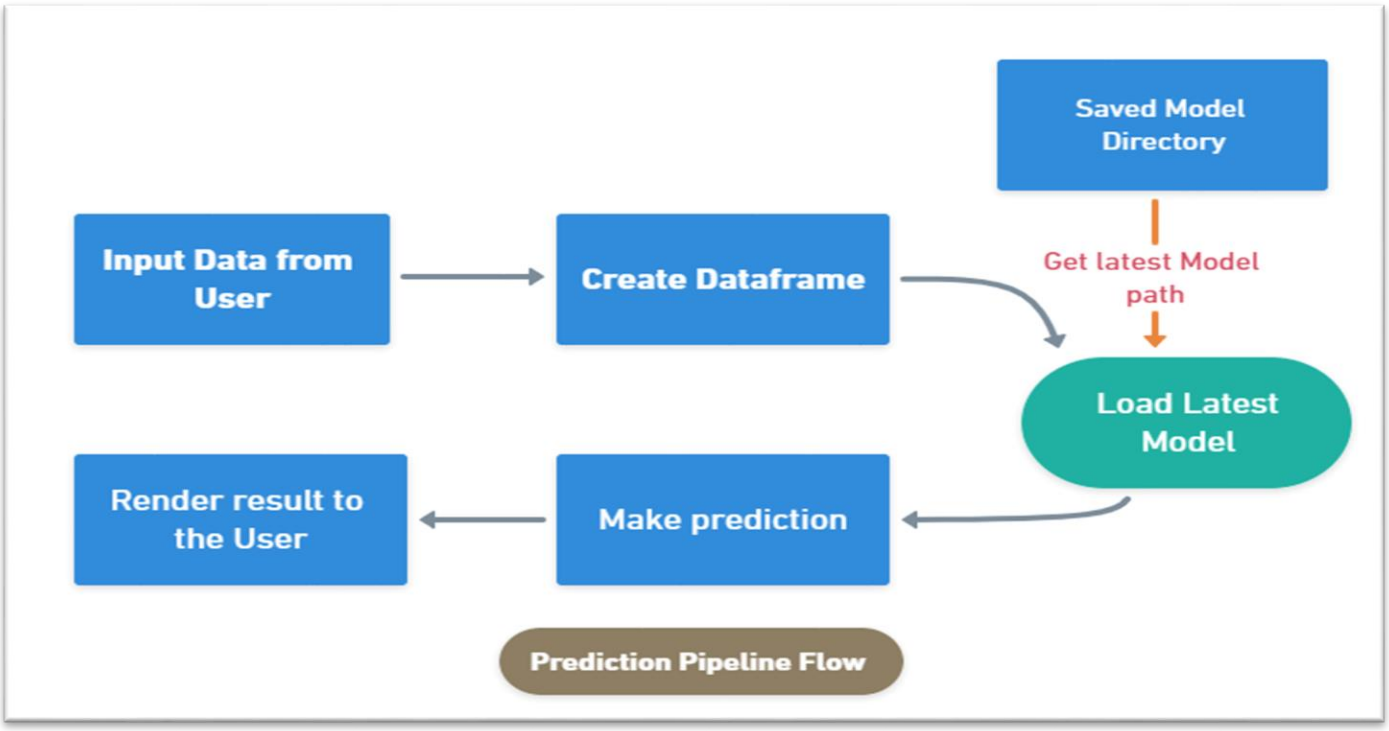
## 1.2. Scope

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

# 2.0 Architecture






## 3. Architecture Description

### 3.1 Data Description:

The primary source of data for this project is from Kaggle. The dataset is comprised of 1338 records with 7 attributes out of which 6 are independent features and one is a target feature('expenses'). The data is in structured format and stored in a CSV file. We will dump all the data into the MongoDB database.

```
df = pd.read_csv('insurance.csv')
df.head()
```

| | age | sex | bmi | children | smoker | region | expenses |
|---|---|---|---|---|---|---|---|
| 0 | 19 | female | 27.9 | 0 | yes | southwest | 16884.92 |
| 1 | 18 | male | 33.8 | 1 | no | southeast | 1725.55 |
| 2 | 28 | male | 33.0 | 3 | no | southeast | 4449.46 |
| 3 | 33 | male | 22.7 | 0 | no | northwest | 21984.47 |
| 4 | 32 | male | 28.9 | 0 | no | northwest | 3866.86 |

```
df.shape
```

```
(1338, 7)
```

### 3.2. Data Ingestion:

In the data Ingestion Process, we will first establish a connection with the MongoDB database to access data continuously. Once the connection is established, we will convert fetched data into Pandas dataframe, after that we will data them into train and test dataset and store it inside the data ingestion artifact.

### 3.3. Data Validation:

In the Data validation step we will validate the ingested data by using 'schema.yaml' config file. First, we will check if all the expected columns are present in the dataset or not. We will also check if all the numerical and categorical columns are present. If any of column is absent, then training pipeline will stop by raising error.

### 3.4 Exploratory Data Analysis:

Once data validation is done, we will perform EDA, ie. Exploring the data by visualizing the distribution of values in numerical columns of the dataset, and the relationships between expenses and other columns and visualizing the distribution of age, BMI (body mass index). Also, by checking the region wise have any differences in the expenses.

### 3.5 Data Transformation:

In this step we will transform our data for this, first we will create sklearn pipeline for Imputing missing values and then use robust scaler for scaling numeric data. Similarly, we will handle categorical data by using one-hot encoding technique so that the machine learning model can understand it. We will save preprocessor object in data transformation artifact to use it for transforming future input data

### 3.6 Model Building:

After getting transformed data, the next stage will be model training where we will train different Machine Learning models for regression and choose model with higher R2 score as the best model among all other different models.

### 3.7 Model Evaluation:

Here the model evaluation will be done on the newly trained model which we got in previous stage. We can define base accuracy of the model and if model accuracy is higher than base accuracy as well as if it is greater than previously saved best model then only, we will accept newly trained model otherwise it will be rejected.

### 3.8 Model Pusher:

The model pusher stage will get only activated if the newly trained model is accepted at the end of the Model evaluation stage. In this stage we will update the saved model directory and store the latest model as best model and this model will be deployed on cloud for making predictions for user.

### 3.9 Model Deployment:

After completing all the above-mentioned stages, the next task is to deploy the model on cloud platforms. Here we will deploy the model on cloud to serve users. User will give input through our hosted API webpage and our deployed model will make predictions for it.

## 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<br>2. Application is deployed | The Application should load completely for the user when the URL is accessed |
| Verify whether user is able to see input fields on application | 1. Application is accessible<br>2. User is logged into the application | User should be able to see input fields on application |
| Verify whether user is able to edit all input fields | 1. Application is accessible<br>2. User is logged in to the application | User should be able to edit all input fields |
| Verify whether user gets Submit button to submit the inputs | 1. Application is accessible<br>2. User is logged in to the application | User should get Submit button to submit the inputs |
| Verify whether user is getting predicted results on clicking submit | 1. Application is accessible<br>2. User is logged in to the application | User should be presented with predicted results on clicking submit |