Machine Learning Project

Ioannis Ioannidis

University of Piraeus
NCSR Demokritos
Athens

13-02-2022

Project Purpose

- **Cardiovascular Diseases** (CVD's) are a group of disorders of the heart and blood vessels
- It constitutes the leading cause of death globally
- $1/3$ of the deaths occur prematurely in people under 70 years old
- This project is a trial of using ML algorithms in order to predict a possible CVD based on typical examinations

Dataset Overview

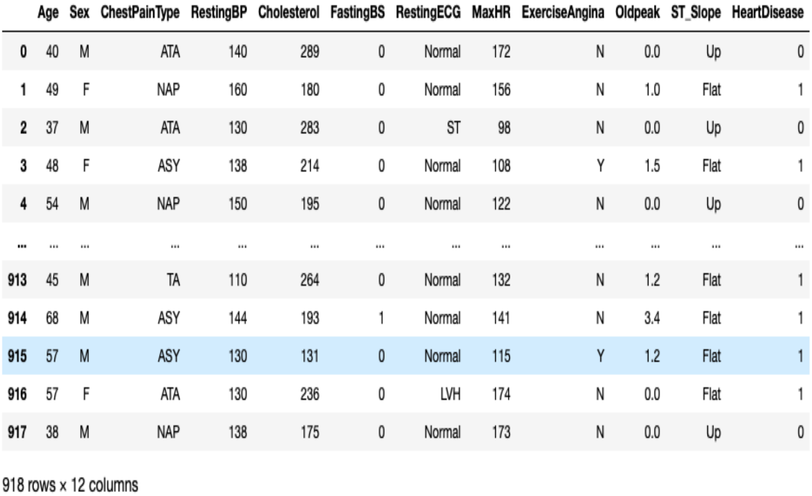
For the training procedure a dataset of 918 observations. Each one contains values of 12 features, where:

- **Age** is the patient's age (years)
- **Sex** is the patient's sex (M:Male, F:Female)
- **ChestPainType** describes a possible chest pain type (TA:Typical Angina, ATA:Atypical Angina, NAP:Non-Anginal Pain, ASY: Asymptomatic)
- **RestingBP** is the patient's resting blood pressure ($mmHg$)
- **Cholesterol** is the patient's serum cholesterol (mm/dl)
- **FastingBS** is the patient's fasting blood sugar level(1:FastingBS $> 120mg/dl$, 0: FastingBS $\leq 120mg/dl$)

Dataset Overview

- **RestingECG** describes patient's resting electrocardiogram results (Normal:Normal, ST:ST-T wave abnormality, LVH:left ventricular hypertrophy)
- **MaxHR** is the patient's maximum heart rate achieved (numeric value between 60 and 202)
- **ExerciseAngina** describes possible exercise-induced angina (Y:Yes, N:No)
- **Oldpeak** is the value of ST depression induced by exercise relative to rest
- **STSlope** describes the slope of the peak exercise ST segment (Up:upsloping, Flat:flat, Down:downsloping)
- **HeartDisease** is the output class (1:heart disease, 0: Normal)

Dataset Overview

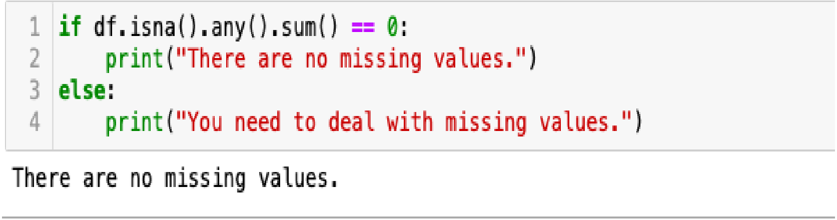


	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0
...	...	...	...	...	...	...	...	...	...	...	...	...
913	45	M	TA	110	264	0	Normal	132	N	1.2	Flat	1
914	68	M	ASY	144	193	1	Normal	141	N	3.4	Flat	1
915	57	M	ASY	130	131	0	Normal	115	Y	1.2	Flat	1
916	57	F	ATA	130	236	0	LVH	174	N	0.0	Flat	1
917	38	M	NAP	138	175	0	Normal	173	N	0.0	Up	0

918 rows x 12 columns

N/A Checking

- First step of the data cleaning is to check if there exist any missing values
- Using the following python script we can make sure that the present dataset does not contain any N/As

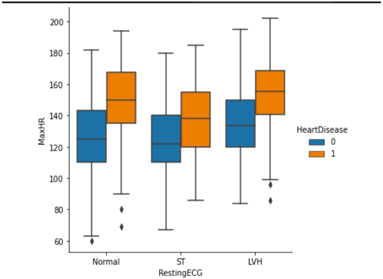


```python
if df.isna().any().sum() == 0:
    print("There are no missing values.")
else:
    print("You need to deal with missing values.")
```

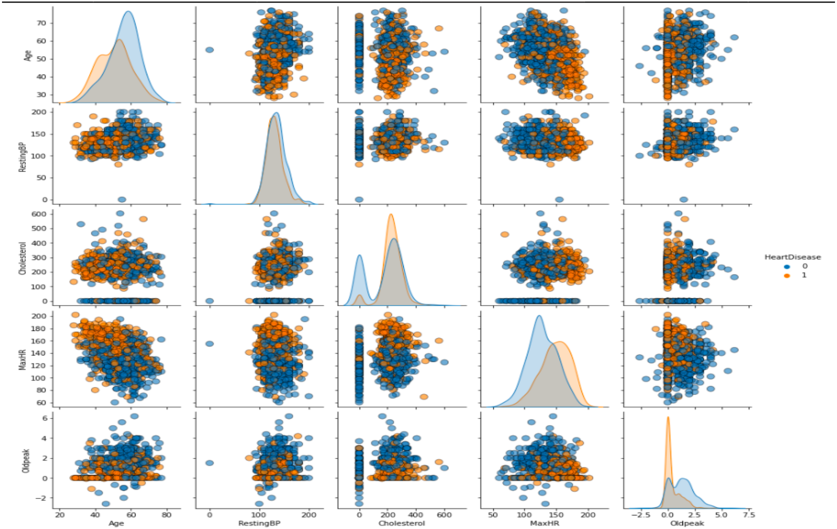
```
There are no missing values.
```

Outliers Detecting

- Another important procedure is to check if there exist any outliers.
- Outliers are nothing but data points or observations that fall outside of an expected distribution or pattern.
- For that purpose we visualize some pairplots and boxplots of the features

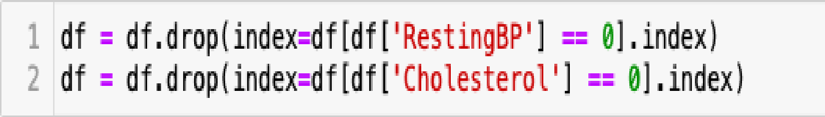


Outliers Detecting



Outliers Discard

- Examining the pairplots someone could easily notice that features ”Cholesterol” and ”RestingBP” include zeros in some observations, which do not correspond to normal values.
- Discarding those data, the remaining observation will be the ones used for the machine learning methods.



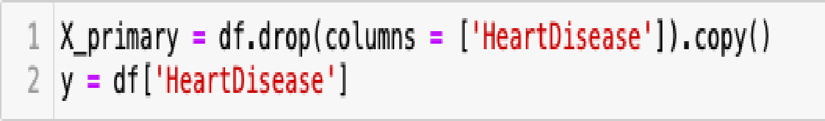
```python
df = df.drop(index=df[df['RestingBP'] == 0].index)
df = df.drop(index=df[df['Cholesterol'] == 0].index)
```

Separating Dependent - Independent Variables

Any machine learning model uses sets of given dependent and independent variables in the learning procedure

- Dependent variable is nothing but the variable which holds the phenomena which we are studying
- Independent variables are the ones which through we are trying to explain the value or effect of the output variable (dependent variable) by creating a relationship between an independent and dependent variable.

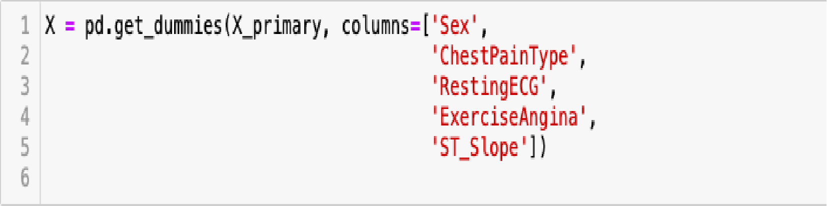
So it is a necessity to split the current dataframe into X, which will contain the features and y, which will contain the ouput class



```
X_primary = df.drop(columns = ['HeartDisease']).copy()
y = df['HeartDisease']
```

Encoding Categorical Data

- Categorical data must be converted to a numerical form, since there exist algorithms which cannot operate on label data directly
- However, even those who can deal with class names, promise more efficient results by following the encoding procedure
- For that purpose, we will apply One-Hot Encoding, which achieves that by replacing label by a binary variable, while the uniqueness is being secured

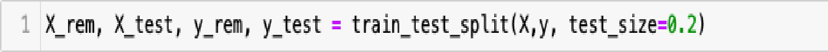


```
X = pd.get_dummies(X_primary, columns=['Sex',
                                      'ChestPainType',
                                      'RestingECG',
                                      'ExerciseAngina',
                                      'ST_Slope'])

```

Train - Test Split

We are, now, going to split the dataset into two subsets, the train and the test set

- Train set will be fitted in the machine learning models
- Test set will be used for evaluating the results
- The split is done following the 80:20 ratio. Thus, 20% of the data is set aside for final evaluation purposes

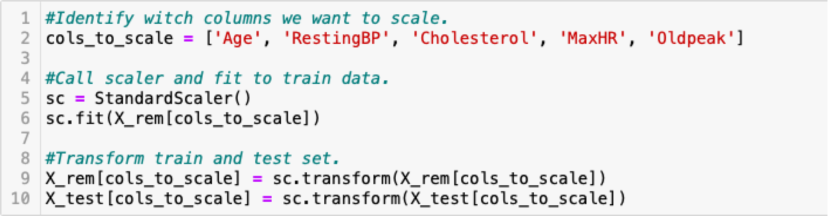


```python
X_rem, X_test, y_rem, y_test = train_test_split(X,y, test_size=0.2)
```

Feature Scaling

The last step is to apply feature scaling

- Feature scaling is a technique to standardize the independent features present in the data in a fixed range
- If it is not done, then a machine learning algorithm tends to weigh greater values, higher and consider smaller values as the lower values, regardless of the unit of the values
- In the present we will apply the standarization technique, which typically means rescaling each attributes data to have a mean of 0 and a standard deviation of 1, using the following tranformation $x_{scaled} = \frac{x-\mu}{\sigma}$



```
#Identify witch columns we want to scale.
cols_to_scale = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']

#Call scaler and fit to train data.
sc = StandardScaler()
sc.fit(X_rem[cols_to_scale])

#Transform train and test set.
X_rem[cols_to_scale] = sc.transform(X_rem[cols_to_scale])
X_test[cols_to_scale] = sc.transform(X_test[cols_to_scale])
```

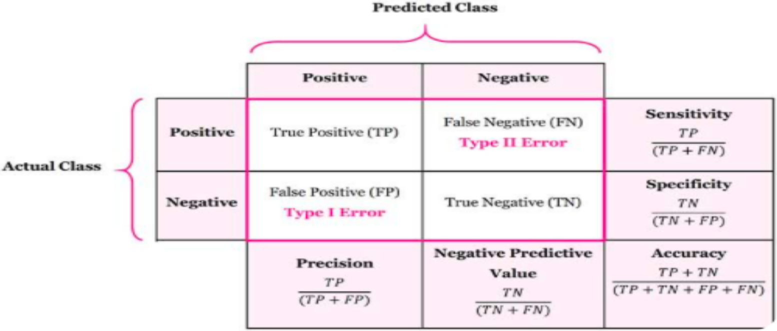
Path Planning

In the present moment we accomplished the data preprocessing procedure. Observations are now ready to train the algorithms. However we haven't specified yet which algorithms we are going to use. We need methods applied in supervised learning for binary classification, as

- Logistic Regression
- K - Nearest Neighbors
- Decision Tree
- Support Vector Machine

Path Planning

- Considering the fact that we are dealing with health data for the evaluation we are going to consult sensitivity (recall) and precision diagrams
- It is about two metrics describing confusion matrix in the following way



		Predicted Class		
		Positive	Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) Type II Error	Sensitivity $\frac{TP}{(TP + FN)}$
	Negative	False Positive (FP) Type I Error	True Negative (TN)	Specificity $\frac{TN}{(TN + FP)}$
		Precision $\frac{TP}{(TP + FP)}$	Negative Predictive Value $\frac{TN}{(TN + FN)}$	Accuracy $\frac{TP + TN}{(TP + TN + FP + FN)}$

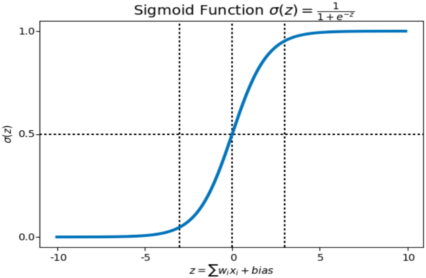
Path Planning

- What we would like to avoid is predicting a patient healthy, while he suffers from CVDs
- For that purpose we focus our attention on the False Negative (FN) Predictions
- Our goal is to minimize FNs, while at the same to time make sure our positive predictions are as valid as possible (maximize TPs)
- Maximizing recall, the metric that provides this ratio, we get attempted result

$$\textbf{Recall} = \frac{\textit{True Positives}}{\textit{True Positives} + \textit{False Negatives}}$$

Logistic Regression

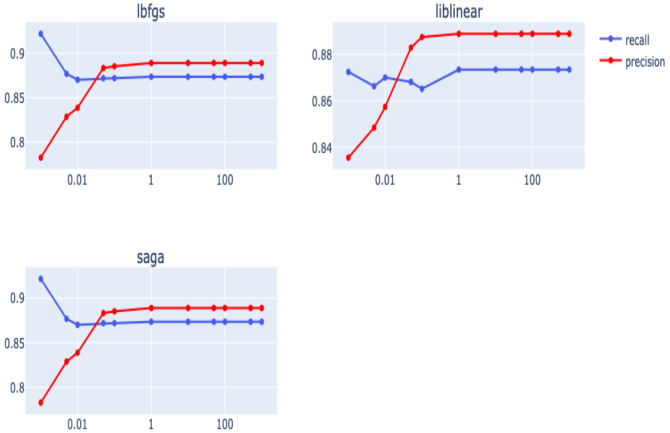
- Logistic regression is a predictive analysis algorithm based on the concept of probability
- In order to map predicted values to this concept it uses sigmoid function



- We expect the classifier to return an output result based on the probability score when passing inputs through the prediction function

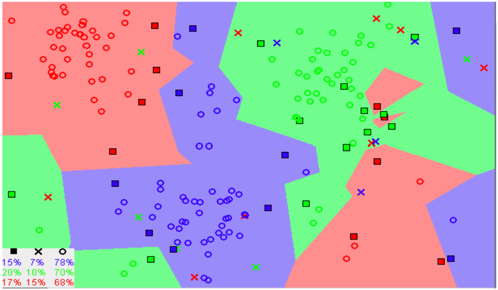
LR Hyperparameters Comparison

- Visualizing recall and precision curves over different solver types (subplots) and values of C (x-axis)



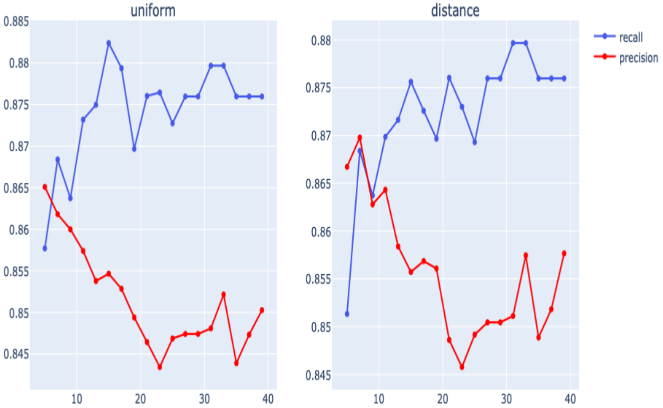
K-Nearest Neighbors

- KNN algorithm expresses the idea of similarity with some pretty easy mathematics
- It calculates the distance between points on a graph
- It assumes that similar things exist in close proximity
- In other words similar things are near to each other



KNN Hyperparameters Comparison

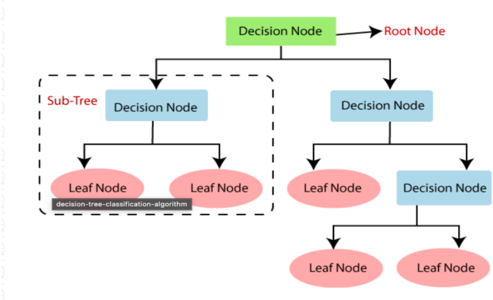
- Visualizing recall and precision curves over different weight types (subplots) and different values for the number of neighbors to use by default for kneighbors queries (x-axis)



Decision Tree

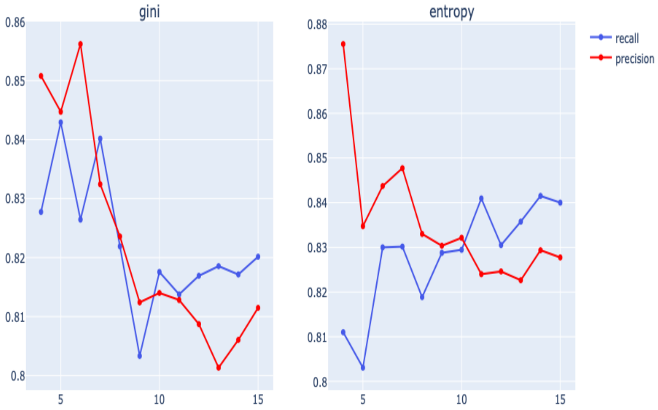
A Decision tree is a flowchart like tree structure, where

- each internal node denotes a test on an attribute
- each branch represents an outcome of the test
- each leaf node (terminal node) holds a class label



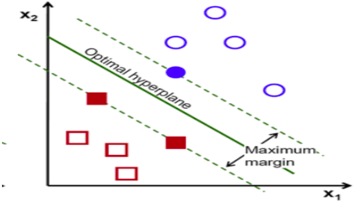
Tree Hyperparameters Comparison

- Visualizing recall and precision curves over different criterion types (subplots) and different values of maximum tree depth (x-axis)



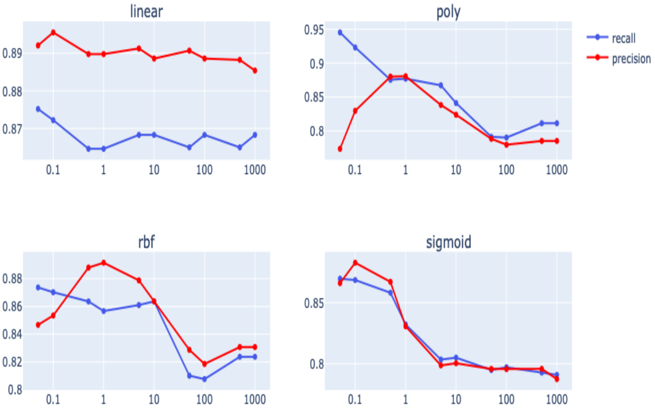
Support Vector Machines

- The objective of the support vector algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points
- To separate the two classes of data points there exist many possible hyperpanes
- SVM finds the one that has the maximum margin
- Maximizing the margin distance provides some reinforcement so that furure data points can be classifies with more confidence



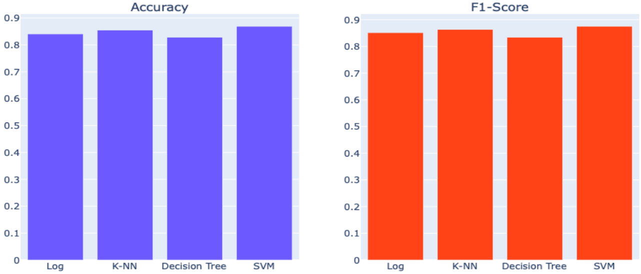
SVM Hyperparameters Comparison

- Visualizing recall and precision curves over different kernel types (subplots) and different values of C (x-axis)



Final Comparison

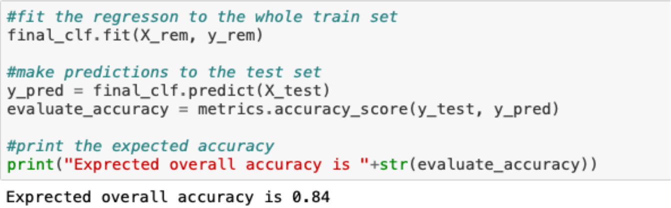
- Using the selected hyperparameters for each algorithm, we move on with a final evaluation testing accuracy and F1-Score



- We finally come up with Support Vector Classifier using polynomial kernel and $C = 0.5$

Evaluation

- In the training and test split we kept the 20% of the data for the final evaluation
- Using this we will measure the expected accuracy that our selected classifier provides

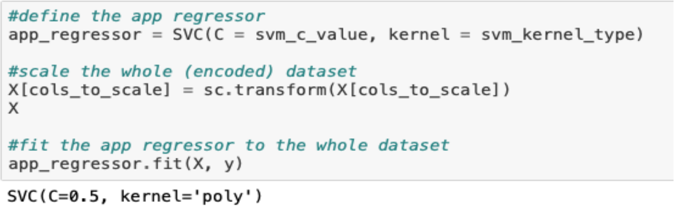


```
#fit the regresson to the whole train set
final_clf.fit(X_rem, y_rem)

#make predictions to the test set
y_pred = final_clf.predict(X_test)
evaluate_accuracy = metrics.accuracy_score(y_test, y_pred)

#print the expected accuracy
print("Exprected overall accuracy is "+str(evaluate_accuracy))
```

```
Exprected overall accuracy is 0.84
```

Fit All The Data

- Reaching the moment we used everything needed for training and validation methods we can now use the whole dataset for a last train for the classifier
- This procedure is necessary in order the final model exported won't lose any information from the available observations

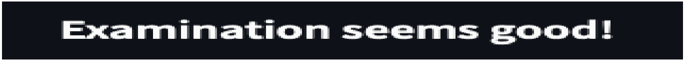


```
#define the app regressor
app_regressor = SVC(C = svm_c_value, kernel = svm_kernel_type)

#scale the whole (encoded) dataset
X[cols_to_scale] = sc.transform(X[cols_to_scale])
X

#fit the app regressor to the whole dataset
app_regressor.fit(X, y)
```

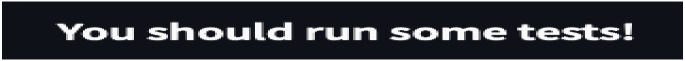
```
SVC(C=0.5, kernel='poly')
```

Result

- Using the final classifier we created a simple web app
- App is hosted by Heroku and you can visit it using the following URL `https://cvdspredictor.herokuapp.com`
- User can insert some of his last medical examination results in order to get a prediction depending on the area he/she is classified

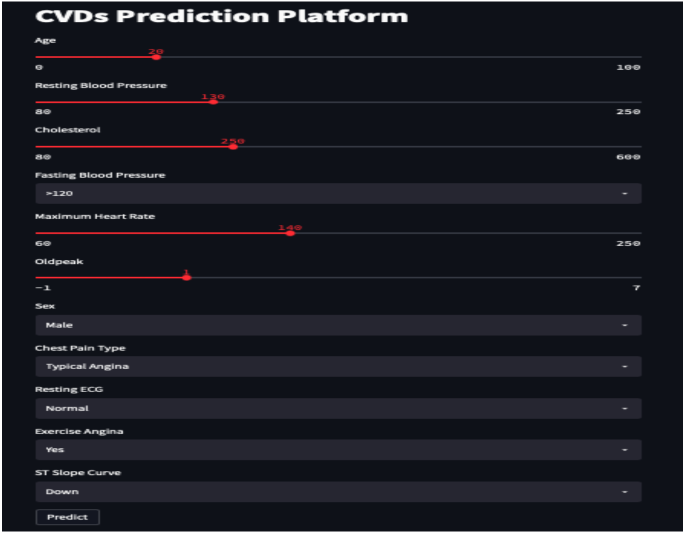


Examination seems good!



You should run some tests!

Web App



CVDs Prediction Platform

Age

20

0 100

Resting Blood Pressure

130

80 250

Cholesterol

250

80 600

Fasting Blood Pressure

>120

Maximum Heart Rate

140

60 250

Oldpeak

1

−1 7

Sex

Male

Chest Pain Type

Typical Angina

Resting ECG

Normal

Exercise Angina

Yes

ST Slope Curve

Down