

1. INTRODUCTION

1.1 Objective

Create a REST service to provide the prediction of a classification model trained to predict the target variable for an input. The model would classify patients with heart disease based on features in the data. The service will have the following requirements:

- Use data and a sample machine learning code from the links below Data:

<https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset>

Sample code:

<https://www.kaggle.com/code/cdabakoglu/heart-disease-classifications-machine-learning>

- Use only 5 important features to build a model
- Take these 5 features as parameters and return its prediction probability in JSON format
- Define the relevant configurations for the deployment of this project.
- Prepare the necessary documentation to make your code and methods more understandable.

1.2 Dataset

The dataset used is the Heart Disease Data Set from the Cleveland database, created in 1988 by V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D. (UCI Machine Learning Repository, n.d.). As Cleveland is one of the major city in United States, the dataset used is relevant for use in prediction of heart disease in United States citizen. The original dataset consists of 76 attributes, but published experiments refer the dataset using a subset of 14 attributes, with 303 instances.

- age: age in years
- sex: 1 = male, 0 = female
- cp (4 values): chest pain type Value 1: typical angina Value 2: atypical angina Value 3: non-anginal pain Value 4: asymptomatic
- trestbps: resting blood pressure in mm Hg on admission to the hospital
- chol: serum cholestoral in mg/dl
- fbs: fasting blood sugar > 120 mg/dl (1 = true; 0 = false)
- restecg: resting electrocardiographic results (values 0,1,2) Value 0: normal Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV) Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
- thalach: maximum heart rate achieved
- exang: exercise induced angina (1 = yes; 0 = no)
- oldpeak: ST depression induced by exercise relative to rest
- slope: the slope of the peak exercise ST segment Value 1: upsloping Value 2: flat Value 3: downsloping
- ca: number of major vessels (0-3) colored by flourosopy
- thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
- target: 1 = presence, 0 = absence

The dataset consisting 5 quantitative attributes which are age, trestbps, chol, thalach and oldpeak ; 9 qualitative categorical attributes which are sex, cp, fbs, restecg, exang, slope, ca, thal and target. ca is considered as qualitative categorical attributes as it only consists of 4 type of unique values.

1.3 Solution Approach

Among the dataset given within the scope of assignment, the most efficient 5 features (cp, thalach, exang, slope, ca) were determined and a dataset in which machine learning models could be trained was created. With this dataset, multiple machine learning models (K- nearest neighbor, Support Vector Machine, Random Forest Classifier, Decision Tree Classifier) were trained and these models were exported. Among these models, Random Forest Classifier, which is suitable for our target and has a high accuracy and F1 score, was selected and integrated into the web application and Rest service created using Flask for backend and React for frontend. With the help of this created web application, the user's inputs are taken with the determined 5 features and it is possible to estimate the probability of the presence or absence of heart disease. In addition, the application was developed and tested to make API call with these 5 features.

2.TOOLS AND LIBRARIES

python == 3.9.7

npm == 8.11.0

numpy == 1.20.3

pickle

sklearn == 0.0

requests == 2.27.1

flask == 1.1.2

pandas == 1.3.4

matplotlib == 3.4.3

scikit-learn == 0.24.2

npm == 8.11.0

React

Moesif CORS (Chrome Extension) (For error number 200 received during prediction from frontend)

3. CODE AND METHODS

3.1 Model

3.1.1 HearthDiseasePrediction.ipynb

In the first cell in the image below contains the part where I import the libraries I use to extract data, analyze and extract Python objects that can be stored and used by other programs or at different runtimes of the same program.

In the second cell, here comes the part where we take my dataset to our notebook using the pandas library.

In the last cell, there is the part that writes the first 5 cells of the dataset we imported into the screen. Thus, we have the chance to briefly examine the dataset.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle
```

```
df = pd.read_csv("heart.csv")
```

```
df.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

In the cell below, for the item "Use only 5 important features to build a model", which is requested from me in the assignment, I first used the Feature Selection methods to select the 5 features in the data set that increase the performance of the models. First of all, for the feature selection method I used, I wanted to use sklearn's "sklearn.feature_selection.SelectKBest" method, which is used for Select features according to the k highest scores. For this, I imported the sklearn library and related methods. Then I divided my dataset into two as X and Y, separated the features other than Target and saved it as an array. Since I want to extract 5 features, I defined my SelectKBest method by determining its k as 5 and using the f_classif score function. After I defined my method, I fitted the X and Y that I had determined and fit my method for the most suitable 5 features. Evaluating the feature selection results, I chose cp, thalach, exang, slope and ca as 5 features from the features with the highest values.

```

from numpy import set_printoptions
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
array = df.values
X = array[:,0:13]
Y = array[:,13]

test = SelectKBest(score_func=f_classif, k=5)
fit = test.fit(X, Y)

set_printoptions(precision=3)
print(fit.scores_)
features = fit.transform(X)

print(features[0:5,:])

```

```

[ 56.785  86.69 238.558 20.087 10.326  1.736 18.838 222.8 242.884
243.451 138.679 174.877 131.803]
[[ 0. 168.  0.  1.  2. ]
 [ 0. 155.  1.  3.1 0. ]
 [ 0. 125.  1.  2.6 0. ]
 [ 0. 161.  0.  0.  1. ]
 [ 0. 106.  0.  1.9 3. ]]

```

```
#cp, thalach, exang, slope, ca
```

In the code below, I first removed the columns other than the 5 features and targets I selected in the first cell from my table. Then, with the code snippet in the second cell, in order to create my train and test data, I first define my target column as an array to x and 5 features as an array to y. In the last cell, there is the code snippet in which I prepared the data, which I allocated 25% to 75% that I can train and test my models.

```
df.drop(df.columns.difference(['cp','thalach','exang','slope','ca','target']), 1, inplace=True)
```

<ipython-input-85-d0a7c209913a>:1: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only

```
df.drop(df.columns.difference(['cp','thalach','exang','slope','ca','target']), 1, inplace=True)
```

```

x = df.drop(['target'],axis=1).values
y = df.iloc[:, -1].values

```

x

```

array([[ 0, 168,  0,  2,  2],
       [ 0, 155,  1,  0,  0],
       [ 0, 125,  1,  0,  0],
       ...,
       [ 0, 118,  1,  1,  1],
       [ 0, 159,  0,  2,  0],
       [ 0, 113,  0,  1,  1]], dtype=int64)

```

y

```
array([0, 0, 0, ..., 0, 1, 0], dtype=int64)
```

```
np.bincount(y_train)
```

```
array([376, 392], dtype=int64)
```

```

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 0)

```

After preparing the data, I start to train my models with the following codes. First I start by using K-Nearest Neighbors (K-NN). In the first cell for K-NN, I first found the optimal `n_neighbors` value using `KNeighborsClassifier` and `GridSearchCV`. `KNeighborsClassifier` is the classifier that implements K-nearest neighbor voting, and `gridsearchcv` is a method that performs an exhaustive search on parameter values specified for an estimator. I defined and trained my K-NN model in cell 3 with the most appropriate `n_neighbors` value displayed in cell 3.

K-Nearest Neighbors (K-NN)

```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

findingBestNeighborsKNN = KNeighborsClassifier()

param_grid = {'n_neighbors': np.arange(1, 25)}

knn_gscv = GridSearchCV(findingBestNeighborsKNN, param_grid, cv=5)

knn_gscv.fit(x, y)

GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
        18, 19, 20, 21, 22, 23, 24])})
```

`knn_gscv.best_params_`

```
{'n_neighbors': 1}
```

```
knnClassifier = KNeighborsClassifier(n_neighbors = 1, metric = 'minkowski', p = 2)
knnClassifier.fit(x_train, y_train)
```

```
KNeighborsClassifier(n_neighbors=1)
```

Then I made a prediction on the test data with `predict` in the first cell below and printed the outputs on the screen. In cell 2, I viewed the classification performance between test and prediction with `confusion_matrix`. In the third cell, calculating the accuracy of the models and the F1 value. The main reason why I especially used the F1 score was not to make an incorrect model selection in data sets that are not evenly distributed.

```
y_pred = knnClassifier.predict(x_test)
y_pred

array([1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1,
       1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0,
       1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1,
       0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1,
       0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1], dtype=int64)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[122,  1],
       [  1, 133]], dtype=int64)
```

```
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix

print("Accuracy of KNN: ", accuracy_score(y_test, y_pred) )
print("F1 Score of KNN: ", f1_score(y_test, y_pred) )
knn_cm = confusion_matrix(y_test, y_pred)
```

```
Accuracy of KNN:  0.9922178988326849
F1 Score of KNN:  0.9925373134328358
```

With the code snippet below, I first created a dataset with 5 features whose target values I know, and with this dataset, I made a prediction with my K-NN model in the 2nd cell. The results came as I expected. Then, in my 3rd cell, I took the dump of my K-NN model with pickle, which creates Python objects that can be stored and used by other programs or at different runtimes of the same program. I made a guessing experiment by calling the pickle file whose dump I took with the 4th cell below and the result came as I wanted.

```
data = [[0,168,0,2,2],[0,161,0,2,1],[0,164,1,2,0],[0,159,0,2,0]]

burak_test = pd.DataFrame(data, columns=['cp','thalach','exang','slope','ca'])

burak_test
```

	cp	thalach	exang	slope	ca
0	0	168	0	2	2
1	0	161	0	2	1
2	0	164	1	2	0
3	0	159	0	2	0

```
burak_pred = knnClassifier.predict(burak_test)
#expected result = [0,0,1,1]
burak_pred
```

```
array([0, 0, 1, 1], dtype=int64)
```

```
pickle.dump(knnClassifier, open('knnModel.pkl','wb'))
```

```
model = pickle.load( open('knnModel.pkl','rb'))
print(model.predict([[0,168,0,2,2]]))
```

```
[0]
```

First, after I took the printouts with my K-NN model, I continued by experimenting with the Support Vector Machine (SVM) model with the same methods. In the first cell below, I first created my model using the Radial basis function kernel and then I trained my model.

Support Vector Machine (SVM)

```
from sklearn.svm import SVC
svmClassifier = SVC(kernel = 'rbf', random_state = 0)
svmClassifier.fit(x_train, y_train)
```

```
SVC(random_state=0)
```

Afterwards, I measured the performance of the model as I did in the KNN model in the code snippets below and took the dumb as a pickle file in the same way.

```
y_pred= svmClassifier.predict(x_test)
y_pred

array([1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0,
       0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0,
       1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1,
       1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0])
```

```
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[ 88,  35],
       [ 31, 103]], dtype=int64)
```

```
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix

print("Accuracy of SVM: ", accuracy_score(y_test, y_pred) )
print("F1 Score of SVM: ", f1_score(y_test, y_pred) )
svmcm = confusion_matrix(y_test, y_pred)
```

```
Accuracy of SVM:  0.7431906614785992
F1 Score of SVM:  0.7573529411764706
```

```
burak_pred = svmClassifier.predict(burak_test.values)
```

```
burak_pred
```

```
array([1, 1, 1, 1], dtype=int64)
```

```
pickle.dump(svmClassifier, open('svmModel.pkl','wb'))
```

```
model = pickle.load(open('svmModel.pkl','rb'))
print(model.predict([[0,168,0,2,2]]))
```

```
[1]
```

After training and estimating with SVM, I continued with Random Forest classifier. I used 30 max_depth and 500 n_estimators that I created in the first cell below. Since these numbers are determined intuitively, I decided after a few tries. Then I trained my data with the classifier.

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rfClassifier = RandomForestClassifier(max_depth = 30, n_estimators = 500)
rfClassifier.fit(x_train, y_train)
```

```
RandomForestClassifier(max_depth=30, n_estimators=500)
```

Afterwards, I measured the performance of the model as I did in the KNN and SVM models in the code snippets below and took the dumb as a pickle file in the same way.

```
y_pred= rfClassifier.predict(x_test)
y_pred

array([[1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1,
        1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
        0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,
        1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1,
        1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0,
        1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0,
        0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1,
        0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1,
        0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1,
        0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0,
        0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1], dtype=int64)
```

```
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[121,  2],
       [ 1, 133]], dtype=int64)
```

```
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix

print("Accuracy of Random Forest Classifier: ", accuracy_score(y_test, y_pred) )
print("F1 Score of Random Forest Classifier: ", f1_score(y_test, y_pred) )
```

```
Accuracy of Random Forest Classifier:  0.9883268482490273
F1 Score of Random Forest Classifier:  0.9888475836431226
```

```
burak_pred = rfClassifier.predict(burak_test.values)

burak_pred
```

```
array([0, 0, 1, 1], dtype=int64)
```

```
pickle.dump(rfClassifier, open('rfModel.pkl','wb'))
```

```
model = pickle.load(open('dtModel.pkl','rb'))
print(model.predict([[0,168,0,2,2]]))
```

```
[0]
```

Finally, I create my model with the following piece of code to train and predict with Decision Tree. Since the interval of the Entropy is [0, 1], by choosing entropy as the criterion, I intuitively set my max_depth value to 15 with a few tries. Then I train my model.

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dtClassifier = DecisionTreeClassifier(criterion = 'entropy',random_state=0,max_depth = 15)
dtClassifier.fit(x_train, y_train)

DecisionTreeClassifier(criterion='entropy', max_depth=15, random_state=0)
```


Afterwards, I measured the performance of the model as I did in the KNN, SVM and Random Forest models in the code snippets below and took the dumb as a pickle file in the same way.

```
y_pred= dtClassifier.predict(x_test)
y_pred

array([[1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1,
        1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
        0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,
        1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1,
        1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0,
        1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0,
        0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1,
        0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1,
        0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0,
        0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1], dtype=int64)

cm = confusion_matrix(y_test, y_pred)
cm

array([[122,  1],
       [ 2, 132]], dtype=int64)

from sklearn.metrics import accuracy_score, f1_score, confusion_matrix

print("Accuracy of Decision Tree Classifier: ", accuracy_score(y_test, y_pred) )
print("F1 Score of Decision Tree Classifier: ", f1_score(y_test, y_pred) )

Accuracy of Decision Tree Classifier:  0.9883268482490273
F1 Score of Decision Tree Classifier:  0.9887640449438201

burak_pred = dtClassifier.predict(burak_test.values)

burak_pred

array([0, 0, 1, 1], dtype=int64)

pickle.dump(dtClassifier, open('dtModel.pkl','wb'))

model = pickle.load(open('dtModel.pkl','rb'))
print(model.predict([[0,168,0,2,2]]))

[0]
```

3.2 Backend

I preferred Flask web framework for the backend of my application. I chose it because it is both easy to use and more pythonic when making machine learning applications.

3.2.1 app.py

In the codes below, I imported the open source libraries that I used first. I added `app = Flask(__name__)` which is the first argument when creating an instance of the Flask object (a Python Flask application). Then I loaded my Random Forest Model, whose dumb I had taken, with pickle.

I rendered the template in my index.html document with the `home()` function.

Afterwards, you can see the predict function, which predicts the requests of the values I send with the post method in my model and renders the results in the HTML GUI.

```

app.py > predict_html
1  import numpy as np
2  from flask import Flask, request, jsonify, render_template
3  import pickle
4
5  app = Flask(__name__)
6  model = pickle.load(open('rfModel.pkl', 'rb'))
7
8  @app.route('/')
9  def home():
10     return render_template('index.html')
11
12  @app.route('/predict_html', methods=['POST'])
13  def predict_html():
14
15     features = [int(x) for x in request.form.values()]
16     print(features)
17     print(type(features))
18     finalFeatures = [np.array(features)]
19     print(finalFeatures)
20     print(type(finalFeatures))
21     prediction = model.predict(finalFeatures)
22
23     output = round(prediction[0])
24     if output == 1:
25         result="You Have Heart Disease"
26     else:
27         result="You Do Not Have Heart Disease"
28
29     return render_template('index.html', prediction_text='Prediction is: {}'.format(result))
30

```

With the predict_api function below, it is possible to make a direct API call with requests.

```

31  @app.route('/predict_api', methods=['POST'])
32  def predict_api():
33
34     data = request.get_json(force=True)
35     prediction = model.predict([np.array(list(data.values()))])
36
37     output = prediction
38     output=output.item()
39     return jsonify(output)
40
41  if __name__ == "__main__":
42     app.run(debug=True)
43

```

3.2.2 templates\index.html

I wanted to add the index.html file, even though it doesn't actually have backend codes. We can make predictions by running the application with a simple frontend locally by rendering the following html codes by directly running app.py without using the frontend I made with React. The following codes simply contain input boxes and predict button that will receive 5 features from the user.

```
templates > index.html > body
1 <!DOCTYPE html>
2 <body>
3 <div class="login">
4 <div>
5
6 </div>
7
8 <!-- Main Input For Receiving Query to Heart Disease Prediction-->
9 <form action="{{ url_for('predict_html')}}" method="post">
10 <label for="Feature-1">Chest Pain Type</label>
11 <input type="text" name="cp" placeholder="1 or 2 or 3 or 4" required="required"/><p><small>1: typical angina / 2: atypical angina / 3: non-anginal pain / 4: asymptomatic</small></p>
12 <br>
13 <label for="Feature-2">Thalach</label>
14 <input type="text" name="thalach" placeholder="Max Heart Rate" required="required"/><p><small>Maximum heart rate achieved</small></p>
15 <br>
16 <label for="Feature-3">Exercise Induced Angina</label>
17 <input type="text" name="exang" placeholder="0 or 1" required="required"/><p><small>Exercise induced angina (1 = yes / 0 = no)</small></p>
18 <br>
19 <label for="Feature-4">Slope</label>
20 <input type="text" name="slope" placeholder="1 or 2 or 3" required="required"/><p><small>the slope of the peak exercise ST segment (1: upsloping / 2: flat / 3: downsloping)</small></p>
21 <br>
22 <label for="Feature-5">Number of Major Vessels</label>
23 <input type="text" name="ca" placeholder="0 or 1 or 2 or 3" required="required"/><p><small>number of major vessels (0-3) colored by flourosopy</small></p>
24 <br>
25 <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
26
27 </form>
28
29 <br>
30 <br>
31 {{ prediction_text }}
32
33 </div>
```

3.3 Frontend

I used react on the frontend and tried to increase the user experience with css. React is an open source javascript library for creating user interfaces.

3.3.1 Page.js

With the code snippet below, firstly react, bootstrap, predict and page.css are imported. Then there is the Page state, this state holds a variable value, and when the state changes, the page is rendered.

```
Frontend > src > JS Page.js > Page > useEffect() callback
1 import React, { useEffect, useRef, useState } from 'react'
2 import { Form, Button } from 'react-bootstrap';
3 import { predict } from './api';
4 import './Page.css';
5
6 const Page = (props) => {
7   const [chestType, setChestType] = useState();
8   const [thalach, setThalach] = useState();
9   const [exang, setExang] = useState();
10  const [slope, setSlope] = useState();
11  const [vessel, setVessel] = useState();
12  const [result, setResult] = useState(null);
13  const [disiaseText, setDisiaseText] = useState("");
14
```

Then, in the code below, there is the function that is called when the button is pressed, it sends a request to the backend and gets the output.

```
15     const handlePredict = () => {
16         if (thalach == 0) {
17             alert("All the fields must be filled.");
18             return;
19         }
20         predict({"cp": chestType, "thalach": thalach, "exang": exang, "slope": slope, "ca": vessel})
21         .then(result => result.json())
22         .then(
23             (result) => {
24                 setResult(result);
25             },
26             (error) => {
27                 alert("Error occurred on API call");
28                 return;
29             }
30         )
31     }
32 }
```

Followed by `useEffect()`, a function in the code below that works in some cases. If the square brackets are empty in the field below, it will work when the page is first opened. If the square brackets are filled, the variable will run every time it is changed.

```
33     useEffect(() => {
34         setChestType(1);
35         setThalach(0);
36         setExang(0);
37         setSlope(1);
38         setVessel(0);
39     }, []);
40
41     useEffect(() => {
42         if (result == 1) {
43             setDisiaseText("You Have Heart Disease");
44         } else if (result == 0) {
45             setDisiaseText("You Do Not Have Heart Disease");
46         } else {
47             return;
48         }
49     }, [result]);
50 }
```

After that, there is the following html part for the forms and buttons on the frontend.

```
51   return (  
52     <Form name="features">  
53       <Form.Label className="pageHeader">Heart Disease Prediction</Form.Label>  
54       <br/>  
55       <Form.Label className="featureLabel">Chest Pain Type</Form.Label>  
56       <Form.Select onChange={ (event) => setChestType(event.target.value)}>  
57         <option value="1">Typical Angina</option>  
58         <option value="2">Atypical Angina</option>  
59         <option value="3">Non-Anginal Pain</option>  
60         <option value="4">Asymptomatic</option>  
61       </Form.Select>  
62  
63       <Form.Label className="featureLabel">Thalach</Form.Label>  
64       <Form.Control  
65         type="text"  
66         placeholder="Max Heart Rate"  
67         onChange={ (event) => setThalach(event.target.value)}  
68       ></Form.Control>  
69  
70       <Form.Label className="featureLabel">Exercise Induced Angina</Form.Label>  
71       <Form.Select onChange={ (event) => setExang(event.target.value)}>  
72         <option value="0" selected>Yes</option>  
73         <option value="1">No</option>  
74       </Form.Select>  
75  
76       <Form.Label className="featureLabel">Slope</Form.Label>  
77       <Form.Select onChange={ (event) => setSlope(event.target.value)}>  
78         <option value="1">Upsloping</option>  
79         <option value="2">Flat</option>  
80         <option value="3">Downsloping</option>  
81       </Form.Select>  
82  
83       <Form.Label className="featureLabel">Number of Major Vessels</Form.Label>  
84       <Form.Select onChange={ (event) => setVessel(event.target.value)}>  
85         <option value="0">0</option>  
86         <option value="1">1</option>  
87         <option value="2">2</option>  
88         <option value="3">3</option>  
89       </Form.Select>  
90  
91       <Button  
92         variant="primary"  
93         style={{ "marginTop": "30px" }}  
94         onClick={handlePredict}  
95       >  
96         Predict  
97       </Button>  
98       <br/>  
99       <Form.Label className="outputText">{diaseText}</Form.Label>  
100     </Form>  
101   );  
102 }  
103  
104 export default Page;
```

3.3.2 index.js

index.js handles app startup, routing and other functions of my application and does require other modules to add functionality.

```
src > JS index.js > ...  
1   import React from 'react';  
2   import ReactDOM from 'react-dom/client';  
3   import Page from './Page';  
4  
5   const root = ReactDOM.createRoot(document.getElementById('root'));  
6   root.render(  
7     <React.StrictMode>  
8       <Page />  
9     </React.StrictMode>  
10  );
```

3.3.3 api.js

The api.js file contains the function used to request the backend. After running the Flask application, a request is sent to the backend.

```
src > JS api.js > ...
1  import React from "react";
2
3  export function predict(body) {
4      return fetch('http://127.0.0.1:5000/predict_api', {
5          method: 'POST',
6          body: JSON.stringify(body),
7      });
8  }
```

3.3.4 Page.css

The Page.css file contains the design configurations on the frontend.

```
src > # Page.css > ...
1  body {
2      background-image: url('static/bioinformatics.jpeg');
3      background-repeat: no-repeat;
4      background-attachment: fixed;
5      background-position: center;
6      background-size: cover;
7      margin: auto;
8      width: 30%;
9  }
10
11  .pageHeader {
12      margin-top: 10%;
13      font-family: 'Roboto Medium', sans-serif;
14      font-size: 30px;
15      color: ■rgb(255, 255, 255);
16      font-weight: 500;
17      text-align: center;
18      line-height: 150%;
19  }
20
21  .featureLabel {
22      margin-top: 30px;
23      color: ■rgb(255, 255, 255);
24  }
25
26  .outputText {
27      margin-top: 30px;
28      margin-left: 50px;
29      color: ■white;
30      font-family: 'Roboto Medium', sans-serif;
31      font-size: 25px;
32  }
```

4. RUNNING THE APPLICATION

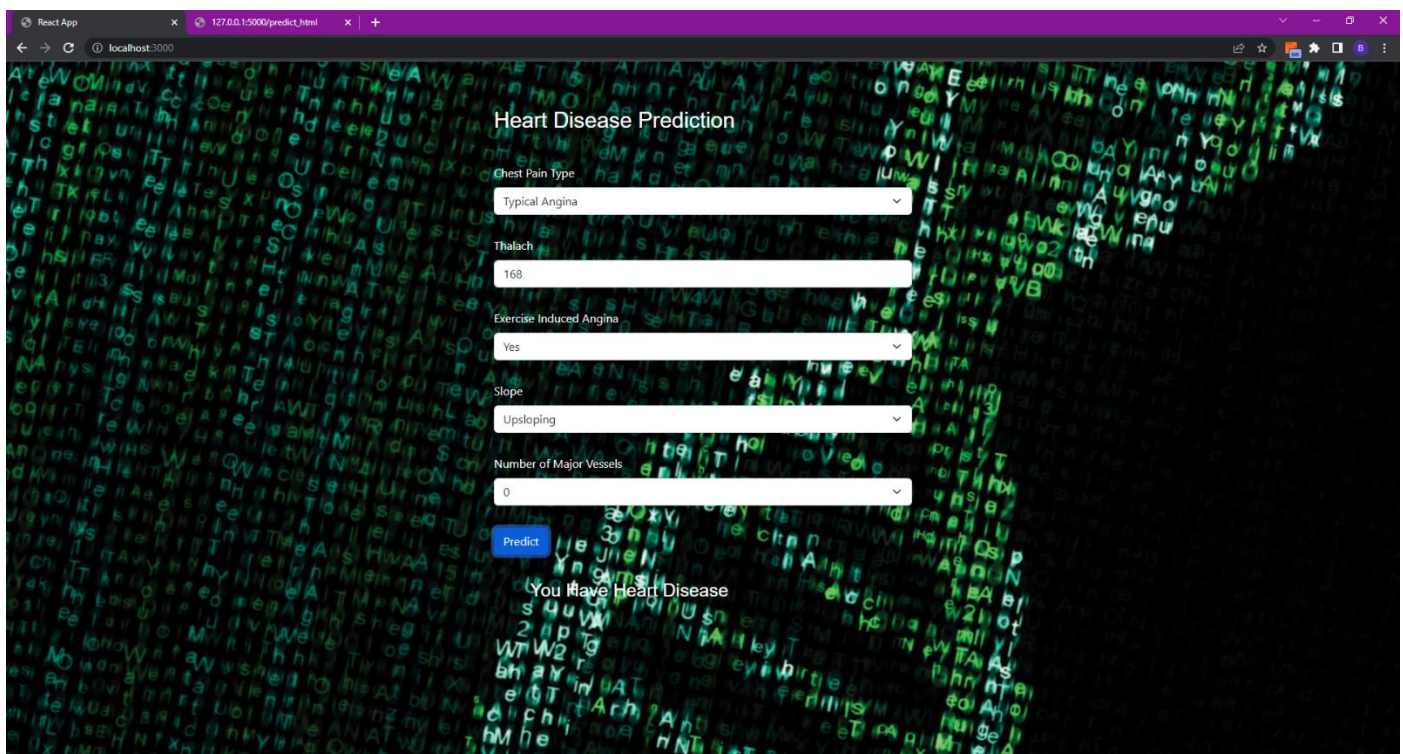
First of all, the tools and libraries in requirement.txt must be installed to run the application. Then we will establish the connection by running both the backend and the frontend side separately.

First, we go to the Heart-Disease-Predictor/Backend/ directory from the terminal and run the "python app.py" command to run the app.py file. Thus, our Flask application will start working at <http://127.0.0.1:5000/>. Then we go to the Heart-Disease-Predictor/Frontend/ directory and run the "npm start" command and our react application will start running at localhost:3000.

We will be able to access the user interface by coming to localhost:3000. API call made from this frontend will go to http://127.0.0.1:5000/predict_api. In addition, we will be able to send requests only via flask with the help of a simple frontend by going to <http://127.0.0.1:5000/>.

5. RESULTS

You can see the user interface where we display the results by logging into localhost:3000 below. If our target value returns 1 after prediction, "You Have Heart Disease" returns, and if it returns 0, "You Do Not Have Disease" returns. target values are first saved as Json.

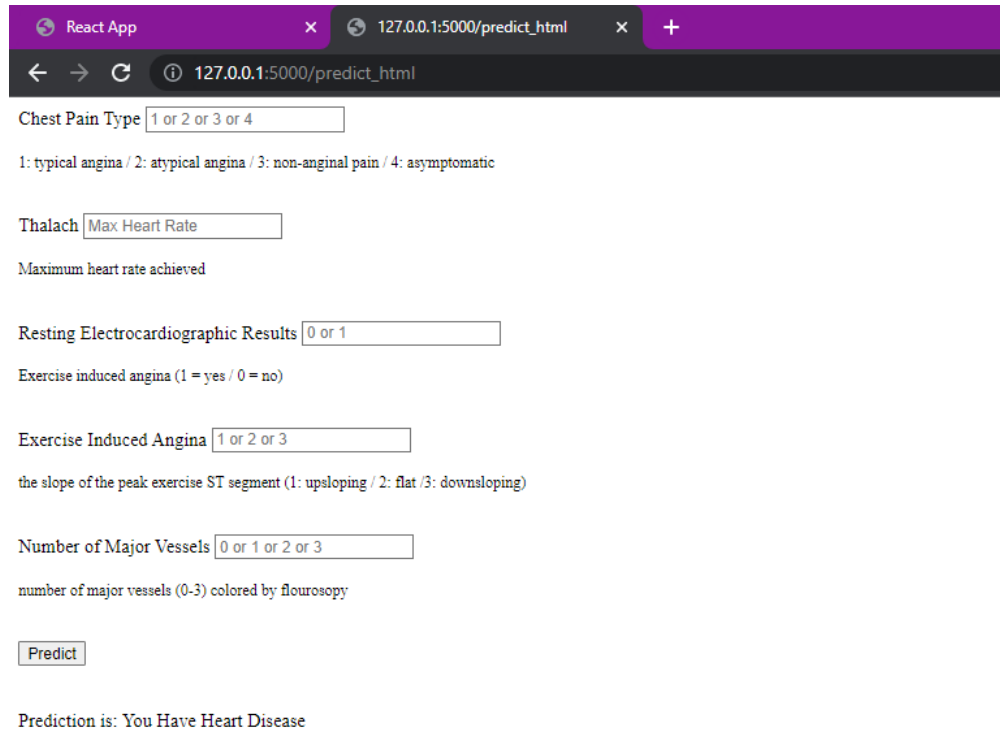


The screenshot shows a web browser window with the title "React App" and the address bar displaying "localhost:3000". The page content is titled "Heart Disease Prediction" and features a form with the following fields:

- Chest Pain Type:** A dropdown menu with "Typical Angina" selected.
- Thalach:** A text input field containing the value "168".
- Exercise Induced Angina:** A dropdown menu with "Yes" selected.
- Slope:** A dropdown menu with "Upsloping" selected.
- Number of Major Vessels:** A dropdown menu with "0" selected.

Below the form is a blue "Predict" button. The result of the prediction is displayed as "You Have Heart Disease". The background of the page is a dark green with a pattern of white and yellow text.

When we go to <http://127.0.0.1:5000/>, a simple bit html gui greets us as follows. Likewise, we can make predictions by entering the values for the relevant 5 features from here. Also in this section, If our target value returns 1 after prediction, "You Have Heart Disease" returns, and if it returns 0, "You Do Not Have Disease" returns. Here, too, we take our target value as Json.



React App x 127.0.0.1:5000/predict_html x +

← → ↻ ⓘ 127.0.0.1:5000/predict_html

Chest Pain Type

1: typical angina / 2: atypical angina / 3: non-anginal pain / 4: asymptomatic

Thalach

Maximum heart rate achieved

Resting Electrocardiographic Results

Exercise induced angina (1 = yes / 0 = no)

Exercise Induced Angina

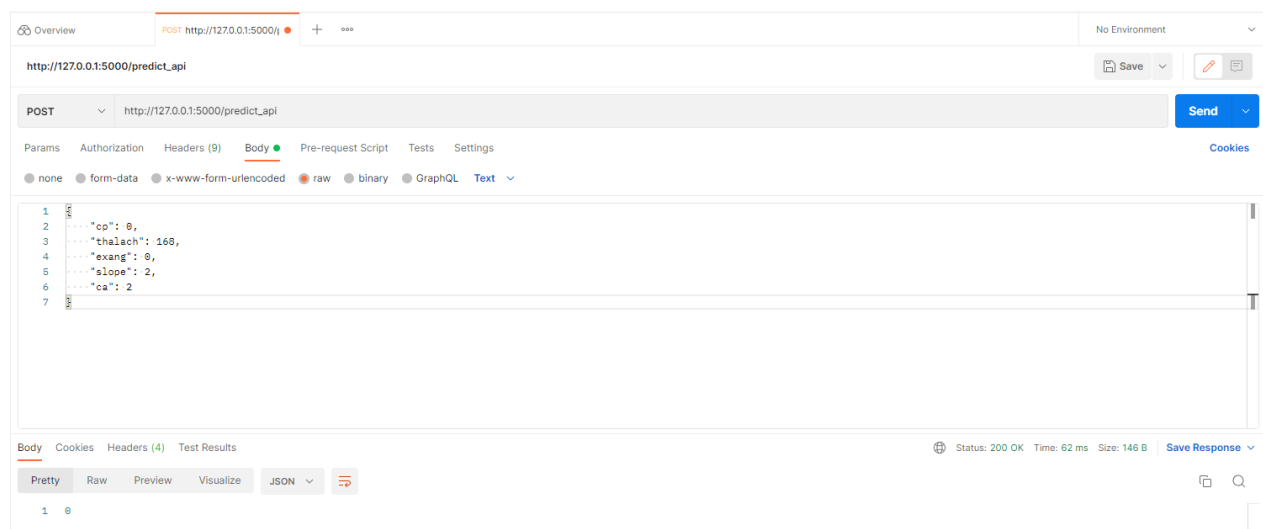
the slope of the peak exercise ST segment (1: upsloping / 2: flat / 3: downsloping)

Number of Major Vessels

number of major vessels (0-3) colored by flourosopy

Prediction is: You Have Heart Disease

I also sent a sample POST request to my Flask application via postman to test my API. You can see the output of a sample Json file under the Heart-Disease-Predictor/Sample Json Post Data/ directory in the postman interface below, where we sent a request to https://127.0.0.1:500/predict_api via POST method. we can save our output as Json.



Overview POST http://127.0.0.1:5000/ + ... No Environment

http://127.0.0.1:5000/predict_api Save

POST http://127.0.0.1:5000/predict_api Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL Text

```
1 {
2   "cp": 0,
3   "thalach": 160,
4   "exang": 0,
5   "slope": 2,
6   "ca": 2
7 }
```

Body Cookies Headers (4) Test Results Status: 200 OK Time: 62 ms Size: 146 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
  "prediction": 1
}
```