**Machine Learning Web App using Bayesian Networks**

***Author: Tudor-Sorin Ciutacu***

***Date: 30 June 2023***

***Hellenic Mediterranean University***

Table of Contents

[1. General description of the project 3](#_Toc139062709)

[2. ML Implementation 3](#_Toc139062710)

[2.1 Data analysis and preprocessing 4](#_Toc139062711)

[2.2. Creating the model 5](#_Toc139062712)

[2.3. Evaluating the model performance 6](#_Toc139062713)

[3. User Interface (UI) implementation 8](#_Toc139062714)

[3.1 Frontend 8](#_Toc139062715)

[3.2 Backend 9](#_Toc139062716)

[4. Bayesian Network Diagram 10](#_Toc139062717)

[5. Architecture 11](#_Toc139062718)

[6. Packages 12](#_Toc139062719)

[7. Conclusions 13](#_Toc139062720)

[8. Bibliography 13](#_Toc139062721)

**Machine Learning Web App using Bayesian Networks**

# **General description of the project**

This project proposes to solve a real world problem, that is detecting *heart disease* in patients. As we all know, early detection can be crucial in saving patients’ lives, by taking the appropriate measures at the right time. This Bayesian Network model implemented in Problog, a probabilistic programming language that allows us to define and reason about uncertain information, performs inference based on user imputed data and makes predictions using its learned parameters from the labeled dataset. The algorithm used for learning the parameters and thus generating the trained model is called LFI (Learning From Interpretations) and was introduced in the following paper: *B. Gutmann, et al.***1**

The project can be divided into 2 parts: Machine Learning model and Web API. Let’s briefly describe each component.

* **ML model using Bayesian Networks**

The ML model is responsible for predicting the presence or absence of Heart Disease in a patient, given some parameters as input. The actual model is written in Problog and it makes use of the Bayesian Networks, the Problog code is generated using a Python script that forms 2 files consisting of the untrained model and the evidence needed for parameter learning. The generation of the trained model file will be covered in the implementation part of the report.

* **Web API**

The Web API can be broken down further in 2 components: the backend which is implemented using the Flask framework and the frontend component implemented using the React framework. They communicate between each other using HTTP requests and the goal is to send the user imputed data to the backend which processes it and feeds it to the ML model and then sends back a response containing the model’s prediction, which will be then displayed on the user’s interface.

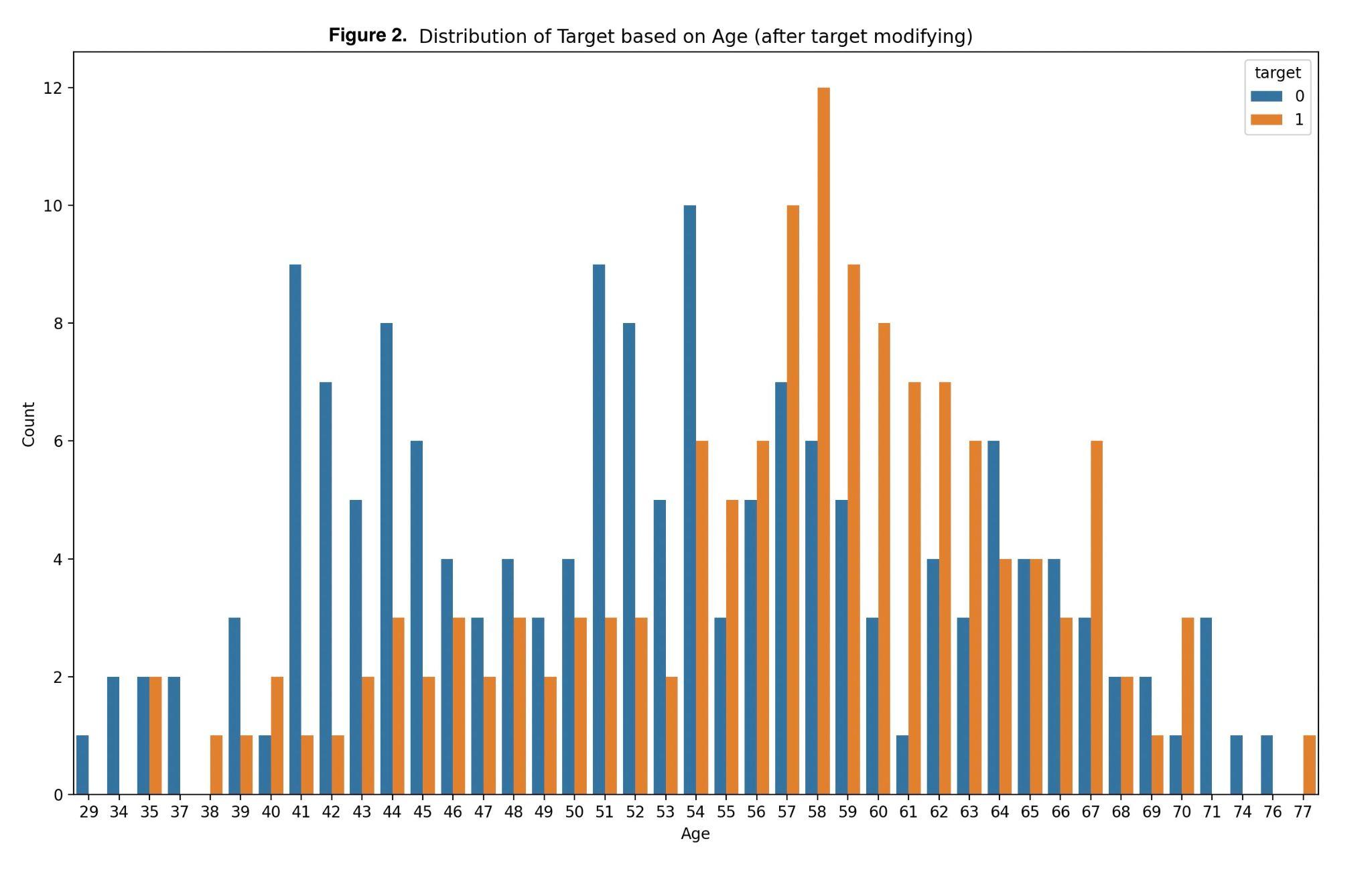
The 2 main components are integrated flawlessly and work in parallel. Let’s delve into details about the implementation of each component.

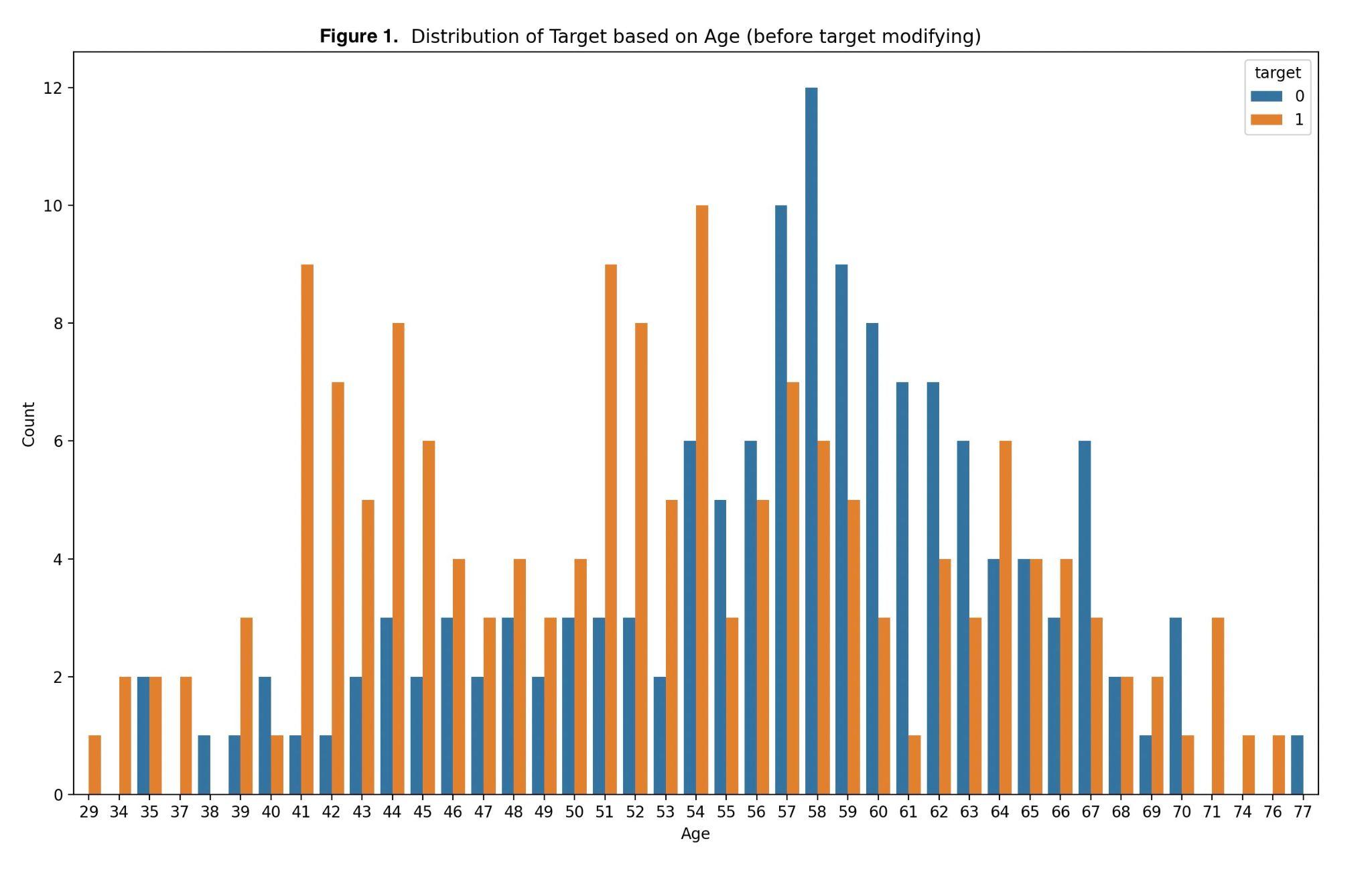
# **ML Implementation**

For the project’s structure I chose to go with 4 folders: ***data*** in which I stored all the files related to the dataset and the performance results, ***src*** directory which contains the Python files along with the Problog ones, ***backend*** a directory made for storing the Flask component, and ***venv*** the folder which has all the necessary files for the virtual environment. I also made a requirements.txt file that stores all the dependencies of the project. In order to run my project, it is recommended to activate the virtual environment and to ensure that everything runs correctly, it is needed to run the following command in a terminal window: *pip install -r requirements.txt* before anything else.

## **2.1 Data analysis and preprocessing**

First things first, I downloaded the dataset from the following web address**2**, and then loaded it into my Python program *preparing\_dataset.py* designed for preprocessing the data. The first thing I observed was that out of the 1025 rows the dataset had 723 duplicated rows, so I removed them and then I proceeded to drop the lines that contained inaccurate data. I then checked for missing values and made sure to fill them up with the mean of that column, in case there were any. Even though in this case there were no missing values, it is a good practice to do this. After that I checked for outliers and found 15 of them, but I decided it's best not to remove them, as they could be representative for the dataset. Next I checked the distribution of the target values, and I made sure the dataset is balanced, but after plotting the distribution of target based on age (Fig. 1), I observed something wrong with the dataset. The target value 1, which represents having heart disease, was dominant among the younger people, which made me realize that the target is reversed. After fixing the target values, by inverting them, I plotted again the distribution of target based on age (Fig. 2), in order to visualize the changes. I saved the modified dataset into a file called *heart\_cleaned.csv* which will be later used in the Python script that generates the model.

Cleaning up the dataset before doing any Machine Learning operations on it is a crucial step, as a bad dataset would highly affect the model’s performance.



## **2.2. Creating the model**

Generating the model is done through a Python script. The newly modified *heart\_cleaned.csv* dataset is loaded into the dataframe, a variable called *program* is initialized with the probabilistic facts, the scope of this variable is to store the entire Problog program. The structure of the Problog program consists of probabilistic facts and rules. After that, the whole dataframe is being parsed and for each label, that is the column name, the values are being encoded into categories using bins and labels, for instance for the *trestbps* column the values range from *94* to *200* so a binning operation is performed as follows: any value between *94* and *119* will be labeled as *l (low),* any value in the *120* to *139* interval will be labeled as *m (medium)* and any value between *140* and *200* will be labeled as *h (high).*

After the binning operations the rules are constructed assembling Problog clauses and using a binary string that generates all the *2^n* possible combinations of parameters, *n* being the number of parameters used, excluding the target. After each iteration a new rule is appended to the *program* string, resulting in the entire Problog program being stored in the string variable.

The next step is to divide the dataset into training samples and testing samples, the ratio chosen was 90-10, save them into separate files: *training\_data.csv* and *testing\_data.csv*, and to start generating evidence statements using the training dataset. Similarly to the Problog program generation, a string variable *evidence* is used to store the entire training evidence. The statements are generated in pairs of 2, one for the *true* statement and one for the *false,* by iterating through the entire training set and for each row parsing the labels.

The 2 strings *program* and *evidence* are then written into 2 different Problog files: *untrained\_model.pl* and *training\_evidence.pl,* and in order to generate the trained model we need to perform parameter learning using the EM algorithm. In the first iteration of EM, each probability is initialized to a set value, specified in the probabilistic facts of the form *t(0.5,A,B).* I used the *0.5* value as an initializer. To start generating the trained model we need to run the following command in the terminal:

*problog lfi untrained\_model.pl training\_evidence.pl -O trained\_model.pl*

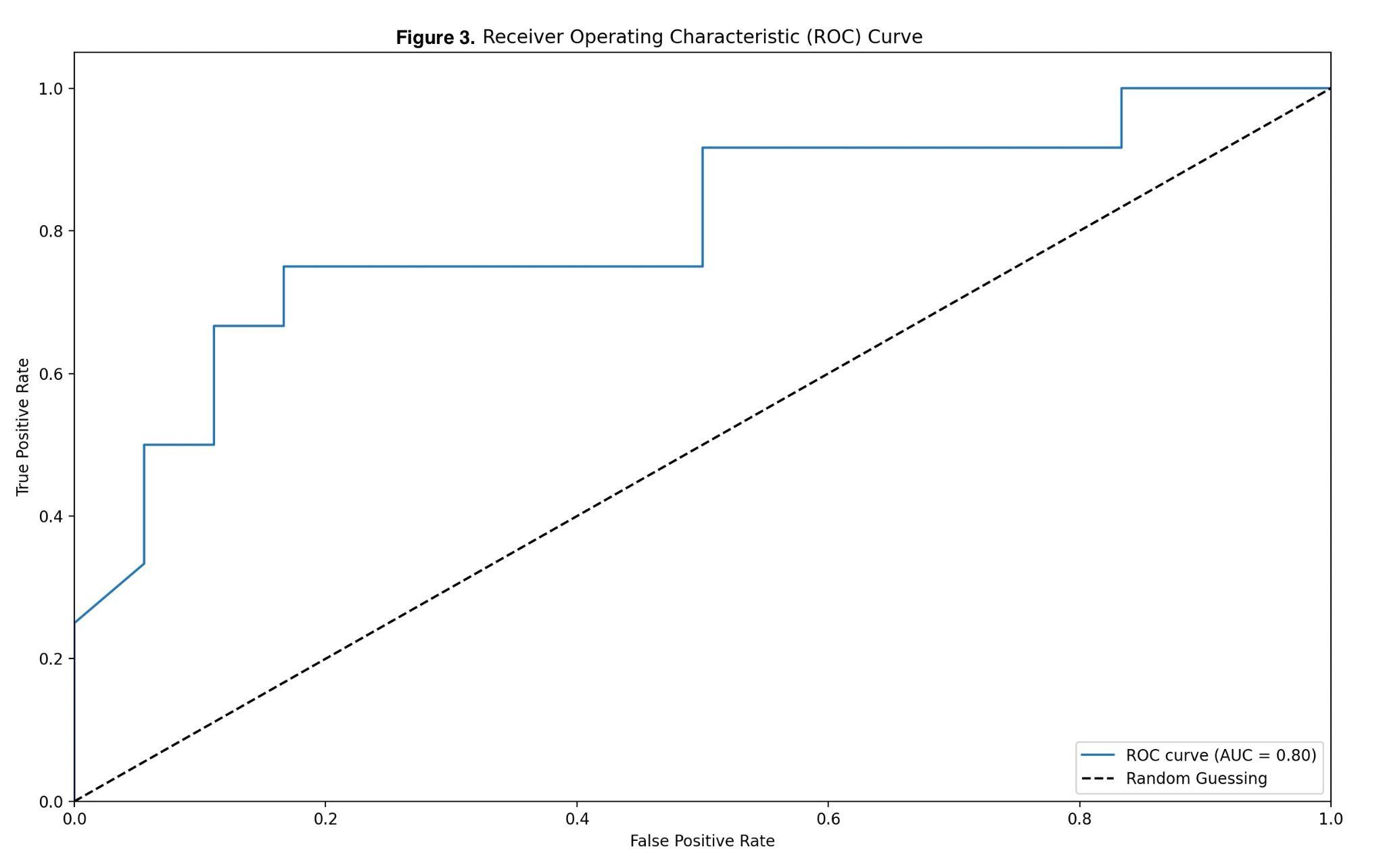
The outcome of this command is the *trained\_model.pl* file generated in our ***src*** folder.

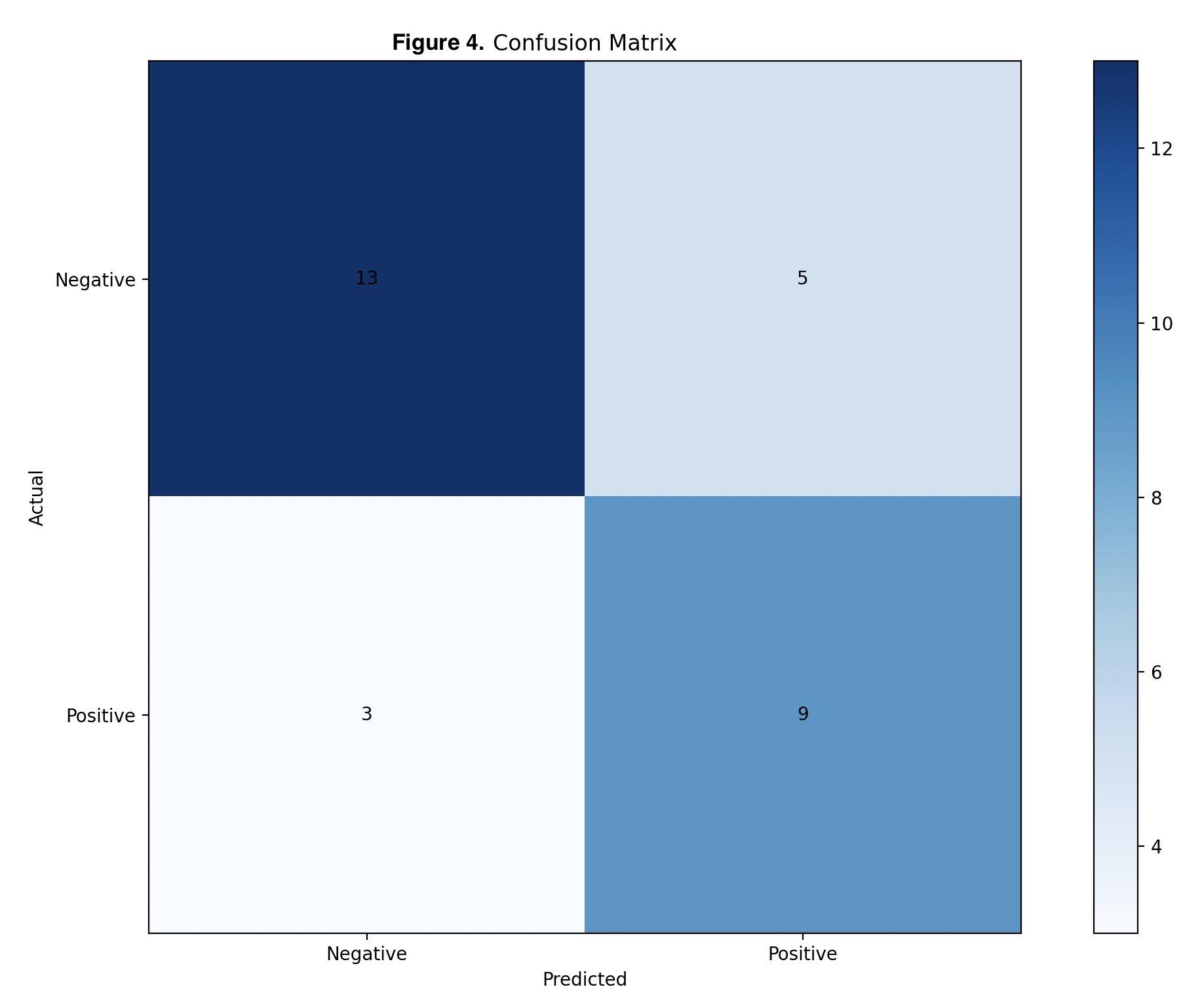
## **2.3. Evaluating the model performance**

Having the trained model generated we can start testing its performance**3**. First we load the model into a string variable *program\_string* and the testing dataset into a dataframe, then we define a dictionary for storing the predictions, and a list for storing the actual classes. We iterate through the dataframe, and for each line we create a query with the current data, then using our *program\_string* we append the query to it and store it into a variable *p* which we then use to generate the *results* object by calling the *evaluate()* function on it.

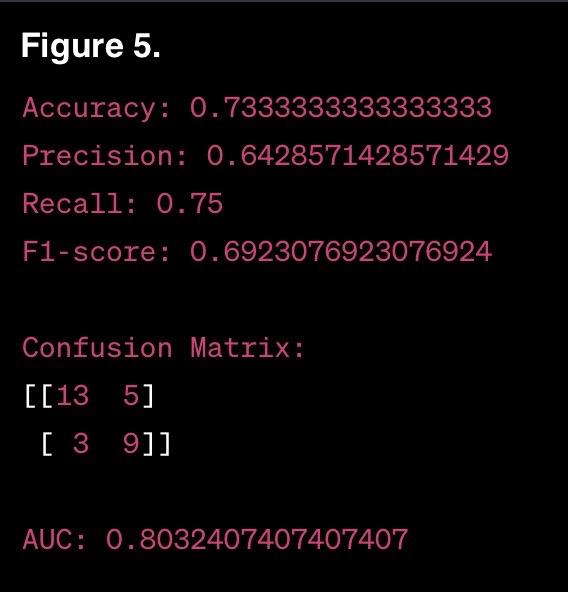
After obtaining the *results* object we iterate through it and for each result we store the floating point number generated by the evaluation into our predictions dictionary, that is the probability of the target being *yes* and the probability of the target being *no,* in which case the former represents having *heart disease* and the latter, not having *heart disease*. The results are also stored into a variable *str\_results* which will later be stored into a txt file called *testing\_results.txt.*

The next step is to calculate performance metrics such as accuracy, precision, recall, F1-score and to obtain the false positive rate along with the true positive rate from calculating the ROC curve and also the AUC score. After calculating these metrics we plot the ROC curve using the Python package matplotlib.pyplot that can be observed in Figure 3. For a thorough evaluation we also calculate and plot the confusion matrix (Fig. 4) to better visualize the predictions.





Last but not least, the computed metrics are stored into the *testing\_results.txt* file (Fig. 5)



# **User Interface (UI) implementation**

## **3.1 Frontend**

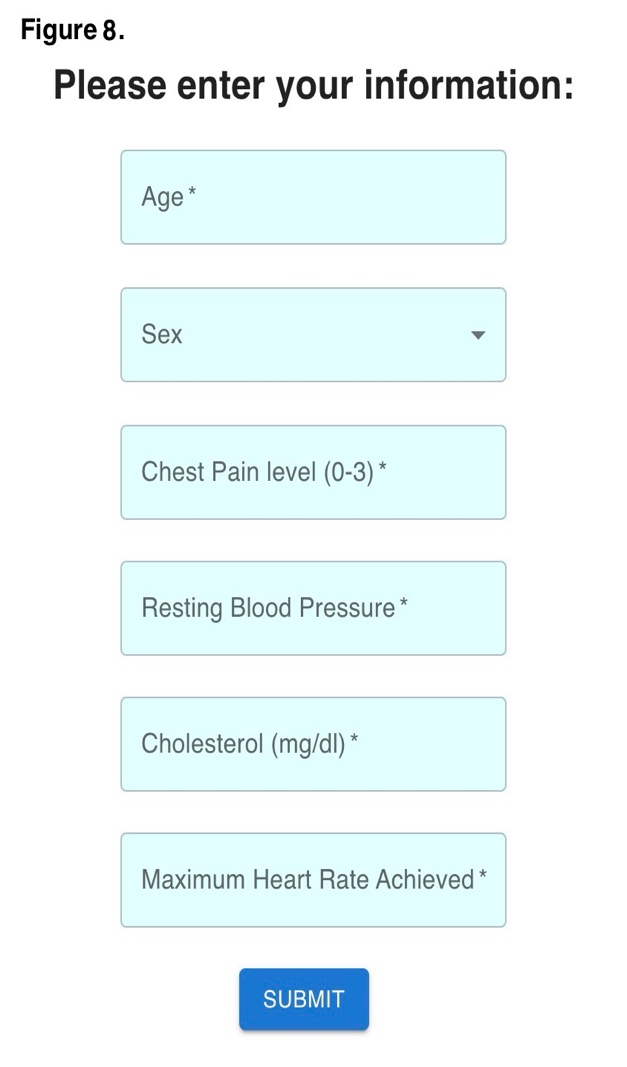
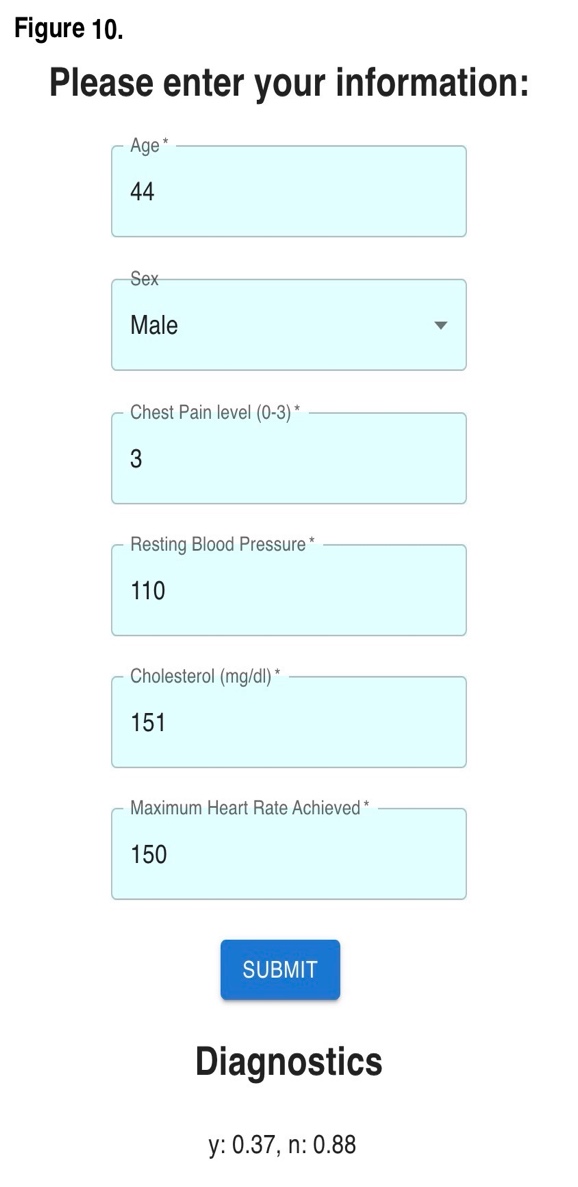
The frontend side of the application is implemented using the React framework and it consists of three Web pages: the ***Home page*** (Fig. 6), which contains 2 buttons helping the user navigate to the right page, ***Software Engineer page*** or ***User page***.

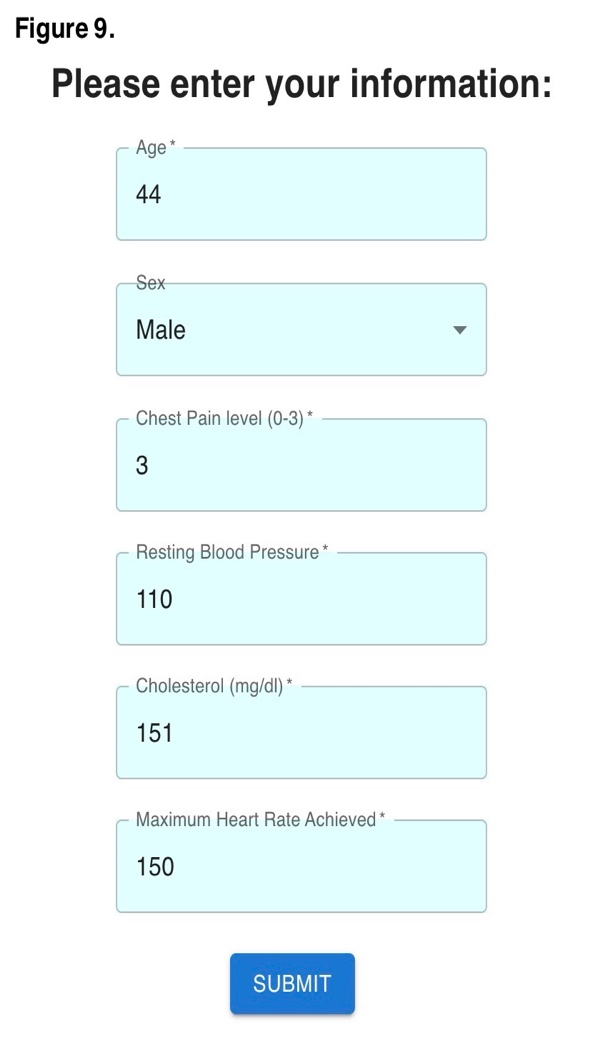
The ***Software Engineer page*** (Fig. 7) displays 3 buttons as follows: **Prepare Dataset**, a button which will trigger the command *python3 preparing\_dataset.py* to be run, thus preparing the dataset and generating the initial plots helping us visualize the data, **Create Model**, a button which will trigger the command *python3 creating\_model.py* to be run, thus generating the untrained model and the training evidence, and finally **Evaluate Performance**, a button which will trigger the command *python3 testing\_model.py* to be run, thus generating the plots that will be displayed on the screen as pop-up windows, and the *testing\_results.txt* file.

The ***User page*** displays a form containing 6 fields: *Age, Sex, Chest Pain level, Resting Blood Pressure, Cholesterol* and *Maximum Heart Rate Achieved* (Fig. 8). The user is prompted to enter his information (Fig. 9) and then, by pressing the ***Submit***button, the frontend makes an API call to the backend which returns the response containing the probabilities of having heart disease (y) and not having heart disease (n) (Fig. 10). In order to start the frontend server, we need to run *npm start* in a separate terminal window while being in the ***frontend*** directory. Before starting the application, we need to make sure all the necessary dependencies are installed, by running the *npm install* command in our terminal window. If everything went right a new tab on our browser should open containing the form.

A screenshot of a software engineer page

Description automatically generated with medium confidence





## **3.2 Backend**

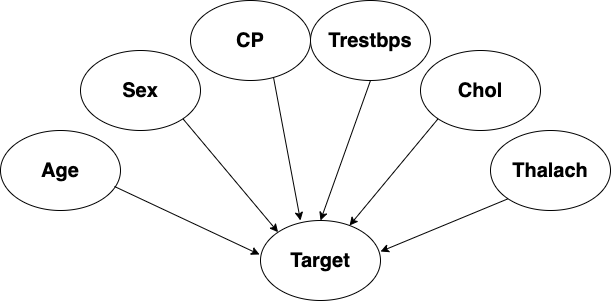
The backend side is implemented through Flask, a Python framework used for building APIs. There is a single Python file in the ***backend*** folder, that is *app.py* and all the operations are being done in this file. First of all, we define the path to the file that contains our trained model, we store it into a variable called *model\_file\_path,* then we store the command for generating the trained model, as explained in the previous sections, in a variable called *generation\_command*, along with the *generation\_directory*. We then check if the trained model doesn’t exist, and in that case we use system commands to change the current directory to the *generation\_directory,* and to run the *generation\_command* in order to generate the trained model. After making sure the trained model exists, we load it into our *program\_string* variable and we start configuring the API.

Before defining the *submit()* function we need to configure the API’s endpoint *'/api/submit'* route to accept the *OPTIONS* and *POST* methods in order to communicate with the frontend. In the *submit()* function we first handle the *OPTIONS* request. We allow access control to the client-side application, which in this case is the frontend side, that makes the request to our backend server. By handling the *OPTIONS* request and setting these headers, the server grants permission to the client-side application to make cross-origin *POST* requests to the *'/api/submit'* endpoint.

After that we handle the *POST* request by saving the data coming through the request into *form\_data* variable, transforming it into *json* format. Next we define 3 functions: *create\_query()* which is responsible for generating the query, *create\_model()* responsible for combining the trained model with the current query, and *process\_prediction()* a functionthat evaluates the Problog program, processes and returns the results string. We store the result in a variable called *eval* and we return it as a response to the *POST* request, back to the frontend side which in term retrieves the data, stores it into a variable and finally displays it on the Web page under the *Diagnosis* field.

In order to start the backend server we need to run the following command in a new terminal window, in the location of our ***backend*** directory: *python3 app.py.* It is important to note that both the frontend server as well as the backend server need to be running at the same time, on different terminal windows, to ensure proper working of the application, and to achieve the desired outcome, which is predicting the presence of heart disease.

# **Bayesian Network Diagram**



# **Architecture**

# **Packages**

In our project we used the following python packages:  
***Flask***:

Flask is a popular web framework for Python. It provides a simple and lightweight way to build web applications. Flask allows us to handle routing, request handling, and templating, making it easy to create dynamic web pages and APIs.

***problog.program***:

problog.program is a package used for working with Probabilistic Logic Programming (PLP) in Python. It provides functionalities to define and manipulate logical programs, and it supports various reasoning tasks, such as inference and query evaluation.

***problog***:

The problog package is an implementation of the Probabilistic Logic Programming language in Python. It allows us to define and solve probabilistic logic programs, which combine logical rules with probabilities. problog.program is a part of this package.

***os***:

The os package is a built-in Python module that provides a way to interact with the operating system. It offers functions for tasks like file and directory manipulation, environment variables, and process management. It is commonly used for tasks that require interaction with the underlying system.

***subprocess***:

The subprocess module is another built-in Python module that provides a way to spawn new processes, connect to their input/output/error pipes, and obtain their return codes. It is often used to run external commands or scripts from within a Python program.

***pandas***:

pandas is a powerful data manipulation and analysis library for Python. It provides data structures such as DataFrames and Series, along with a wide range of functions for data cleaning, transformation, and exploration. pandas is commonly used in data science and data analysis projects.

***seaborn***:

seaborn is a data visualization library for Python. It is built on top of matplotlib and provides a high-level interface for creating attractive and informative statistical graphics. seaborn simplifies the process of creating visualizations and offers additional functionalities for complex visualizations.

***matplotlib.pyplot***:

matplotlib.pyplot is a sub-module of the matplotlib library, which is a widely used plotting library in Python. It provides a MATLAB-like interface for creating various types of plots and visualizations. matplotlib.pyplot is commonly used for creating line plots, scatter plots, bar plots, histograms, and more.

***sklearn.metrics***:

sklearn.metrics is a module within the scikit-learn library, which is a popular machine learning library for Python. The sklearn.metrics module provides various evaluation metrics and scoring functions for assessing the performance of machine learning models. It includes metrics for classification, regression, clustering, and more.

***numpy***:

numpy is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. numpy is widely used in numerical computing, data manipulation, and machine learning tasks.

# **Conclusions**

This project provided a comprehensive dive into the paradigm of Machine Learning, requiring the development of a classification model using Bayesian Networks, encompassing two different domains of programming, i.e., Artificial Intelligence and Web Development, combining multiple programming languages such as Python, Problog, JavaScript to name a few. Needless to say, I learned a lot in the making of this project and it is a scalable application. The number of parameters chosen for the model was due to my system’s limitations as I first tried implementing a model with all 13 of the parameters but while evaluating the model my system crashed. The implementation that I chose allows for scaling up to many more parameters, making a more robust and complex model that can also be trained on larger datasets for better performance. This project is more of a prototype that offered a complex and unique learning experience following a trial-and-error approach.

# **Bibliography**

1. *B. Gutmann, I. Thon and L. De Raedt.* ***Learning the parameters of probabilistic logic programs from interpretations****. Machine Learning and Knowledge Discovery in Databases, volume 6911, pp. 581 - 596, Springer, 2011.*
2. [*https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset*](https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset)
3. Fatima, M. and Pasha, M. (2017) Survey of Machine Learning Algorithms for Disease Diagnostic. *Journal of Intelligent Learning Systems and Applications*, **9**, 1-16.
4. Chang V., Bhavani V.R., Xu A.Q., Hossain M. An artificial intelligence model for heart disease detection using machine learning algorithms Healthc. Anal., 2 (2022), Article 100016
5. [*https://problog.readthedocs.io/en/latest/*](https://problog.readthedocs.io/en/latest/)