CSE 587 - Data Intensive Computing
Project 1
Phase 3 Report

Name: Mrunmayee Vijay Rane
UBIT : mrane

Name: Steve Thomas
UBIT : stevetho

Problem Statement:

- Analyze various NHANES datasets and predict people who are at risk of developing diabetes and depression.

Overview:

We have created a web application which runs locally, where the user receives his medical analysis for depression and diabetes based on certain inputs. This application can predict the possibility of having both depression and diabetes based on gender, age, ethnicity, sleep hours, diastolic pressure, systolic pressure, weight, height, speech frequency hearing loss and high frequency hearing loss. It uses XgBoost classifier model in the background for classifying a user based on their inputs to 4 categories such as no depression and no diabetes, no depression but detected with diabetes, no diabetes but detected with depression, detected with both depression and diabetes.

Instructions for Running Phase 3:

Backend:

Install python along with pip
Go to backend folder and run the following commands in command line:

```
pip install -r requirements.in
pip install -r requirements.txt
uvicorn api_DPDB:app --reload
```

Frontend:

Install npm
Go to form-app(frontend) folder
Run following commands in command line:

```
npm install --legacy-peer-deps
npm run start
```

Phase #3:

a. How you specifically used the models from phase 2, which models did your product end up using, tuning of any relevant parameters?

Phase 2:

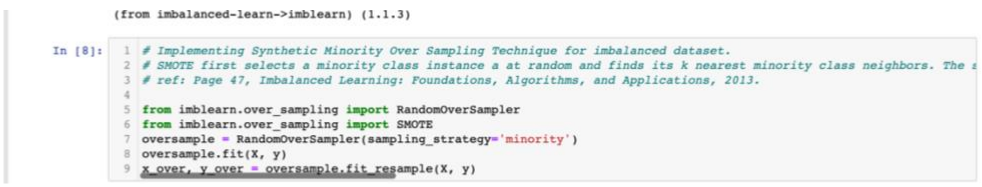
We tried various classifier models for the National Health and Nutritional Examination Survey (NHANES) dataset for the years 2015 - 2016 and 2017 - March 2020. The highest accuracies were given by random forest, xgboost, and decision tree classifier in phase 2. The accuracies gained by the classifier model in phase 2 were not as much as expected. Initially, the highest-scoring accuracy classifiers in phase 2 were not showing expected accuracy due to a few imbalances in the dataset. To deal with the Imbalanced dataset we used Synthetic Minority Oversampling Technique (SMOTE).

One way to solve the problem of an imbalanced dataset is to oversample the examples in the minority class. This can be achieved by simply duplicating examples from the minority class in the training dataset prior to fitting a model. This can balance the class distribution.

SMOTE first selects a minority class instance at random and finds its k nearest minority class neighbors. The synthetic instance is then created by choosing one of the k nearest neighbors b at random and connecting a and b to form a line segment in the feature space. The synthetic instances are generated as a convex combination of the two chosen instances a and b.
SMOTE Implementation:

We defined a SMOTE instance with default parameters that will balance the minority class and then fit and apply it in one step to create a transformed version of our dataset.

Code:



```
(from imbalanced-learn->imblearn) (1.1.3)

In [8]:
# Implementing Synthetic Minority Over Sampling Technique for imbalanced dataset.
# SMOTE first selects a minority class instance a at random and finds its k nearest minority class neighbors. The s
# ref: Page 47, Imbalanced Learning: Foundations, Algorithms, and Applications, 2013.

from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
oversample = RandomOverSampler(sampling_strategy='minority')
oversample.fit(X, y)
X_over, y_over = oversample.fit_resample(X, y)
```

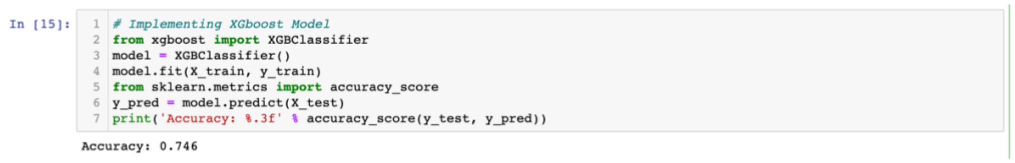
XGboost Classifier Model Implementation**:**

Extreme Gradient Boosting is the abbreviation for XGBoost. XGBoost is a distributed gradient boosting library that has been developed to be very effective, adaptable, and portable. It uses the Gradient Boosting framework to implement machine learning algorithms. It offers a parallel tree boosting to address a variety of data science challenges quickly and accurately.

Boosting is an ensemble learning strategy that creates a strong classifier out of a number of successively weak classifiers. In order to address the bias-variance trade-off, boosting techniques are essential. Boosting algorithms regulate all the components of a model—bias and variance—and are thought to be more effective than bagging algorithms, which solely account for excessive variance.

This model on our dataset gave us 74.6% accuracy.

Code:



```
In [15]:
# Implementing XGboost Model
from xgboost import XGBClassifier
model = XGBClassifier()
model.fit(X_train, y_train)
from sklearn.metrics import accuracy_score
y_pred = model.predict(X_test)
print('Accuracy: %.3f' % accuracy_score(y_test, y_pred))

Accuracy: 0.746
```

After applying SMOTE we gained following accuracy.
1. Decision Tree Classifier, Accuracy: 70.9
2. Random Forest Classifier, Accuracy: 73.9
3. XGBoost Classifier, Accuracy: 74.6

Hence, we ended up using the XGBoost classifier model with highest accuracy of 74.6%.

b. What can users learn from your product, how does it help them solve problems related to your problem statement, other ideas for how to extend your project, or other avenues that could be explored related to the problem?

   Solving Problem statement:
   According to our problem statement, we are predicting the chances of being diagnosed by diabetes and depression based on gender, age, ethnicity, sleep hours, diastolic pressure, systolic pressure, weight, height, speech frequency hearing loss and high frequency hearing loss as user inputs. We used the XgBoost classifier model to predict a user with these inputs as depressed, diabetic or both.

   Future directions:

   This project could also be used by health care practitioners currently to determine diabetes and depression. In order to extend this project, we can add NHANES dataset of previous years and include other parameters such as dietary data (dietary interview & dietary supplements used in 30 days) and lab data(arsenic, chromium, cobalt & iron contents) and see correlation between them. However, we can use our dataset to classify hearing losses as well. Additionally, by adding the hepatitis data column in NHANES dataset to our current dataset. We can incorporate diagnosis of hepatitis and hearing losses as well. Summing up, this project could be used as a user health evaluation system for their respective health inputs such as physical, mental health, lab, dietary data.

UI Application:

The frontend of the application is made using React framework and makes use of fetch API to communicate with the backend. Once all required information is given to the application, the application fires a POST request to the backend on pressing the submit button. The backend then processes this information and returns a message containing the likelihood of the person being affected by diabetes and/or depression. This message is then displayed to the user.

UI Home.JS code:

```
import "./App.css";
import { useState } from "react";
import { Popup } from "./components/Popup/Popup";
import { Gallerypopup } from "./components/Gallerypopup/Gallerypopup";

function Home() {
  const [message, setMessage] = useState("");
  const [gender, setGender] = useState("");
  const [age, setAge] = useState("");
  const [ethinicity, setEthinicity] = useState("");
  const [sleep, setSleep] = useState("");
  const [diastolic, setDiastolic] = useState("");
  const [systolic, setSystolic] = useState("");
  const [SFHL, setSFHL] = useState("");
  const [HFHL, setHFHL] = useState("");
  const [weight, setWeight] = useState("");
  const [height, setHeight] = useState("");
  const [open, setOpen] = useState(false);

  let handleSubmit = async (e) => {
    e.preventDefault();
    try {
      let res = await fetch("http://127.0.0.1:8000/classify", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          gen: parseFloat(gender),
          age: parseFloat(age),
          eth: parseFloat(ethinicity),
          sleep_hrs: parseFloat(sleep),
          dpress: parseFloat(diastolic),
          spress: parseFloat(systolic),
          sfhl: parseFloat(SFHL),
          hfhl: parseFloat(HFHL),
          height: parseFloat(height),
          weight: parseFloat(weight),
        }),
      });
      let resJson = await res.json();
      if (res.status === 200) {
        setGender("");
        setAge("");
        setEthinicity("");
        setSleep("");
        setDiastolic("");
        setSystolic("");
        setSFHL("");
        setHFHL("");
        setMessage(resJson["message"]);
      } else {
        setMessage("Some error occured");
      }
    } catch (err) {
      console.log(err);
    }
  };

  return (
    <div className="App">
      <form onSubmit={handleSubmit}>
        <button onClick={() => setOpen(true)}> Graphs</button>
        {open ? <Gallerypopup closePopup={() => setOpen(false)} /> : null}
        <br />
        <br />
        <br />
        <label>
          Biological Sex
          <select
            value={gender}
            onChange={(e) => setGender(e.target.value)}
            required
          >
            <option value="" disabled="true"></option>
            <option value="1">male</option>
            <option value="2">female</option>
          </select>
        </label>
        <label>
          Age
          <input
        type="number"
        value={age}
        placeholder="Age"
        min="18"
        max="120"
        onChange={(e) => setAge(e.target.value)}
        required
      />
    </label>
    <label>
      Ethinicity
      <select
        value={ethinicity}
        onChange={(e) => setEthinicity(e.target.value)}
        required
      >
        <option value="" disabled="true"></option>
        <option value="1">Mexican American</option>
        <option value="2">Other Hispanic</option>
        <option value="3">White</option>
        <option value="4">Black</option>
        <option value="6">Asian</option>
        <option value="7">Mixed</option>
      </select>
    </label>
    <label>
      Height
      <input
        type="number"
        value={height}
        placeholder="Height in cm"
        min="0"
        max="400"
        step="0.1"
        onChange={(e) => setHeight(e.target.value)}
        required
      />
    </label>
    <label>
      Weight
      <input
        type="number"
        value={weight}
        placeholder="Weight in kg"
        min="0"
        max="600"
        step="0.1"
        onChange={(e) => setWeight(e.target.value)}
        required
      />
    </label>
    <label>
      Average hours of sleep
      <input
        type="number"
        value={sleep}
        placeholder="Sleep"
        min="0"
        max="24"
```

```
          step="0.1"
          onChange={(e) => setSleep(e.target.value)}
          required
        />
      </label>
      <label>
        Diastolic Pressure
        <input
          type="number"
          value={diastolic}
          placeholder="Diastolic Pressure in mmHg"
          min="0"
          max="300"
          step="0.01"
          onChange={(e) => setDiastolic(e.target.value)}
          required
        />
      </label>
      <label>
        Systolic Pressure
        <input
          type="number"
          value={systolic}
          placeholder="Systolic Pressure in mmHg"
          min="0"
          max="300"
          step="0.01"
          onChange={(e) => setSystolic(e.target.value)}
          required
        />
      </label>

      <label>
        Speech-Frequency Hearing Loss
        <input
          type="number"
          value={SFHL}
          placeholder="SFHL in Hz"
          min="-50"
          max="1500"
          step="0.001"
          onChange={(e) => setSFHL(e.target.value)}
          required
        />
      </label>

      <label>
        High-Frequency Hearing Loss
        <input
          type="number"
          value={HFHL}
          placeholder="HFHL in Hz"
          min="-50"
          max="1500"
          step="0.001"
          onChange={(e) => setHFHL(e.target.value)}
          required
        />
      </label>
```

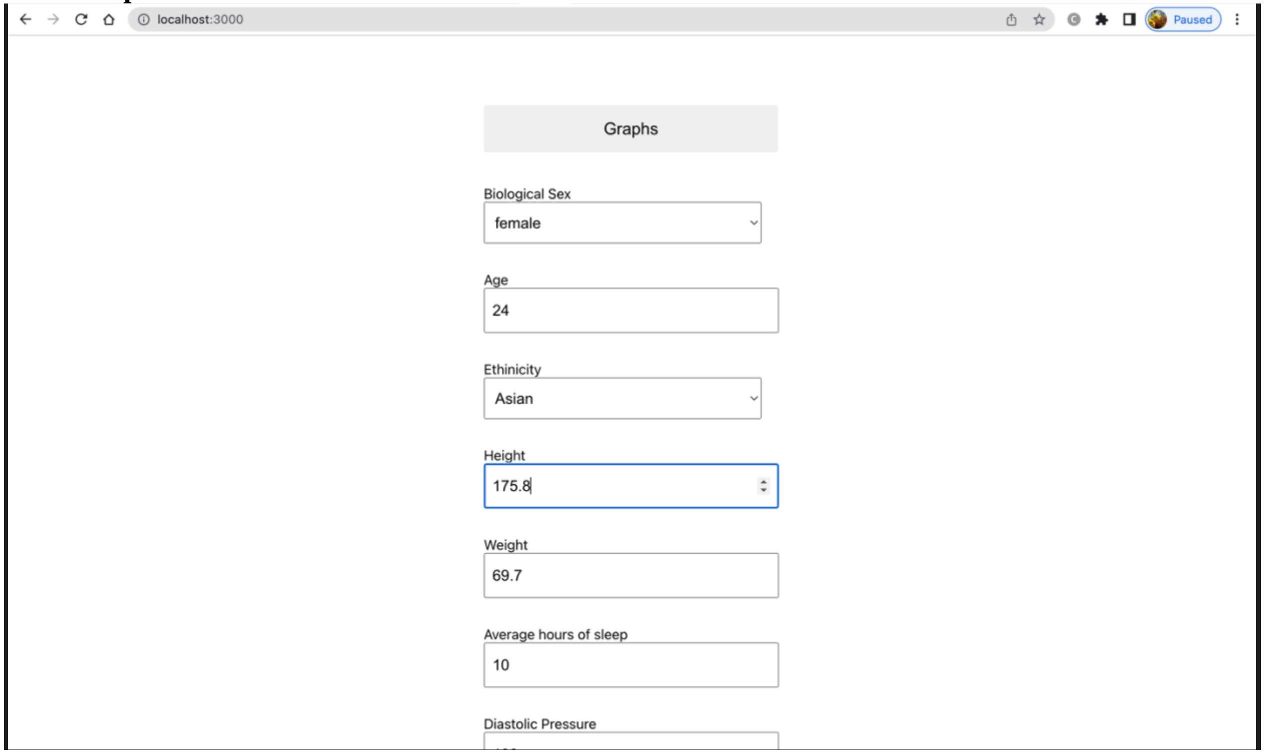
```
        <button type="submit">Submit</button>

        <div className="message">
          {message ? (
            <Popup text={message} closePopup={() => setMessage(false)} />
          ) : null}
        </div>
      </form>
    </div>
  );
}

export default Home;
```

UI Screenshots:

Model Inputs:



Biological Sex
female

Age
24

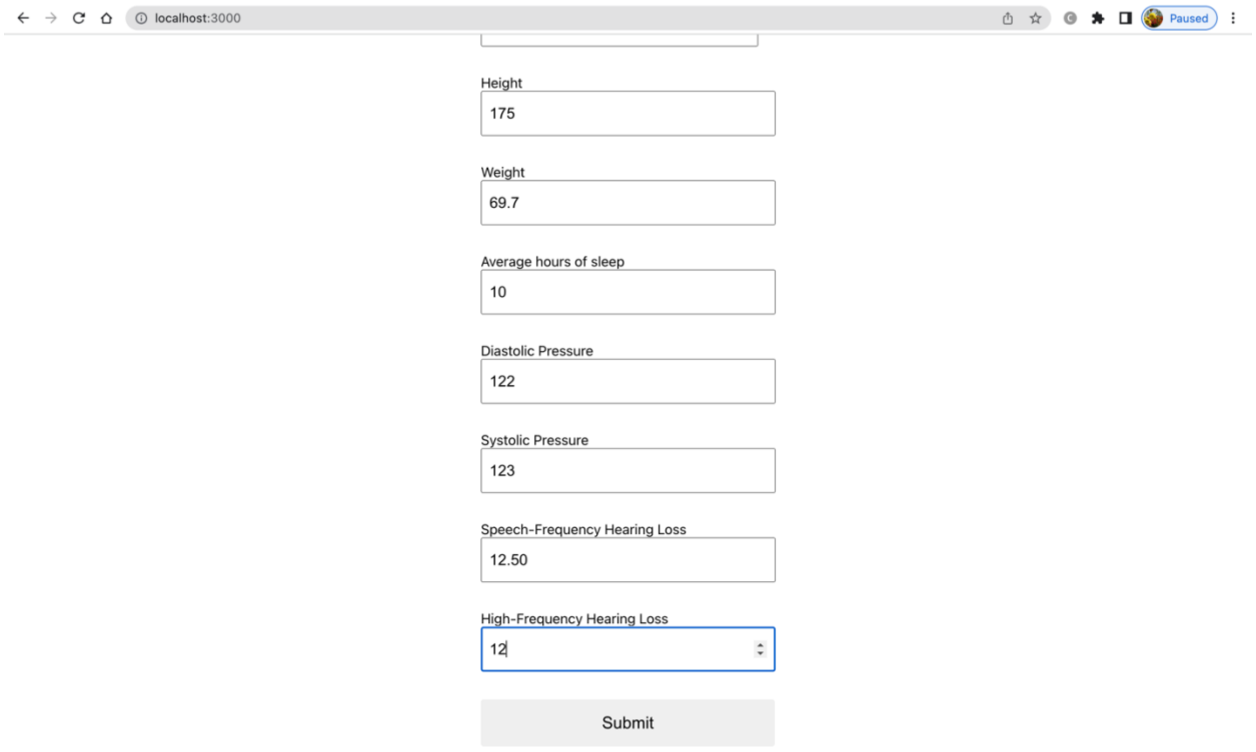
Ethinicity
Asian

Height
175.8

Weight
69.7

Average hours of sleep
10

Diastolic Pressure



localhost:3000

Height
175

Weight
69.7

Average hours of sleep
10

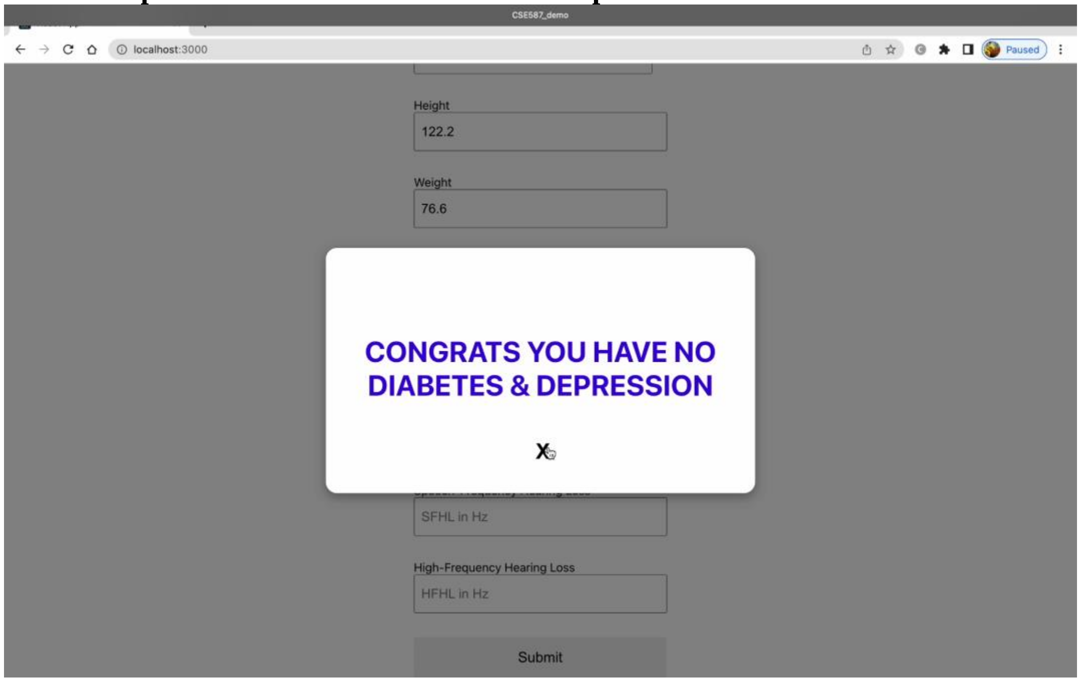
Diastolic Pressure
122

Systolic Pressure
123

Speech-Frequency Hearing Loss
12.50

High-Frequency Hearing Loss
12

Model Output for Case 0 No Diabetes & No Depression:



CSE587_demo

localhost:3000

Height
122.2

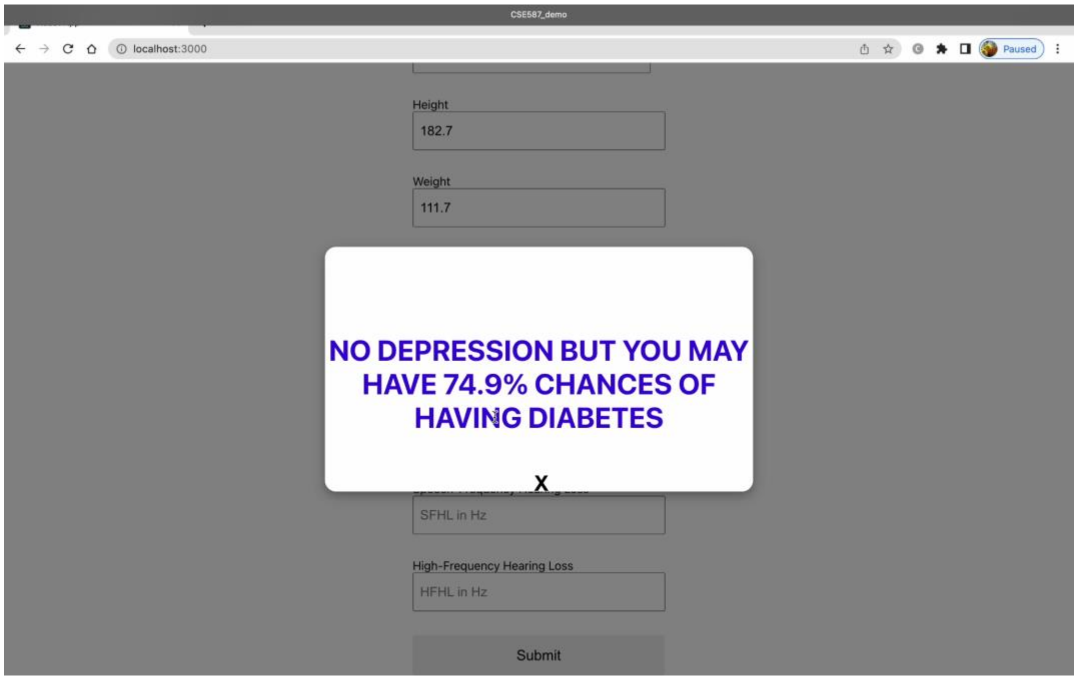
Weight
76.6

CONGRATS YOU HAVE NO DIABETES & DEPRESSION

SFHL in Hz

High-Frequency Hearing Loss
HFHL in Hz

Model Output for Case 1 Diabetes & No Depression:



Height

182.7

Weight

111.7

NO DEPRESSION BUT YOU MAY HAVE 74.9% CHANCES OF HAVING DIABETES

X

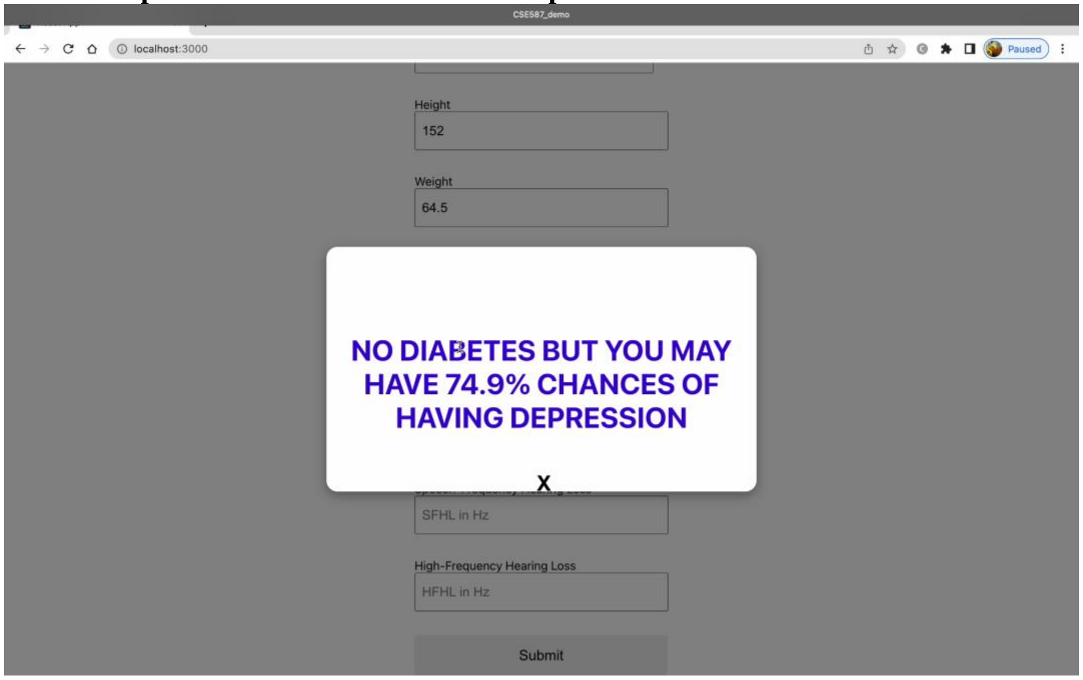
SFHL in Hz

High-Frequency Hearing Loss

HFHL in Hz

Submit

Model Output for Case 2 No Diabetes & Depression:



Height

152

Weight

64.5

NO DIABETES BUT YOU MAY HAVE 74.9% CHANCES OF HAVING DEPRESSION

X

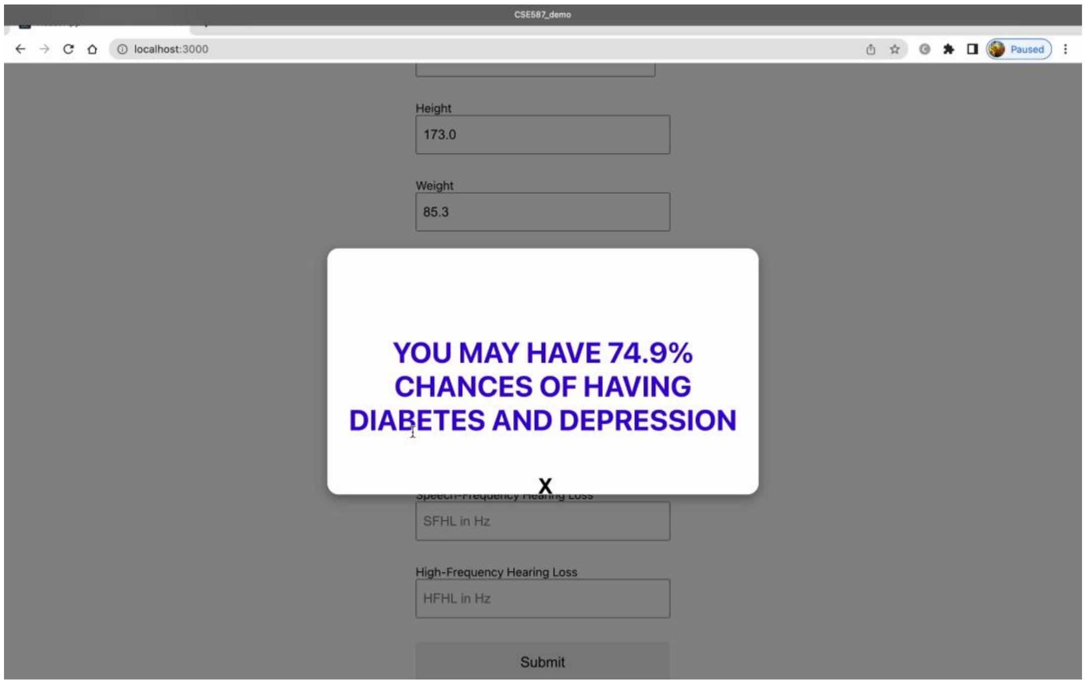
SFHL in Hz

High-Frequency Hearing Loss

HFHL in Hz

Submit

Model Output for Case 3 Both Diabetes & Depression:



Height
173.0

Weight
85.3

YOU MAY HAVE 74.9% CHANCES OF HAVING DIABETES AND DEPRESSION

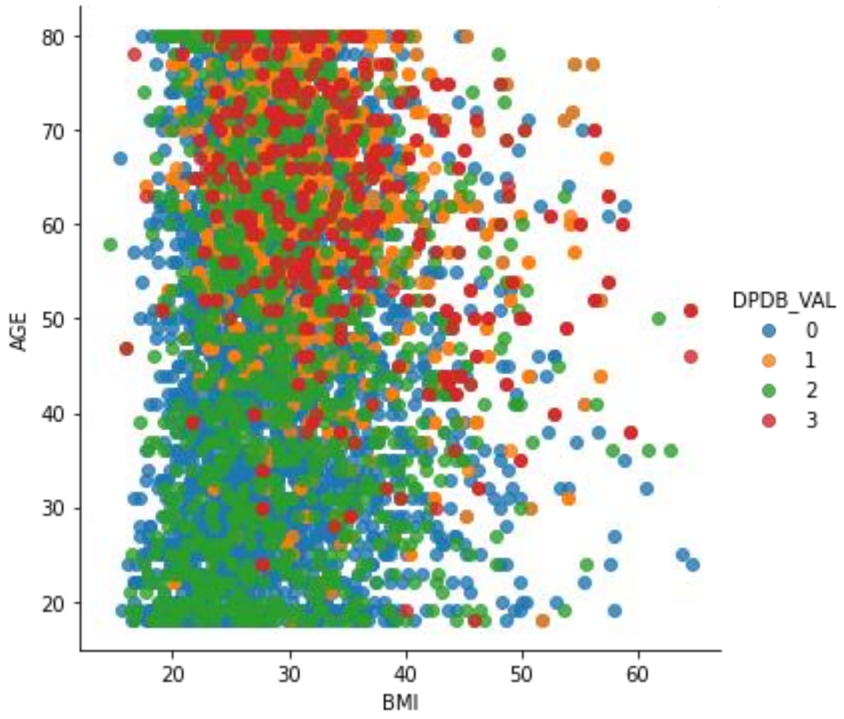
X

Speech-Frequency Hearing Loss
SFHL in Hz

High-Frequency Hearing Loss
HFHL in Hz

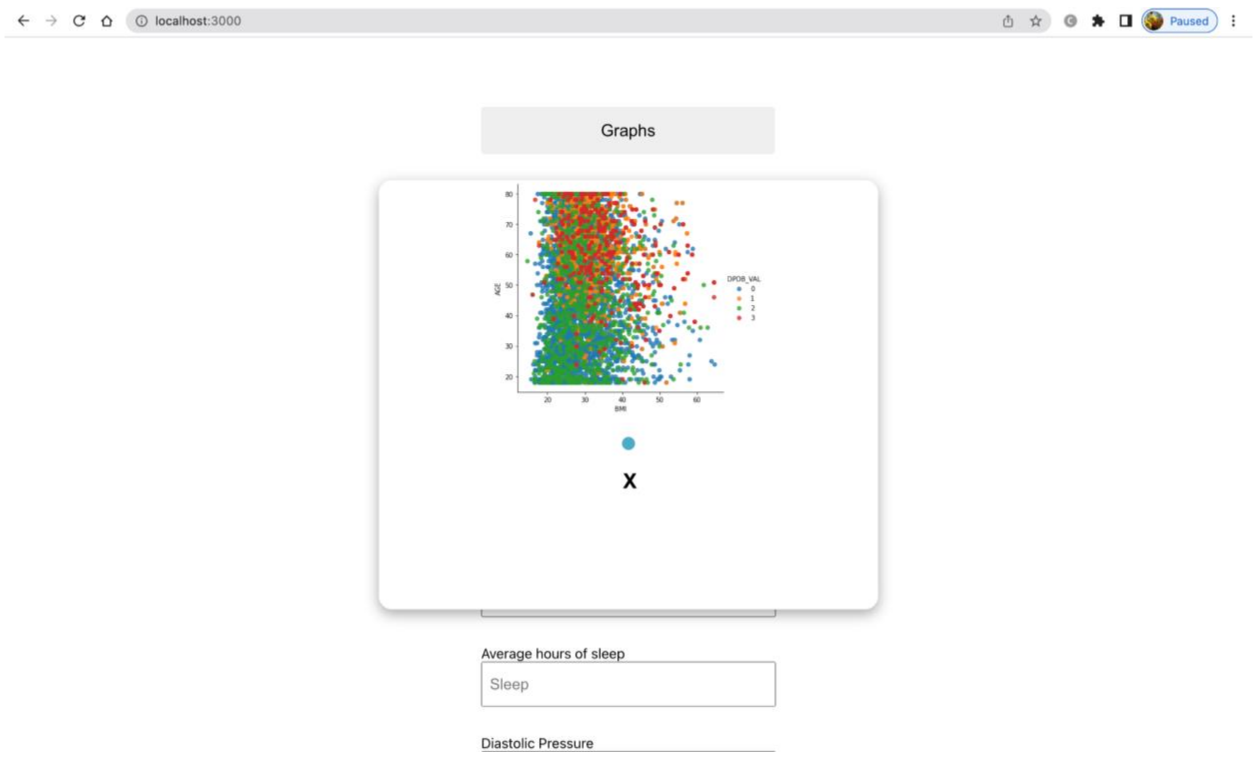
Submit

Graph:
Plot of age on y axis and BMI on x axis:



Mappings:

0: NDPNDB: No Depression & No Diabetes
1: NDPDB: No Depression & Diabetes
2: DPNDB: Depression & No Diabetes
3: DPDB: Depression & Diabetes



Average hours of sleep

Sleep

Diastolic Pressure

Conclusion:

This web application could predict its user based on its respective input in to four categories such as depressive, diabetic, both depressive and diabetic and both non depressive and non-diabetic.

REFERENCES:

1. Page 47, Imbalanced Learning: Foundations, Algorithms, and Applications, 2013.
2. https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification
3. https://fastapi.tiangolo.com/
4. https://reactjs.org/