**Low-Level Design Documentation (LLD)**



**Project On:**

**Title: Insurance Premium Prediction**

**By: - Vaibhav Bhosale**

**Date: - 31/06/2023**

**Contents**

1. **Abstract**
2. **Introduction**
   1. **Importance of Low-Level Design (LLD)?**
   2. **Scope of (LLD)**
3. **Architecture**
   1. **Architecture Design**
   2. **Data Requirement.**
   3. **Data Collection.**
   4. **About the dataset.**
   5. **Data Description**
   6. **Tools / Software Used.**
   7. **Importing data into databases**
   8. **Exporting data from the databases**
   9. **Data pre-processing**
   10. **Modeling**
   11. **UI integration**
   12. **Data from user**
   13. **Data validation**
   14. **Rendering the results**
4. **Deployment**
5. **Abstract.**

This project represents a machine learning-based health insurance prediction system. Recently, many attempts have been made to solve this problem, as after Covid-19 pandemic, health insurance has become one of the most prominent areas of research. We have used the USA's medical cost personal dataset from kaggle, having 1338 entries. Features in the dataset that are used for the prediction of insurance cost include: Age, Gender, BMI, Smoking Habit, number of children etc. We used linear regression and also determined the relation between price and these features. We trained the system using a 80-20 split and achieved an accuracy of 86%.

1. **Introduction.**

2.1. What is a Low-Level design document?

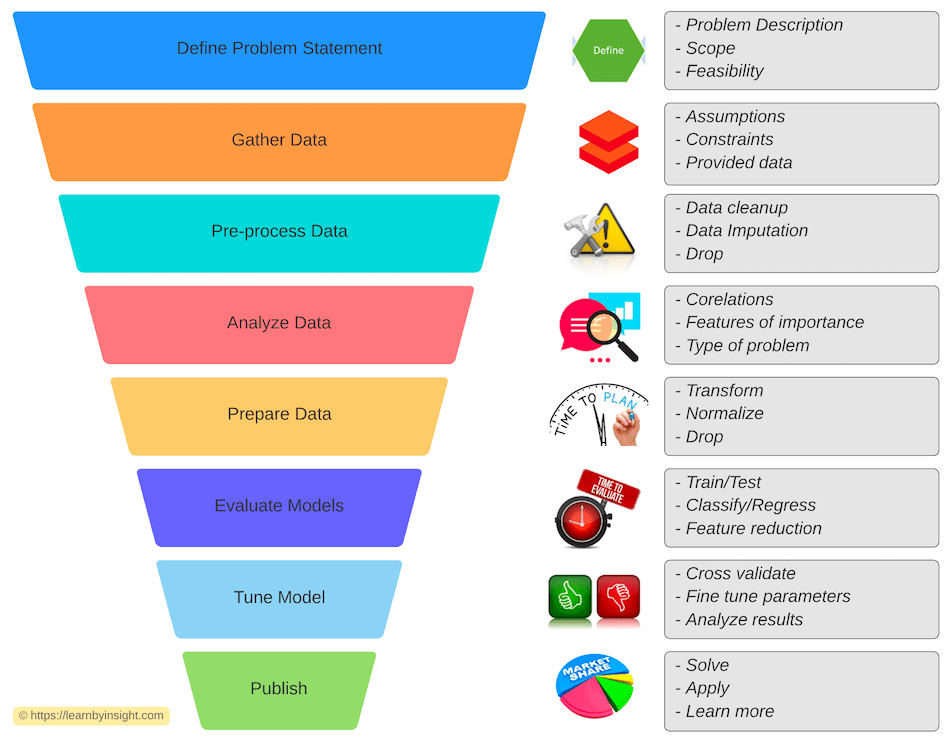
The goal of LLD or a low-level design document (LLDD) is to give the internal

Logical design of the actual program code for Food Recommendation 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.

2.2. Scope of (LLD)

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

1. **Architecture.**



**3.1 ARCHITECTURE DESIGN**

This project is completely based on the life cycle of machine learning, where we will be predicting the insurance premium. The tools used in this project are Python, Pandas, NumPy, Matplotlib, Seaborn, Scikit learn,

For the Version Control system Git was used and for deployment AWS (amazon web service) was used.

**3.2 Data Requirement.**

Whenever we are working on any project the data is completely dependent on the requirement of the problem statement. For this project the problem statement was to create a Hyper tuned Regression machine learning model which can predict the insurance premium

**3.3 Data Collection.**

The data which is used in this project was taken from Kaggle

Dataset link:[*https://www.kaggle.com/datasets/noordeen/insurance-premium-prediction*](https://www.kaggle.com/datasets/noordeen/insurance-premium-prediction)

**3.4 About the dataset.**

File Contain.

- Readme.txt

Insurance.csv - contains 1338 rows and 7 columns

**3.5 Data Description.**

Below are the various attributes for the Insurance Dataset. Please go through the insurance.csv for better clarity of the attributes when reading this document.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |

* **age**: contains the age of the policyholders from various types of policies.
* **sex** : determines the Gender of the population taking a particular policy
* **bmi**: BMI being the most important factor in the domain of Health Insurance, depending upon which the insurance company decides the premium to be paid by the policyholder.
* **children:** It is taken as no. of beneficiaries added for a particular insurance of a policyholder.
* **smoker**: Like BMI, the smoking parameter is an important factor for calculating the premium of a particular policy for a policyholder.
* **region**: Various insurance companies divide their serving in the form of regions, which in turn decides the copay or any extra premium to be given by the policyholder.
* **expenses**: this is the total premium paid by the policyholder for a given given policy schedule.

**3.6 Tools / Software Used:-**

* Python version used for this project 3.8 or higher ( This may get updated and some features might not be available in new version. )
* Python libraries such as NumPy, pandas, matplotlib, seaborn and scikit-learn ( Used for implementation of machine learning algorithms. )
* Jupyter Visual studio code is used as an IDE for writing the code.
* Github is used as the version control system.
* AWS is used for deployment.
* flask is used for Web-page

**3.7 Importing data into the databases.**

MongoDB was used for loading the dataset using Pandas Library was used for training and making the machine-learning model.

**3.8 Exporting data to the database**

The data has been dumped to the MongoDB database..

**3.9 Data Preprocessing**

Have taken the insurance\_main\_dataset.csv file as my dataset.

* All the necessary libraries were imported first such as Numpy, Pandas, Matplotlib, and Seaborn.
* Checking the basic profile of the dataset. To get a better understanding of the dataset.
  + Using Info method
  + Using Describe method
  + Checking for unique values of each column.
* Checking for null values, There are no null values present in our dataset.
* The categorical variable has been encoded with the help of a label encoder.
* After performing all the above steps, the dataset is ready and can be processed into the stage of modelling.

**3.10 Modeling**

* After this the data was split into 2 sets X and y. X contains all the columns except the target column in our case (expenses), and y contains only the Target column.
* Using train test split we first split the dataset into X\_train, X\_test, y\_train, and y\_test.
* Standard scaling has been used to bring the data on the same scale
* The following libraries were imported to create Regression models.
  + from sklearn.ensemble import(AdaBoostRegressor, GradientBoostingRegressor, RandomForestRegressor)
  + from sklearn.tree import DecisionTreeRegressor
  + from xgboost import XGBRegressor
  + from sklearn.linear\_model import LinearRegression

**3.11 UI Integration**

Apache Airflow can be used to monitor the model and predict the new batch dataset. A flask webapp has been created to get the insurance premium amount based on certain inputs like age, BMI, sex, smoker etc.

**3.12 Data from the user**

User can give the required input and get the insurance premium amount as a result in Streamlit webapp. Data from the user is retrieved using the batch file and using Apache Airflow our machine learning model to give the predicted result.

**3.13 Data Validation**

The data which is entered by the user is validated by the data\_validation.py file which is built using inside the components folder under insurance and then this data is transformed using data\_transformation.py under the same path and finally transferred to our model.

**3.14 Rendering the result**

The result for the predictions can be obtained in the flask webapp and also result for our model can be seen in the prediction file generated after running the prediction.

**4. Deployment**

This model is deployed on AWS Elasticbean. The following are the steps to deploy the model on the AWS platform:

* **Create Elastic Beanstalk Environment**

Navigate to Elastic Beanstalk.

Click Create application.

On the Configure environment page, set the following values:

* Application name: Insurance\_Premium\_Prediction
* Platform: Python
* Application code: Sample application

Click Next.

Select Use an existing service role and choose the existing service role that contains RootRole.

For EC2 instance profile, select the existing profile that contains InstanceLoggingProfile.

Click Next.

Under Virtual Private Cloud (VPC), select the listed VPC (from the dropdown).

Under Public IP address, check Activated.

In the Instance subnets section, select all three available subnets.

Click Next.

Under EC2 Security groups, check the non-default security group name.

Click Next.

Click Next.

Review your configurations and click Submit. It will take a few minutes to complete creation.

* **Create an S3 Bucket**

Be sure to download the files listed in the lab instructions for later use in this lab.

Navigate to S3.

Click Create bucket.

In the Bucket name field, type a unique DNS-compliant name.

Click Next.

On the Configure options screen, enable versioning.

Click Next > Next > Create bucket.

Upload the ZIP file that was downloaded earlier in this lab to the new S3 bucket.

Click on the uploaded file name, and copy the key to a text file for use later in this lab.

* **Create an AWS CodePipeline**

Navigate to CodePipeline.

On the Welcome page, click Create pipeline.

If this is your first time using CodePipeline, click Get Started.

On the Choose pipeline settings page, in Pipeline name, enter the name for your pipeline.

In Service role, choose New service role to allow CodePipeline to create a new service role in IAM.

Click Next.

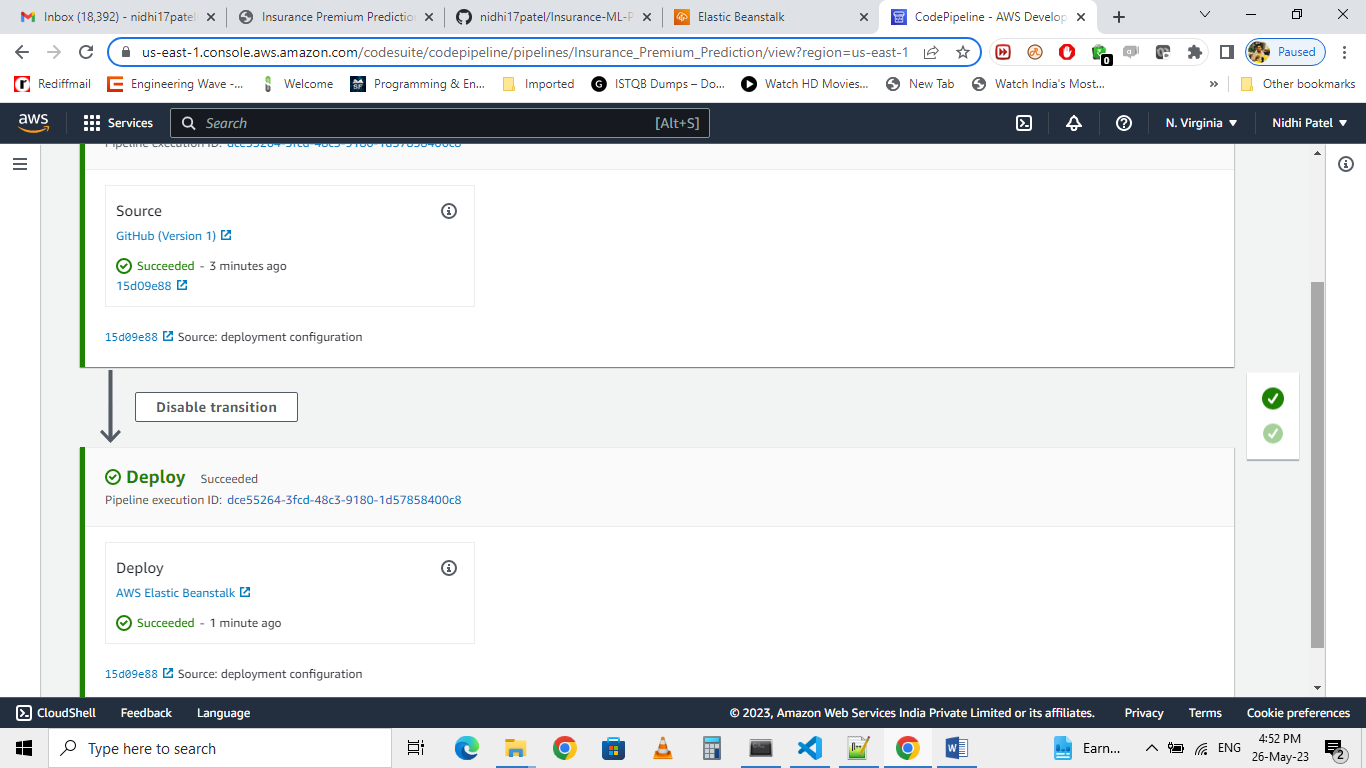
On the Add source stage page, in Source provider, choose S3, specify its required options, and then click Next.

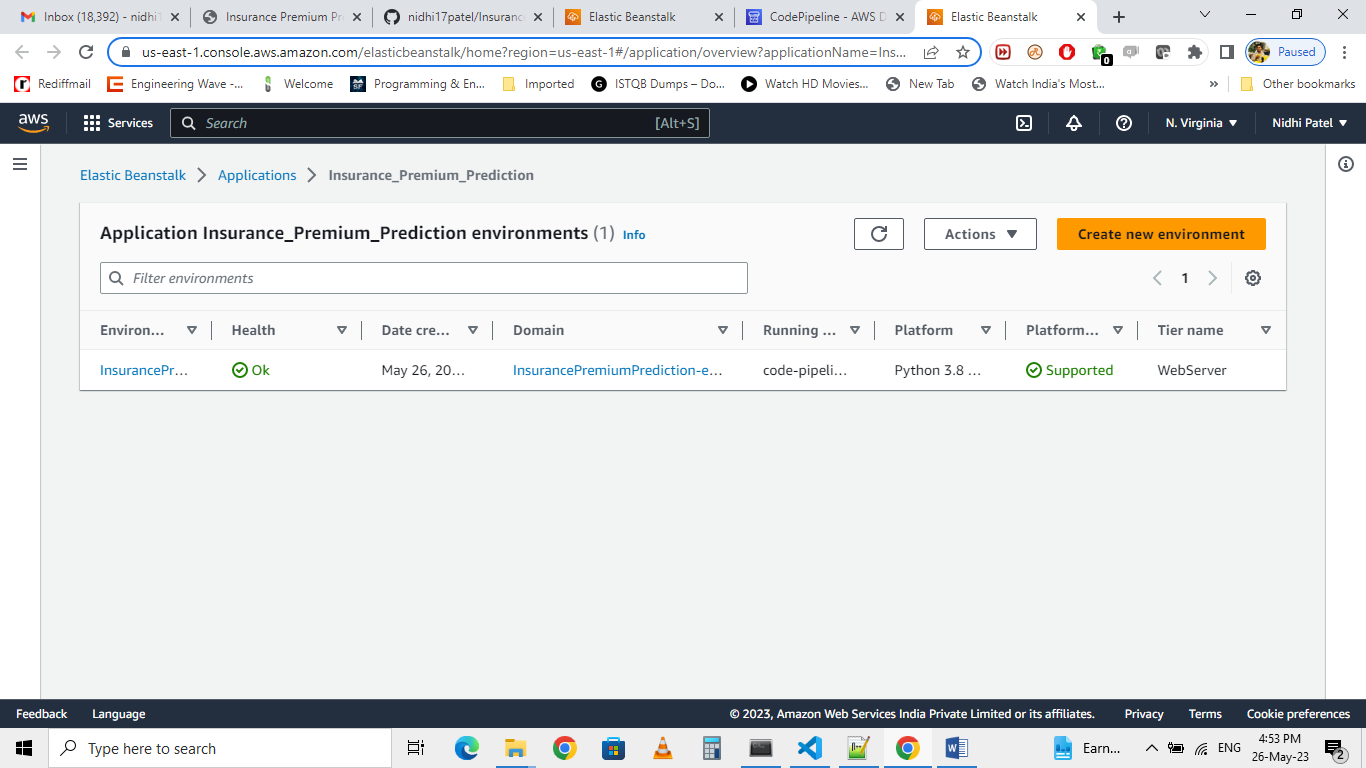
Click Skip build stage.

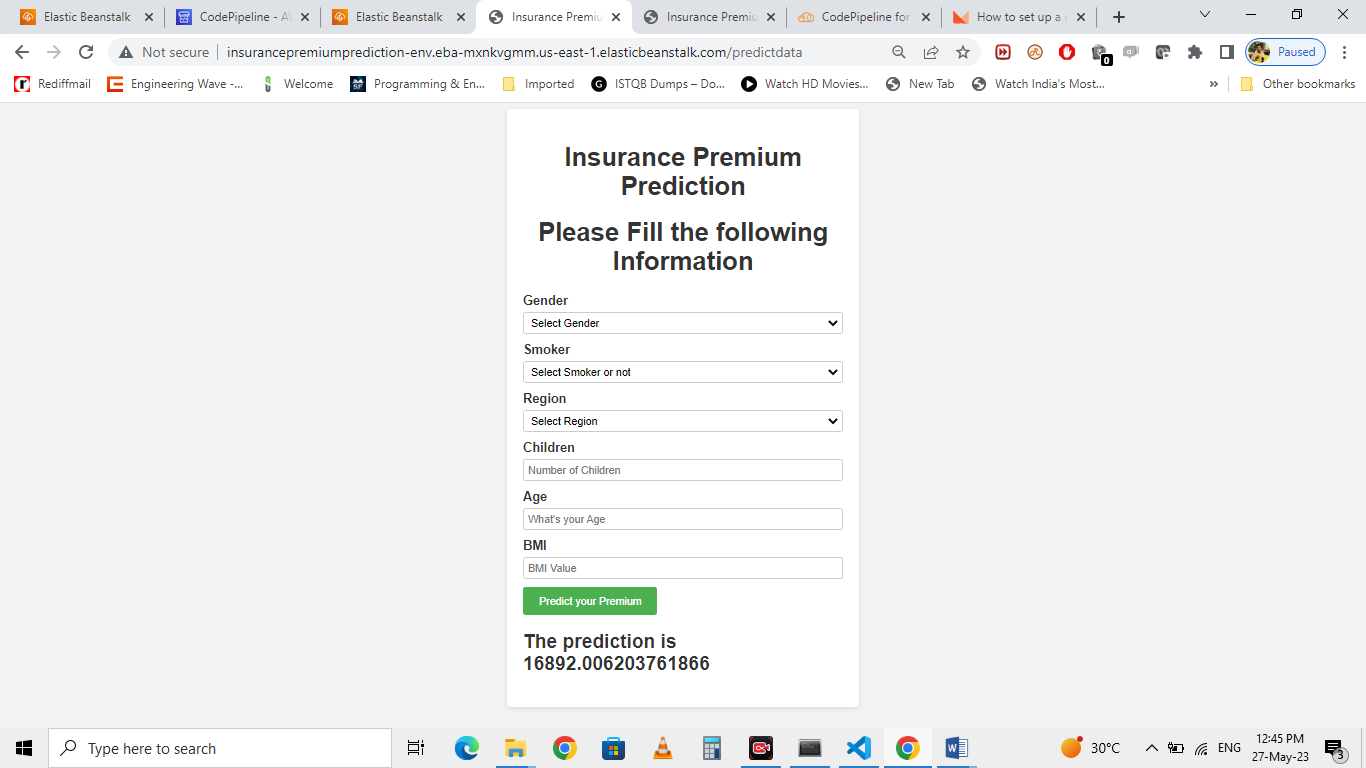
On the Add deploy stage page, set AWS Elastic Beanstalk as the Deploy provider. In Application name, enter or choose the name of an existing Elastic Beanstalk application. In Environment name, enter an environment for the application. Click Next.

On the Review page, review your pipeline configuration, and then click Create pipeline to create the pipeline.

* A web page has been created and deployed using flask







Application is deployed at URL:

**http://insurancepremiumprediction-env.eba-mxnkvgmm.us-east-1.elasticbeanstalk.com/predictdata**