Sri Lanka Institute of Information Technology

Final Report

Fundamentals of Data Mining – IT3051

Mini Project

Group-15

Predicting Risk Factors for Cardiovascular Heart Disease

IT21924750	Senevirathna W.P.U.
IT21468360	Rizvi F. A
IT21276200	Samarakoon S.M.R.S.B
IT21329760	Nanayakkara N.W.B.S

Declaration

This project report is our original work, and the content is not plagiarized from any other resource. References for all the content taken from external resources are correctly cited. To the best of our knowledge, this report does not contain any material published or written by third parties, except as acknowledged in the text.

Acknowledgement

We would like to convey our heartfelt gratitude to everyone who helped us to complete this project. We are especially grateful to our lecturer, Mr. Prasanna Sumathipala and our supervisors, Mr. Samadhi Chathuranga, Mr. Chan, and Ms. Supipi, who have supported and directed us in carrying out the project and have been observant since the first stage of the project, providing stimulating recommendations and encouragement. We would also like to express our heartfelt gratitude to our parents and friends.

Table of Contents

Background

Cardiovascular diseases (CVDs) are the leading cause of death globally, taking an estimated 17.9 million lives each year. CVDs are a group of disorders of the heart and blood vessels and include coronary heart disease, cerebrovascular disease, rheumatic heart disease, and other conditions. [1]

Individuals need to be aware of their risk factors and take steps to prevent and manage cardiovascular diseases to improve their overall health and longevity. Regular check-ups with healthcare providers can also help in the early detection and management of CVD (cardiovascular diseases) risk factors. [2]

To address this vital problem, our data mining project intends to use advanced analytics and machine learning to predict the risk of cardiovascular diseases by uncovering the hidden patterns within a diverse dataset.

The dataset provides extensive information regarding the multitude of elements that influence the likelihood of cardiovascular disease. It incorporates in-depth profiles of more than 70,000 individuals, encompassing details such as their age, gender, height, weight, blood pressure measurements, cholesterol levels, blood glucose levels, smoking habits, alcohol consumption, physical activity levels, and whether they have been diagnosed with cardiovascular diseases. [3]

We intend to develop an accurate and reliable approach for predicting the risk of cardiovascular disease that can be used both by healthcare professionals and individuals concerned about their heart condition. It will help predict risks and provide significant insight into the root causes of CVDs.

Scope of work

Our project consists of 5 layers.

1. User Interface Layer
2. Data wrangling and data cleansing layer
3. Data Mining Layer
4. Model building and Analysis Layer
5. Data Visualizing layer

1. User Interface Layer

Objective: To create a user-friendly environment, enhancing user interaction with the backend analytics.

The User Interface Layer constitutes the system's front end, serving as the interface through which users interact. It facilitates data selection and input and provides users with the analytics they need.

2. Data wrangling and data cleansing layer

Objective: Transform the raw data into a more suitable and valuable format for downstream analytical purposes.

In the Data Wrangling and Data Cleansing Layer, the focus is on cleaning and preprocessing data. This phase involves identifying and rectifying corrupt or inaccurate records while also pinpointing incomplete, inaccurate, or irrelevant data segments.

3. Data Mining Layer

Objective: Apply data mining techniques to the prepared data to uncover useful insights and patterns.

The Data Mining Layer involves the analysis of datasets using algorithms to extract valuable numeric information. Its main function is to unearth insights from the data and convert this information into a structured format that is easily comprehensible, paving the way for further analysis.

4. Model building and Analysis layer

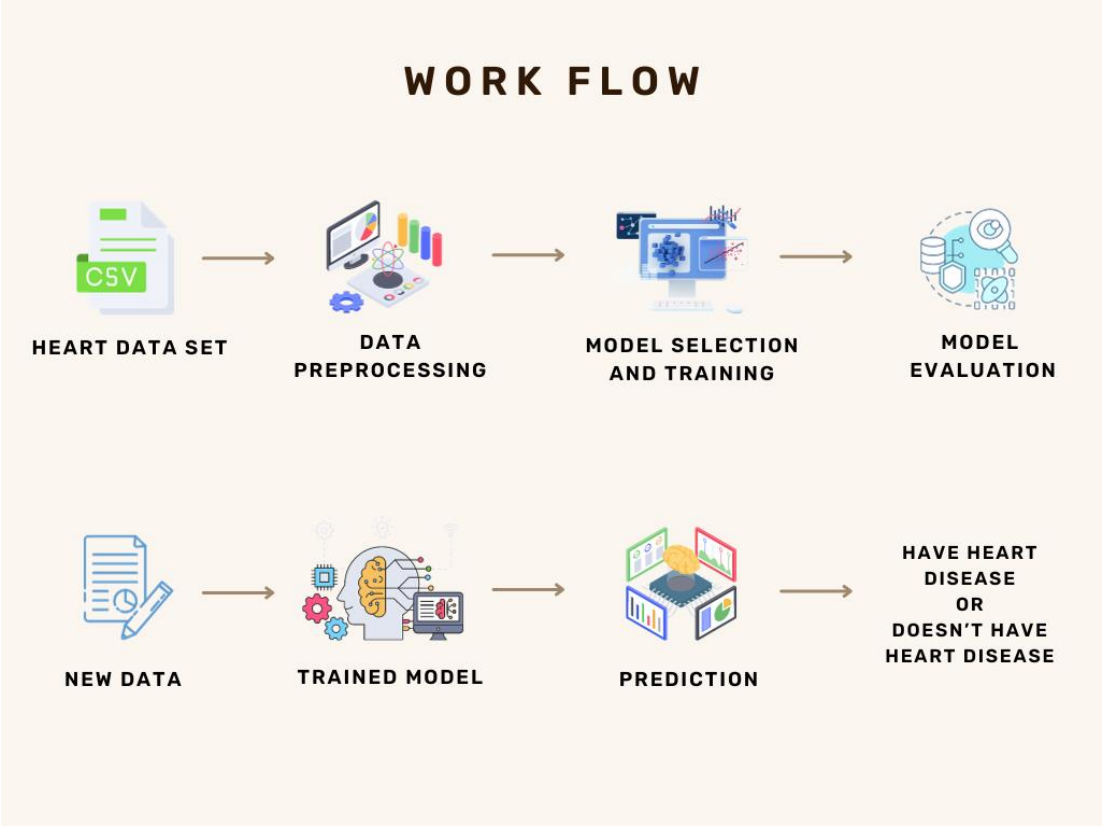
Objective: Develop predictive models and conduct in-depth analysis to aid decision-making.

This is the process of modeling data to predict whether a person has been diagnosed with cardiovascular disease. This layer creates predictive models that use the chosen dataset to predict desired outcomes from new data**.**

5. Data Visualizing Layer

Objective: Present the analyzed data and model outputs in graphical form for user comprehension.

The Data Visualizing Layer focuses on graphically representing the final analyzed results of characteristic data in an appealing and easily understandable manner for users. It utilizes appropriate graphs and a user-friendly interface to accomplish this task.



Methodology

1. Data Gathering
2. Data Pre-processing
3. Model Selection and Model Training
4. Model Evaluation
5. Model Deployment

1. Data Gathering

We used a public data set to build our heart disease prediction system.

Dataset Name – Risk Factors for Cardiovascular Heart Disease
Provided by – Kaggle.
Original Author – Kuzak Dempsy
Link to the dataset - Risk Factors for Cardiovascular Heart Disease (kaggle.com)

Details on the risk factors for cardiovascular disease are included in this dataset. Over 70 thousand people's age, gender, height, weight, blood pressure readings, cholesterol, glucose, smoking and drinking habits are all included in the data. It also states whether the person is active or not and whether they have any cardiovascular conditions.

1.1. Variable List

Attributes	Description	Type
Age	Age	Integer (Days)
Gender	Gender	Integer 1=male,0=female
Height	Height measured in centimeters	Integer
Weight	Weight measured in kilograms	Float
Ap_hi	Systolic blood pressure reading taken from patient	Integer
Ap_lo	Diastolic blood pressure reading taken from patient	Integer
Cholesterol	Total cholesterol level read as mg/dl on a scale 0 – 5+ units	Integer 1 – normal, 2- above normal, 3 – well above normal
Gluc	Glucose level read as mmol/l on a scale 0 – 16+ units	Integer 1 – normal, 2- above normal, 3 – well above normal
Smoke	Whether person smokes or not	Binary 1= yes, 0=no
Alco	Whether person drinks alcohol or not	Binary 1= yes, 0=no
Active	Whether person physically active or not	Binary 1= yes, 0=no
Cardio	Whether person suffers from cardiovascular diseases or not	Binary 1= yes, 0=no

2. Data Preprocessing

2.1. Load Dataset

- Loading the dataset.

```
[ ]  #loading the csv data to a pandas DataFrame
     heart_data = pd.read_csv('Heart_dataSet.csv')
```

```
[ ]  #print fist 5 rows of the data set
     heart_data.head()
```

	index	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0

- Checking info about the dataset

```
#getting some infor about data
heart_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   index        70000 non-null  int64  
 1   id           70000 non-null  int64  
 2   age          70000 non-null  int64  
 3   gender       70000 non-null  int64  
 4   height       70000 non-null  int64  
 5   weight       70000 non-null  float64
 6   ap_hi        70000 non-null  int64  
 7   ap_lo        70000 non-null  int64  
 8   cholesterol  70000 non-null  int64  
 9   gluc         70000 non-null  int64  
 10  smoke        70000 non-null  int64  
 11  alco         70000 non-null  int64  
 12  active       70000 non-null  int64  
 13  cardio       70000 non-null  int64  
dtypes: float64(1), int64(13)
memory usage: 7.5 MB
```

2.2.Data Cleaning

- Checking for missing values.

```
[ ]  #cheacking for missing values
     heart_data.isnull().sum()
```

```
index          0
id             0
age            0
gender         0
height         0
weight         0
ap_hi          0
ap_lo          0
cholesterol    0
gluc           0
smoke          0
alco           0
active         0
cardio         0
dtype: int64
```

- Checking for duplicate values.

```
[ ]  #CHECK DUPLICATE VALUES
     heart_data.duplicated().sum()
```

```
0
```

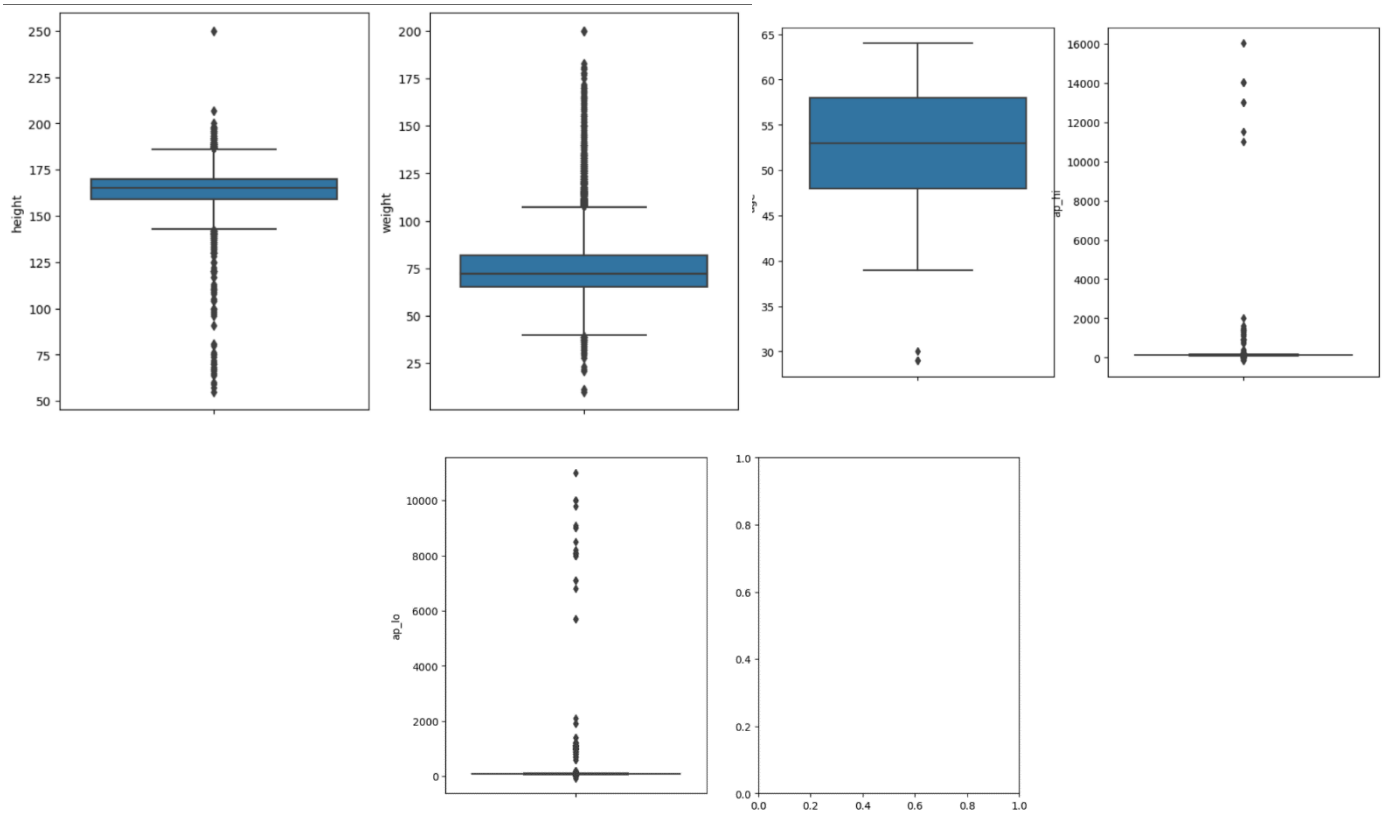
- Check unique values of specified Columns.

```
[ ]  # Data cleaning
     colmuns_with_labels = ["gender", "cholesterol", "gluc", "smoke", "alco", "active"]

     for i in colmuns_with_labels:
         x = heart_data[i].unique()
         x.sort()
         print(i + " has values: " + str(x))
```

```
gender has values: [1 2]
cholesterol has values: [1 2 3]
gluc has values: [1 2 3]
smoke has values: [0 1]
alco has values: [0 1]
active has values: [0 1]
```

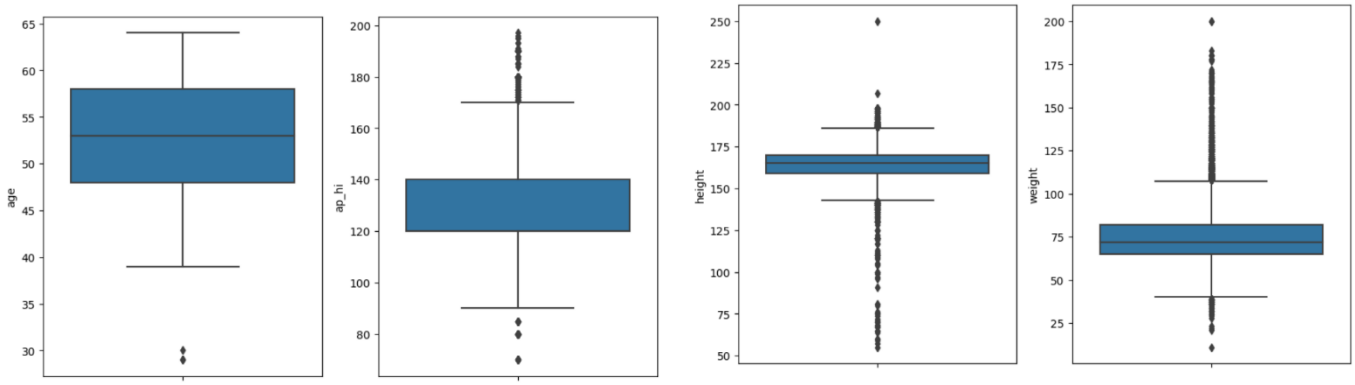
- Check for outliers.

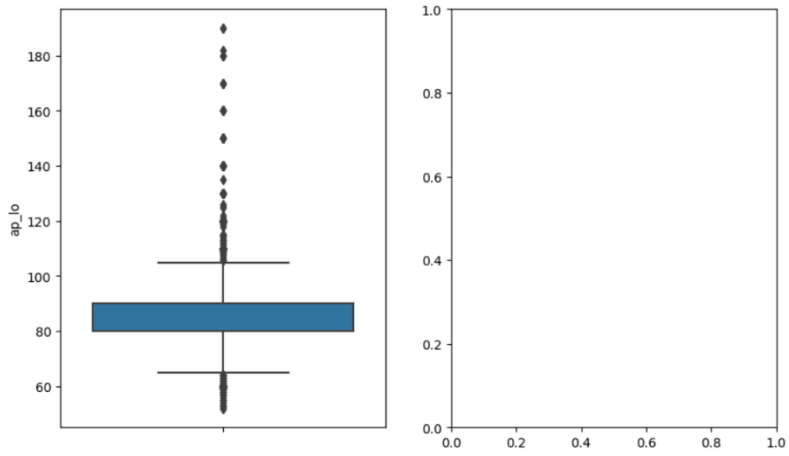
```
#check outlier
heart_data_physio = heart_data[["height","weight","age","ap_hi", "ap_lo"]] # these variables have range
fig, axes = plt.subplots(3,2,figsize=(10,20))

ind = 1
for i in heart_data_physio.columns:
    if ind <= 6:
        plt.subplot(3,2,ind)
        sns.boxplot(data = heart_data_physio, y=i )
    ind = ind+1
```



- Solve the outlier problem.

```
#resolve outlier

heart_data = heart_data[(heart_data["ap_hi"] < 200) & (heart_data["ap_hi"] > 50) & (heart_data["ap_lo"] < 200) & (heart_data["ap_lo"] > 50)]
```





- Statistical Measures of data

```
#statistical measures about the data
heart_data.describe()
```

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
count	68550.000000	68550.000000	68550.000000	68550.000000	68550.000000	68550.000000	68550.000000	68550.000000	68550.000000	68550.000000	68550.000000	68550.000000
mean	52.827936	1.348607	164.366798	74.105689	126.465631	81.350343	1.363968	1.225514	0.087819	0.053494	0.803399	0.494311
std	6.769248	0.476533	8.179705	14.300444	16.329647	9.480968	0.678518	0.571600	0.283034	0.225018	0.397431	0.499971
min	29.000000	1.000000	55.000000	11.000000	70.000000	52.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000
25%	48.000000	1.000000	159.000000	65.000000	120.000000	80.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000
50%	53.000000	1.000000	165.000000	72.000000	120.000000	80.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000
75%	58.000000	2.000000	170.000000	82.000000	140.000000	90.000000	1.000000	1.000000	0.000000	0.000000	1.000000	1.000000
max	64.000000	2.000000	250.000000	200.000000	197.000000	190.000000	3.000000	3.000000	1.000000	1.000000	1.000000	1.000000

- Checking for the distribution of the target variable

```
# cheacking the distributing of target variable
heart_data['cardio'].value_counts()
```

```
0    34665
1    33885
Name: cardio, dtype: int64
```

1--> Defective heart

0--> Healthy Haart

2.3. Data Analysis

- Transformed the values in specific columns to more interpretable labels or categories as it makes the data more human-readable and facilitates better understanding during data analysis and exploration.

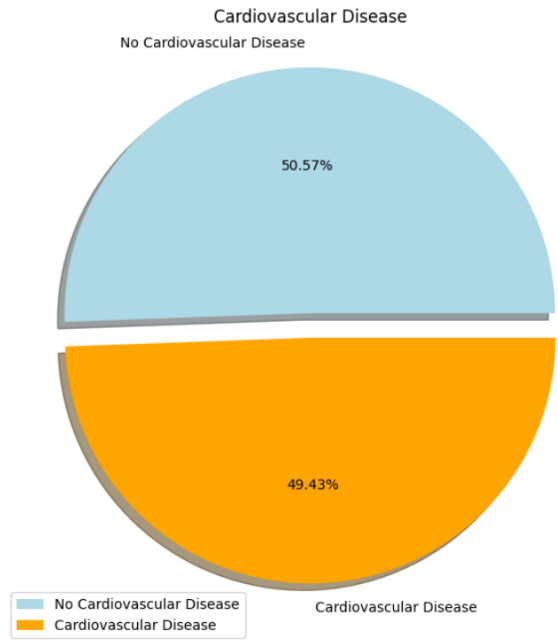
Analyze **data**

```
heart_data_analyze = heart_data.copy()
heart_data_analyze["gender"]= heart_data_analyze["gender"].apply(lambda x: "male" if x == 1 else "female")
heart_data_analyze["alco"]= heart_data_analyze["alco"].apply(lambda x: "No" if x == 0 else "Yes")
heart_data_analyze["smoke"]= heart_data_analyze["smoke"].apply(lambda x: "No" if x == 0 else "Yes")
heart_data_analyze["active"]= heart_data_analyze["active"].apply(lambda x: "No" if x == 0 else "Yes")

def transform_value(val):
    if val == 1:
        return "type 1"
    elif val == 2:
        return "type 2"
    else:
        return "type 3"

heart_data_analyze['cholesterol']= heart_data_analyze['cholesterol'].apply(transform_value)
heart_data_analyze['gluc']= heart_data_analyze['gluc'].apply(transform_value)
```

```
heart_data_analyze.head()
heart_data_test = heart_data_analyze
```

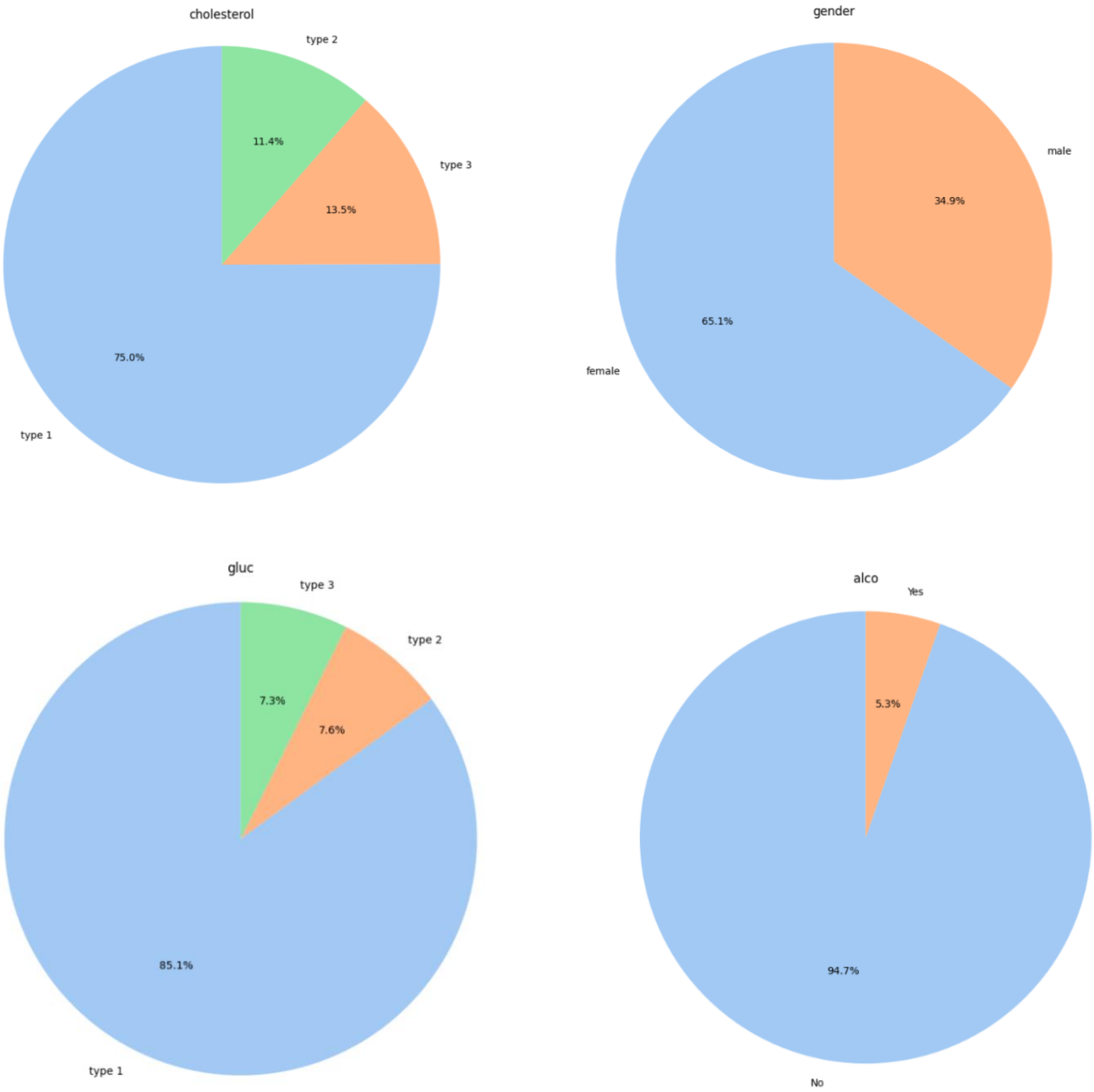
- Visualized the distribution of cardiovascular disease.

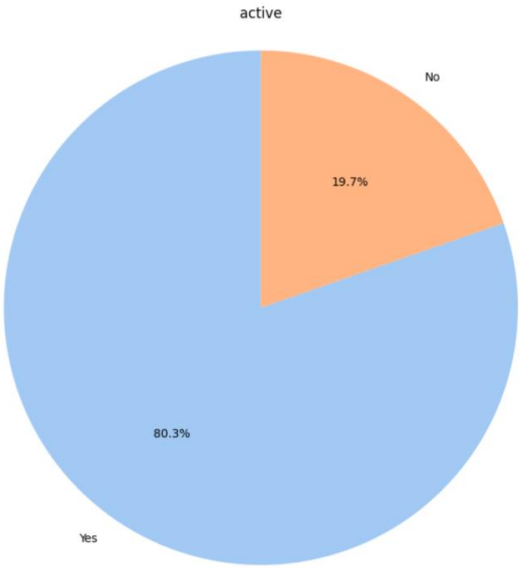
```
plt.figure(figsize = (10, 8))
plt.title('Cardiovascular Disease')
plt.pie(heart_data_analyze['cardio'].value_counts(), labels = ['No Cardiovascular Disease', 'Cardiovascular Disease'], explode = (0.1, 0.0), colors = ['lightblue', 'orange'],autopct = '%1.2f%
plt.legend(loc = 'best')
```

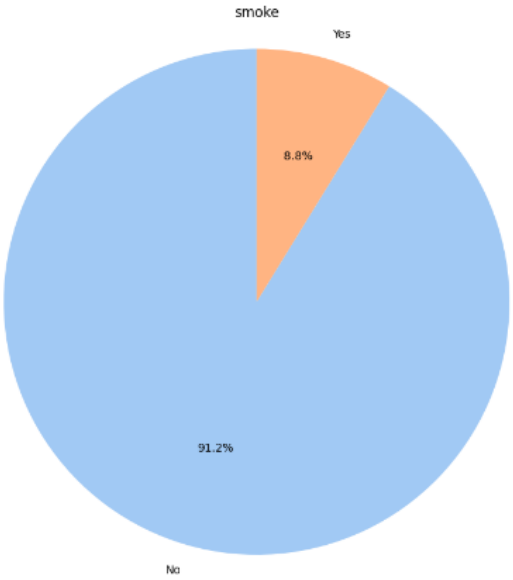


- Checked the distribution for categorical features.

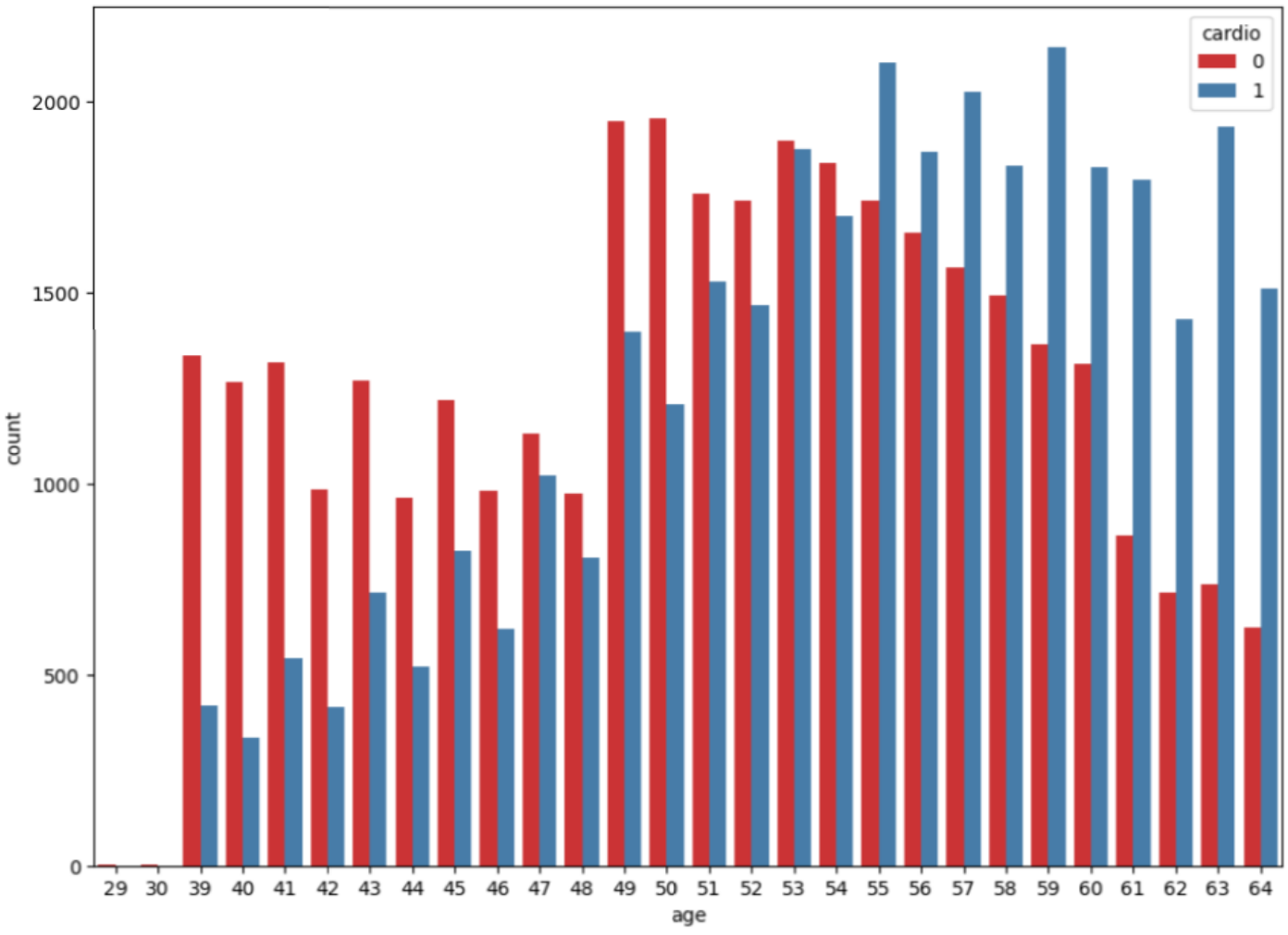
```
[ ]  plt.figure(figsize=(30, 50))

     ind = 0
     for i in colmuns_with_labels:
         ind = ind + 1
         plt.subplot(7, 2, ind)

         plt.pie(heart_data_analyze[i].value_counts(), labels=heart_data_analyze[i].unique(), autopct='%1.1f%%', colors=sns.color_palette('pastel'), startangle=90)

         plt.axis('equal')
         plt.title(i)

     plt.tight_layout()
     plt.show()
```







- Distribution of cardio heart disease for age groups.



2.4.Feature Selection and Data Transformation

- Removed the ‘index’ and ‘id’ columns from the dataset.
- Converted age from days to years by dividing each age value by 365 and then rounded down to the nearest integer.
- Converted weight values from float to integer.

```
[ ]  heart_data.drop(['index', 'id'], axis = 1, inplace = True)

[ ]  #convert age into int , weight in to int
     heart_data['age'] = heart_data['age'].apply(lambda x : int(x/365))
     heart_data['weight'] =heart_data['weight'].apply(lambda x : int(x))
```

- Calculated BMI and Pulse Pressure values as they are useful features in health care related data sets as they provide insights into factors affecting cardiovascular health.

Splitting the Features and target / Feature Enginnering

```
[ ]  #BMI, risk level scores, and pulse pressure are useful features in healthcare-related datasets as they provide insights into factors affecting cardiovascular health.
     heart_data['pp'] = heart_data['ap_hi']-heart_data['ap_lo']
     heart_data["bmi"] = heart_data["weight"]/((heart_data["height"]/100)*(heart_data["height"]/100))
```

- Calculated health risk score, according to assigned weight for different health risk factors.

Heakth Risk Metric

Cholesterol Level: Weight = 3 (higher weight because cholesterol is a significant health risk factor).

Smoking Status: Weight = 3 (higher weight due to the strong association with health risks).

Glucose Level: Weight = 2 (moderate weight as elevated glucose levels are concerning).

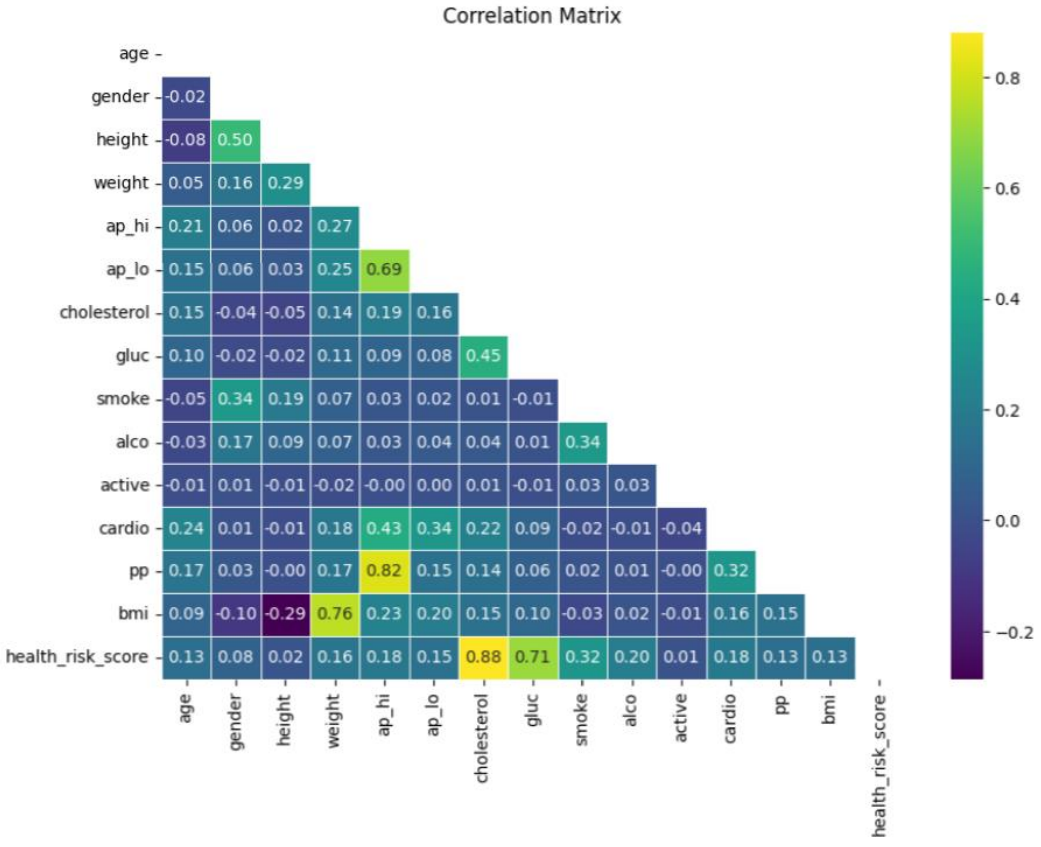
Alcohol Consumption: Weight = 1 (lower weight as it has a less direct impact compared to the other factors).

```
weights = {
    'Chol': 3,
    'Smoke': 3,
    'Gluc': 2,
    'Alco': 1
}

heart_data['health_risk_score'] = heart_data['cholesterol'] * weights['Chol'] + \
                          heart_data['gluc'] * weights['Gluc'] + \
                          heart_data['smoke'] * weights['Smoke'] + \
                          heart_data['alco'] * weights['Alco']
```

- Computed the pairwise correlation of columns.

```
heart_data_corr = heart_data.corr()
heart_data_corr
```

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio	pp	bmi	health_risk_score
age	1.000000	-0.023766	-0.081747	0.053579	0.209980	0.152448	0.154858	0.098591	-0.047765	-0.029125	-0.010318	0.239244	0.166412	0.086115	0.130530
gender	-0.023766	1.000000	0.499046	0.155996	0.059334	0.064723	-0.037030	-0.021268	0.338259	0.170557	0.005251	0.006580	0.029805	-0.096834	0.077431
height	-0.081747	0.499046	1.000000	0.292858	0.017793	0.033899	-0.050681	-0.018871	0.188046	0.094769	-0.008166	-0.011058	-0.002587	-0.287351	0.019283
weight	0.053579	0.155996	0.292858	1.000000	0.266995	0.246855	0.140194	0.106325	0.067095	0.068213	-0.017189	0.178579	0.169429	0.763749	0.164265
ap_hi	0.209980	0.059334	0.017793	0.266995	1.000000	0.692645	0.194188	0.091041	0.025367	0.030652	-0.000874	0.429148	0.819053	0.229669	0.180907
ap_lo	0.152448	0.064723	0.033899	0.246855	0.692645	1.000000	0.159600	0.075195	0.023523	0.042391	0.000746	0.336871	0.153504	0.204935	0.150935
cholesterol	0.154858	-0.037030	-0.050681	0.140194	0.194188	0.159600	1.000000	0.450928	0.009945	0.035384	0.009424	0.221245	0.139088	0.145595	0.880024
gluc	0.098591	-0.021268	-0.018871	0.106325	0.091041	0.075195	0.450928	1.000000	-0.005554	0.011120	-0.007240	0.089405	0.064915	0.101659	0.705701
smoke	-0.047765	0.338259	0.188046	0.067095	0.025367	0.023523	0.009945	-0.005554	1.000000	0.338539	0.025229	-0.016469	0.016043	-0.027833	0.321911
alco	-0.029125	0.170557	0.094769	0.068213	0.030652	0.042391	0.035384	0.011120	0.338539	1.000000	0.025111	-0.008381	0.008274	0.015268	0.204700
active	-0.010318	0.005251	-0.008166	-0.017189	-0.000874	0.000746	0.009424	-0.007240	0.025229	0.025111	1.000000	-0.037308	-0.001790	-0.013179	0.013017
cardio	0.239244	0.006580	-0.011058	0.178579	0.429148	0.336871	0.221245	0.089405	-0.016469	-0.008381	-0.037308	1.000000	0.319978	0.163605	0.183957
pp	0.166412	0.029805	-0.002587	0.169429	0.819053	0.153504	0.139088	0.064915	0.016043	0.008274	-0.001790	0.319978	1.000000	0.151637	0.127786
bmi	0.086115	-0.096834	-0.287351	0.763749	0.229669	0.204935	0.145595	0.101659	-0.027833	0.015268	-0.013179	0.163605	0.151637	1.000000	0.134495
health_risk_score	0.130530	0.077431	0.019283	0.164265	0.180907	0.150935	0.880024	0.705701	0.321911	0.204700	0.013017	0.183957	0.127786	0.134495	1.000000



- Dropped features with the absolute value of correlation with cardio < 0.1.

Drop features with the absulute value of correlation with cardio < 0.1

```
[31] heart_data = heart_data.drop(["gender","height","smoke","alco","gluc","active"], axis="columns")
```

- Checked the data set after feature engineering.

```
heart_data.head()
```

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio	pp	bmi	health_risk_score
0	50	2	168	62	110	80	1	1	0	0	1	0	30	21.967120	5
1	55	1	156	85	140	90	3	1	0	0	1	1	50	34.927679	11
2	51	1	165	64	130	70	3	1	0	0	0	1	60	23.507805	11
3	48	2	169	82	150	100	1	1	0	0	1	1	50	28.710479	5
4	47	1	156	56	100	60	1	1	0	0	0	0	40	23.011177	5

3. Model Selection and Model Training

- Split the dataset into train and test data.

```
Split in to test and train data

[ ]  X = heart_data.drop(columns='cardio', axis=1)
     Y = heart_data['cardio']
```

3.1. Logistic Regression

```
# Split the data
# X = heart_data.drop('cardio', axis=1)
# y = heart_data['cardio']
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

#Initialize the model1
model1 = LogisticRegression(max_iter=1000)  # Adjust max_iter as needed

# Lists to store training and testing metrics
train_accuracy = []
test_accuracy = []
train_loss = []
test_loss = []
precision_scores = []
recall_scores = []

# Train the model1 and collect metrics
for i in range(100):  # You can adjust the number of iterations
    model1.fit(X_train, y_train)

    # Training accuracy
    X_train_prediction = model1.predict(X_train)
    training_data_accuracy = accuracy_score(X_train_prediction, y_train)
    train_accuracy.append(training_data_accuracy)

    # Testing accuracy
    y_pred = model1.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    test_accuracy.append(accuracy)

    # Precision and Recall
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    precision_scores.append(precision)
    recall_scores.append(recall)

    # You can also add code here to collect loss values if the XGBoost library you're using provides them.
    # For example, you can collect them after each boosting round.

print('Accuracy on Training data: ',training_data_accuracy)
print("Accuracy on Testing data:", accuracy)
print("precission",precision)
print("recall : ",recall)
```

```
Accuracy on Training data:  0.7256746900072939
Accuracy on Testing data: 0.7277899343544858
precission 0.7487512487512488
recall :  0.6691964285714286
```

- We also checked the accuracy for the trained logistic regression model using the data before cleaning. We observed that the accuracy was low compared to the model we trained using cleaned and transformed data set.

```
[ ]  # Accuracy on training data
     # X_train_prediction = model.predict(X_train)
     # training_data_accuracy = accuracy_score(X_train_prediction, Y_train)

[ ]  # print('Accuracy on Training data: ',training_data_accuracy)

     Accuracy on Training data:  0.5939095550692924

[ ]  # Accuracy on test data
     # X_test_prediction = model.predict(X_test)
     # test_data_accuracy = accuracy_score(X_test_prediction, Y_test)

[ ]  # print('Accuracy on Test data: ',test_data_accuracy)

     Accuracy on Test data:  0.5943107221006565
```

3.2.XGB - XGBoost

```
# Split the data
X = heart_data.drop('cardio', axis=1)
y = heart_data['cardio']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the model2
model2 = xgb.XGBClassifier()

# Lists to store training and testing metrics
train_accuracy = []
test_accuracy = []
train_loss = []
test_loss = []
precision_scores = []
recall_scores = []

# Train the model2 and collect metrics
for i in range(100):  # You can adjust the number of iterations
    model2.fit(X_train, y_train)

    # Training accuracy
    X_train_prediction = model2.predict(X_train)
    training_data_accuracy = accuracy_score(X_train_prediction, y_train)
    train_accuracy.append(training_data_accuracy)

    # Testing accuracy
    y_pred = model2.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    test_accuracy.append(accuracy)

    # Precision and Recall
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    precision_scores.append(precision)
    recall_scores.append(recall)

    # You can also add code here to collect loss values if the XGBoost library you're using provides them.
    # For example, you can collect them after each boosting round.

print('Accuracy on Training data: ',training_data_accuracy)
print("Accuracy on Testing data:", accuracy)
print("precission",precision)
print("recall : ",recall)
```

```
Accuracy on Training data:  0.7584062727935813
Accuracy on Testing data: 0.7324580598103574
precission 0.7477272727272727
recall :  0.6854166666666667
```

3.3.Decision Tree

Decision Tree

```python
# Split the data
X = heart_data.drop('cardio', axis=1)
y = heart_data['cardio']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#Initialize the model3
model3 = DecisionTreeClassifier()

#Lists to store training and testing metrics

train_accuracy = []
test_accuracy = []
train_loss = []
test_loss = []
precision_scores = []
recall_scores = []

# Train the model3 and collect metrics
for i in range(100):  # You can adjust the number of iterations
    model3.fit(X_train, y_train)

    # Training accuracy
    X_train_prediction = model3.predict(X_train)
    training_data_accuracy = accuracy_score(X_train_prediction, y_train)
    train_accuracy.append(training_data_accuracy)

    # Testing accuracy
    y_pred = model3.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    test_accuracy.append(accuracy)

    # Precision and Recall
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    precision_scores.append(precision)
    recall_scores.append(recall)
```

```python
print('Accuracy on Training data: ',training_data_accuracy)
print("Accuracy on Testing data:", accuracy)
print("precission",precision)
print("recall : ",recall)
```

```
Accuracy on Training data:  0.9653355215171407
Accuracy on Testing data: 0.6409190371991247
precission 0.6398879028491359
recall :  0.6116071428571429
```

3.4.Support Vector Classifier

Support Vector Classifier

```
[39] from sklearn.svm import SVC
     import numpy as np

     # Split the data
     X = heart_data.drop('cardio', axis=1)
     y = heart_data['cardio']
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

     # Initialize the model4
     model4 = SVC()

     # Lists to store training and testing metrics
     train_accuracy = []
     test_accuracy = []
     precision_scores = []
     recall_scores = []

     # Train the model4 for a single iteration
     model4.fit(X_train, y_train)

     # Training accuracy
     X_train_prediction = model4.predict(X_train)
     training_data_accuracy = accuracy_score(X_train_prediction, y_train)
     train_accuracy.append(training_data_accuracy)

     # Testing accuracy
     y_pred = model4.predict(X_test)
     accuracy = accuracy_score(y_test, y_pred)
     test_accuracy.append(accuracy)

     # Precision and Recall
     precision = precision_score(y_test, y_pred)
     recall = recall_score(y_test, y_pred)
     precision_scores.append(precision)
     recall_scores.append(recall)
```

```
print('Accuracy on Training data: ',training_data_accuracy)
print("Accuracy on Testing data:", accuracy)
print("precission",precision)
print("recall : ",recall)
```

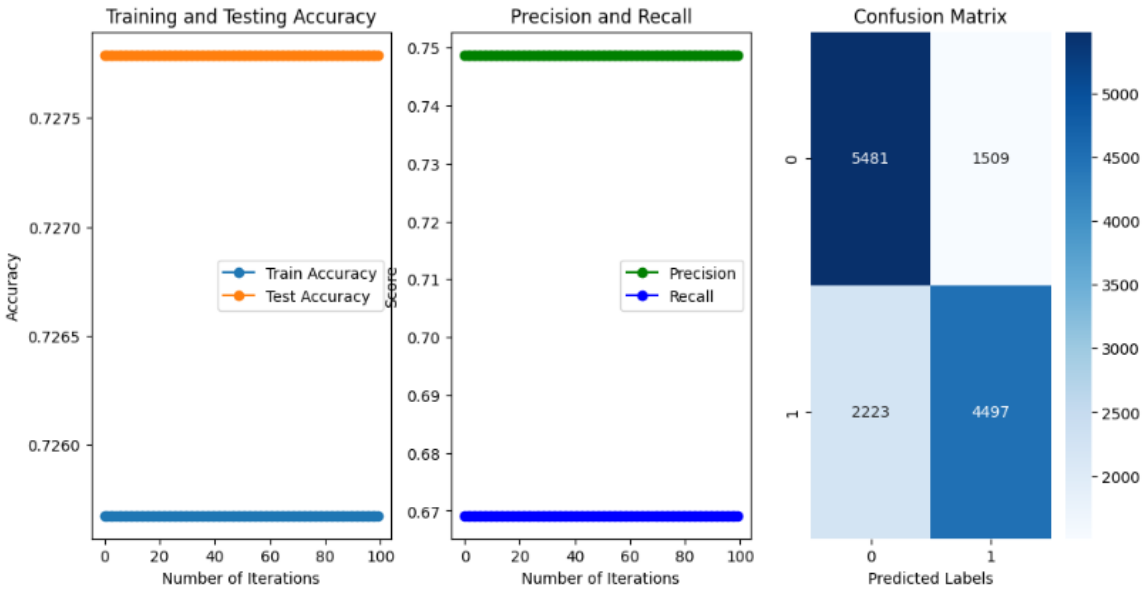
```
Accuracy on Training data:  0.7237235594456601
Accuracy on Testing data: 0.7261852662290299
precission 0.7608160393950053
recall :  0.64375
```

4. Model Evaluation

- We used Accuracy, Precision and Recall as evaluation metrices.

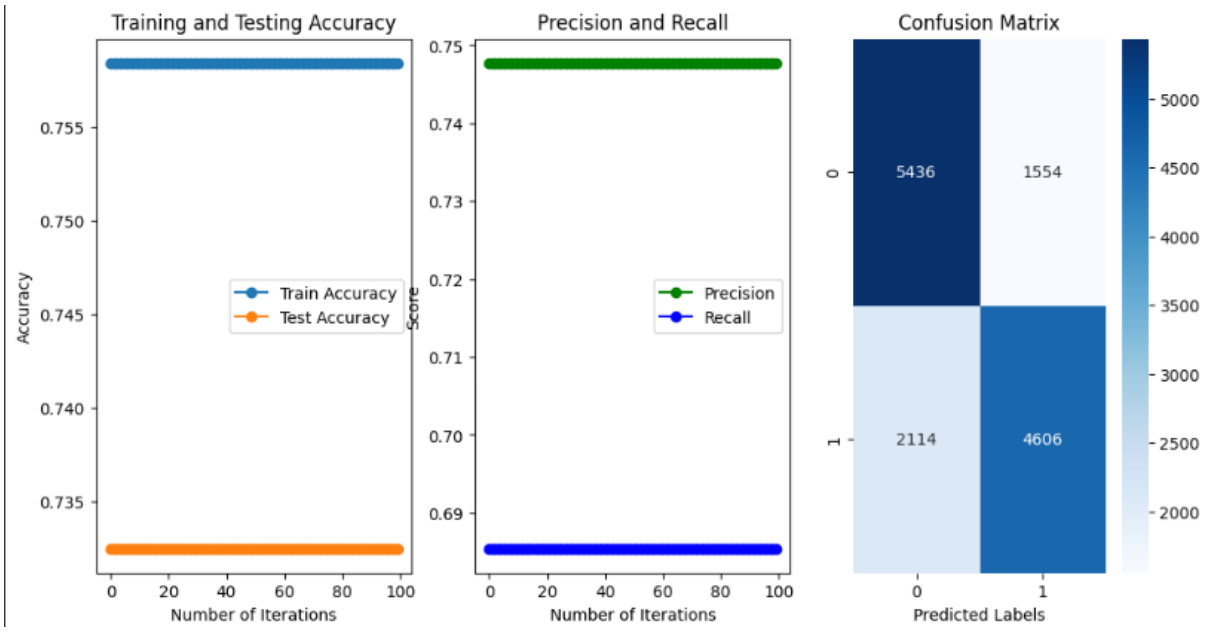
4.1. Logistic Regression

```
# Create a plot
plt.figure(figsize=(12, 6))

plt.subplot(1, 3, 1)
plt.plot(train_accuracy, label='Train Accuracy', marker='o')
plt.plot(test_accuracy, label='Test Accuracy', marker='o')
plt.xlabel('Number of Iterations')
plt.ylabel('Accuracy')
plt.title('Training and Testing Accuracy')
plt.legend()

plt.subplot(1, 3, 2)
plt.plot(precision_scores, label='Precision', marker='o', color='g')
plt.plot(recall_scores, label='Recall', marker='o', color='b')
plt.xlabel('Number of Iterations')
plt.ylabel('Score')
plt.title('Precision and Recall')
plt.legend()

plt.subplot(1, 3, 3)
# Plot train and test loss if available

# Display the confusion matrix
cm_LogisticRegression = confusion_matrix(y_test, y_pred)
sns.heatmap(cm_LogisticRegression, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted Labels')
plt.title('Confusion Matrix')

plt.show()
```



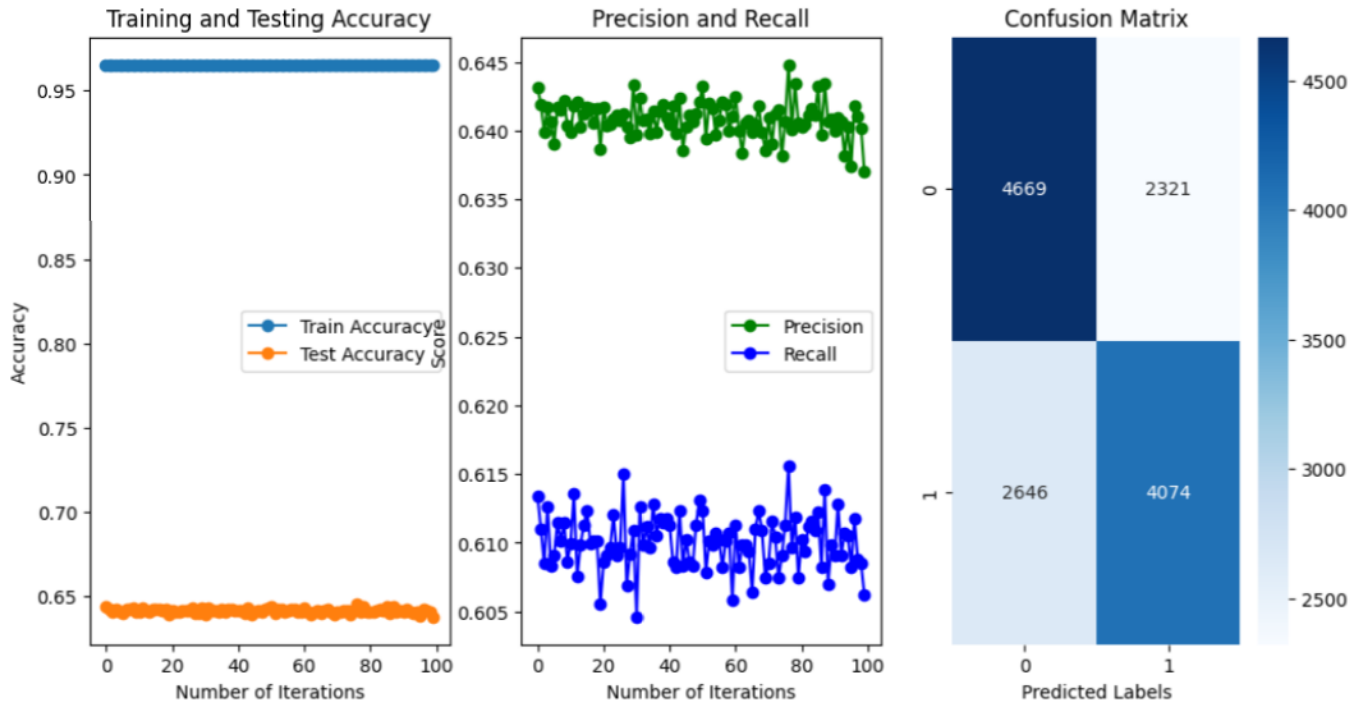
4.2.XGB – XGBoost

```
# Create a plot
plt.figure(figsize=(12, 6))

plt.subplot(1, 3, 1)
plt.plot(train_accuracy, label='Train Accuracy', marker='o')
plt.plot(test_accuracy, label='Test Accuracy', marker='o')
plt.xlabel('Number of Iterations')
plt.ylabel('Accuracy')
plt.title('Training and Testing Accuracy')
plt.legend()

plt.subplot(1, 3, 2)
plt.plot(precision_scores, label='Precision', marker='o', color='g')
plt.plot(recall_scores, label='Recall', marker='o', color='b')
plt.xlabel('Number of Iterations')
plt.ylabel('Score')
plt.title('Precision and Recall')
plt.legend()

plt.subplot(1, 3, 3)
# Plot train and test loss if available

# Display the confusion matrix
cm_xgb = confusion_matrix(y_test, y_pred)
sns.heatmap(cm_xgb, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted Labels')
plt.title('Confusion Matrix')

plt.show()
```



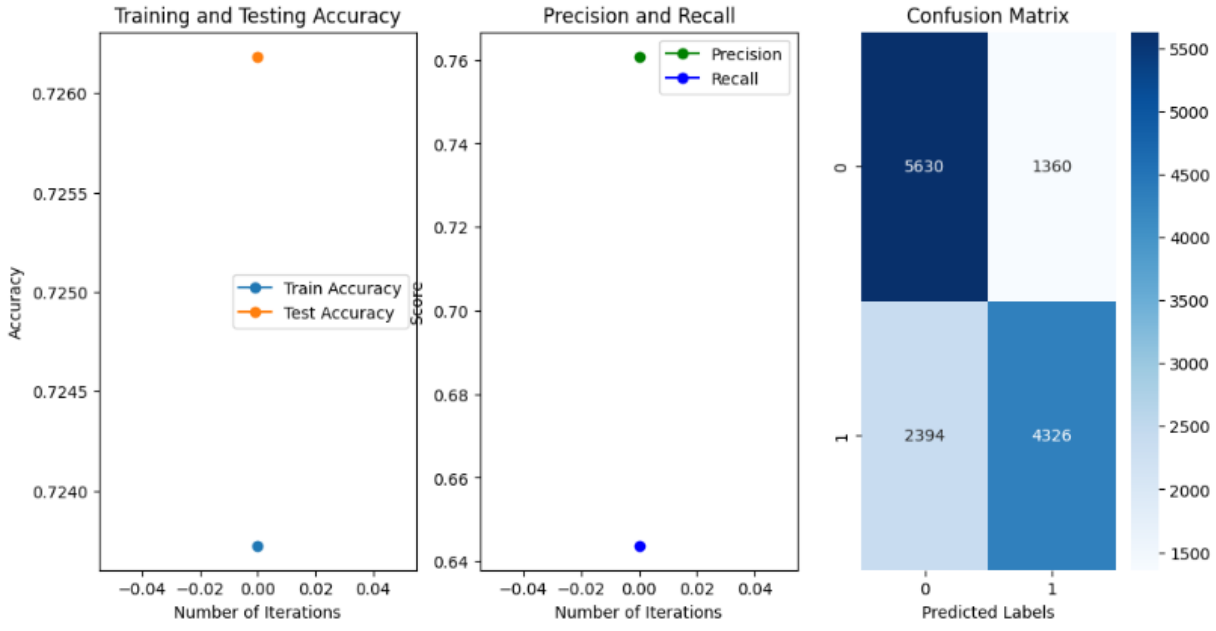
4.3.Decision Tree

```
# Create a plot
plt.figure(figsize=(12, 6))

plt.subplot(1, 3, 1)
plt.plot(train_accuracy, label='Train Accuracy', marker='o')
plt.plot(test_accuracy, label='Test Accuracy', marker='o')
plt.xlabel('Number of Iterations')
plt.ylabel('Accuracy')
plt.title('Training and Testing Accuracy')
plt.legend()

plt.subplot(1, 3, 2)
plt.plot(precision_scores, label='Precision', marker='o', color='g')
plt.plot(recall_scores, label='Recall', marker='o', color='b')
plt.xlabel('Number of Iterations')
plt.ylabel('Score')
plt.title('Precision and Recall')
plt.legend()

plt.subplot(1, 3, 3)
# Plot train and test loss if available

# Display the confusion matrix
cm_tree = confusion_matrix(y_test, y_pred)
sns.heatmap(cm_tree, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted Labels')
plt.title('Confusion Matrix')

plt.show()
```



4.4.Support Vector Classifier

```
# Create a plot
plt.figure(figsize=(12, 6))

plt.subplot(1, 3, 1)
plt.plot(train_accuracy, label='Train Accuracy', marker='o')
plt.plot(test_accuracy, label='Test Accuracy', marker='o')
plt.xlabel('Number of Iterations')
plt.ylabel('Accuracy')
plt.title('Training and Testing Accuracy')
plt.legend()

plt.subplot(1, 3, 2)
plt.plot(precision_scores, label='Precision', marker='o', color='g')
plt.plot(recall_scores, label='Recall', marker='o', color='b')
plt.xlabel('Number of Iterations')
plt.ylabel('Score')
plt.title('Precision and Recall')
plt.legend()

plt.subplot(1, 3, 3)
# Plot train and test loss if available

# Display the confusion matrix
cm_svc = confusion_matrix(y_test, y_pred)
sns.heatmap(cm_svc, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted Labels')
plt.title('Confusion Matrix')

plt.show()
```

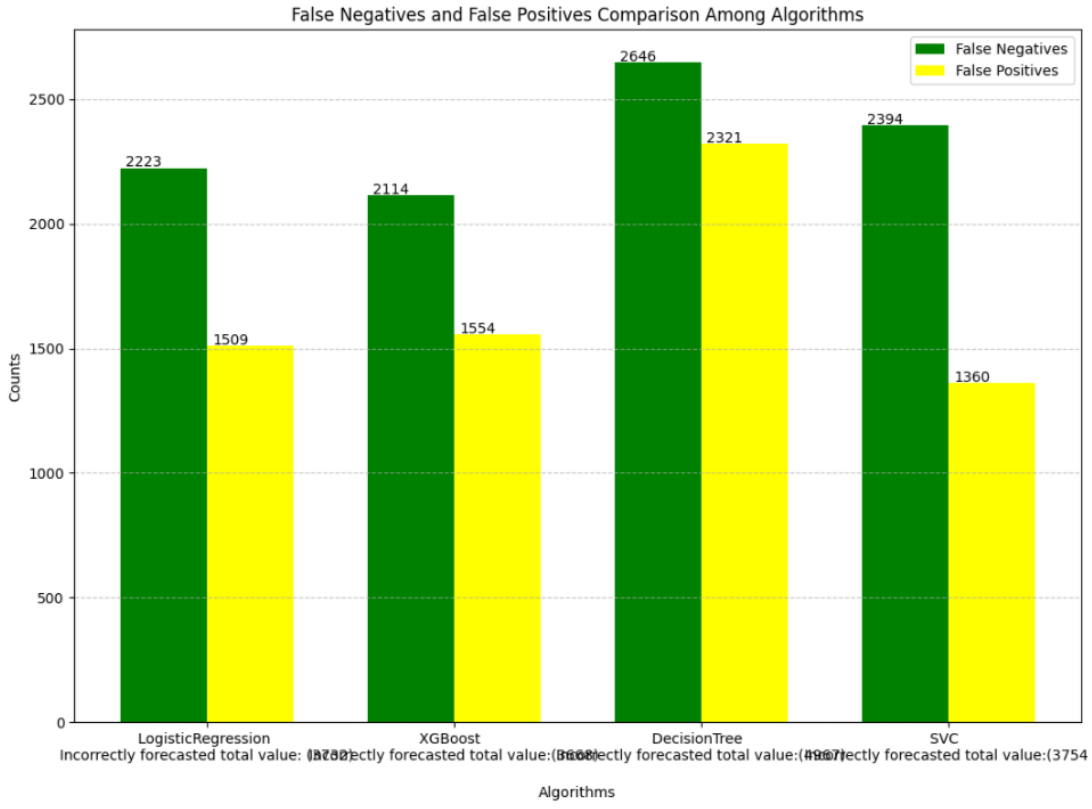


❖ Comparing results from all the models

- Comparing False Negatives and False Positives among the algorithms

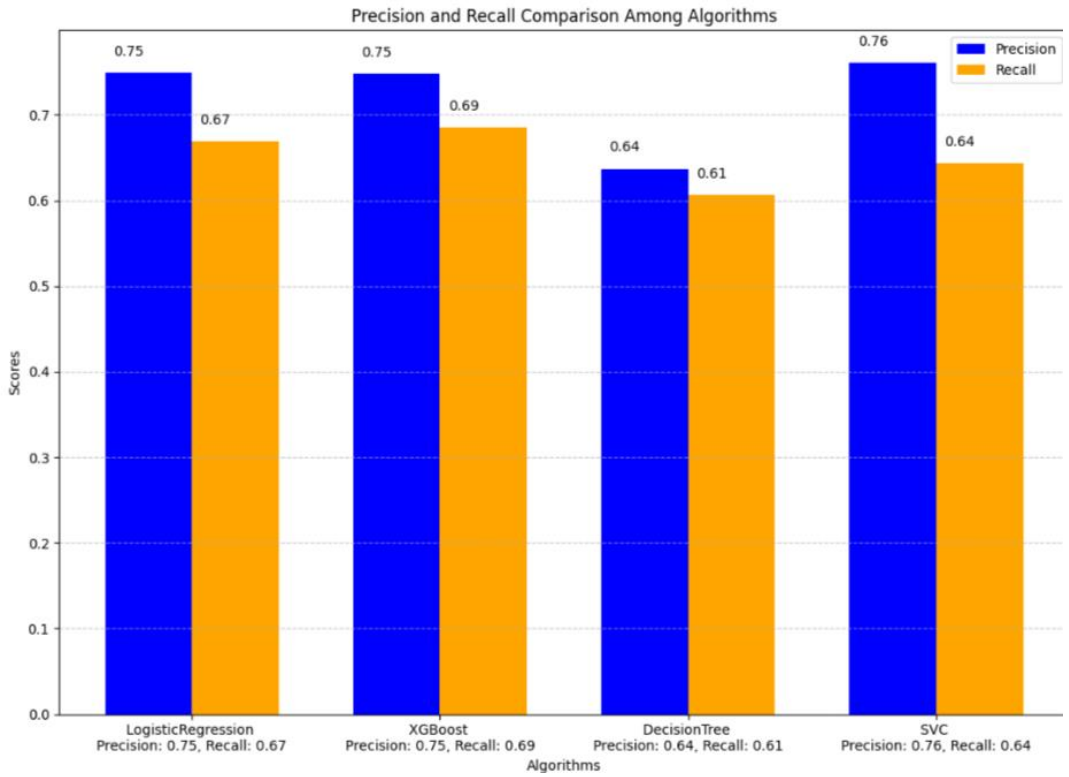
Comparing results form all models

```
tn_lg, fp_lg, fn_lg, tp_lg = cm_LogisticRegression.ravel()
tn_xgb, fp_xgb, fn_xgb, tp_xgb = cm_xgb.ravel()
tn_tree, fp_tree, fn_tree, tp_tree = cm_tree.ravel()
tn_svc, fp_svc, fn_svc, tp_svc = cm_svc.ravel()

fn_values = [fn_lg, fn_xgb, fn_tree, fn_svc]
fp_values = [fp_lg, fp_xgb, fp_tree, fp_svc]

total_values = [fn + fp for fn, fp in zip(fn_values, fp_values)]

model_names = ["LogisticRegression", "XGBoost", "DecisionTree","SVC"]

x = np.arange(len(model_names))
custom_labels = [f"LogisticRegression \nIncorrectly forecasted total value: ({total_values[0]})",
                 f"XGBoost  \nIncorrectly forecasted total value:({total_values[1]})\n",
                 f"DecisionTree  \nIncorrectly forecasted total value:({total_values[2]})\n",
                 f"SVC  \nIncorrectly forecasted total value:({total_values[3]})"]


fn_color = 'green'
fp_color = 'yellow'

bar_width = 0.35
fig, ax = plt.subplots()
bar1 = ax.bar(x - bar_width / 2, fn_values, width=bar_width, color=fn_color, label='False Negatives')
bar2 = ax.bar(x + bar_width / 2, fp_values, width=bar_width, color=fp_color, label='False Positives')

ax.set_xticks(x)
ax.set_xticklabels(custom_labels)
```



- Comparing Precision and Recall among Algorithms

```python
import matplotlib.pyplot as plt
import numpy as np

# Assuming you have the confusion matrix values for each model
tn_lg, fp_lg, fn_lg, tp_lg = cm_LogisticRegression.ravel()
tn_xgb, fp_xgb, fn_xgb, tp_xgb = cm_xgb.ravel()
tn_tree, fp_tree, fn_tree, tp_tree = cm_tree.ravel()
tn_svc, fp_svc, fn_svc, tp_svc = cm_svc.ravel()

# Calculate precision and recall for each model
precision_lg = tp_lg / (tp_lg + fp_lg) if (tp_lg + fp_lg) > 0 else 0
recall_lg = tp_lg / (tp_lg + fn_lg) if (tp_lg + fn_lg) > 0 else 0

precision_xgb = tp_xgb / (tp_xgb + fp_xgb) if (tp_xgb + fp_xgb) > 0 else 0
recall_xgb = tp_xgb / (tp_xgb + fn_xgb) if (tp_xgb + fn_xgb) > 0 else 0

precision_tree = tp_tree / (tp_tree + fp_tree) if (tp_tree + fp_tree) > 0 else 0
recall_tree = tp_tree / (tp_tree + fn_tree) if (tp_tree + fn_tree) > 0 else 0

precision_svc = tp_svc / (tp_svc + fp_svc) if (tp_svc + fp_svc) > 0 else 0
recall_svc = tp_svc / (tp_svc + fn_svc) if (tp_svc + fn_svc) > 0 else 0

# Assuming you have lists to store these precision and recall values
precision_scores = [precision_lg, precision_xgb, precision_tree, precision_svc]
recall_scores = [recall_lg, recall_xgb, recall_tree, recall_svc]


model_names = ["LogisticRegression", "XGBoost", "DecisionTree", "SVC"]

x = np.arange(len(model_names))
custom_labels = [f"{model} \nPrecision: {precision:.2f}, Recall: {recall:.2f}" for model, precision, recall in zip(model_names, precision_scores, recall_scores)]

precision_color = 'blue'
recall_color = 'orange'

bar_width = 0.35
fig, ax = plt.subplots()
bar1 = ax.bar(x - bar_width / 2, precision_scores, width=bar_width, color=precision_color, label='Precision')
bar2 = ax.bar(x + bar_width / 2, recall_scores, width=bar_width, color=recall_color, label='Recall')

ax.set_xticks(x)
ax.set_xticklabels(custom_labels)
```



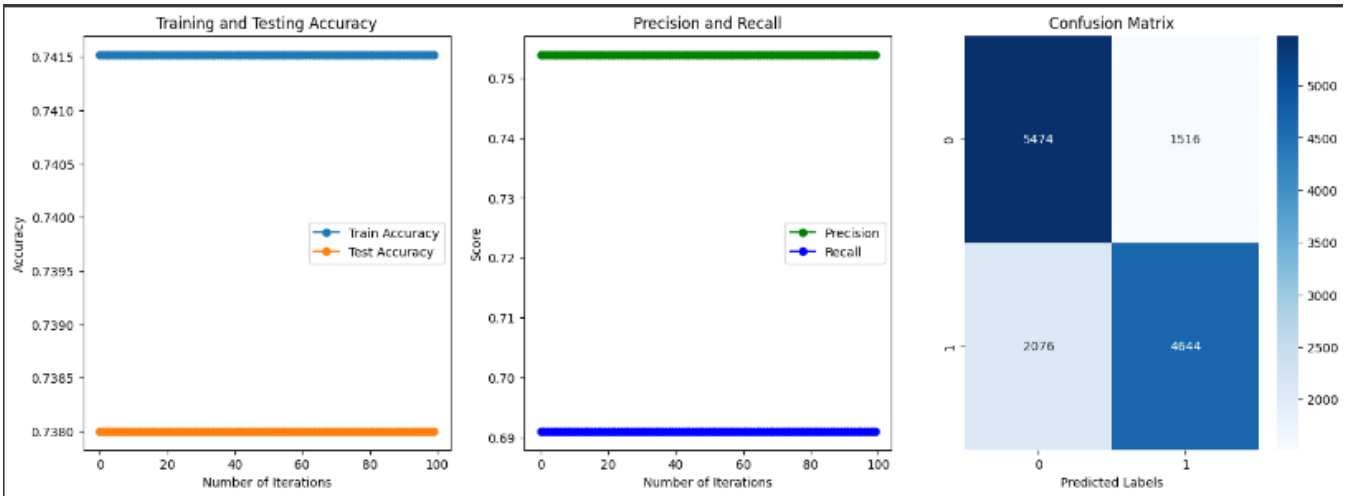
❖ Hyper Parameter Tunning for XGBoost Model

- After comparing the models, we selected XGBoost model to continue with our deployment.
- So, we tuned the hyper parameters to increase the test accuracy, precision and recall of the model and retrained the model.

```
# Split the data
X = heart_data.drop('cardio', axis=1)
y = heart_data['cardio']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Hyperparameter Tuning using GridSearchCV
param_grid = {
    'learning_rate': [0.1, 0.01],
    'max_depth': [3, 5, 7],
    'min_child_weight': [1, 3, 5],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0]
}

model = xgb.XGBClassifier()
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, scoring='accuracy', cv=3, n_jobs=-1)
grid_result = grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_result.best_params_
print("Best Parameters:", best_params)

# Model Training and Metrics Collection
model2 = xgb.XGBClassifier(**best_params)  # Using the best parameters from GridSearchCV
```

```
# Lists to store training and testing metrics
train_accuracy = []
test_accuracy = []
precision_scores = []
recall_scores = []

# Train the model and collect metrics
for i in range(100):  # You can adjust the number of iterations
    model2.fit(X_train, y_train)

    # Training accuracy
    X_train_prediction = model2.predict(X_train)
    training_data_accuracy = accuracy_score(X_train_prediction, y_train)
    train_accuracy.append(training_data_accuracy)

    # Testing accuracy
    y_pred = model2.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    test_accuracy.append(accuracy)

    # Precision and Recall
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    precision_scores.append(precision)
    recall_scores.append(recall)

print("Accuracy on Testing data:", accuracy)
print("Precision:", precision)
print("Recall : ", recall)
```

```
Best Parameters: {'colsample_bytree': 0.9, 'learning_rate': 0.1, 'max_depth': 5, 'min_child_weight': 1, 'subsample': 0.8}
Accuracy on Testing data: 0.738001458789205
Precision: 0.7538961038961038
Recall :  0.6910714285714286
```



5. Model Deployment

- Saved the selected trained model.

```
[69] import pickle

     filename = 'trained_model.sav'

     #writing the model in binary format
     pickle.dump(model2, open(filename, 'wb'))
```

```
[70] #loading the saved model
     loaded_model = pickle.load(open('trained_model.sav', 'rb')) #reading the binary format
```

- Downloaded the trained model file.
- Created a new file in Spyder IDE (Integrated Development Environment) to develop the ‘heart disease prediction web app.’
- Loaded the trained model into the new file.
- Built a predictive system.

```python
#Creating a function for prediction
def heart_prediction(modified_input_data):


    # Unpack the tuple into individual variables
    age_in_days, weight, ap_hi, ap_lo, cholesterol, pp, bmi, health_risk_score  = modified_input_data

    #Change the input data to a numpy array
    input_data_as_numpy_array = np.asarray(modified_input_data)

    # Reshape the numpy array as we are predicting for only on instance
    input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

    #Select the model
    selected_model = loaded_model

    prediction = selected_model.predict(input_data_reshaped)
    print(prediction)

    if (prediction[0]==0):
      return'The Person does not have a Heart Disease'
    else:
      return'The Person has a Heart Disease'
```

- Developed the front-end using Stream lit.

```python
def main():

    # Giving a title
    st.title("Heart Disease Prediction Web App")

    # Getting the input data from the user

    #for dropdowns

    # Map numerical values to labels
    gender_mapping = {1: 'Male', 2: 'Female'}
    cholestrol_mapping = {1: 'normal', 2: 'above normal', 3: 'well above normal'}
    gluc_mapping = {1: 'normal', 2: 'above normal', 3: 'well above normal'}
    smoke_mapping = {1:'smoker', 0:'non-smoker'}
    alco_mapping = {1:'yes', 0:'no'}
    active_mapping = {1:'active', 0:'inactive'}

    # Create a list of numerical values from your dataset
    gender_values = [1, 2]
    cholestrol_values = [1,2,3]
    gluc_values = [1,2,3]
    smoke_values = [0,1]
    alco_values = [0,1]
    active_values = [0,1]

    # Use a list comprehension to create a list of corresponding labels
    gender_options = [gender_mapping[value] for value in gender_values]
    cholestrol_options = [cholestrol_mapping[value] for value in cholestrol_values]
    gluc_options = [gluc_mapping[value] for value in gluc_values]
    smoke_options = [smoke_mapping[value] for value in smoke_values]
    alco_options = [alco_mapping[value] for value in alco_values]
    active_options = [active_mapping[value] for value in active_values]
```

- Used drop downs and text fields to get user input.

```python
#age
age_str = st.text_input('Age in years')
age = int(age_str) if age_str else 0  # Default to 0 if input is empty


#gender
# Use st.selectbox to create a dropdown menu for gender
selected_gender_label = st.selectbox('Select Gender', gender_options)
# Reverse map the selected label to the numerical value
gender = {label: value for value, label in gender_mapping.items()}[selected_gender_label]
# Display the selected gender (numerical value)
st.write(f'You selected gender: {selected_gender_label} (numerical value: {gender})')

#height
height_str = st.text_input('Height measured in centimeters ')
height = int(height_str) if height_str else 0  # Default to 0 if input is empty

#weight
weight_str = st.text_input('Weight measured in kilograms ')
weight = int(weight_str) if weight_str else 0  # Default to 0 if input is empty

#ap_hi
ap_hi_str = st.text_input('Systolic blood pressure ')
ap_hi = int(ap_hi_str) if ap_hi_str else 0  # Default to 0 if input is empty

#ap_lo
ap_lo_str = st.text_input('Diastolic blood pressure ')
ap_lo = int(ap_lo_str) if ap_lo_str else 0  # Default to 0 if input is empty

#Cholestrol
selected_cholestrol_label = st.selectbox('Select cholestrol Level', cholestrol_options)
cholestrol = {label: value for value, label in cholestrol_mapping.items()}[selected_cholestrol_label]
st.write(f'selected cholestrol level: {selected_cholestrol_label} (numerical value: {cholestrol})')

#Glucose
selected_gluc_label = st.selectbox('Select glucose Level', gluc_options)
gluc = {label: value for value, label in gluc_mapping.items()}[selected_gluc_label]
st.write(f'selected gluc level: {selected_gluc_label} (numerical value: {gluc})')

#smoke
selected_smoke_label = st.selectbox('Do you smoke? ', smoke_options)
smoke = {label: value for value, label in smoke_mapping.items()}[selected_smoke_label]
st.write(f'Smokes or not: {selected_smoke_label} (numerical value: {smoke})')

#alco
selected_alco_label = st.selectbox('Do you consume Alcohol? ', alco_options)
alco = {label: value for value, label in alco_mapping.items()}[selected_alco_label]
st.write(f'Drinks alcohol or not: {selected_alco_label} (numerical value: {alco})')

#active
selected_active_label = st.selectbox('Are you physically active? ', active_options)
active = {label: value for value, label in active_mapping.items()}[selected_active_label]
st.write(f'Physically active or not: {selected_active_label} (numerical value: {active})')
```

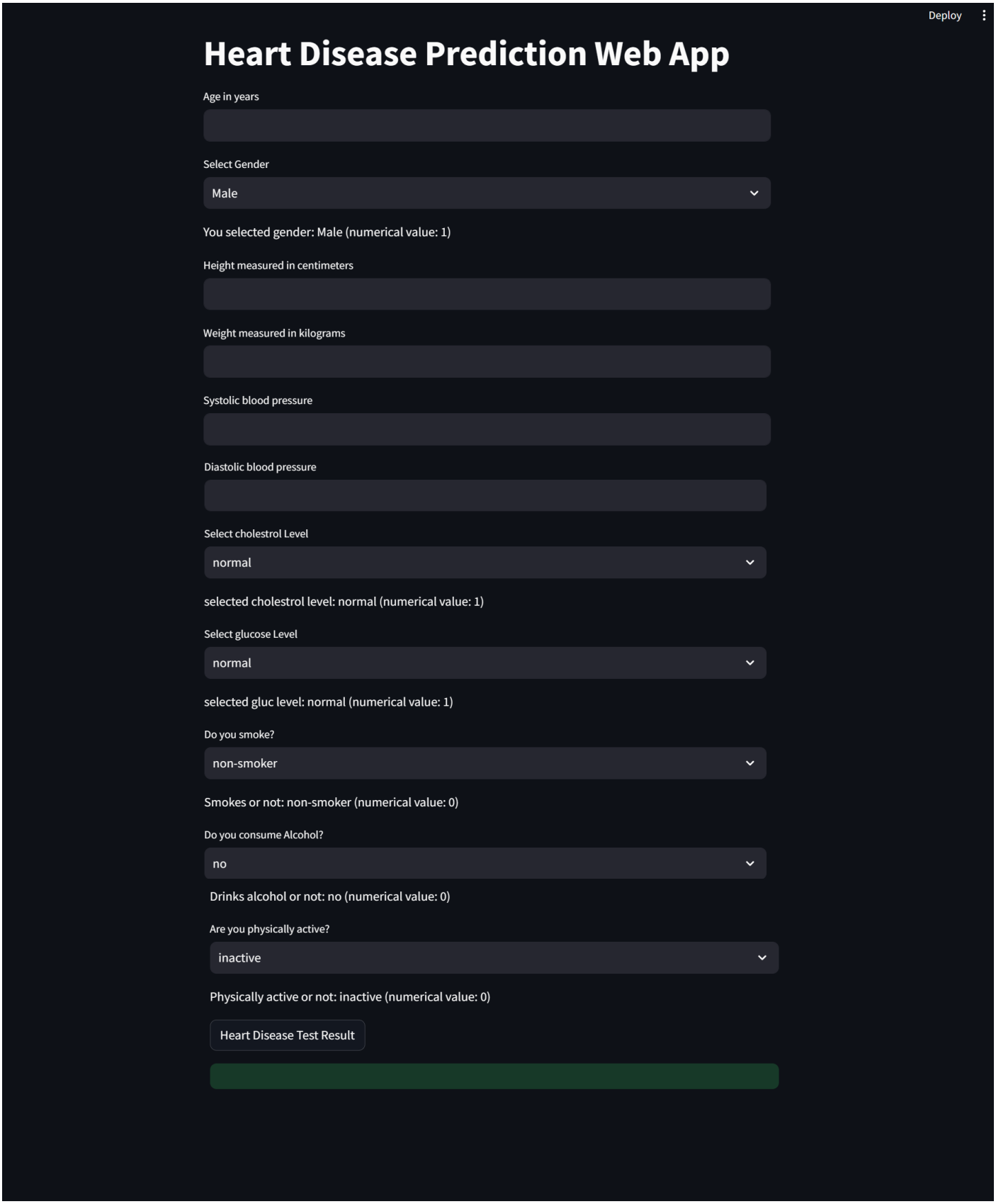
- Took the user input and converted it, as necessary.
- Sent the modified input data as an argument to the heart_prediciton function to make the prediction and display it in the web view.

```
    # Code for prediction

    #return variable will be stored here
    diagnosis = ''


    #create a button
    if st.button('Heart Disease Test Result'):
        # Pass the input data to the function as a tuple
        input_data = (
            age,
            gender,
            height,
            weight,
            ap_hi,
            ap_lo,
            cholestrol,
            gluc,
            smoke,
            alco,
            active
        )

        # Unpack the tuple into individual variables
        age, gender, height, weight, ap_hi, ap_lo, cholesterol, gluc, smoke, alco, active = input_data

        # Convert age from years to days
        age_in_days = age * 365  # Assuming an average year has 365 days

        # BMI and pulse pressure
        pp = ap_hi - ap_lo
        bmi = weight / ((height / 100) * (height / 100))

        # Calculating health risk score
        weights = {
            'Chol': 3,
            'Smoke': 3,
            'Gluc': 2,
            'Alco': 1
        }

        health_risk_score = cholesterol * weights['Chol'] + gluc * weights['Gluc'] + smoke * weights['Smoke'] + alco * weights['Alco']

        # Create modified input data as a tuple
        modified_input_data = (age_in_days, weight, ap_hi, ap_lo, cholesterol, pp, bmi, health_risk_score)

        # call the function
        diagnosis = heart_prediction(modified_input_data)


    st.success(diagnosis)
```

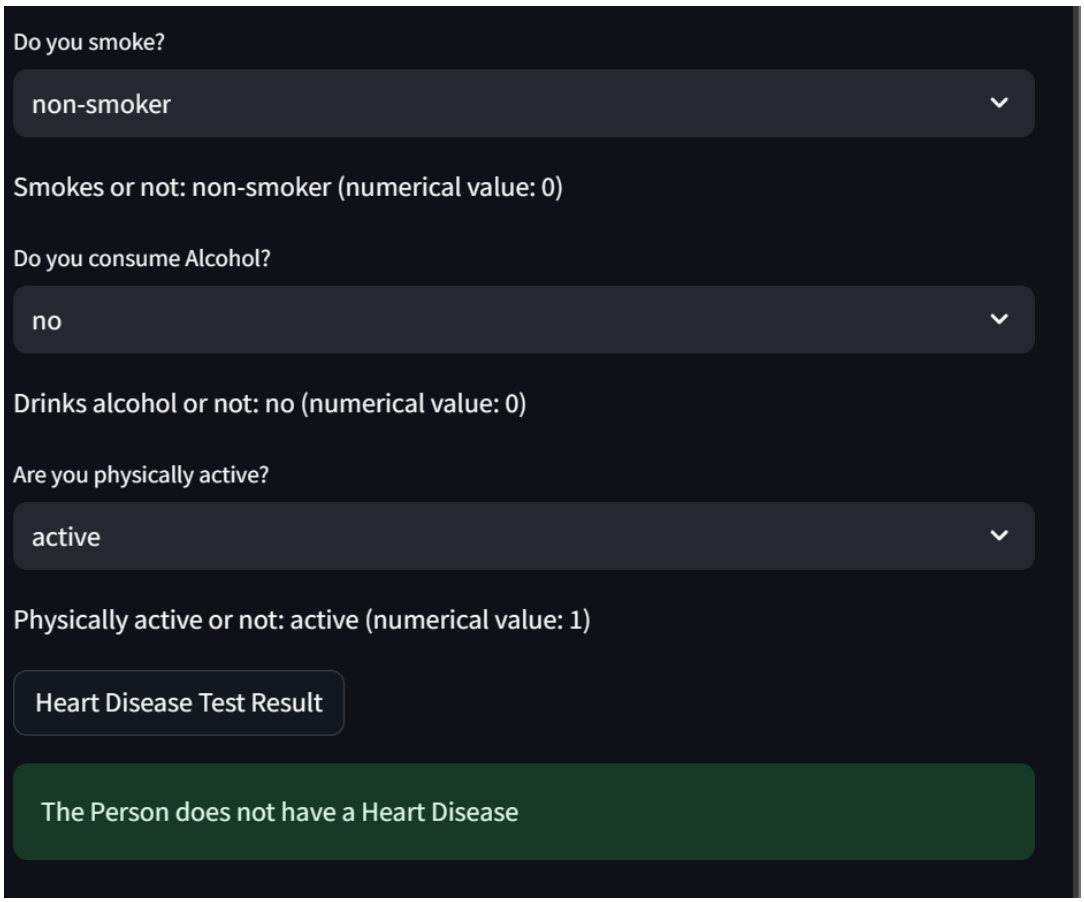
- Display the SHAP summary plot for each user.

```
    # Display the plot here
    if diagnosis:  # Display plot only if there is a diagnosis

        # Create a DataFrame for the user's data
        user_df = pd.DataFrame([modified_input_data], index=['Value'], columns=['age', 'weight', 'ap_hi', 'ap_lo', 'cholesterol', 'pp','bmi','health_risk_score'])

        # Initialize an explainer with the loaded model
        explainer = shap.Explainer(loaded_model)

        # Calculate SHAP values for the user's data
        shap_values = explainer.shap_values(user_df)

        # Plot the summary plot
        fig, ax = plt.subplots()
        shap.summary_plot(shap_values, user_df, plot_type="bar", show=False)
        plt.title("Feature Importance for Heart Disease Prediction")
        st.pyplot(fig, clear_figure=True)  # Display the plot in Streamlit app



#Run the file using a command prompt or as a standalone file
if __name__ == '__main__':
    main()
```

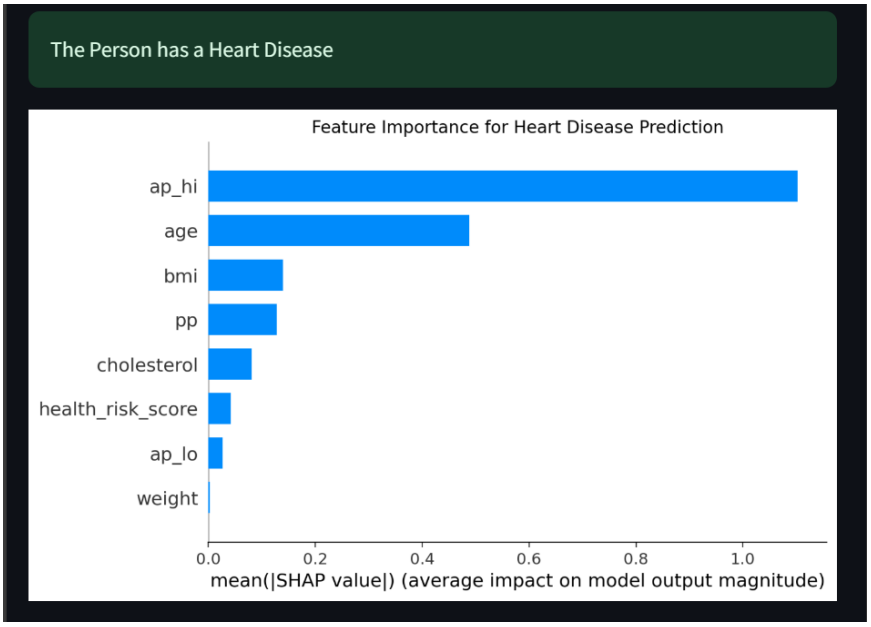
❖ Web View


Deploy
Heart Disease Prediction Web App
Age in years
Select Gender
Male
You selected gender: Male (numerical value: 1)
Height measured in centimeters
Weight measured in kilograms
Systolic blood pressure
Diastolic blood pressure
Select cholestrol Level
normal
selected cholestrol level: normal (numerical value: 1)
Select glucose Level
normal
selected gluc level: normal (numerical value: 1)
Do you smoke?
non-smoker
Smokes or not: non-smoker (numerical value: 0)
Do you consume Alcohol?
no
Drinks alcohol or not: no (numerical value: 0)
Are you physically active?
inactive
Physically active or not: inactive (numerical value: 0)
Heart Disease Test Result

- Prediction



- The SHAP summary plot displays the impact of different features on the prediction of heart disease.



Conclusion

Through the analysis of a diverse and complex data set, we have gained valuable insights into the factors influencing the risk of CVD. Our project revealed the interplay of several factors, including age, gender, blood pressure, cholesterol levels, lifestyle choices, and medical history.

We have successfully developed and fine-tuned predictive models that accurately estimate an individual's risk of developing CVD. These models can be a powerful tool for both healthcare professionals and individuals who want to proactively manage their cardiovascular health.

We have successfully selected the best model (XGBoost) and deployed it in Stream lit in a user-friendly manner so that users can check whether they have heart disease or not by easily entering their details.

Addressing the global burden of CVD, our data mining project highlights the power of advanced analytics and machine learning to improve public health. By providing accurate risk predictions and insights into factors contributing to CVD, we offer a promising path to reducing the prevalence of these diseases and improving overall health and longevity. In the future, we hope to see our work translated into practical applications that empower individuals to take control of their cardiovascular health, help healthcare professionals provide more targeted care, and contribute to a world with fewer lives affected by CVD. The journey does not end here; continues its ongoing research, collaboration, and shared commitment to a healthier future.

Appendix

Demonstration video: Dimonstration_vedio_link

GitHub Link: https://github.com/it21924750/FDM_mini_project_Group_15.git

Link to the dataset: Risk Factors for Cardiovascular Heart Disease (kaggle.com)

References

[1] "Cardiovascular diseases (CVDs)," 11 June 2021. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds).

[2] "Prediction models for cardiovascular disease risk in the general population: systematic review," 16 May 2016. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4868251/.

[3] [Online]. Available: Risk Factors for Cardiovascular Heart Disease (kaggle.com).