```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
import seaborn as sns
```

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

```
df.head(5) # limited number of rows
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDiseas
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	

```
#LE: Sex, FastingBS, ExerciseAngina
#Categorical: ChestPainType, RestingECG, ST_Slope
#Integer: Age, RestingBP, Cholestrol, MaxHR, Oldpeak
```

```
df.shape #To find the shape(row,columns) 4240 instances and 16 features
```

```
(918, 12)
```

```
df.size #Total number of cells
#total number of cells
```

```
11016
```

```
df.info() #find the values, shows features
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    object
 2   ChestPainType   918 non-null    object
 3   RestingBP       918 non-null    int64
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    object
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    object
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object
 11  HeartDisease    918 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

```
df.isnull()#null values
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	Exerci
0	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	
913	False	False	False	False	False	False	False	False	
914	False	False	False	False	False	False	False	False	
915	False	False	False	False	False	False	False	False	
916	False	False	False	False	False	False	False	False	
917	False	False	False	False	False	False	False	False	

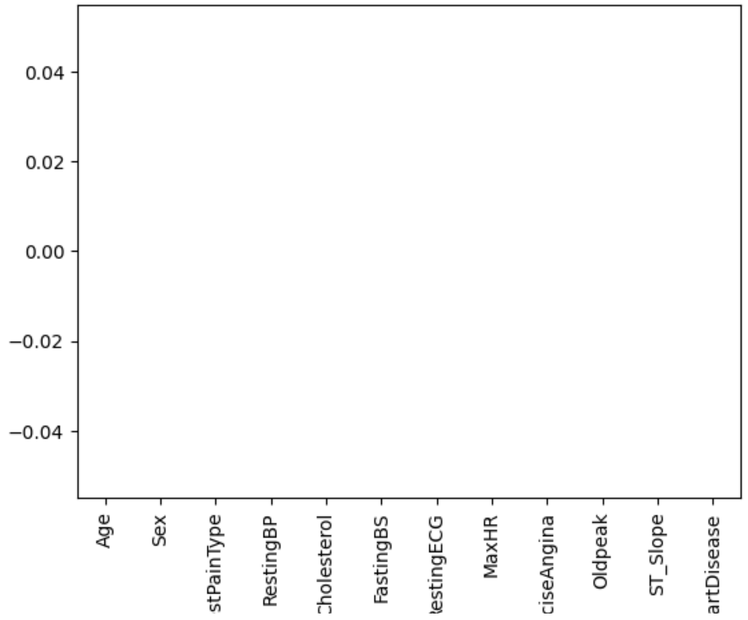
918 rows × 12 columns

```
df.isnull().sum()
```

```
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

A bar plot of the number of null values in the dataset

```
df.isnull().sum().plot.bar()
plt.show()
```



Double-click (or enter) to edit

To show the columns or features of the dataset

```
df.columns
```

```
    Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
           'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
           'HeartDisease'],
          dtype='object')
```

To display unique values before Encoding

```
for i in df:
  print(i,np.unique(df[i]))
```

```
    Age [28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51
     52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
     76 77]
    Sex ['F' 'M']
    ChestPainType ['ASY' 'ATA' 'NAP' 'TA']
    RestingBP [  0  80  92  94  95  96  98 100 101 102 104 105 106 108 110 112 113 114
     115 116 117 118 120 122 123 124 125 126 127 128 129 130 131 132 133 134
     135 136 137 138 139 140 141 142 143 144 145 146 148 150 152 154 155 156
     158 160 164 165 170 172 174 178 180 185 190 192 200]
    Cholesterol [  0  85 100 110 113 117 123 126 129 131 132 139 141 142 147 149 152 153
     156 157 159 160 161 163 164 165 166 167 168 169 170 171 172 173 174 175
     176 177 178 179 180 181 182 183 184 185 186 187 188 190 192 193 194 195
     196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213
```

```
 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231
 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249
 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267
 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285
 286 287 288 289 290 291 292 293 294 295 297 298 299 300 302 303 304 305
 306 307 308 309 310 311 312 313 315 316 318 319 320 321 322 325 326 327
 328 329 330 331 333 335 336 337 338 339 340 341 342 344 347 349 353 354
 355 358 360 365 369 384 385 388 392 393 394 404 407 409 412 417 458 466
 468 491 518 529 564 603]
FastingBS [0 1]
RestingECG ['LVH' 'Normal' 'ST']
MaxHR [ 60  63  67  69  70  71  72  73  77  78  80  82  83  84  86  87  88  90
  91  92  93  94  95  96  97  98  99 100 102 103 104 105 106 107 108 109
 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145
 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163
 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181
 182 184 185 186 187 188 190 192 194 195 202]
ExerciseAngina ['N' 'Y']
Oldpeak [-2.6 -2.  -1.5 -1.1 -1.  -0.9 -0.8 -0.7 -0.5 -0.1  0.   0.1  0.2  0.3
  0.4  0.5  0.6  0.7  0.8  0.9  1.   1.1  1.2  1.3  1.4  1.5  1.6  1.7
  1.8  1.9  2.   2.1  2.2  2.3  2.4  2.5  2.6  2.8  2.9  3.   3.1  3.2
  3.4  3.5  3.6  3.7  3.8  4.   4.2  4.4  5.   5.6  6.2]
ST_Slope ['Down' 'Flat' 'Up']
HeartDisease [0 1]
```

Label Encoding all the categorical values

We decided to go with Label Encoding as the the number of categories in each feature with categorical values were minimal. If there were many we would have chosen One-Hot encoding.

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
```

```
x=list(df['Sex'].unique())
```

```
le.fit(x)
```

```
▾ LabelEncoder
LabelEncoder()
```

```
df['Sex']=le.transform(df.Sex)
```

```
x=list(df['ChestPainType'].unique())
le.fit(x)
df['ChestPainType']=le.transform(df.ChestPainType)
```

```
x=list(df['RestingECG'].unique())
le.fit(x)
df['RestingECG']=le.transform(df.RestingECG)
```

```
x=list(df['ExerciseAngina'].unique())
le.fit(x)
df['ExerciseAngina']=le.transform(df.ExerciseAngina)
```

```
x=list(df['ST_Slope'].unique())
le.fit(x)
df['ST_Slope']=le.transform(df.ST_Slope)
```

All the above are steps to label encode

Final encoded dataset

```
df
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseA
0	40	1	1	140	289	0	1	172	
1	49	0	2	160	180	0	1	156	
2	37	1	1	130	283	0	2	98	
3	48	0	0	138	214	0	1	108	
4	54	1	2	150	195	0	1	122	
...	...	...	...	...	...	...	...	...	
913	45	1	3	110	264	0	1	132	
914	68	1	0	144	193	1	1	141	
915	57	1	0	130	131	0	1	115	
916	57	0	1	130	236	0	0	174	
917	38	1	2	138	175	0	1	173	

918 rows × 12 columns

To display unique values after encoding

```
for i in df:
  print(i,np.unique(df[i]))#CHECKS UNIQUE VALUES
```

```
Age [28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51
 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
 76 77]
Sex [0 1]
ChestPainType [0 1 2 3]
RestingBP [  0  80  92  94  95  96  98 100 101 102 104 105 106 108 110 112 113 114
 115 116 117 118 120 122 123 124 125 126 127 128 129 130 131 132 133 134
 135 136 137 138 139 140 141 142 143 144 145 146 148 150 152 154 155 156
 158 160 164 165 170 172 174 178 180 185 190 192 200]
Cholesterol [  0  85 100 110 113 117 123 126 129 131 132 139 141 142 147 149 152 153
```

```
  156 157 159 160 161 163 164 165 166 167 168 169 170 171 172 173 174 175
  176 177 178 179 180 181 182 183 184 185 186 187 188 190 192 193 194 195
  196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213
  214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231
  232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249
  250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267
  268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285
  286 287 288 289 290 291 292 293 294 295 297 298 299 300 302 303 304 305
  306 307 308 309 310 311 312 313 315 316 318 319 320 321 322 325 326 327
  328 329 330 331 333 335 336 337 338 339 340 341 342 344 347 349 353 354
  355 358 360 365 369 384 385 388 392 393 394 404 407 409 412 417 458 466
  468 491 518 529 564 603]
 FastingBS [0 1]
 RestingECG [0 1 2]
 MaxHR [ 60  63  67  69  70  71  72  73  77  78  80  82  83  84  86  87  88  90
   91  92  93  94  95  96  97  98  99 100 102 103 104 105 106 107 108 109
  110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
  128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145
  146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163
  164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181
  182 184 185 186 187 188 190 192 194 195 202]
 ExerciseAngina [0 1]
 Oldpeak [-2.6 -2.  -1.5 -1.1 -1.  -0.9 -0.8 -0.7 -0.5 -0.1  0.   0.1  0.2  0.3
   0.4  0.5  0.6  0.7  0.8  0.9  1.   1.1  1.2  1.3  1.4  1.5  1.6  1.7
   1.8  1.9  2.   2.1  2.2  2.3  2.4  2.5  2.6  2.8  2.9  3.   3.1  3.2
   3.4  3.5  3.6  3.7  3.8  4.   4.2  4.4  5.   5.6  6.2]
 ST_Slope [0 1 2]
 HeartDisease [0 1]
```

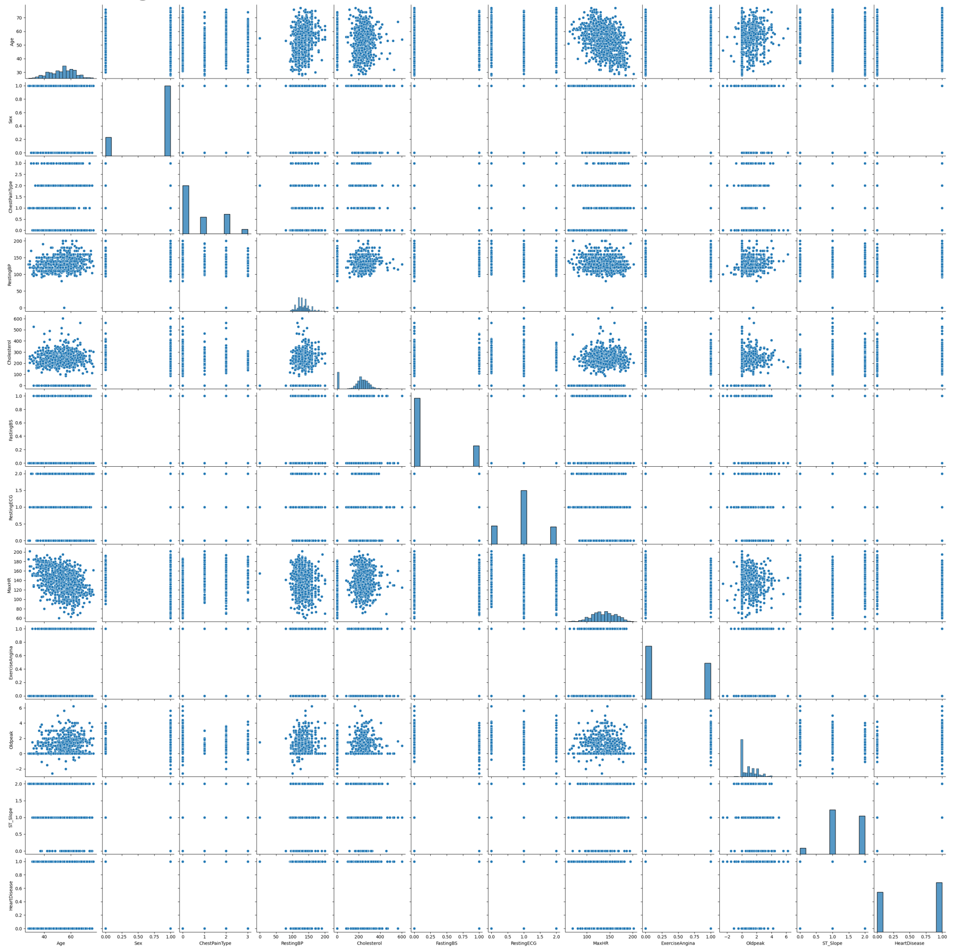
From this pairplot, the following conclusions may be drawn:

1. Age correlates positively with RestingBP and Cholesterol.
2. Age and RestingBP have a negative connection with MaxHR.
3. Age, RestingBP, Cholesterol, MaxHR, and Oldpeak seem to be distributed differently across people with and without heart disease.
4. There is no obvious linear connection between the variables, indicating that a nonlinear model may be required for forecasting.
5. The association between ChestPainType and the other variables in the dataset is unclear.
6. The relationship between Oldpeak and MaxHR is negative.
7. There seems to be a link between resting blood pressure and cholesterol.

Overall, the pairplot may help us uncover any potential correlations between the variables and is a valuable tool for selecting features in machine learning models.

```
sns.pairplot(df, diag_kind="hist")
```

```
<seaborn.axisgrid.PairGrid at 0x7fcec5d12c70>
```

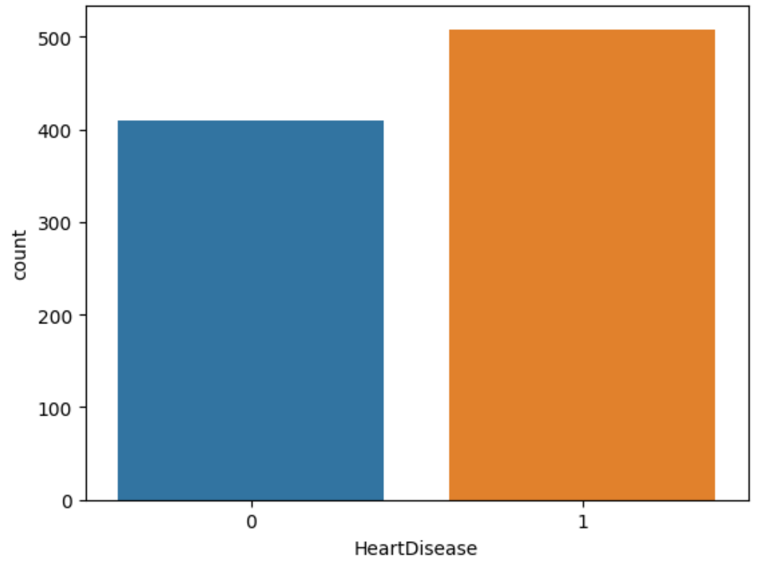

Age
Sex
ChestPainType
RestingBP
Cholesterol
FastingBS
RestingECG
MaxHR
ExerciseAngina
Oldpeak
ST_Slope
HeartDisease

```
df.columns
```

```
Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')
```

```
sns.countplot(x='HeartDisease', data=df)
```

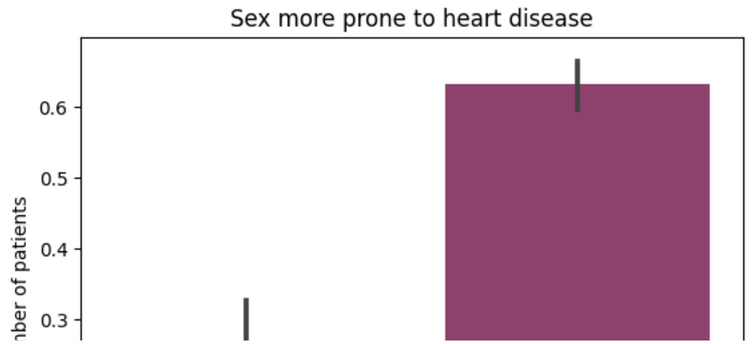
```
<Axes: xlabel='HeartDisease', ylabel='count'>
```



THE ABOVE GRAPH DEPICTS THAT IN OUR DATASET, APPROXIMATELY OVER 500 PEOPLE ARE ACTUALLY PRONE TO HEART DISEASE GIVEN THE FEATURES AND ABOUT 400 PEOPLE ARE NOT

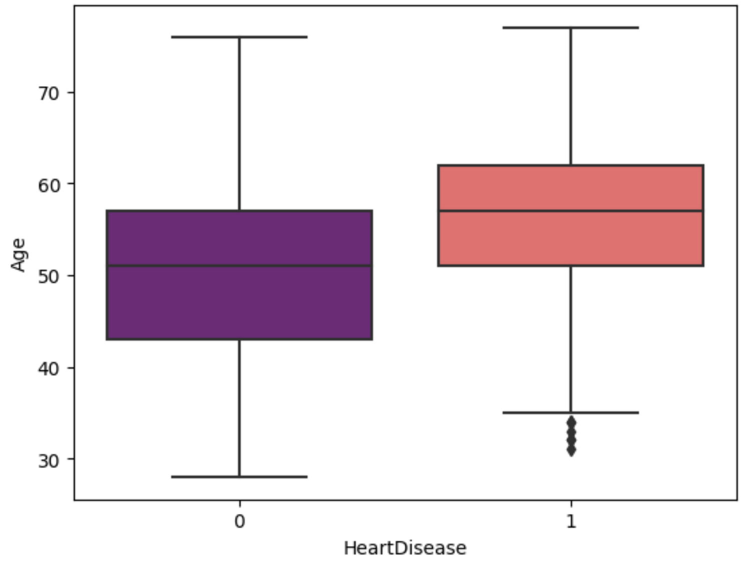
```
sns.barplot(x='Sex', y='HeartDisease', data=df, palette="flare")
plt.title('Sex more prone to heart disease')
plt.xlabel('Sex')
plt.ylabel('Number of patients')
```

```
Text(0, 0.5, 'Number of patients')
```



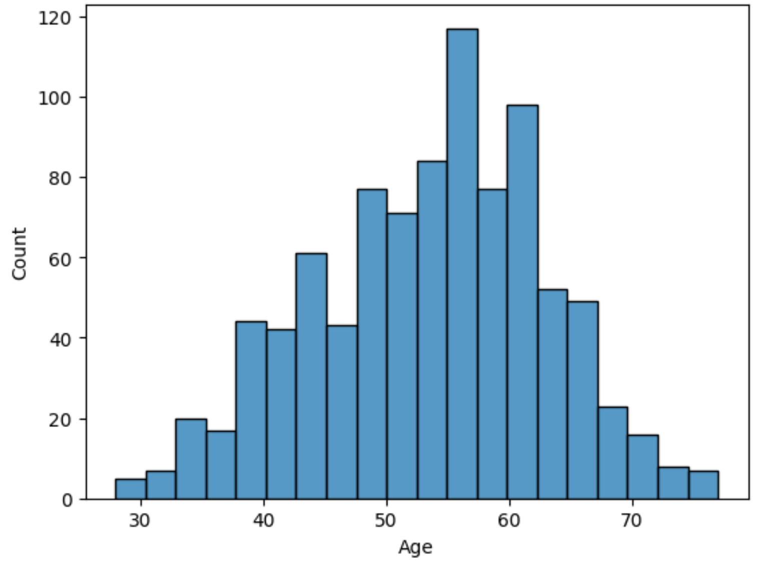
INFERENCE: Given that there are more male patients with heart illness than female patients, it may be concluded from this image that men are more likely than women to have heart disease. The fact that the standard for men with heart disease is greater than the bar for women with heart disease serves as the foundation for this assumption. It's crucial to remember that this conclusion is based on the information in this specific dataset and could not necessarily apply to the whole population.

```
sns.boxplot(data=df, x='HeartDisease', y='Age', palette="magma")
plt.show()
```



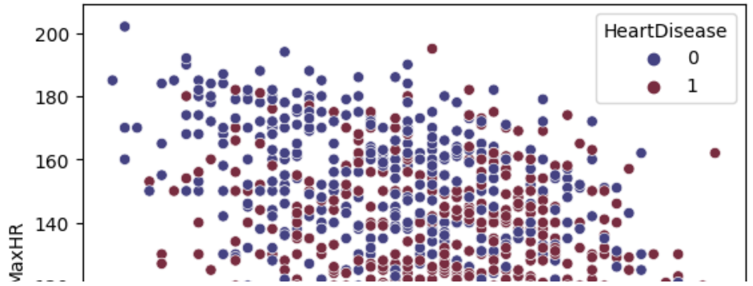
INFERENCE: This boxplot suggests that people with heart disease(1) are often older than those without heart disease. The distribution of ages for patients with heart disease is more skewed towards older ages than the distribution for patients without heart disease, and this inference is based on the facts that the median (represented by the line inside the box) for patients with heart disease is higher than the median for patients without heart disease. It's crucial to remember that this conclusion is based on the information in this specific dataset and could not necessarily apply to the whole population. The box plot may also be used to spot any extreme values or outliers in the dataset. The box for individuals with heart disease contains numerous dots above the bottom border, which signify probable outliers. Understanding the nature of these outliers and their possible influence on the association between ageing and heart disease may need more investigation.

```
sns.histplot(data=df, x='Age', bins=20)
plt.show()
```



INFERENCE: The distribution of ages in the dataset is inferred from this representation to be about normally distributed, with a peak occurring between the ages of 55 and 65. This conclusion is based on the histogram's resemblance to a bell curve. Also, the dataset seems to include some patients who are much older than the bulk of patients based on the histogram's extended tail towards older ages. Another conclusion drawn from this visualisation is that there aren't many patients in the dataset who are under the age of 40. This is supported by the fact that there are few patients in the youngest age bins of the histogram (i.e., ages less than 40). This can be as a result of the lower prevalence of cardiac disease in younger people or a dataset restriction (i.e., it may not include many younger patients).

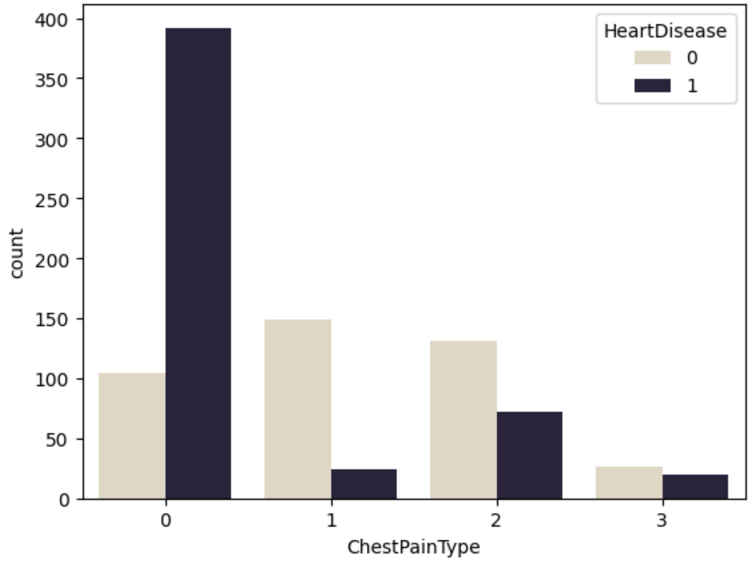
```
sns.scatterplot(data=df, x='Age', y='MaxHR', hue='HeartDisease', palette="icefire")
plt.show()
```



INFERENCE: This graph suggests that individuals with heart illness often have lower maximal heart rates than those without heart disease. This is due to the fact that people with heart disease tend to cluster towards the lower end of the maximum heart rate range, while healthy individuals are more evenly distributed across the maximum heart rate range.

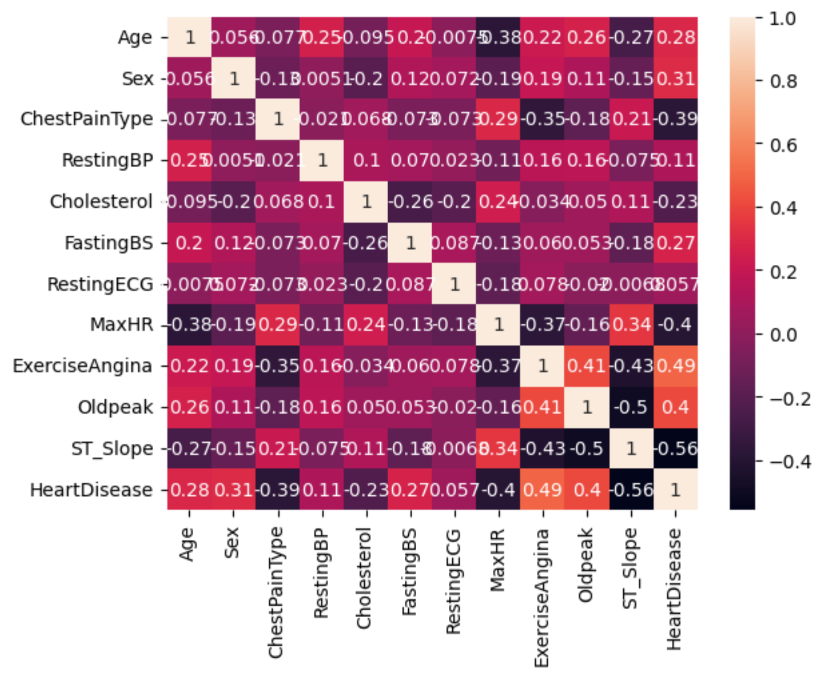
Another conclusion drawn from this visualisation is that, whether or not patients have cardiac disease, there is a universally negative correlation between age and maximal heart rate. This is based on the scatterplot's downward trend from left to right, which reveals that maximal heart rate tends to decline as age rises.

```
sns.countplot(data=df, x='ChestPainType', hue='HeartDisease', palette="ch:s=-.2,r=.6")
plt.show()
```



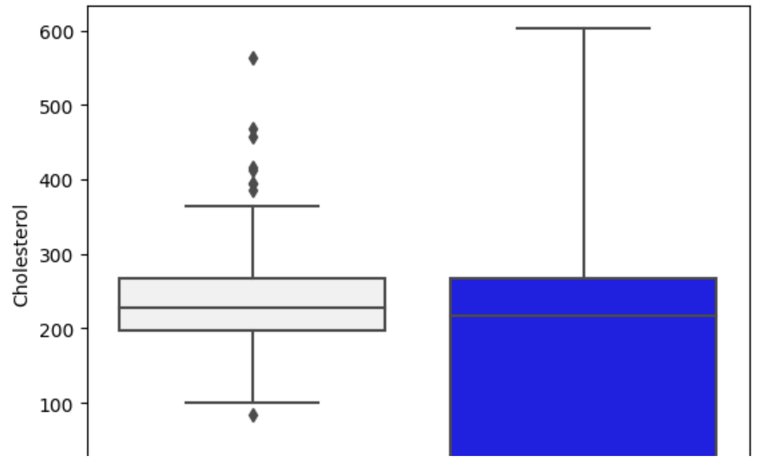
INFERENCE: This grap suggests that people with type 0 and type 1 chest pain are more likely to suffer heart disease than those with type 2 and type 3 chest pain. This is based on the observation that patients with chest pain kinds 0 and 1 are more likely to have heart disease than those without, but patients with chest pain types 2 and 3 are more likely to have heart disease. Another conclusion drawn from this visualisation is that regardless of whether a patient has cardiac disease, type 3 chest pain is the most prevalent form of chest pain reported by patients in the sample. This is based on the fact that the plot's y-axis shows that chest pain type 3 has the largest count.

```
sns.heatmap(df.corr(), annot=True)
plt.show()
```



The heatmap is an effective tool for showing the strength and direction of the linear connection between variable pairs in the dataset. A correlation coefficient that is positive shows a positive association between two variables, while a correlation value that is negative suggests a negative relationship. A correlation value of 0 shows that there is no linear association. Age, resting blood pressure (RestingBP), and cholesterol levels are favourably connected with one another, but age, RestingBP, and cholesterol levels are negatively correlated with maximum heart rate obtained during exercise (MaxHR). This shows that older individuals with greater RestingBP and cholesterol levels may have a higher risk of developing heart disease, but those with a higher maximum heart rate obtained during exercise may have a lower risk of developing heart disease. There is a somewhat significant link between chest pain type and heart disease, which suggests that people with particular forms of chest pain may be more likely to develop heart disease. In addition, there is a modest negative connection between the slope of the ST segment during peak exertion (ST Slope) and heart disease, suggesting that individuals with a more downwardly sloped ST segment during exercise may be less likely to develop heart disease.

```
sns.boxplot(data=df, x='HeartDisease', y='Cholesterol', palette="light:b")
plt.show()
```



INFERENCE:The x-axis indicates the prevalence or absence of cardiac disease, whilst the y-axis indicates cholesterol levels. The box plot reveals that the median cholesterol levels of individuals with heart disease are somewhat higher than those of people without heart disease. In addition, the box plot reveals that there are a few heart disease patients with abnormally high or low cholesterol levels (outliers). Consequently, we may deduce that high cholesterol levels may be a risk factor for heart disease, and additional research is required to determine the nature of the association between cholesterol levels and heart disease.

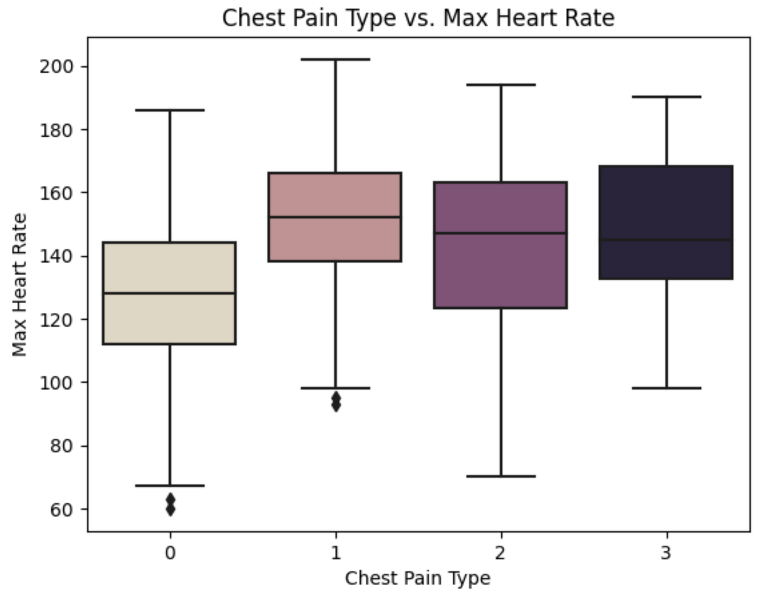
```
sns.scatterplot(data=df, x='Cholesterol', y='RestingBP', palette="vlag")
plt.title('Cholesterol vs. Resting Blood Pressure')
plt.xlabel('Cholesterol')
plt.ylabel('Resting Blood Pressure')
```

```
<ipython-input-95-dfe88af54dbb>:1: UserWarning: Ignoring `palette` because no `hue` variable
  sns.scatterplot(data=df, x='Cholesterol', y='RestingBP', palette="vlag")
Text(0, 0.5, 'Resting Blood Pressure')
```

Inference: There is no apparent linear association between cholesterol levels and resting blood pressure, as seen by the scatter figure. Yet, we can observe that there is a concentration of data points with elevated cholesterol and blood pressure during rest. This implies that increased cholesterol levels may be related with greater resting blood pressure, although other variables also influence resting blood pressure. Consequently, further research is required to comprehend the association between cholesterol levels and resting blood pressure, as well as the function of other possible heart disease risk factors.

```
sns.boxplot(data=df, x='ChestPainType', y='MaxHR', palette="ch:s=-.2,r=.6")
plt.title('Chest Pain Type vs. Max Heart Rate')
plt.xlabel('Chest Pain Type')
plt.ylabel('Max Heart Rate')
```

```
Text(0, 0.5, 'Max Heart Rate')
```

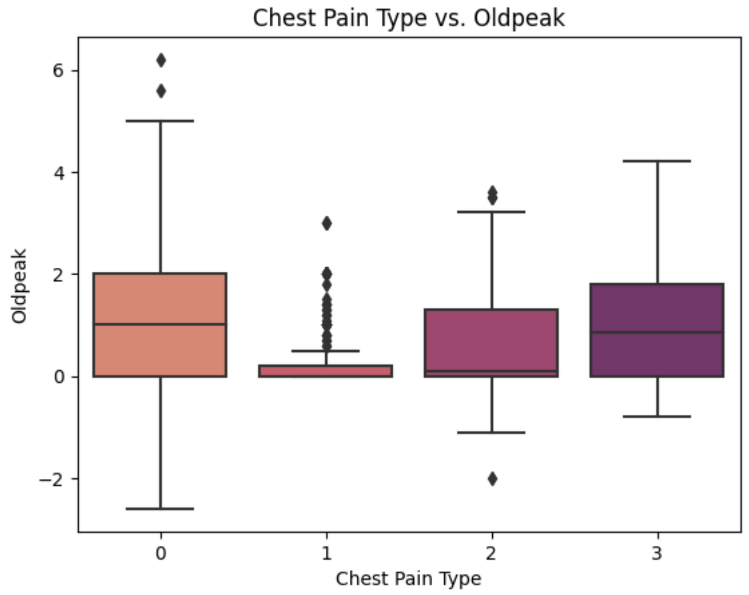


Inference:Patients with chest pain types 0 and 1 tend to have a greater maximal heart rate than those with chest pain types 2 and 3. This shows that the kind of chest discomfort may be associated with the highest heart rate reached during exercise.

Yet, there is a substantial overlap between the various forms of chest pain, suggesting that chest pain type alone may not be a reliable predictor of the highest heart rate obtained during exercise. Consequently, further research is required to comprehend the association between chest pain kind and maximal heart rate obtained during exercise, as well as the function of other possible heart disease risk factors.

```
sns.boxplot(data=df, x='ChestPainType', y='Oldpeak', palette="flare")
plt.title('Chest Pain Type vs. Oldpeak')
plt.xlabel('Chest Pain Type')
plt.ylabel('Oldpeak')
```

```
    Text(0, 0.5, 'Oldpeak')
```



Inference:The boxplot illustrates the association between the variables Chest Pain Type and Oldpeak. Oldpeak indicates the ST depression generated by activity compared to rest and is hence a significant predictor of heart disease. Patients with Chest Pain Type 0 and 1 had a greater median Oldpeak value compared to patients with Chest Pain Types 2 and 3. This indicates that people with Chest Pain Types 0 and 1 may be more susceptible to cardiac disease than those with Chest Pain Types 2 and 3. Yet, further research and statistical tests are need to validate this association.

```
X = df.drop(['HeartDisease'], axis= 1)
y = df['HeartDisease']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size = 0.3,random_state=42)
```

```
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
X_train= st_x.fit_transform(X_train)
X_test= st_x.transform(X_test)
```

Support Vector Machine(SVM)

```
from sklearn.svm import SVC # "Support vector classifier"
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(X_train, y_train)
```

```
▾                 SVC
SVC(kernel='linear', random_state=0)
```

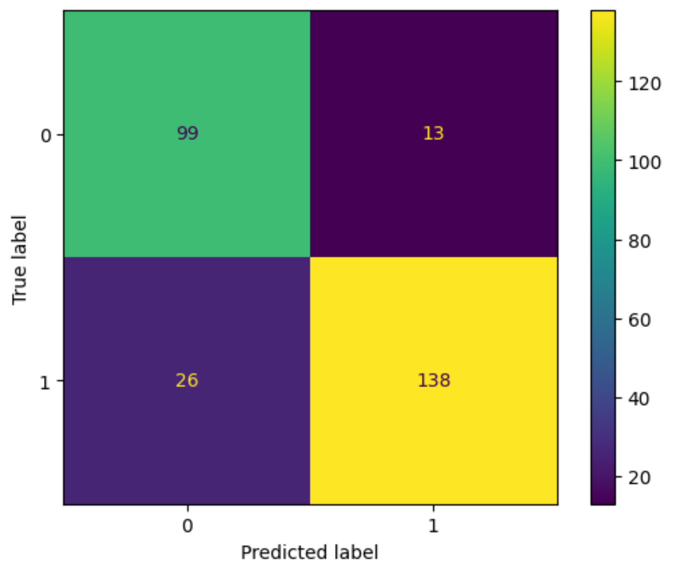
```
y_pred= classifier.predict(X_test)
```

Importing necessary performance metrics

```
from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    ConfusionMatrixDisplay,
    f1_score,
    roc_curve,
    roc_auc_score
)
```

To display the confusion matrix for the model

```
labels = [0,1]
confusion_matrix_svm= confusion_matrix(y_test, y_pred, labels=labels)
disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix_svm, display_labels=labels)
disp.plot();
```



Accuracy of the model

```
accuracy_score_svm= accuracy_score(y_test, y_pred)
accuracy_score_svm
```

```
    0.8586956521739131
```

F1 score of the model

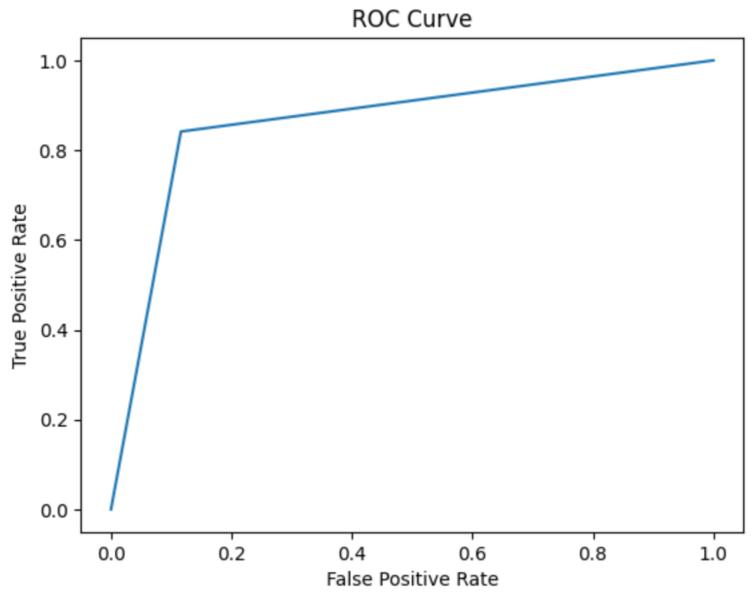
```
f1_svm= f1_score(y_pred, y_test, average="weighted")
f1_svm
```

```
    0.8577360204768019
```

ROC Curve and Area under the ROC Curve

```
fpr_svm, tpr_svm, _ = roc_curve(y_test, y_pred)

#create ROC curve
plt.plot(fpr_svm,tpr_svm)
plt.title("ROC Curve")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



```
score_svm= roc_auc_score(y_test, y_pred)
score_svm
```

```
0.8626959930313588
```

Performing K-Fold cross validation to see if accuracy can be further improved

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

```
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
```

```
results_svm= cross_val_score(classifier, X, y, cv=kfold)
```

```
print(results_svm)
```

```
[0.83695652 0.88043478 0.88043478 0.81420765 0.8579235 ]
```

Random Forest

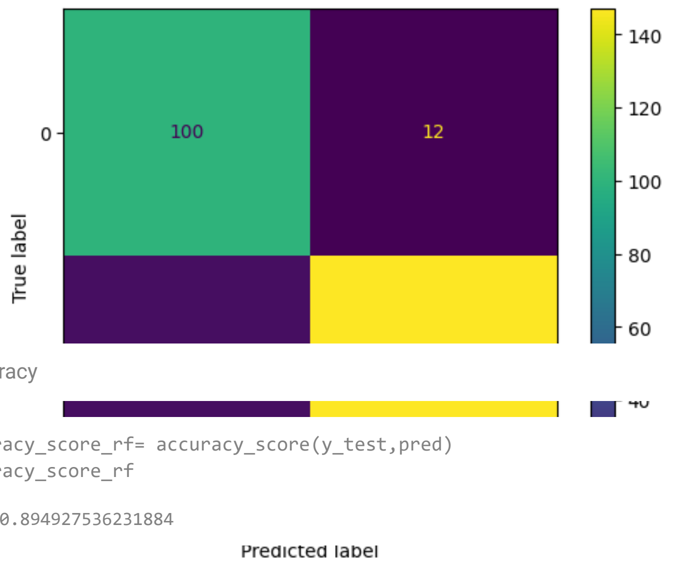
```
from sklearn.ensemble import RandomForestClassifier
# Initializing a Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Fitting the classifier to the training data
clf.fit(X_train, y_train)

# Using the classifier to make predictions on the testing data
pred = clf.predict(X_test)
```

To display the confusion matrix

```
labels = [0,1]
confusion_matrix_rf= confusion_matrix(y_test,pred, labels=labels)
disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix_rf, display_labels=labels)
disp.plot();
```



Accuracy

```
accuracy_score_rf= accuracy_score(y_test,pred)
accuracy_score_rf
```

```
    0.894927536231884
```

F1 score

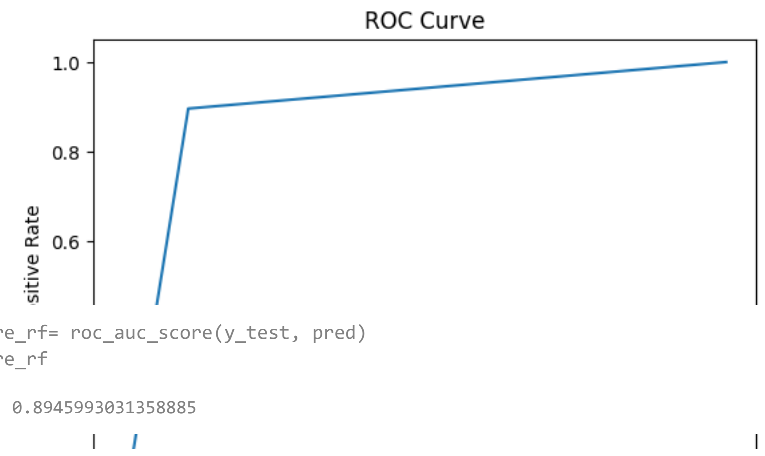
```
f1_rf= f1_score(y_pred, y_test, average="weighted")
f1_rf
```

```
    0.8577360204768019
```

ROC Curve and Area under the ROC curve

```
fpr_rf, tpr_rf, _ = roc_curve(y_test, pred)

#create ROC curve
plt.plot(fpr_rf,tpr_rf)
plt.title("ROC Curve")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



```
score_rf= roc_auc_score(y_test, pred)
score_rf
```

```
    0.8945993031358885
```

INFERENCE:

From the performance metrics applied on both models we can see that Random Forest Algorithm works better.The margin is negligible but our topic necessitates maximum accuracy possible as it predicts if you are at risk of getting a stroke.

Creating functions to predict to facilitate the development of our website.

Without taking inputs

```
def predicted(age, sex, c_pain, rBP, Ch, FBs, RECG, MHR, EA, OP, ST_S):
  output=clf.predict([[age, sex, c_pain, rBP, Ch, FBs, RECG, MHR, EA, OP, ST_S]])
  if (output[0]==0):
    print("No risk of stroke or heart disease")
  else:
    print("Patient is at risk of stroke or heart disease")
```

```
predicted(45, 1, 3, 120, 180, 0, 2, 176, 1, 2.5, 1)
```

```
    Patient is at risk of stroke or heart disease
```

Taking inputs

```
def predicter(clf):
  ip1=int(input("Enter age: "))
  ip2=int(input("Enter sex(0-Female,1-Male): "))
  ip3=int(input("Enter ChestPain type(TA: Typical Angina-3, ATA: Atypical Angina-1, NAP: Non-Anginal Pain-2, ASY: Asymptomatic-0): "))
  ip4=int(input("Enter Resting BP: "))
  ip5=int(input("Enter Cholestrol: "))
  ip6=int(input("Enter Fasting Bs(1: if FastingBS > 120 mg/dl, 0: otherwise): "))
  ip7=int(input("Enter Resting ECG(Normal: Normal-1, ST: having ST-T wave abnormality-2, LVH: showing left ventricular hypertrophy by Estes' criteria-0): "))
  ip8=int(input("Enter Max Heart rate: "))
  ip9=int(input("Enter whether Excercise Angina(No-0,Yes-1): "))
  ip10=float(input("Enter Oldpeak: "))
  ip11=int(input("Enter ST_Slope(Up: upsloping-2, Flat: flat-1, Down: downsloping-0): "))

  output=clf.predict([[ip1,ip2,ip3,ip4,ip5,ip6,ip7,ip8,ip9,ip10,ip11]])
  if output[0]==0:
    print("No risk of stroke or heart disease")
  else:
    print("Patient is at risk of stroke or heart disease")
```

To link our python code to the website

```
from flask import Flask, render_template, request
import pandas as pd
import numpy as np


app = Flask(__name__)

@app.route('/')
def home():
    return render_template('original.html')


@app.route("/predict", methods=['GET','POST'])
def predict():
    if request.method == 'POST':
        age = float(request.form['age'])
        sex = float(request.form['sex'])
        cp = float(request.form['cp'])
        trestbps = float(request.form['trestbps'])
        chol = float(request.form['chol'])
        fbs= float(request.form['fbs'])
        restecg = float(request.form['restecg'])
        thalach = float(request.form['thalach'])
        exang = float(request.form['exang'])
        oldpeak = float(request.form['oldpeak'])
        slope = float(request.form['slope'])

        args = [age,sex,cp,trestbps,chol,fbs,restecg,thalach,exang,oldpeak,slope]

        res = predicted([args])
    return render_template('predict.html', prediction = res)
if __name__ == '__main__':
    app.run(debug=True)
```

```
'app = Flask(__name__)\n\n@app.route(\'/\')\ndef home():\n    return render_template(\'orig
inal.html\')\n\n\n@app.route("/predict", methods=[\'GET\',\'POST\'])\ndef predict():\n    i
f request.method == \'POST\':\n        age = float(request.form[\'age\'])\n        sex = fl
oat(request.form[\'sex\'])\n        cp = float(request.form[\'cp\'])\n        trestbps = fl
oat(request.form[\'trestbps\'])\n        chol = float(request.form[\'chol\'])\n        fbs=
float(request.form[\'fbs\'])\n        restecg = float(request.form[\'restecg\'])\n        t
```