Software Design Specification Document: Machine Learning Medical Insurance Predictor

Revision 1.0

Joshua Desmond, Javan Huett, Ellie Brinkman, Fatim Diabate, and Ibrima Njie

Table of Contents

1. Introduction

1.1 Purpose

This software aims to offer medical insurance companies and their customers a readily accessible app that enables users to predict medical insurance quotes. With the ever-growing applications and implementations of Machine Learning (ML) and other subfields of AI into the Healthcare and Finance industries, utilizing ML models and algorithms for this medical insurance predictor allows for swifter and more accurate predictions. This medical insurance predictor not only provides medical insurance companies with critical insight into their business practices and premium pricing, but it can also be an invaluable tool for the company's current and potential customers. In addition, the predictor generates a swift and personalized insurance premium catered to the individual.

1.2 System Overview

This software utilizes machine learning through Python's Scikit-learn library to train and use the XGBoost algorithm to predict insurance quotes based on characteristics like age, BMI or Body Mass Index, smoking status, etc. The Scikit-learn library supplies different functions for various Machine Learning (ML) models and algorithms like XGBoost that allow programmers to train the models based on data and produce new predictions using these trained models. The XGBoost algorithm is an efficient machine learning algorithm where a prediction is improved by a series of decision trees that learn from each other. Developing an interactive user interface through the Python library Tkinter enables users to add and submit their characteristics through fillable boxes and view their predictions and other data. Tkinter is a library that allows programmers to create user interfaces, similar to mobile and desktop apps' new screens and windows. The Tkinter library has an assortment of different buttons, menus, windows, icons, and more that allow for great customization and function.

1.3 Design Map

This insurance predictor is an accessible, easy-to-use application that allows users to quickly and accurately predict their insurance premiums. The system architecture resembles that of a Model-View-Controller or MVC, which is often used when there are multiple avenues to interact with and view the data. MVC is a type of system architecture that aims to separate the presentation and interaction from the system data. This architecture has three main components: the Model – which manages the data and operations on that data, the View – which defines and manages how the data is presented, and the Contoller – which manages user interaction like the clicking of a button, for instance. In essence, the GUI or the Graphical User Interface acts like the controller – managing the user's interactions with the buttons and menus. The GUI is an interface that allows users to view and interact with the program through various buttons, menus, windows, and icons. At the same time, we have functions that define how the data is displayed within the GUI. Lastly, the different ML models, created data structures, and operations constitute the Model component. The project has two main interfaces: the main data window and the predictor window. The main data window allows viewing the data frame and metrics, as well as reading about the different data analyses regarding the data fields. This window also allows users to view the accuracy scores of various ML models. The predictor window allows users to input their data characteristics like age, BMI, and smoking status, and then view their predicted insurance charge.

1.4 Acronyms and Definitions

GUI – Graphical User Interface; an interface which allows users to directly interact with and view the program's inputs, outputs, and components through the integration of buttons, menus, windows, and various icons.

XGBoost – An algorithm that optimizes the gradient boosting algorithm, which sequentially builds several decision trees or rather weak learners, that aim to minimize the errors of the previous learner through loss functions.

ML – Machine Learning; a subfield of Artificial Intelligence that focuses on automating computer learning by discovering patterns, making predictions, and classifying, for instance.

MVC – Model-View-Controller; a type of system architecture that focuses on isolating the system data from the presentation and interaction; has three main components: the Model, View, and the Controller

NaN – Not a Number; this is a value that is used to indicate that the data is missing or there was an undefined calculation.

Tkinter – A Python library that allows programmers to create an interface with windows, buttons, and menus to allow users to interact with a program

2. Design Considerations

2.1 Assumptions

Users have knowledge of basic app usage with buttons, menus, and fillable fields, however, they lack the understanding and knowledge of programming and computer functions.

2.2 Constraints

Constraints on the development, as well as the predictor itself, were the limited meetings between team members, often restricted to class time to communicate. In addition to this, uploaded data must follow the exact structure of the default dataset and be the same file type (.csv), otherwise the predictor will not be able to produce an output. Another constraint is that imported datasets cannot be too large – as they can jeopardize the integrity of the model and its predictions due to noisy data or the data limit of the model. Additionally, another constraint of this program was the user's understanding. When assuming the user has no technological background or knowledge, the GUI must explain and describe data analytics and avoid programming or technology-heavy functions on the user's part.

2.3 System Environment

The Medical Insurance Predictor software utilizes Python 3.10, as well as several other imported Python libraries such as Scikit-learn, Tkinter, Pandas, Seaborn, Matplotlib, to name a few. As mentioned previously, the Scikit-learn library allows programmers to use and train various Machine Learning models and algorithms. Meanwhile, the Tkinter library allows programmers to create user interfaces, which allow users to interact with the program through buttons, menus, windows, etc. The Pandas library enabled us to create a frame of the data, as well as clean and encode the data. In other words, the library allowed programmers to manipulate and correct errors, inconsistencies, and more. Both Seaborn and Matplotlib are graph-based libraries that create and display graphs based on the data.

2.4 Risks and Volatile Areas

No volatile areas have been identified in the application. However, there are several risks within the application. For instance, there is a risk that with large datasets, the data quality, as well as the accuracy of the predictions, can be degraded as the model may not be suited for the size. In addition to this, processing and training models with larger datasets may result in longer wait times, reducing the overall user experience. Another risk for this application is that missing fields or columns in the provided dataset could produce errors, as the program is specifically manipulating and viewing this missing data. Although, it is important to note that with additional alterations, in addition to added functionalities and features come with the risk of new bugs and potential issues that can make the development process longer, make the app too complex for the user – hindering their ability to use the app's features, and degrade the software.

3. Project Development

3.1 Development Methodology

The development methodology or architecture employed in the development of this medical insurance predictor utilizes several aspects from several different software engineering structures. For instance, most of the programming is done in pairs – with a pair responsible for creating functions for training the model, cleaning the data, and utilizing the model, while another pair designs and implements these functions into a simplistic GUI. This allows members to not only build off of each other's work, decreasing the workload, but also allows others to aid and provide support when encountering errors and issues. This specific aspect is an example of Agile Software Architecture: Extreme Programming. In addition to Extreme Programming's pair programming, we also utilize plan-driven development, especially in the development of the GUI, using a step-by-step approach. In this case, for instance, we begin with planning and identifying which features should be implemented, like graphs, specific buttons, and fields for accessing .csv files, then implementing, testing, and assessing them. Overall, the used methodology greatly resembles that of the Incremental Developmental model, which is a mixture of both agile and plan-driven developments that simultaneously specify, develop, and validate the software between versions.

3.2 Development Challenges

3.2.1 Data Cleaning and Visualization Issues

The majority of the issues during early development were errors with importing and calling libraries and modules, specifically with the Seaborne library and the ArbitraryOutlierCaper module from the features_outliers library, despite pip installing these libraries. Some solutions came from reinstalling the library/module in the filepath, using an alternative library, and restarting the kernel. In addition to these small issues and errors, we also faced a challenge mapping the data from the dataset where NaNs, or Not A Number, appear instead of an expected numerical value. NaNs in data indicate that the values are missing or are the result of undefined calculations, often signifying an error with the data. When training ML models, it is important to avoid NaNs, as they can skew the data and produce inaccurate results, or can result in an error when the model requires numerical data. Through many trials and errors, we discovered that when skewing the data, we were rounding (to make it more visually appealing). By not rounding the data, we successfully returned numbers rather than NaNs.

3.2.2 GUI Development Issues

Similar to the issues during the alpha phase of data cleaning and visualizing, we also had minor errors with modules not being recognized, but it was easily remedied by reinstalling the library or reloading the kernel. We also ran into an issue when implementing buttons that, when clicked, would train the dataset and output the accuracies using several models, which was left unfunctional in the prototype. The solution instead was to directly output the values on the data window. After the second version of this app, there was an issue with making the predictions – we had accidentally hardcoded the prediction feature, which was originally planned to be implemented through the CSV. However, the import CSV feature instead imported the dataset that the model was trained on. This was a major problem as one of the main features of the predictor was individualized predictions based on the user's characteristics and qualities. The solution was to create a second window, which would make the app less cluttered and enhance the visuals and usage of the app. This second window had fillable text boxes and dropdown boxes for the different qualities of the user; once clicking on the "OK" button, the prediction would be shown below the inputs.

3.3 ML Model Considerations

When selecting a model or algorithm for this project, we explored several different algorithms and models: Linear Regression Model, Support Vector Regression Model, Random Forest Regressor, and Extreme Gradient Boosting, commonly known as XGBoost. The Linear Regression ML Model models a linear relationship between two variables through the method of least-squares, whereas the Support Vector Regression Model extends the linear regression line of best fit with a defined margin of tolerance. Random Forest is a powerful ensemble-based regression method that constructs multiple decision trees and aggregates their outputs for improved accuracy. Finally, XGboost is an optimized algorithm that sequentially builds several decision trees or weak learners that aim to minimize the errors of the previous learner through loss functions.

3.3.1 ML Model Selection

For the predictions, we selected the XGBoost algorithm to predict the insurance charges based on the individual's characteristics. Through testing the accuracy of the aforementioned models and algorithms, we found that XGBoost had the highest accuracy scores. In addition to this, XGBoost is robust – accurately capturing complex interactions and relationships among variables such as age, BMI, and smoking habits, as well as efficiently handling large datasets with potential missing or noisy data points. This makes XGBoost well-suited to real-world medical applications, providing more reliable, scalable, and precise predictions than traditional Scikit-learn regression models, like those models previously mentioned.

4. Project Evolution

4.1 Software Revisions & Feedback

Key revisions and feedback for the initial prototype included fixing the functions of several buttons, removing the gradient boosting button as it was redundant, as well as improving the simplicity, usability, and finally improving the understanding of the user. One such small, but essential step toward improving the app’s usability and the user’s experience is through adding a supplementary description of graphs, describing their purpose and importance to the user. Another revision was added once an issue was discovered, which was how users made predictions. This revision included making a separate window where users could input their information like BMI, Age, etc., using textboxes and dropdown menus, in addition to displaying the prediction. This new window makes it more accessible, as users no longer need to scroll to the bottom of the cluttered main window, instead being able to see the prediction window from the beginning.

4.2 Future Revisions

Future Revisions of this Medical Insurance Predictor could include optimizing the display time of the various graphs after the dataset is imported, as it takes a couple of seconds to view, and providing an example dataset structure to demonstrate how the dataset should be structured. Another revision could be having a default dataset to make predictions from the start, while also keeping the option to upload another dataset. Additionally, other potential revisions could include expanding the expected dataset to consider additional factors other than BMI, region, age, smoking status, etc., streamlining the layout, as well as customizing the interface to make it more personalized. This customization could also be utilized by the user with he use of buttons, sliders, and other menus to customize the window's brightness, font size, and more – all of which add to the app's accessibility.

5. High-Level Design

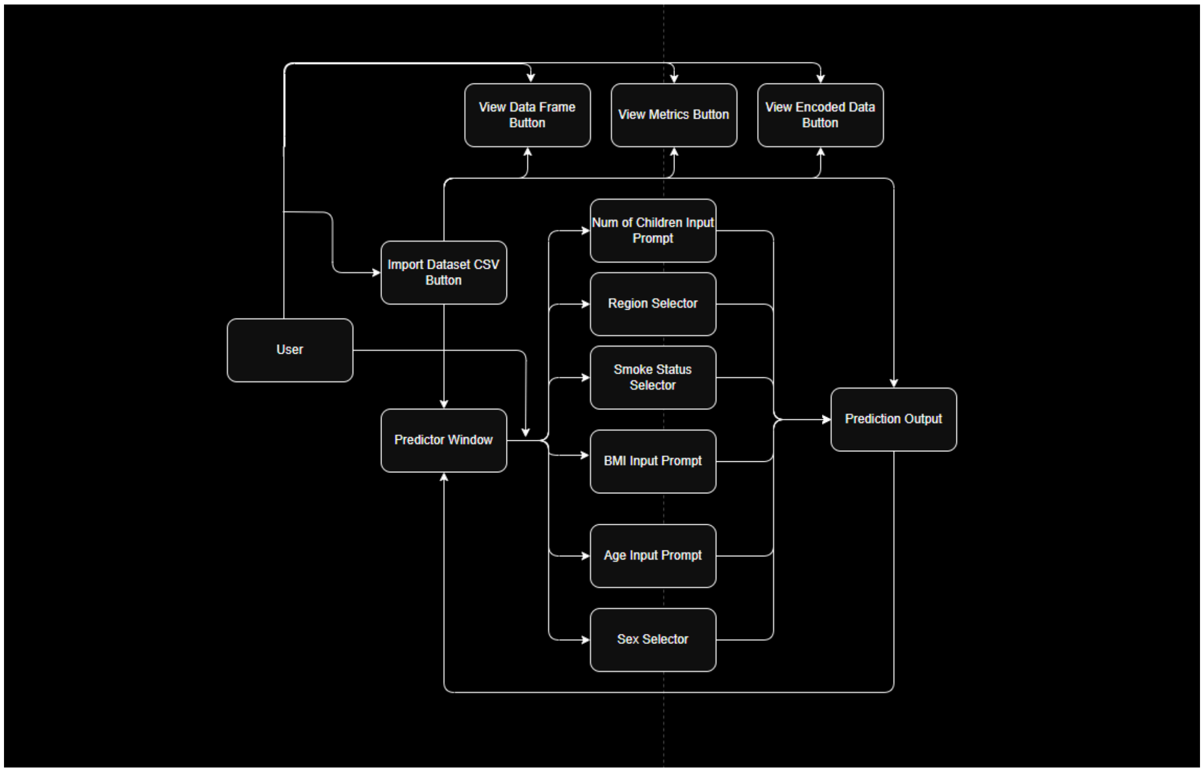


Figure 5: Pictured above is a diagram of the high-level use case, displaying the interactions of both the users and high-level components of this application. The User interacts with the "Import Dataset CSV" button, which then allows users to interact with the newly opened predictor window and several other buttons: " View Data Frame", "View Metrics", and "View Encoded Data". These windows and buttons use the data provided by the dataset button. Furthermore, the predictor window has several textboxes and drop-down menus that the user can input – all of which the results of the training based on this inputted data are displayed in the Predictor Output, which the user can view via the Predictor Window.

6. Low-Level Design

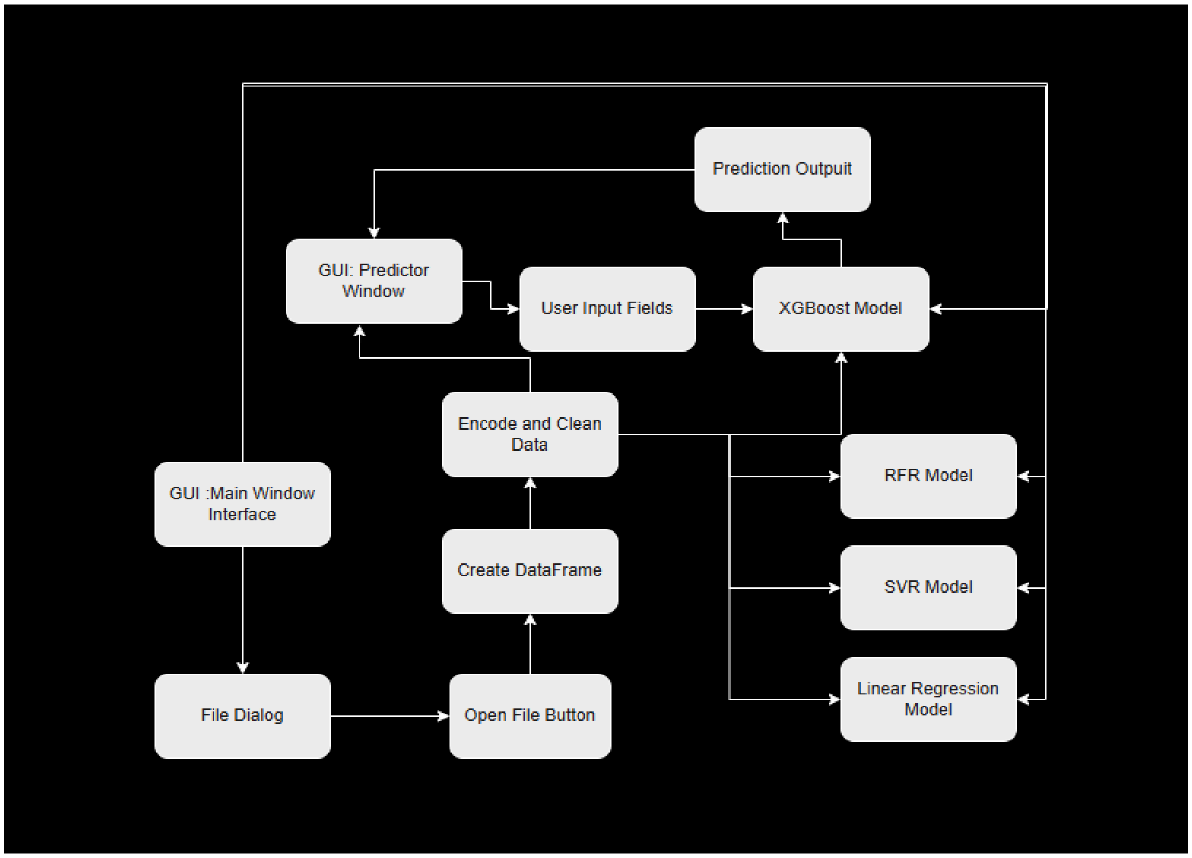
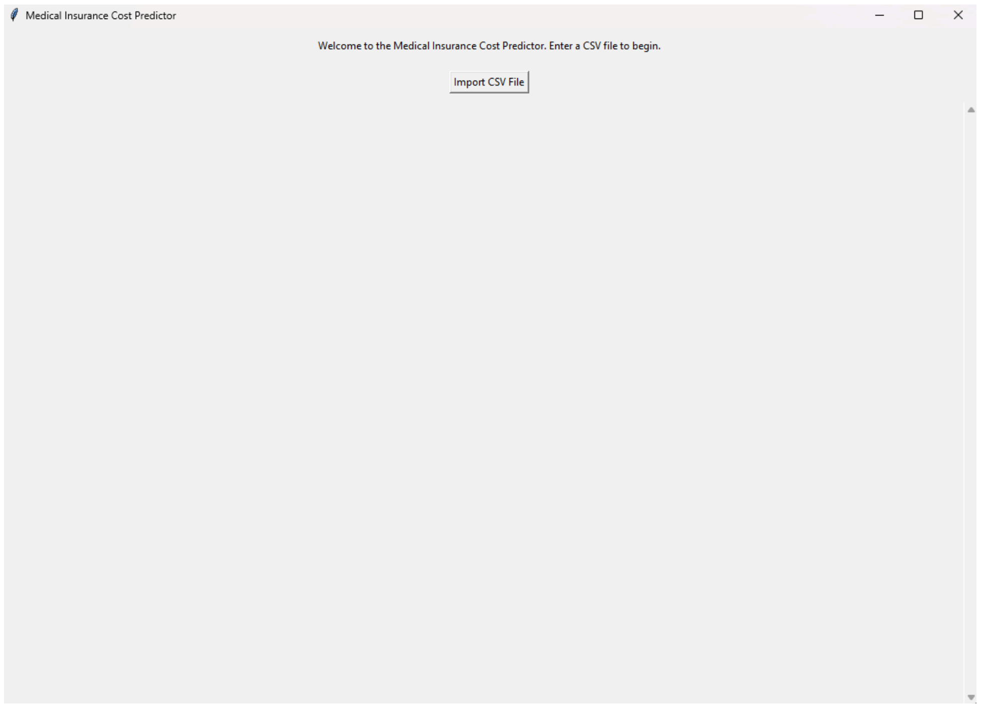


Figure 6: Pictured above is the low-level diagram visualizing the different algorithms, models, major functions, data structures, and objects and their interactions and relationships within the application that the GUI utilizes behind the scenes. This diagram has two main GUI windows: the Main Window and the Predictor Window. The Main Window interface uses a file dialog to allow users to select the dataset file on the user's computer. Once the file is selected, the data within the file is made into a dataframe, where the data is cleaned and encoded. In other words, the data is modified to correct any inconsistencies or errors, as well as map different non-numerical categories like smoking status as numbers (this allows ML models to understand better and process this information to identify patterns). Following this, the Predictor Window is opened, and the XGBoost, Random Forest Regression (RFR), Linear Regression, and Support Vector Regression (SVR) models are trained with the cleaned data. However, the XGBoost also receives input data from the user via the fields provided in the Predictor Window, which allows the model to produce a prediction that is displayed in the Predictor Window GUI.

7. User Interface Design

7.1 Screen 1: Data Window



Welcome to the Medical Insurance Cost Predictor. Enter a CSV file to begin.

Figure 7.1.1: This is the main window, where users import a CSV file. After importing a CSV file, different statistics, graphs, and data descriptions appear on this scrollable window. This window has other buttons related to the data, like view dataframe, view metrics, and view encoded data.

7.2 Screen 2: Predictor Window

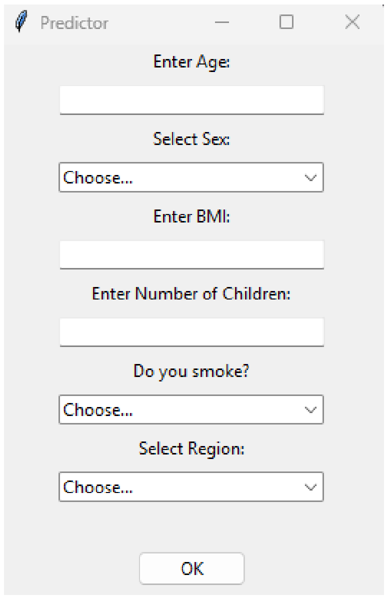


Figure 7.2.1: Pictured is the second window of the app, which appears after the dataset is imported. Here, the user inputs and selects the different choices and values of the appropriate fields, and then the window displays the insurance prediction.

Appendix A: Project Timeline

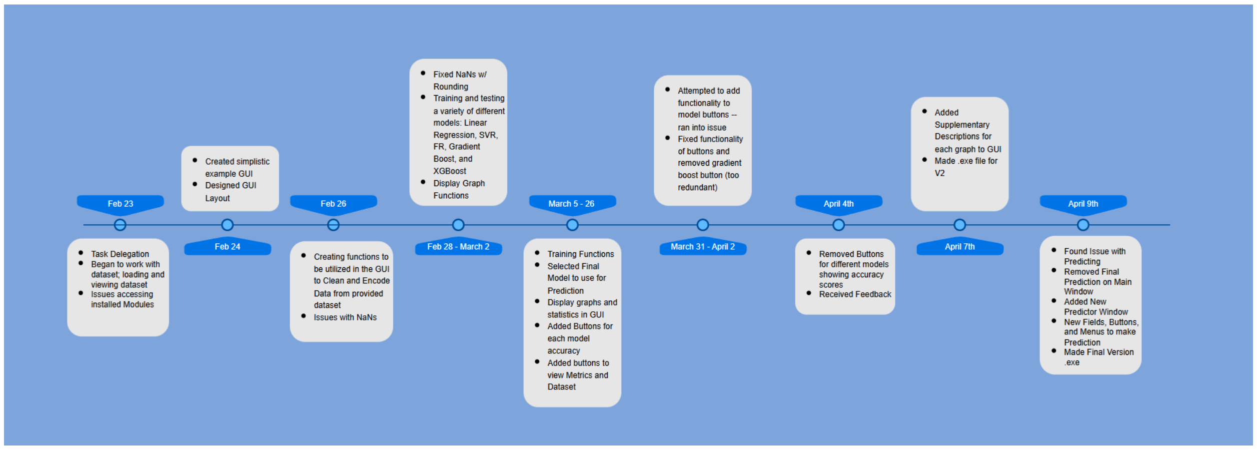
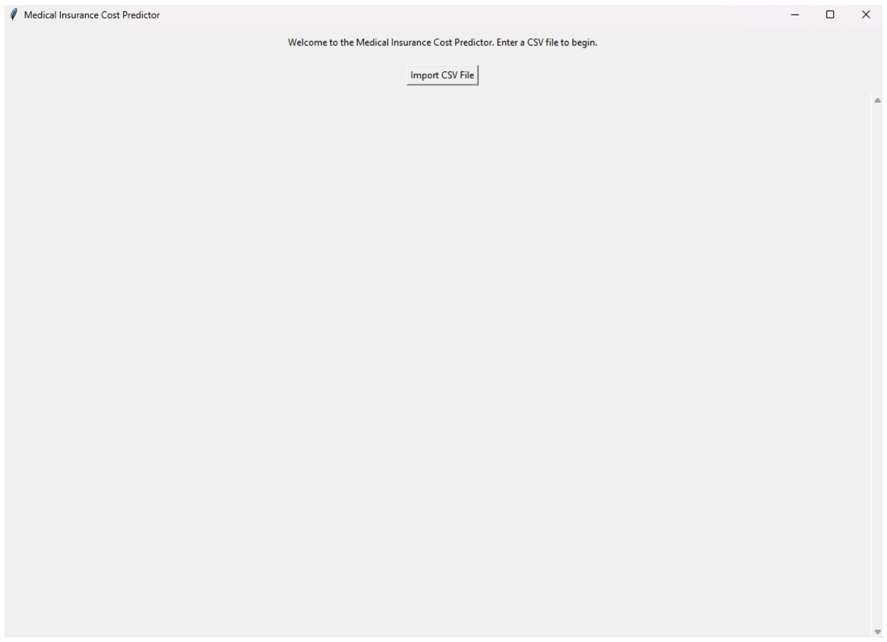

Feb 23
Task Delegation
Began to work with dataset; loading and viewing dataset
Issues accessing installed Modules
Feb 24
Created simplistic example GUI
Designed GUI Layout
Feb 26
Creating functions to be utilized in the GUI to Clean and Encode Data from provided dataset
Issues with NaNs
Feb 28 - March 2
Fixed NaNs w/ Rounding
Training and testing a variety of different models: Linear Regression, SVR, FR, Gradient Boost, and XGBoost
Display Graph Functions
March 5 - 26
Training Functions
Selected Final Model to use for Prediction
Display graphs and statistics in GUI
Added Buttons for each model accuracy
Added buttons to view Metrics and Dataset
March 31 - April 2
Attempted to add functionality to model buttons -- ran into issue
Fixed functionality of buttons and removed gradient boost button (too redundant)
April 4th
Removed Buttons for different models showing accuracy scores
Received Feedback
April 7th
Added Supplementary Descriptions for each graph to GUI
Made .exe file for V2
April 9th
Found Issue with Predicting
Removed Final Prediction on Main Window
Added New Predictor Window
New Fields, Buttons, and Menus to make Prediction
Made Final Version .exe

Figure 1: Generalized, Approximate Project Timeline marking major progress, changes in functionality, and brief mentions of bugs and issues that appeared through development.

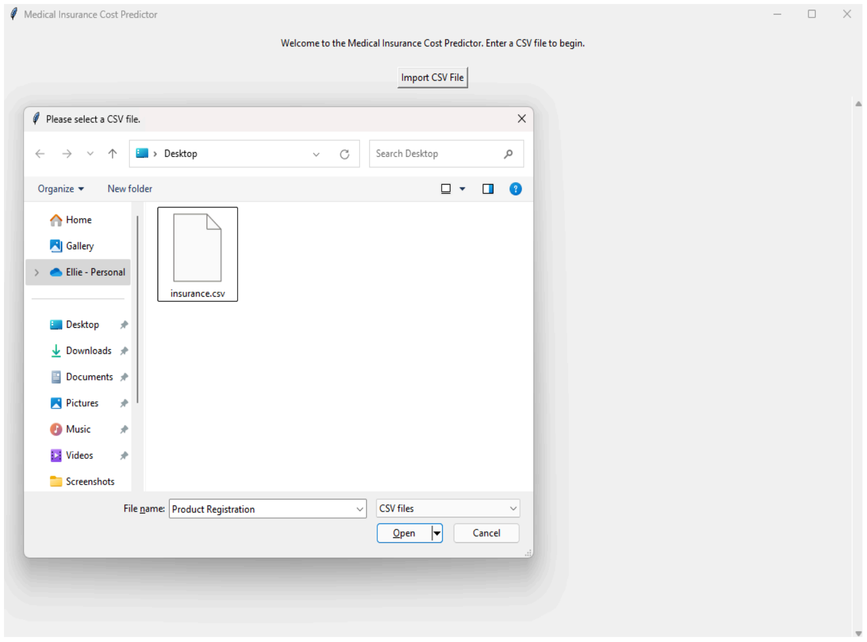
Appendix B: User Manual

1. Opening, Loading, and Viewing Data



Welcome to the Medical Insurance Cost Predictor. Enter a CSV file to begin.

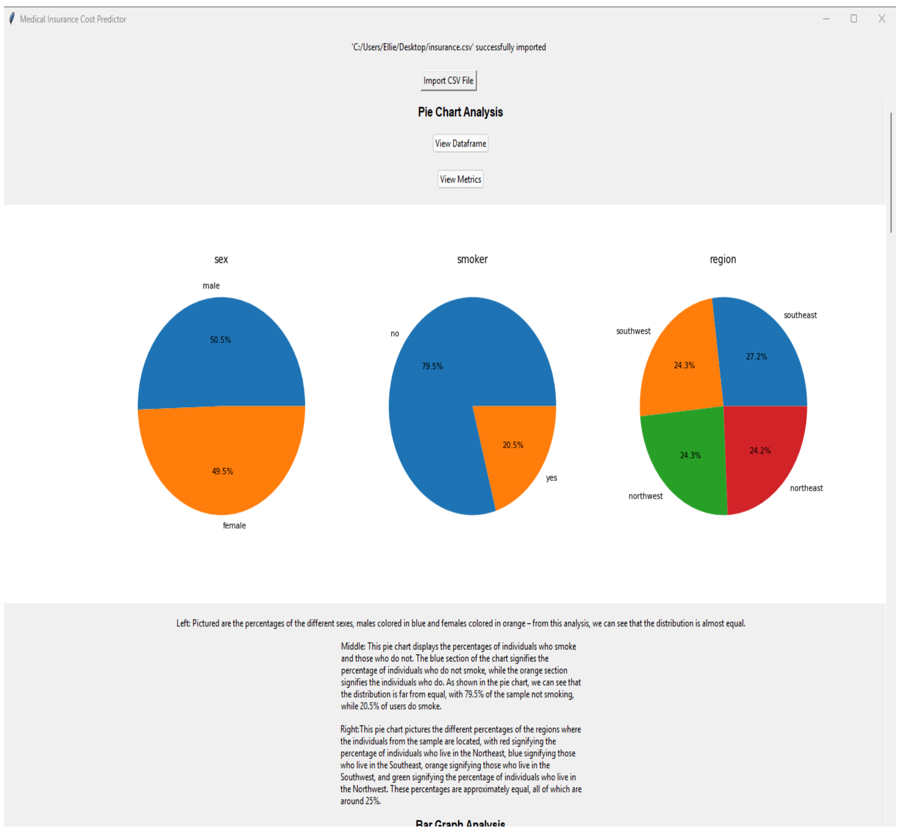
Users should first click the “Import CSV File” Button. This will prompt another window to open to select the CSV file.



Welcome to the Medical Insurance Cost Predictor. Enter a CSV file to begin.

File name: Product Registration

Once the dataset in the form of a CSV file is selected, press the “Open” button on the bottom right



'C:/Users/Ellie/Desktop/insurance.csv' successfully imported

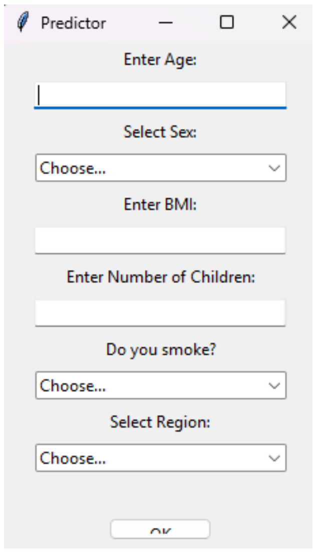
Pie Chart Analysis

Left: Pictured are the percentages of the different sexes, males colored in blue and females colored in orange – from this analysis, we can see that the distribution is almost equal.

Middle: This pie chart displays the percentages of individuals who smoke and those who do not. The blue section of the chart signifies the percentage of individuals who do not smoke, while the orange section signifies the individuals who do. As shown in the pie chart, we can see that the distribution is far from equal, with 79.5% of the sample not smoking, while 20.5% of users do smoke.

Right:This pie chart pictures the different percentages of the regions where the individuals from the sample are located, with red signifying the percentage of individuals who live in the Northeast, blue signifying those who live in the Southeast, orange signifying those who live in the Southwest, and green signifying the percentage of individuals who live in the Northwest. These percentages are approximately equal, all of which are around 25%.

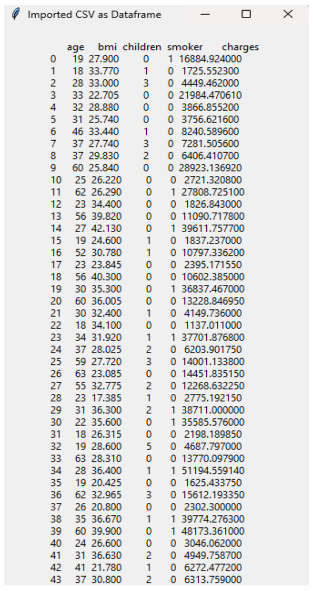
Bar Graph Analysis



Once the CSV has been opened a loaded, the main window will update completely with graphs and other analyses, as well as a new window to make predictions. Users can scroll down the main window, exploring various graphs, proportions, and analyses complete with detailed descriptions.

Another window will also pop up where Users will be able to input their characteristics and view their prediction. This will be detailed later in the next section: Making Predictions.

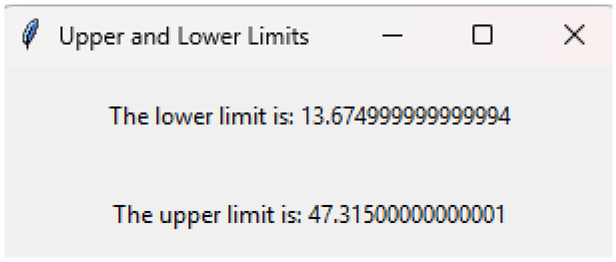
The main window now has three additional buttons: "View DataFrame" (top middle), "View Metrics" (top), and "View Encoded Data" (bottom button below graphs)..



```
    age    bmi  children  smoker       charges
0    19  27.900        0       1  16884.924000
1    18  33.770        1       0   1725.552300
2    28  33.000        3       0   4449.462000
3    33  22.705        0       0  21984.470610
4    32  28.880        0       0   3866.855200
5    31  25.740        0       0   3756.621600
6    46  33.440        1       0   8240.589600
7    37  27.740        3       0   7281.505600
8    37  29.830        2       0   6406.410700
9    60  25.840        0       0  28923.136920
10   25  26.220        0       0   2721.320800
11   62  26.290        0       1  27808.725100
12   23  34.400        0       0   1826.843000
13   56  39.820        0       0  11090.717800
14   27  42.130        0       1  39611.757700
15   19  24.600        1       0   1837.237000
16   52  30.780        1       0  10797.336200
17   23  23.845        0       0   2395.171550
18   56  40.300        0       0  10602.385000
19   30  35.300        0       1  36837.467000
20   60  36.005        0       0  13228.846950
21   30  32.400        1       0   4149.736000
22   18  34.100        0       0   1137.011000
23   34  31.920        1       1  37701.876800
24   37  28.025        2       0   6203.901750
25   59  27.720        3       0  14001.133800
26   63  23.085        0       0  14451.835150
27   55  32.775        2       0  12268.632250
28   23  17.385        1       0   2775.192150
29   31  36.300        2       1  38711.000000
30   22  35.600        0       1  35585.576000
31   18  26.315        0       0   2198.189850
32   19  28.600        5       0   4687.797000
33   63  28.310        0       0  13770.097900
34   28  36.400        1       1  51194.559140
35   19  20.425        0       0   1625.433750
36   62  32.965        3       0  15612.193350
37   26  20.800        0       0   2302.300000
38   35  36.670        1       1  39774.276300
39   60  39.900        0       1  48173.361000
40   24  26.600        0       0   3046.062000
41   31  36.630        2       0   4949.758700
42   41  21.780        1       0   6272.477200
43   37  30.800        2       0   6313.759000
```

To the left is an example output of when the "View DataFrame" button is clicked.

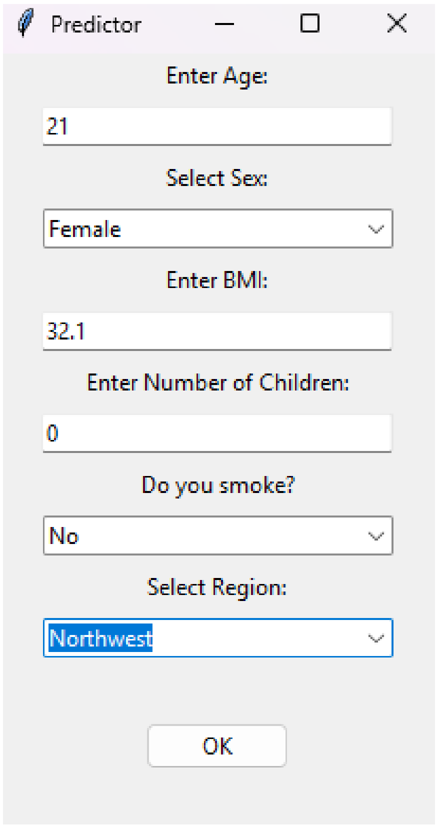
Below is an example output of when the "View Metrics" button is clicked.



The lower limit is: 13.674999999999994

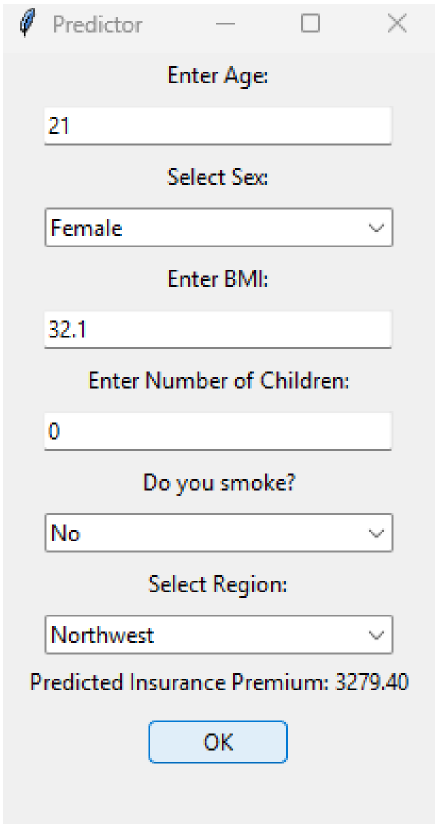
The upper limit is: 47.31500000000001

2. Making Predictions



Enter Age: 21
Select Sex: Female
Enter BMI: 32.1
Enter Number of Children: 0
Do you smoke? No
Select Region: Northwest

Users will input data by filling textboxes and using drop-down menus. Once done, Users will click the "OK" button, which will display the predicted insurance charge as shown to the right, for example.



Enter Age: 21
Select Sex: Female
Enter BMI: 32.1
Enter Number of Children: 0
Do you smoke? No
Select Region: Northwest
Predicted Insurance Premium: 3279.40