**Insurance Prediction based on Income, Age, Bmi**

Revision Number : 1. 0

Last of Revision: 24/08/2024

# Document Version Control

|  |  |  |  |
| --- | --- | --- | --- |
| Date Issued | Version | Description | Author |
| 22/08/2024 | 1 | Initial HLD — V1.0 | T Bhosale |
| 24/08/2024 | 2 | Updated KPI — V1.1 | T Bhosale |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Contents

Document Version Control

Abstract

1. Introduction

1.1 Why this High-Level Design Document ?  
 1.2 Scope.  
 1.3 Definitions

2. General Description.   
 2.1 Problem Statement  
 2.2 Proposed Solution  
 2.3 Technical Requirements  
 2.4 Data Requirements  
 2.5 Tools Used.

3. Design Details  
 3.1 Process Flow.  
 3.1.1 Model Training and Evaluation  
 3.1.2 Deployment Process  
 3.2 Event Log

3.3 Error Handling

3.4 Performance

3.5 Reusability

3.7 Resource utilization

3.8 Deployment

4. Dashboards

5. Conclusion

Abstract

The Project aims to provide personalized health cost estimates based on individual health situations, enabling customers to focus on the health aspects of insurance policies. By offering these estimates, the project helps individuals compare health insurance plans and perks more effectively, ensuring that their decisions are informed by a clear understanding of projected healthcare costs rather than non-essential factors.

**1. Introduction**

**1.1 Why this High-Level Design Document?**

The purpose of this High-Level Design (HLD) Document is to add the necessary detail to the current project description to represent a suitable model for coding. This document is also intended to help detect contradictions prior to coding, and can be used as a reference manual for how the modules interact at a high level.

The HLD will:

* Present all of the design aspects and define them in detail.
* Describe the user interface being implemented.
* Describe the hardware and software interfaces.
* Describe the performance requirements.
* Include design features and the architecture of the project
* List and describe the non-functional attributes like
  + Reliability
  + Maintainability
  + Portability
  + Reusability
  + Application compatibility
  + Resource utilization
  + Serviceability

##### 1.2 Scope

The HLD documentation presents the structure of the system, such as the database architecture, application architecture (layers), application flow (Navigation), and technology architecture. The HLD uses non-technical to mildly-technical terms which should be understandable to the administrators of the system.

##### **1.3** Definitions

*Term UGV Description*

*Database Collection of all the information*

*monitored by this system.*

*IDE AWS*  Amazon Web Services

2. General Description

**2.1 Product Statement**

The goal of this project is to give people an estimate of how much they need based on

their individual health situation. After that, customers can work with any health

insurance carrier and its plans and perks while keeping the projected cost from our

study in mind. This can assist a person in concentrating on the health side of an

insurance policy rather than the ineffective part.

**2.2 PROPOSED SOLUTION**

The proposed solution involves developing a machine learning model that analyzes

various health factors and personal medical data to generate accurate cost estimates for

healthcare needs. The process begins by collecting individual health data, including

medical history, current health status, lifestyle factors, and other relevant metrics. This

data is then processed and analyzed using the machine learning model, which has been

trained on a diverse dataset of healthcare costs and outcomes.

The model generates a personalized health cost estimate, which reflects the likely

expenses an individual might incur based on their specific health situation. This estimate

serves as a guideline for users as they explore different health insurance plans and perks

offered by various carriers. The solution emphasizes transparency and accuracy, enabling

users to prioritize health-related benefits in their insurance selection process.

**2.3 Technical Requirements**

* **Data Management:** Secure storage and management of health data using a

robust database, ensuring compliance with privacy standards (e.g., HIPAA).

* **Machine Learning:** Develop and train models using libraries like TensorFlow

or scikit-learn for accurate health cost predictions.

* **Backend/API:** Implement a backend (e.g. Flask) with RESTful APIs to handle data processing and predictions.
* **Frontend:** Create an intuitive user interface for data input, cost estimates, and insurance plan comparison.
* **Insurance Integration:** Integrate external APIs for accessing health

insurance plan details and comparison.

* **Cloud Deployment:** Use cloud services (e.g., AWS) for hosting, scalability, and CI/CD pipelines for continuous deployment.
* **Compliance:** Ensure the solution meets healthcare regulations (e.g., HIPAA, GDPR) and includes user consent mechanisms.
* **Monitoring:** Implement tools for system monitoring, logging, and regular updates to maintain performance and accuracy.

**2.4 Data Requirements**

For the successful development and training of our machine learning model, we require datasets provided in CSV or similar formats that allow for easy integration and processing. The dataset must include specific columns such as:

* **Age:** The individual's age, which is a critical factor in determining healthcare needs and costs.
* **BMI (Body Mass Index):** A measure of body fat based on height and weight, influencing health risk predictions.
* **Gender:** The gender of the individual, which can impact health outcomes and associated costs.
* **Smoking Status:** Indicated as 'Yes' or 'No,' this column captures whether the individual is a smoker, a significant factor in health risk assessments.
* **Number of Children:** The number of dependents or children an individual has, which can affect insurance needs and healthcare cost
* **Region** : The region where people lives.
* **Expenses** : This column represents the target variables in our dataset. It captures the total healthcare costs or expenditures associated with each individual, based on the various factors such as age, BMI, gender, smoking status and number of children.

2.5 Tools used

Python programming language and frameworks such as NumPy, Pandas, Scikit-learn, flask are used to build the whole model.



* **Python** is used coding and implementation.
* **Aws** is used for deployment of the model.
* For visualization of plots, **Matplotlib**, **Seaborn** and **plotly** are used.
* **Pandas** are used for dataframe creations.
* **MySQL/MongoDB** is used to retrieve, insert, delete and update the database**.**
* **Github** is used as version control system.

3. Design Details

3.1 Process Flow

1. **Data cleaning:** The process of identifying and correcting inaccuracies or inconsistencies in data to ensure quality and reliability.

2. **Data Ingestion:** The process of importing and integrating data from various sources into a centralized system for analysis.

3. **Data Transformation:** The process of converting data into a suitable format or structure for analysis and modeling, including normalization, aggregation, and feature engineering.

4. **Model Building:** The process of creating statistical or machine learning models using the transformed data to make predictions.

5. **Model Testing:** The process of evaluating the performance of a model using test data to ensure it generalizes well to new, unseen data.

3.2 Deployment Process

Deploying an ML model on AWS elastic Beanstalk. Below are steps in details:

* **Prepare Your Model**: Train your machine learning model locally or in the cloud and save it in a format like .pkl (e.g. using scikit-learn)
* **Create a Web Application**: Develop a web application (e.g., using Flask) that serves your model predictions via REST API endpoints. Ensure the application can load the model and handle inference requests.
* **Package the Application**: Package your web application along with any necessary dependencies into a zip file. Ensure all necessary environment configurations (e.g., environment variables, configuration files) are included.
* **Create an Elastic Beanstalk Environment**: Log into the AWS Management Console and create a new Elastic Beanstalk environment. Choose the appropriate platform (e.g., Python) for your application.
* **Deploy the Application**: Upload your packaged application to Elastic Beanstalk. Elastic Beanstalk will handle the deployment process, including provisioning resources like EC2 instances, load balancers, and auto-scaling groups.
* **Configure Environment Variables**: Set up environment variables in the Elastic Beanstalk console for configurations like database connections, API keys, or model paths.
* **Test the Application**: Once deployed, test your application by sending requests to the API endpoint to ensure that the model is correctly loaded and can make predictions.
* **Monitor and Scale**: Use Elastic Beanstalk's monitoring tools to keep track of application performance, and configure auto-scaling if needed to handle variable traffic.
* **Set Up CI/CD (Optional)**: Integrate your Elastic Beanstalk environment with CI/CD tools (like AWS CodePipeline) to automate deployments whenever there are updates to the application code or model.
* **Secure the Application**: Ensure your application is secure by configuring SSL certificates, setting up IAM roles and policies, and using AWS services like WAF (Web Application Firewall) if necessary.

3.2 Event log

The system should log every event so that the user will know what process is running internally.

3.3 Event Handling

If errors be encountered, an explanation will be displayed as to what went wrong? An error will be defined as anything that falls outside the normal and intended usage.

4. Performance

* **Performance Metrics**: Key metrics like response time, throughput, and resource usage.
* **Expected Load**: Anticipated user load and peak conditions.
* **Scalability**: How the system will scale to handle increased load.
* **Latency and Response Time**: Target response times and optimization strategies.
* **Resource Management**: How resources will be allocated and monitored.
* **Fault Tolerance**: Strategies for maintaining performance during failures.
* **Testing**: Load and stress testing approaches.
* **Optimization**: Techniques for performance improvement.

4.1 Reusability

The code written and the components used should have the ability to be reused with no problems.

**4.2 Application Compatibility**

The different models for this project will be using python as an interface between them. Each model will have its own task to perform, and it is the job of the python to ensure proper transfer of information.

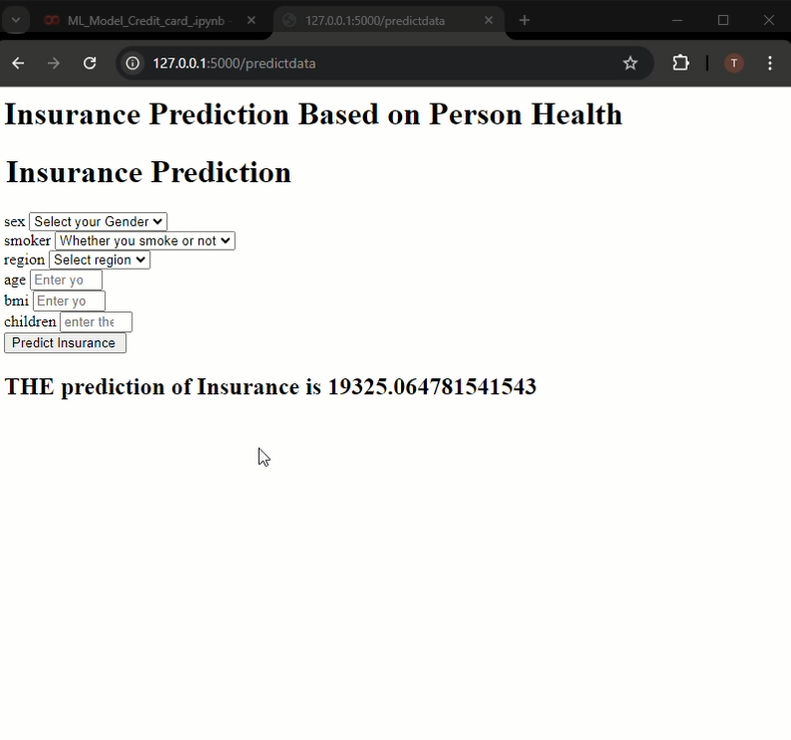
#### **4.3 Resource Utilization**

When any task is performed, it will likely use all the processing power available until that function is finished.

#### **4.4 Deployment**



**5 Final Output on web page**



**6 Conclusion**

The health cost estimation tool developed in this project provides individuals with a personalized estimate of their healthcare expenses based on their specific health situations. By offering a tailored projection, the tool empowers users to make more informed decisions when selecting health insurance plans. This approach allows customers to focus on the health-related aspects of insurance policies, rather than solely on financial considerations.

The insights provided by the tool can significantly enhance the decision-making process, ensuring that individuals are better prepared to choose insurance plans that align with their needs and preferences. Ultimately, this empowers users to make choices that support their overall well-being and financial security, while also facilitating a more focused and efficient interaction with health insurance carriers.