Heart Disease Prediction

- Importing libraries
- Importing the dataset
- Dataset information (Pandas Profiling)
- Spliting the Train & Test datasets
- Feature Scaling and Feature Selection

```
# Random Forest Classifier  Model

1.   Importing the libraries
2.   Testing and training data
3.   Decision Tree
4.   Confusion matrix
5.   Sensitivity and specificity
6.   ROC(Receiver Operator Curve) and accuracy
```

```
# Logistic Regression

1.   Importing the libraries
2.   Training data
3.   Prediction
4.   Confusion matrix
5.   Sensitivity and specificity
6.   ROC, Accuracy and AUC(area under curve)
```

```
# KNN

1.   Importing libraries
2.   Importing the dataset
3.   Feature Selection
4.   Data processing
5.   ROC and score
```

```
# ANN

1.   Building the ANN
2.   Adding the input layer and the first hidden layer
     Adding the second hidden la
```

```
   Adding the output layer
3. Training the ANN on the Training set
4. Making the predictions and evaluating the model
5. Confusion matrix and accuracy
6. Sensitivity and specificity
```

- Testing the model

Importing the libraries

```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns #for plotting
from sklearn.ensemble import RandomForestClassifier #for the model
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz #plot tree
from sklearn.metrics import roc_curve, auc #for model evaluation
from sklearn.metrics import classification_report #for model evaluation
from sklearn.metrics import confusion_matrix #for model evaluation
from sklearn.model_selection import train_test_split #for data splitting
```

```
tf.__version__
```

```
'2.19.0'
```

Random Forest Classifier Algorithm Model

Part 1 - Data Preprocessing

Importing the dataset

```
dataset = pd.read_csv('/content/heart_disease_data.csv')
X = dataset.iloc[:, 3:-1].values
y = dataset.iloc[:, -1].values
```

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
```

```
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

```
dataset.head(10)
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slop
0	63	1	3	145	233	1	0	150	0	2.3	
1	37	1	2	130	250	0	1	187	0	3.5	
2	41	0	1	130	204	0	0	172	0	1.4	
3	56	1	1	120	236	0	1	178	0	0.8	
4	57	0	0	120	354	0	1	163	1	0.6	
5	57	1	0	140	192	0	1	148	0	0.4	
6	56	0	1	140	294	0	0	153	0	1.3	
7	44	1	1	120	263	0	1	173	0	0.0	
8	52	1	2	172	199	1	1	162	0	0.5	
9	57	1	2	150	168	0	1	174	0	1.6	

The above set of data is transparent and self-explanatory. Yet, the interpretation of few of the column headers is not obvious. Here's what they mean,

age: The age of person in years

cp: The chest pain experienced (Value 1: typical angina, Value 2: atypical angina, Value 3: non-anginal pain, Value 4: asymptomatic) trestbps: The person's resting blood pressure (mm Hg on admission to the hospital)

chol: The person's cholesterol measurement in mg/dl

fbs: The person's fasting blood sugar (> 120 mg/dl, 1 = true; 0 = false)

restecg: Resting electrocardiographic measurement (0 = normal, 1 = having ST-T wave abnormality, 2 = showing probable or definite left ventricular hypertrophy by Estes' criteria)

thalach: The person's maximum heart rate achieved

exang: Exercise induced angina (1 = yes; 0 = no)

oldpeak: ST depression induced by exercise relative to rest ('ST' relates to positions on the ECG plot. See more here)

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

ca: The number of major vessels (0-3)

thal: A blood disorder called thalassemia (3 = normal; 6 = fixed defect; 7 = reversable defect)

target: Heart disease (0 = no, 1 = yes)

Diagnosis: The diagnosis of heart disease is done on a combination of clinical signs and test results. The types of tests run will be chosen on the basis of what the physician thinks is going on 1, ranging from electrocardiograms and cardiac computerized tomography (CT) scans, to blood tests and exercise stress tests 2.

Looking at information of heart disease risk factors led me to the following: *high cholesterol, high blood pressure, diabetes, weight, family history and smoking* . According to another source , the major factors that can't be changed are: *increasing age, male gender and heredity. Note that thalassemia, one of the variables in this dataset, is heredity*. Major factors that can be modified are: *Smoking, high cholesterol, high blood pressure, physical inactivity, and being overweight and having diabetes*. Other factors include *stress, alcohol and poor diet/nutrition*.

I can see no reference to the 'number of major vessels', but given that the definition of heart disease is "**...what happens when your heart's blood supply is blocked or interrupted by a build-up of fatty substances in the coronary arteries"**, it seems logical the more major vessels is a good thing, and therefore will reduce the probability of heart disease.

Given the above, I would hypothesis that, if the model has some predictive ability, we'll see these factors standing out as the most important.

Let's change the column names to be a bit clearer,

```
dataset.columns = ['age', 'sex', 'chest_pain_type', 'resting_blood_press
       'exercise_induced_angina', 'st_depression', 'st_slope', 'num_majo
```

Now, I am revising the values of categorical variables, for clearer and easy understability

```
dataset['sex'][dataset['sex'] == 0] = 'female'
dataset['sex'][dataset['sex'] == 1] = 'male'

dataset['chest_pain_type'][dataset['chest_pain_type'] == 0] = 'typical a
dataset['chest_pain_type'][dataset['chest_pain_type'] == 1] = 'atypical
```

```
dataset['chest_pain_type'][dataset['chest_pain_type'] == 2] = 'non-angin
dataset['chest_pain_type'][dataset['chest_pain_type'] == 3] = 'asymptoma

dataset['fasting_blood_sugar'][dataset['fasting_blood_sugar'] == 0] = 'l
dataset['fasting_blood_sugar'][dataset['fasting_blood_sugar'] == 1] = 'g

dataset['rest_ecg'][dataset['rest_ecg'] == 0] = 'normal'
dataset['rest_ecg'][dataset['rest_ecg'] == 1] = 'ST-T wave abnormality'
dataset['rest_ecg'][dataset['rest_ecg'] == 2] = 'left ventricular hypert

dataset['exercise_induced_angina'][dataset['exercise_induced_angina'] ==
dataset['exercise_induced_angina'][dataset['exercise_induced_angina'] ==

dataset['st_slope'][dataset['st_slope'] == 0] = 'upsloping'
dataset['st_slope'][dataset['st_slope'] == 1] = 'flat'
dataset['st_slope'][dataset['st_slope'] == 2] = 'downsloping'

dataset['thalassemia'][dataset['thalassemia'] == 1] = 'normal'
dataset['thalassemia'][dataset['thalassemia'] == 2] = 'fixed defect'
dataset['thalassemia'][dataset['thalassemia'] == 3] = 'reversable defect
```

```
/tmp/ipython-input-4269461875.py:1: FutureWarning: ChainedAssignmentErro
You are setting values through chained assignment. Currently this works
A typical example is when you are setting values in a column of a DataFr

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assigr

See the caveats in the documentation: https://pandas.pydata.org/pandas-d

  dataset['sex'][dataset['sex'] == 0] = 'female'
/tmp/ipython-input-4269461875.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-d
  dataset['sex'][dataset['sex'] == 0] = 'female'
/tmp/ipython-input-4269461875.py:1: FutureWarning: Setting an item of in
  dataset['sex'][dataset['sex'] == 0] = 'female'
/tmp/ipython-input-4269461875.py:4: FutureWarning: ChainedAssignmentErro
You are setting values through chained assignment. Currently this works
A typical example is when you are setting values in a column of a DataFr

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assigr

See the caveats in the documentation: https://pandas.pydata.org/pandas-d

  dataset['chest_pain_type'][dataset['chest_pain_type'] == 0] = 'typica
/tmp/ipython-input-4269461875.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-d
  dataset['chest_pain_type'][dataset['chest_pain_type'] == 0] = 'typica
/tmp/ipython-input-4269461875.py:4: FutureWarning: Setting an item of in
  dataset['chest_pain_type'][dataset['chest_pain_type'] == 0] = 'typica
```

```
/tmp/ipython-input-4269461875.py:9: FutureWarning: ChainedAssignmentErr
You are setting values through chained assignment. Currently this works
A typical example is when you are setting values in a column of a DataF

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assig

See the caveats in the documentation: https://pandas.pydata.org/pandas-

  dataset['fasting_blood_sugar'][dataset['fasting_blood_sugar'] == 0] =
/tmp/ipython-input-4269461875.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
  dataset['fasting_blood_sugar'][dataset['fasting_blood_sugar'] == 0] =
/tmp/ipython-input-4269461875.py:9: FutureWarning: Setting an item of in
  dataset['fasting_blood_sugar'][dataset['fasting_blood_sugar'] == 0] =
/tmp/ipython-input-4269461875.py:12: FutureWarning: ChainedAssignmentErr
You are setting values through chained assignment. Currently this works
A typical example is when you are setting values in a column of a DataF
```

Review the datatypes,

```
dataset.dtypes
```

	0
age	int64
sex	object
chest_pain_type	object
resting_blood_pressure	int64
cholesterol	int64
fasting_blood_sugar	object
rest_ecg	object
max_heart_rate_achieved	int64
exercise_induced_angina	object
st_depression	float64
st_slope	object
num_major_vessels	int64
thalassemia	object
target	int64

dtype: object

Some of those aren't quite right. The code below changes them into categorical variables,

```
dataset['sex'] = dataset['sex'].astype('object')
dataset['chest_pain_type'] = dataset['chest_pain_type'].astype('object')
dataset['fasting_blood_sugar'] = dataset['fasting_blood_sugar'].astype('
dataset['rest_ecg'] = dataset['rest_ecg'].astype('object')
dataset['exercise_induced_angina'] = dataset['exercise_induced_angina'].
dataset['st_slope'] = dataset['st_slope'].astype('object')
dataset['thalassemia'] = dataset['thalassemia'].astype('object')
```

```
dataset.dtypes
```

	0
age	int64
sex	object
chest_pain_type	object
resting_blood_pressure	int64
cholesterol	int64
fasting_blood_sugar	object
rest_ecg	object
max_heart_rate_achieved	int64
exercise_induced_angina	object
st_depression	float64
st_slope	object
num_major_vessels	int64
thalassemia	object
target	int64

dtype: object

For the categorical varibles, we need to create dummy variables. I'm also going to drop the first category of each. For example, rather than having 'male' and 'female', we'll have 'male' with values of 0 or 1 (1 being male, and 0 therefore being female).

```
dataset = pd.get_dummies(dataset, drop_first=True)
```

Now, lets see

```
dataset.head()
```

	age	resting_blood_pressure	cholesterol	max_heart_rate_achieved	st_d
0	63	145	233	150	
1	37	130	250	187	
2	41	130	204	172	
3	56	120	236	178	
4	57	120	354	163	

Dataset information (Pandas Profiling)

```
! pip install https://github.com/pandas-profiling/pandas-profiling/archi
```

Show hidden output

```
import pandas as pd
import pandas_profiling
from pandas_profiling import ProfileReport
```

Upgrade to ydata-sdk

Improve your data and profiling with ydata-sdk, featuring data quality scoring, redundancy detection, outlier identification, text validation, and synthetic data generation.

```
/tmp/ipython-input-3916717816.py:2: DeprecationWarning: `import pandas_pr
```

```
import pandas_profiling as pp
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

```
pp.ProfileReport(dataset, title = 'Pandas Profiling report of "dataset"'
```

Show hidden output

Splitting the dataset into the Training set and Test set

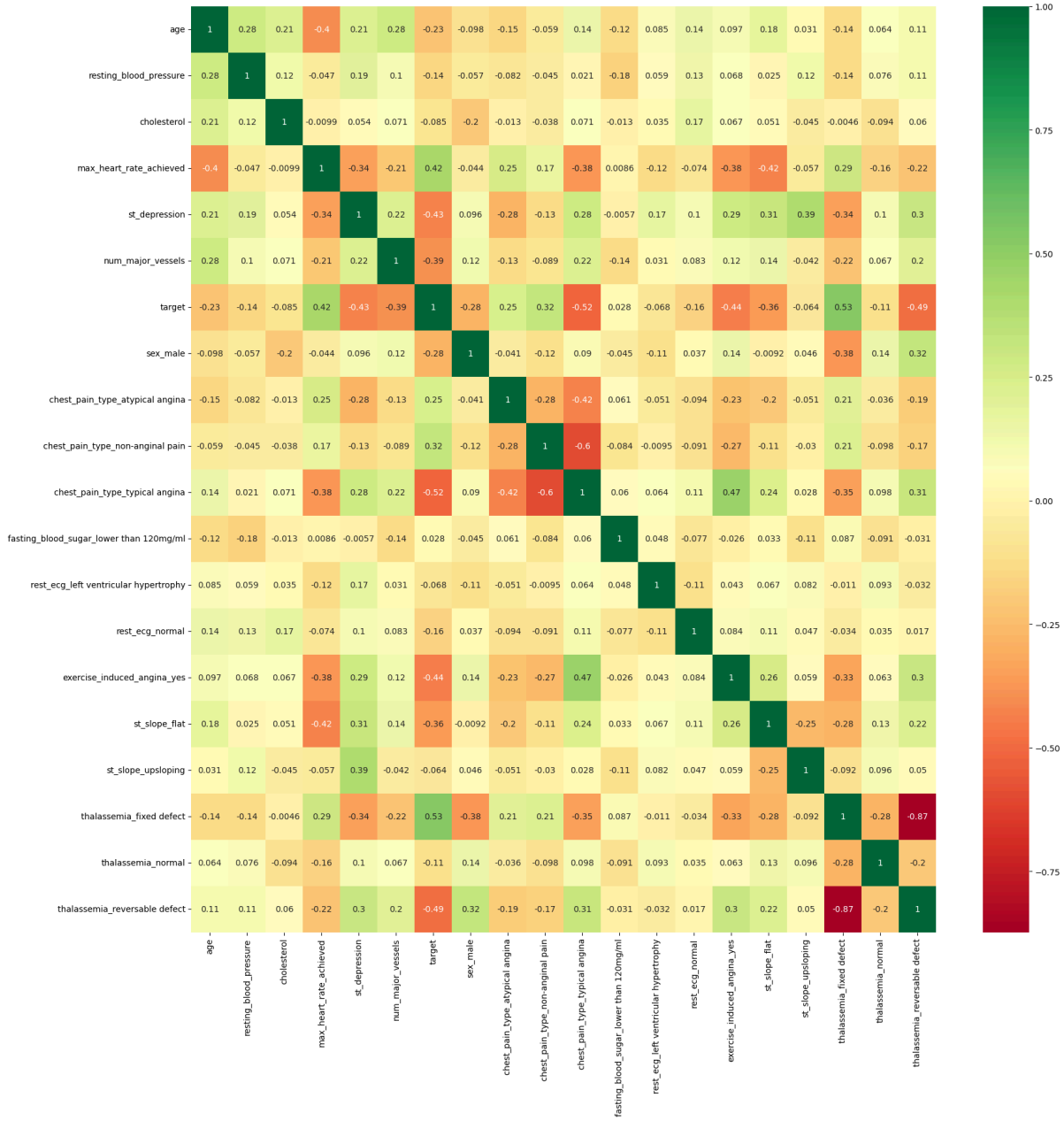
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
```

Feature Scaling

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

Feature Selection

```
import seaborn as sns
#get correlations of each features in dataset
corrmat = dataset.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(dataset[top_corr_features].corr(),annot=True,cmap="RdYlGn"
```

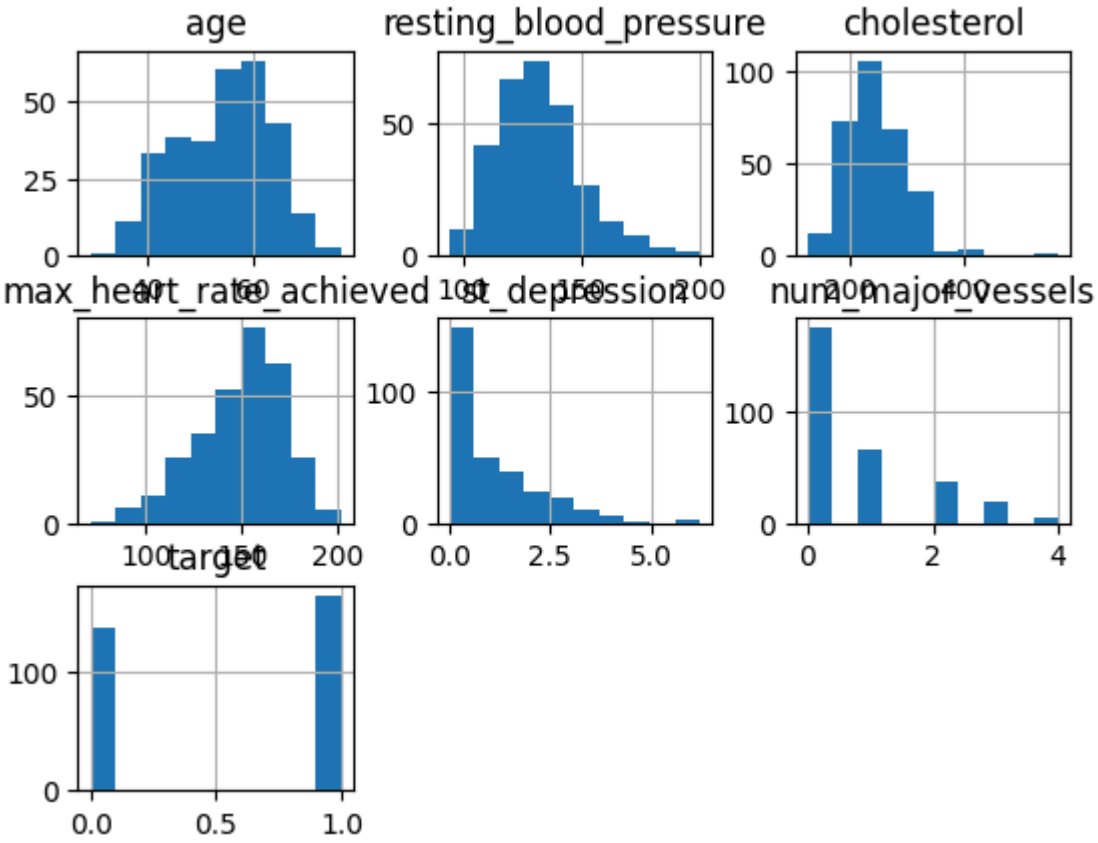



```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

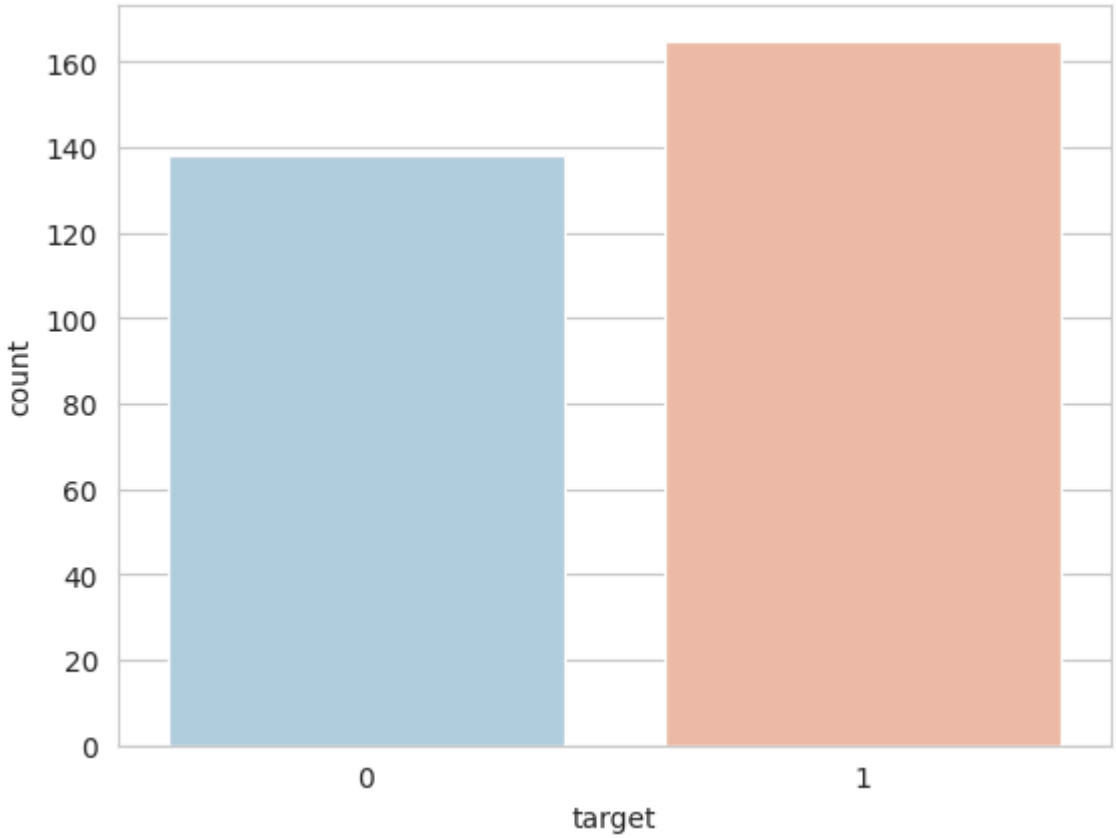
```
dataset.hist()
```

```
array([[<Axes: title={'center': 'age'}>,
        <Axes: title={'center': 'resting_blood_pressure'}>,
        <Axes: title={'center': 'cholesterol'}>],
       [<Axes: title={'center': 'max_heart_rate_achieved'}>,
        <Axes: title={'center': 'st_depression'}>,
        <Axes: title={'center': 'num_major_vessels'}>],
       [<Axes: title={'center': 'target'}>, <Axes: >, <Axes: >]],
      dtype=object)
```




```
sns.set_style('whitegrid')
sns.countplot(x='target',data=dataset,palette='RdBu_r')
```

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

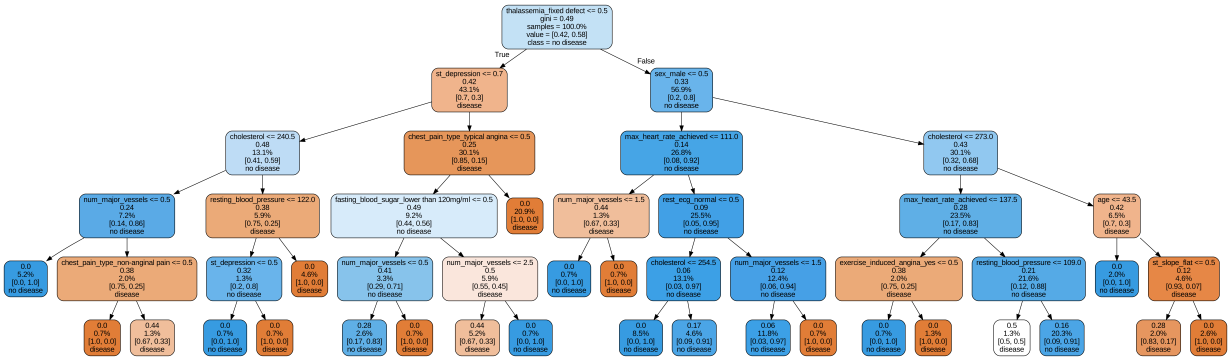


This gives us on explainability tool. However, I can't glance at this and get a quick sense of the most important features. We'll revisit those later.

Random Forest Classifier and Decision tree Model

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import export_graphviz
from IPython.display import Image
from subprocess import call

# Assuming 'dataset' is already defined with your data
X = dataset.drop('target', axis=1)  # Drop 'target' column, use axis=1 f
y = dataset['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.

model = RandomForestClassifier(max_depth=5)
model.fit(X_train, y_train)


# Decision Tree
estimator = model.estimators_[1]
feature_names = X_train.columns  # Use columns from X_train

y_train_str = y_train.astype('str')
```

```
y_train_str[y_train_str == '0'] = 'no disease'
y_train_str[y_train_str == '1'] = 'disease'

export_graphviz(estimator, out_file='tree.dot',
                feature_names = feature_names,
                class_names = y_train_str,
                rounded = True, proportion = True,
                label='root',
                precision = 2, filled = True)

call(['dot', '-Tpng', 'tree.dot', '-o', 'tree.png', '-Gdpi=600'])

Image(filename = 'tree.png')
```




```
y_predict = model.predict(X_test)
y_pred_quant = model.predict_proba(X_test)[:, 1]
y_pred_bin = model.predict(X_test)
```

Assess the fit with a confusion matrix,

```
confusion_matrix = confusion_matrix(y_test, y_pred_bin)
confusion_matrix
```

```
array([[25,  4],
       [ 4, 28]])
```

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred_bin)
```

```
print("Accuracy:", accuracy)
accuracy_rf = accuracy
```

```
Accuracy: 0.8688524590163934
```

```
from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred_bin)
print("Precision:", precision)
```

```
Precision: 0.875
```

```
from sklearn.metrics import f1_score

f1 = f1_score(y_test, y_pred_bin)
print("F1 Score:", f1)
```

```
F1 Score: 0.875
```

```
from sklearn.metrics import recall_score, accuracy_score, precision_scor
recall = recall_score(y_test, y_pred_bin)
print("Recall:", recall)
```

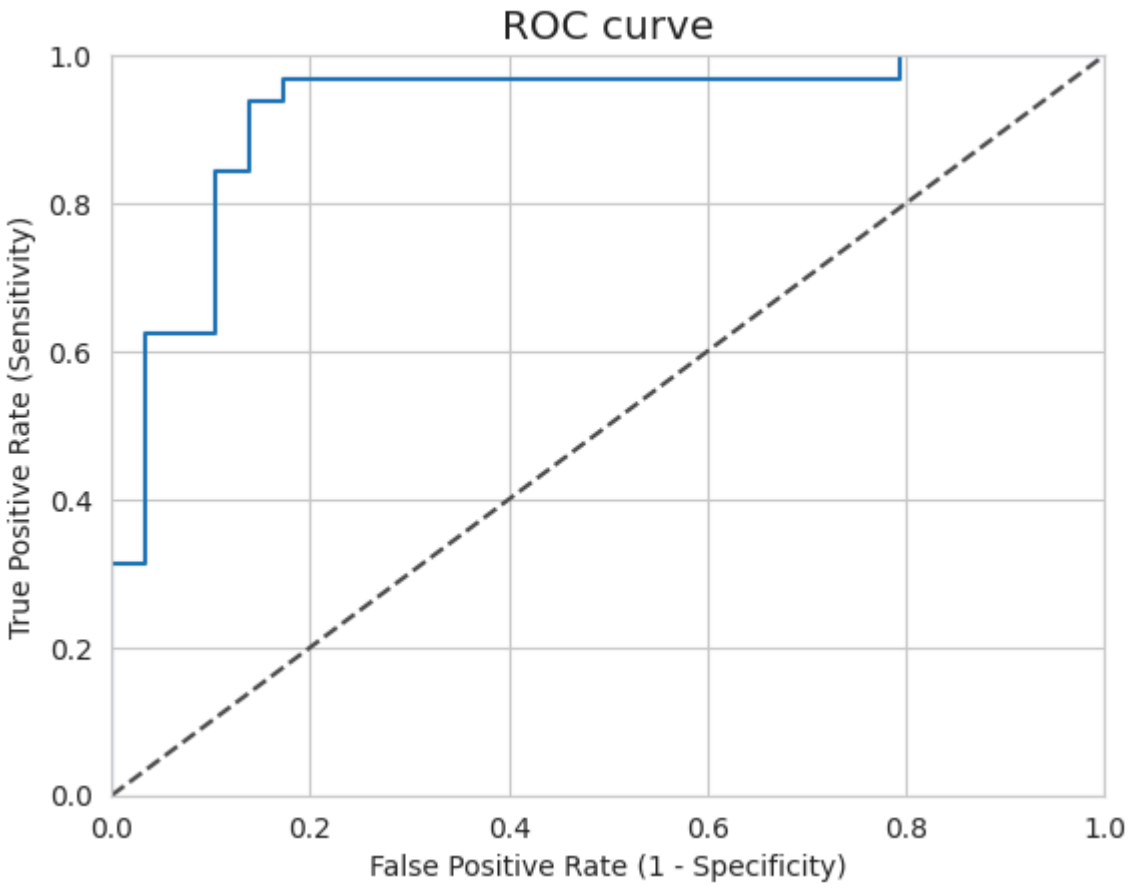
```
Recall: 0.875
```

Diagnostic tests are often sold, marketed, cited and used with sensitivity and specificity as the headline metrics. Sensitivity and specificity are defined as,

```
total=sum(sum(confusion_matrix))

sensitivity = confusion_matrix[0,0]/(confusion_matrix[0,0]+confusion_mat
print('Sensitivity : ', sensitivity )

specificity = confusion_matrix[1,1]/(confusion_matrix[1,1]+confusion_mat
print('Specificity : ', specificity)
```

```
Sensitivity :  0.8620689655172413
Specificity :  0.875
```

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_quant)

fig, ax = plt.subplots()
ax.plot(fpr, tpr)
ax.plot([0, 1], [0, 1], transform=ax.transAxes, ls="--", c=".3")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.rcParams['font.size'] = 12
plt.title('ROC curve')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.grid(True)
```



That seems reasonable. Let's also check with a Receiver Operator Curve (ROC),

Another common metric is the Area Under the Curve, or AUC. This is a convenient way to capture the performance of a model in a single number, although it's not without certain issues. As a rule of thumb, an AUC can be classed as follows,

0.90 - 1.00 = excellent

0.80 - 0.90 = good

0.70 - 0.80 = fair

0.60 - 0.70 = poor

0.50 - 0.60 = fail

Let's see what the above ROC gives us,

```
auc(fpr, tpr)
```

```
np.float64(0.9234913793103449)
```

Cross Validation

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import cross_val_score, StratifiedKFold
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_scor


cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

metrics = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']
results = {}
for metric in metrics:
  scores = cross_val_score(model, X, y, cv=cv, scoring=metric)
  results[metric] = scores

# Print the cross-validation results
for metric in metrics:
  print(f'{metric.capitalize()}: {results[metric].mean():.4f} (std: {res
```

```
Accuracy: 0.8483 (std: 0.0579)
Precision: 0.8396 (std: 0.0768)
Recall: 0.8909 (std: 0.0454)
F1: 0.8484 (std: 0.0566)
Roc_auc: 0.9161 (std: 0.0423)
```

Logistic regression

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
```

```
datasetlr = pd.read_csv('/content/heart_disease_data.csv')
X_lr = datasetlr.iloc[:, 3:-1].values
y_lr = datasetlr.iloc[:, -1].values
```

```
X_trainlr, X_testlr, y_trainlr, y_testlr = train_test_split(X_lr, y_lr,
```

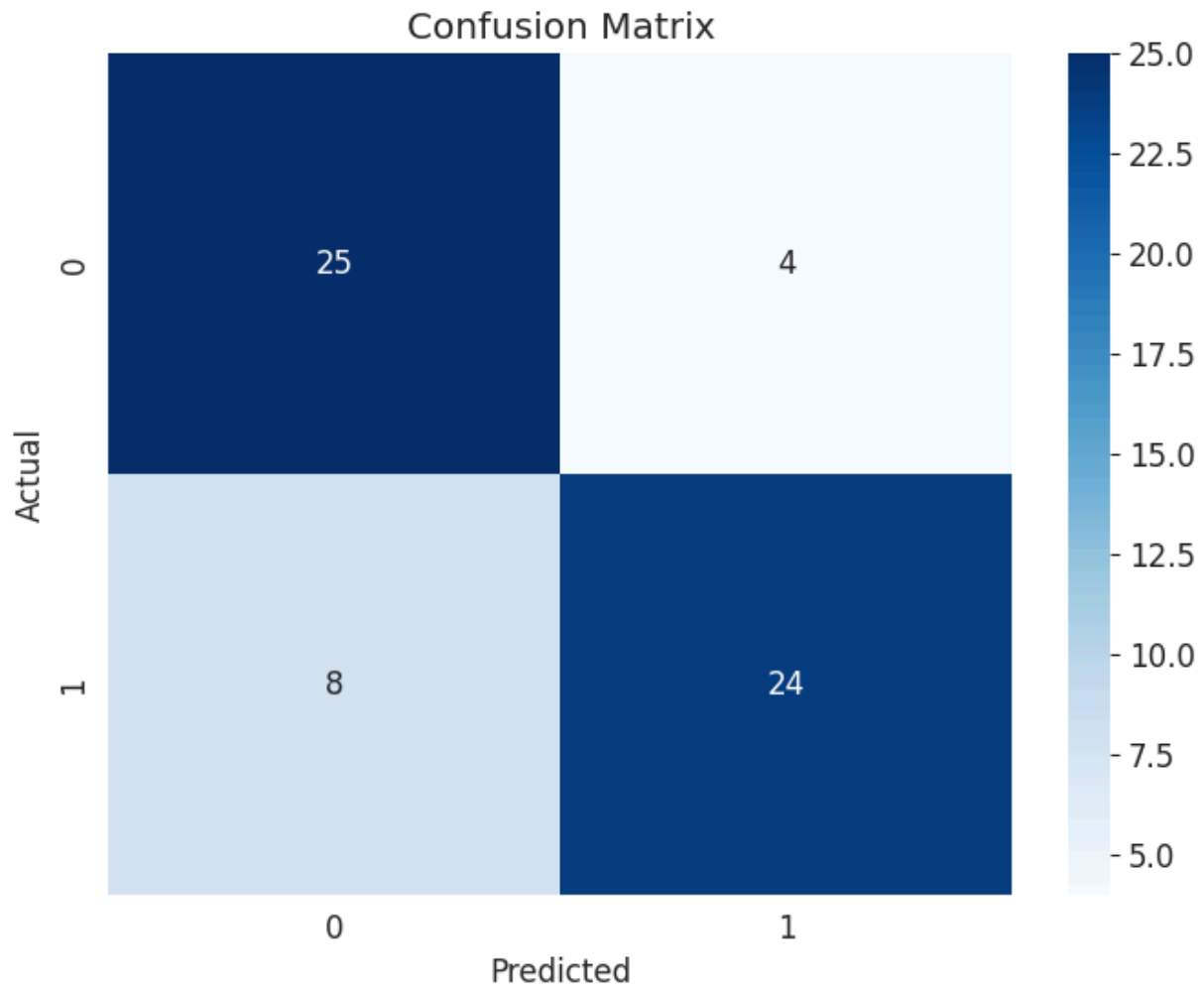
```
# Create and train the Logistic Regression model
logreg = LogisticRegression()
logreg.fit(X_trainlr, y_trainlr)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
# Make predictions
y_predlr = logreg.predict(X_testlr)
```

```
# Confusion Matrix
cmlr = confusion_matrix(y_testlr, y_predlr)
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(cmlr, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

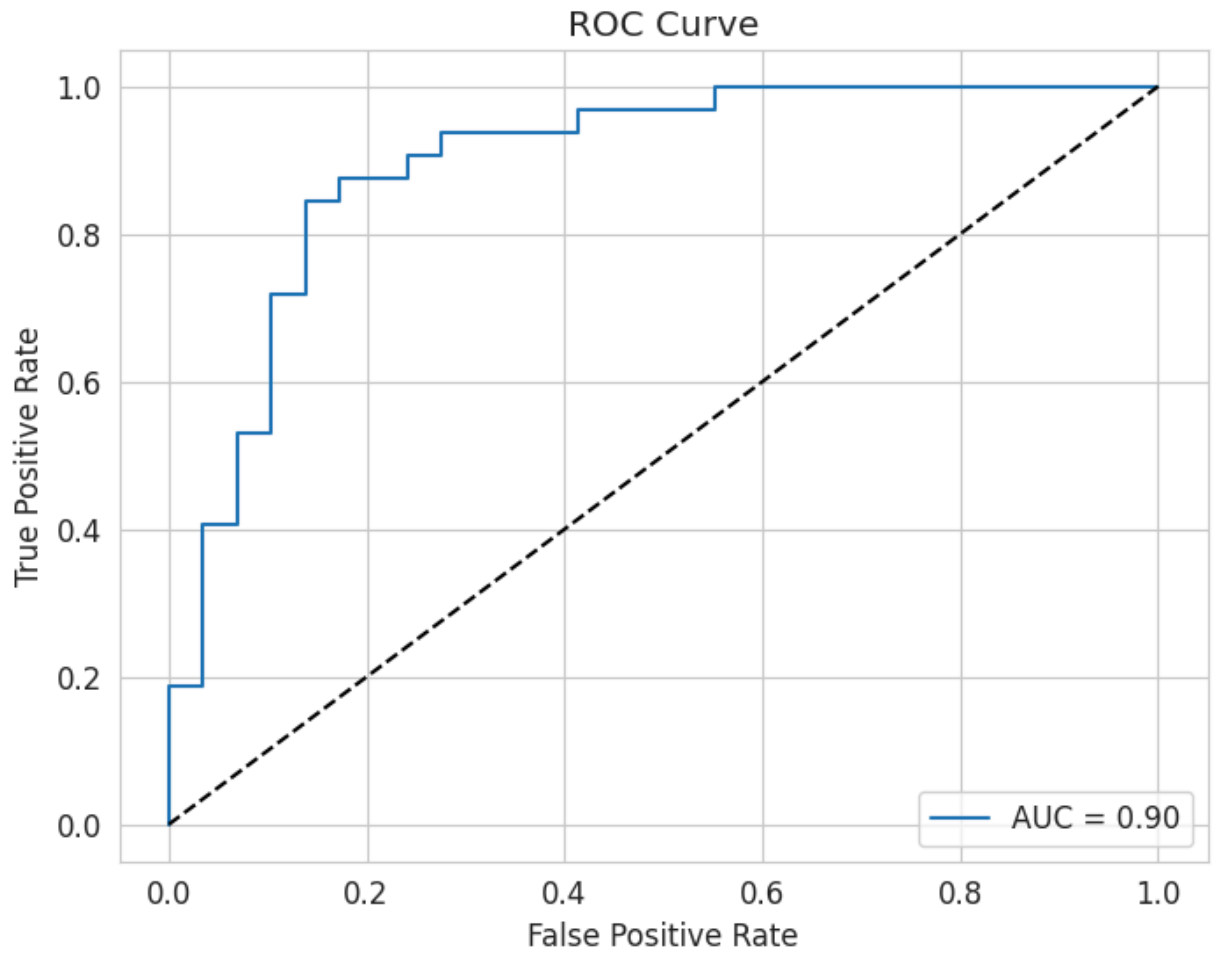



```
# Calculate sensitivity
sensitivity = cmlr[0, 0] / (cmlr[0, 0] + cmlr[1, 0])
print('Sensitivity:', sensitivity)

# Calculate specificity
specificity = cmlr[1, 1] / (cmlr[1, 1] + cmlr[0, 1])
print('Specificity:', specificity)
```

```
Sensitivity: 0.7575757575757576
Specificity: 0.8571428571428571
```

```
# ROC Curve and AUC
y_predlr_proba = logreg.predict_proba(X_testlr)[:, 1]
y_predlr = logreg.predict(X_testlr)
y_predlr_quant = logreg.predict_proba(X_testlr)[:, 1]
```

```
fpr, tpr, thresholds = roc_curve(y_testlr, y_predlr_quant)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```




```
# Calculate accuracy
accuracy = (cmlr[0, 0] + cmlr[1, 1]) / total
print('Accuracy:', accuracy)
accuracy_lr = accuracy
```

```
Accuracy: 0.8032786885245902
```

```
from sklearn.metrics import precision_score
```

```
precision = precision_score(y_testlr, y_predlr)
print("Precision:", precision)
```

```
Precision: 0.8571428571428571
```

```
f1 = f1_score(y_testlr, y_predlr)
print("F1 Score:", f1)
```

```
F1 Score: 0.8
```

```
from sklearn.metrics import recall_score

recall = recall_score(y_testlr, y_predlr)
print("Recall:", recall)
```

```
Recall: 0.75
```

```
print(f'AUC: {roc_auc:.2f}')
```

```
AUC: 0.90
```

```
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_scor

# Create a Logistic Regression model
logreg = LogisticRegression()

# Define the cross-validation strategy
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Define the metrics to evaluate
metrics = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']
results = {}

# Perform cross-validation for each metric
for metric in metrics:
  scores = cross_val_score(logreg, X_lr, y_lr, cv=cv, scoring=metric)
  results[metric] = scores

# Print the cross-validation results
for metric in metrics:
  print(f'{metric.capitalize()}: {results[metric].mean():.4f} (std: {res
```

```
Accuracy: 0.7755 (std: 0.0499)
Precision: 0.7735 (std: 0.0703)
Recall: 0.8485 (std: 0.0469)
F1: 0.8060 (std: 0.0337)
Roc_auc: 0.8652 (std: 0.0529)
```

KNN Model

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
df1 = pd.read_csv('/content/heart_disease_data.csv')
```

```
df1.info()
```

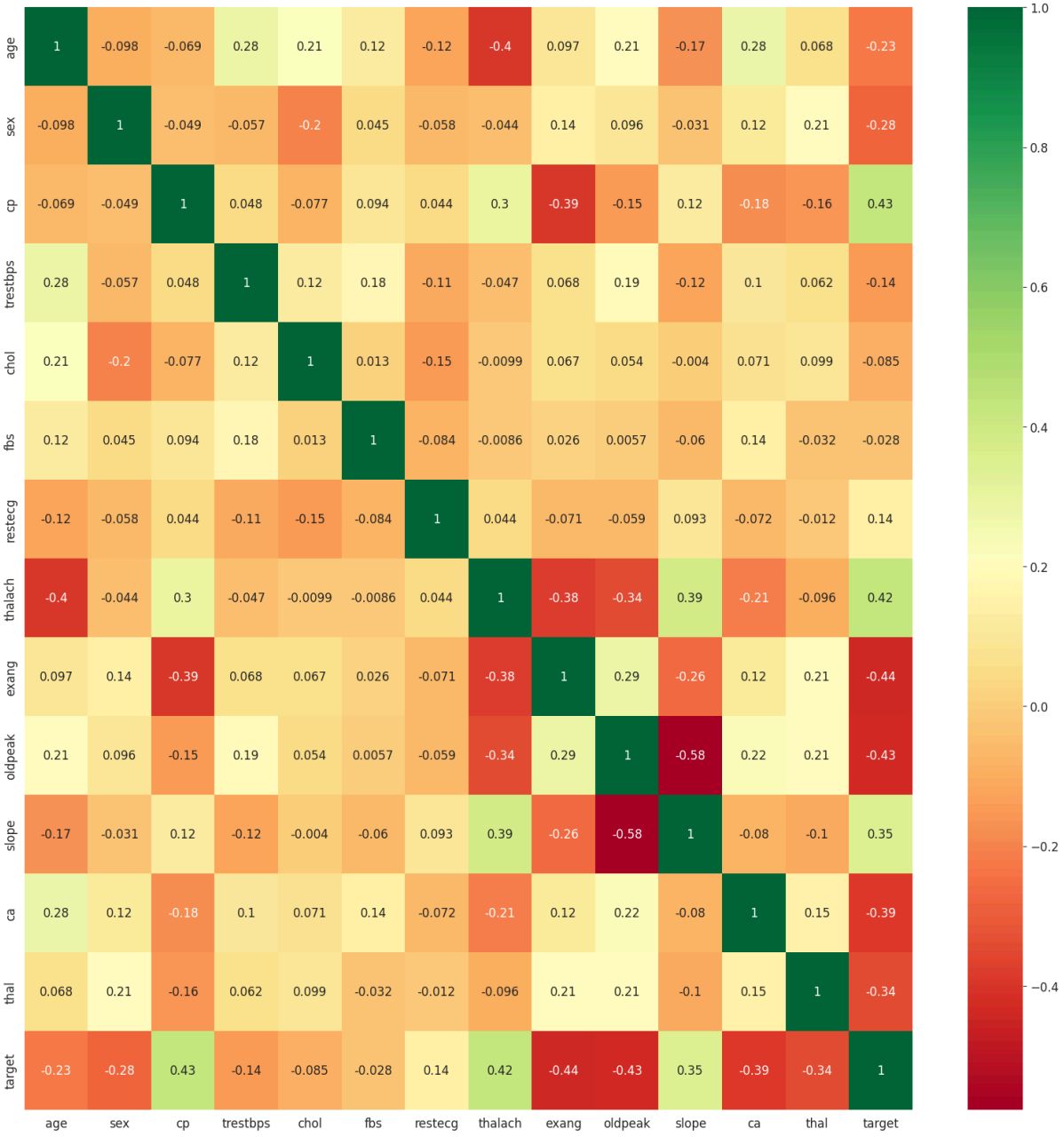
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

```
df1.describe()
```

	age	sex	cp	trestbps	chol	fbs	re
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.

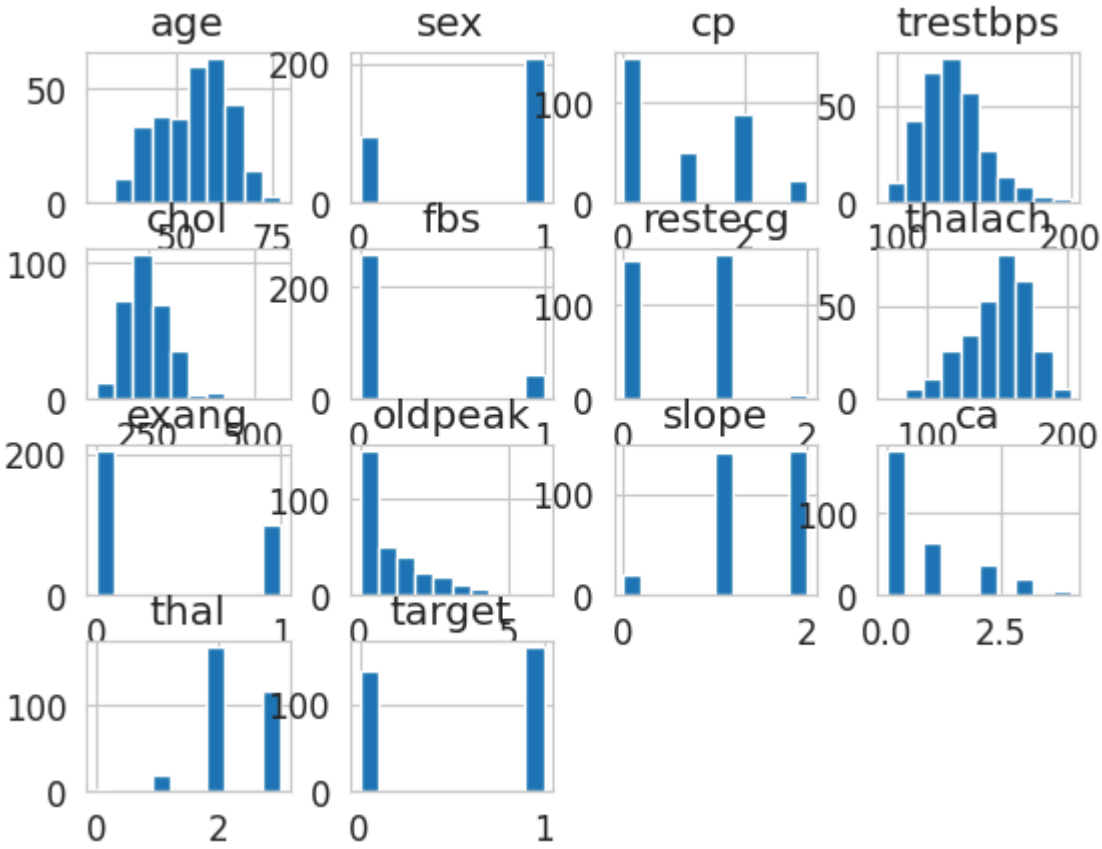
Feature selection

```
import seaborn as sns
#get correlations of each features in dataset
corrmat = df1.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(df1[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```


age 1 -0.098 -0.069 0.28 0.21 0.12 -0.12 -0.4 0.097 0.21 -0.17 0.28 0.068 -0.23
sex -0.098 1 -0.049 -0.057 -0.2 0.045 -0.058 -0.044 0.14 0.096 -0.031 0.12 0.21 -0.28
cp -0.069 -0.049 1 0.048 -0.077 0.094 0.044 0.3 -0.39 -0.15 0.12 -0.18 -0.16 0.43
trestbps 0.28 -0.057 0.048 1 0.12 0.18 -0.11 -0.047 0.068 0.19 -0.12 0.1 0.062 -0.14
chol 0.21 -0.2 -0.077 0.12 1 0.013 -0.15 -0.0099 0.067 0.054 -0.004 0.071 0.099 -0.085
fbs 0.12 0.045 0.094 0.18 0.013 1 -0.084 -0.0086 0.026 0.0057 -0.06 0.14 -0.032 -0.028
restecg -0.12 -0.058 0.044 -0.11 -0.15 -0.084 1 0.044 -0.071 -0.059 0.093 -0.072 -0.012 0.14
thalach -0.4 -0.044 0.3 -0.047 -0.0099 -0.0086 0.044 1 -0.38 -0.34 0.39 -0.21 -0.096 0.42
exang 0.097 0.14 -0.39 0.068 0.067 0.026 -0.071 -0.38 1 0.29 -0.26 0.12 0.21 -0.44
oldpeak 0.21 0.096 -0.15 0.19 0.054 0.0057 -0.059 -0.34 0.29 1 -0.58 0.22 0.21 -0.43
slope -0.17 -0.031 0.12 -0.12 -0.004 -0.06 0.093 0.39 -0.26 -0.58 1 -0.08 -0.1 0.35
ca 0.28 0.12 -0.18 0.1 0.071 0.14 -0.072 -0.21 0.12 0.22 -0.08 1 0.15 -0.39
thal 0.068 0.21 -0.16 0.062 0.099 -0.032 -0.012 -0.096 0.21 0.21 -0.1 0.15 1 -0.34
target -0.23 -0.28 0.43 -0.14 -0.085 -0.028 0.14 0.42 -0.44 -0.43 0.35 -0.39 -0.34 1
age sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal target
1.0 0.8 0.6 0.4 0.2 0.0 −0.2 −0.4

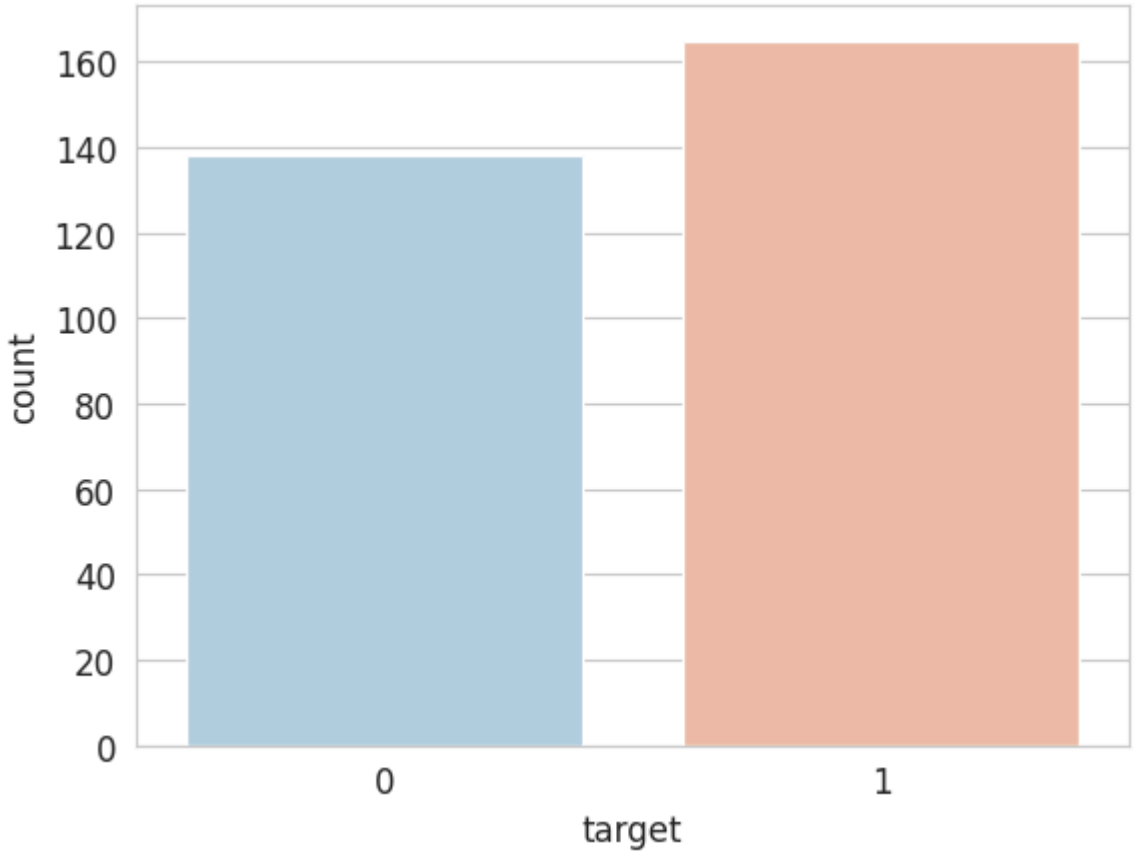
```
df1.hist()
```

```
array([[<Axes: title={'center': 'age'}>, <Axes: title={'center':
'sex'}>,
        <Axes: title={'center': 'cp'}>,
        <Axes: title={'center': 'trestbps'}>],
       [<Axes: title={'center': 'chol'}>,
        <Axes: title={'center': 'fbs'}>,
        <Axes: title={'center': 'restecg'}>,
        <Axes: title={'center': 'thalach'}>],
       [<Axes: title={'center': 'exang'}>,
        <Axes: title={'center': 'oldpeak'}>,
        <Axes: title={'center': 'slope'}>,
        <Axes: title={'center': 'ca'}>],
       [<Axes: title={'center': 'thal'}>,
        <Axes: title={'center': 'target'}>, <Axes: >, <Axes: >]],
      dtype=object)
```




```
sns.set_style('whitegrid')
sns.countplot(x='target',data=df1,palette='RdBu_r')
```

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



Data Processing

```
dataset1 = pd.get_dummies(df1, columns = ['sex', 'cp', 'fbs', 'restecg',
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
standardScaler = StandardScaler()
columns_to_scale = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
dataset1[columns_to_scale] = standardScaler.fit_transform(dataset1[colum
```

```
dataset1.head()
```

	age	trestbps	chol	thalach	oldpeak	target	sex_0	sex_1	cp_
0	0.952197	0.763956	-0.256334	0.015443	1.087338	1	False	True	Fals
1	-1.915313	-0.092738	0.072199	1.633471	2.122573	1	False	True	Fals
2	-1.474158	-0.092738	-0.816773	0.977514	0.310912	1	True	False	Fals
3	0.180175	-0.663867	-0.198357	1.239897	-0.206705	1	False	True	Fals
4	0.290464	-0.663867	2.082050	0.583939	-0.379244	1	True	False	Tru

5 rows × 31 columns

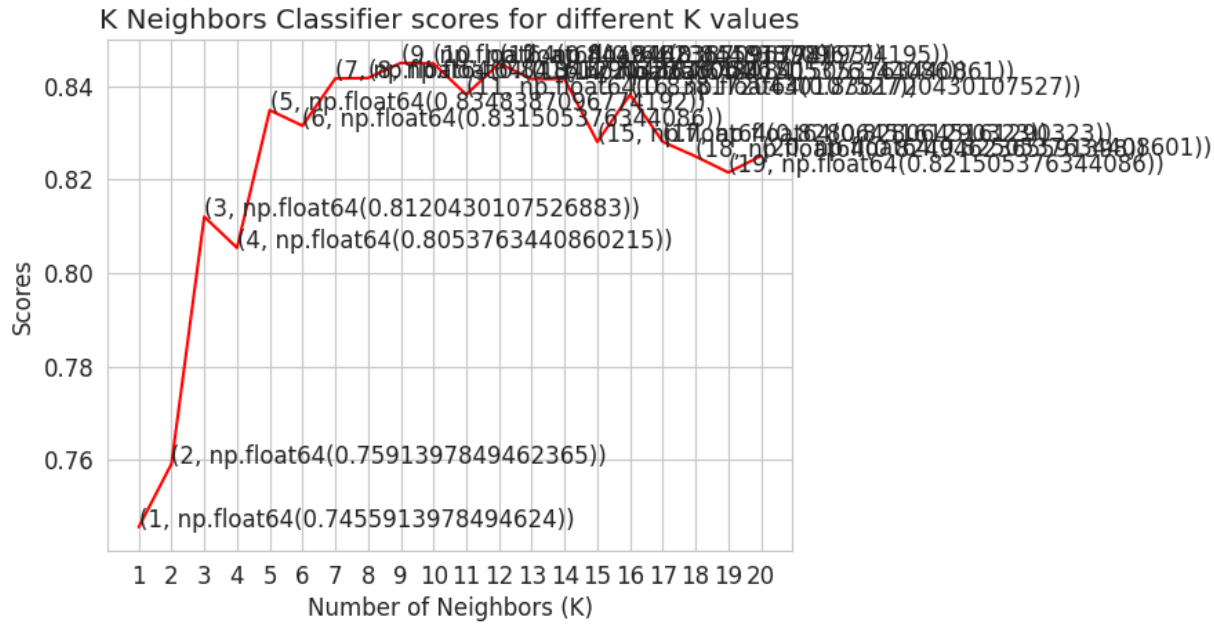
```
y1 = dataset1['target']
X1 = dataset1.drop(['target'], axis = 1)
```

```
from sklearn.model_selection import train_test_split
X_trainknn, X_testknn, y_trainknn, y_testknn = train_test_split(X1, y1,
```

```
from sklearn.model_selection import cross_val_score
knn_scores = []
for k in range(1,21):
    knn_classifier = KNeighborsClassifier(n_neighbors = k)
    score=cross_val_score(knn_classifier,X1,y1,cv=10)
    knn_scores.append(score.mean())
```

```
plt.plot([k for k in range(1, 21)], knn_scores, color = 'red')
for i in range(1,21):
    plt.text(i, knn_scores[i-1], (i, knn_scores[i-1]))
plt.xticks([i for i in range(1, 21)])
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Scores')
plt.title('K Neighbors Classifier scores for different K values')
```

```
Text(0.5, 1.0, 'K Neighbors Classifier scores for different K values')
```




```
knn_classifier = KNeighborsClassifier(n_neighbors = 12)
score=cross_val_score(knn_classifier,X1,y1,cv=10)
```

```
score.mean()
accuracy_knn = score.mean()
```

```
from sklearn.metrics import precision_score

knn_classifier.fit(X_trainknn, y_trainknn)
y_pred_knn = knn_classifier.predict(X_testknn)

precision_knn = precision_score(y_testknn, y_pred_knn)
print("Precision:", precision_knn)
```

```
Precision: 0.90625
```

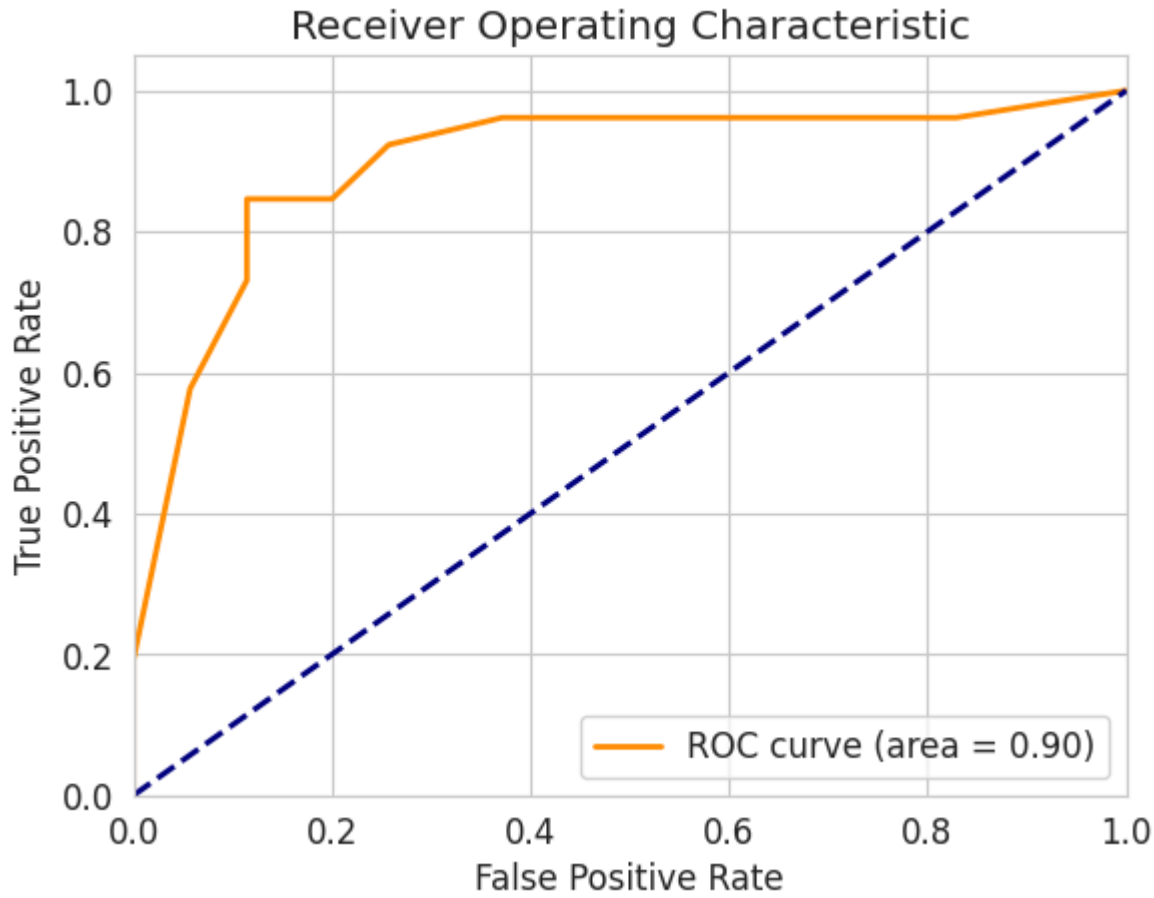
```
f1_knn = f1_score(y_testknn, y_pred_knn)
print("F1 Score:", f1_knn)
```

```
F1 Score: 0.90625
```

```
from sklearn.metrics import recall_score

recall_knn = recall_score(y_testknn, y_pred_knn)
print("Recall:", recall_knn)
```

```
Recall: 0.90625
```

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score

knn_classifier = KNeighborsClassifier(n_neighbors = 12)
score=cross_val_score(knn_classifier,X1,y1,cv=10)

X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_siz
knn_classifier.fit(X1_train, y1_train)
y_pred_proba = knn_classifier.predict_proba(X1_test)[:,1]
fpr, tpr, thresholds = roc_curve(y1_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```




```
auc(fpr, tpr)
```

```
np.float64(0.8978021978021978)
```

Cross validation

```
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_scor


# Create a KNN classifier with the best k value (found earlier)
knn_classifier = KNeighborsClassifier(n_neighbors=12)

# Define the cross-validation strategy
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Define the metrics to evaluate
metrics = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']
results = {}

# Perform cross-validation for each metric
for metric in metrics:
  if metric == 'roc_auc':
    scores = cross_val_score(knn_classifier, X1, y1, cv=cv, scoring='roc
  else:
    scores = cross_val_score(knn_classifier, X1, y1, cv=cv, scoring=metr
```

```
    results[metric] = scores

# Print the cross-validation results
for metric in metrics:
  print(f'{metric.capitalize()}: {results[metric].mean():.4f} (std: {res
```

```
Accuracy: 0.8346 (std: 0.0476)
Precision: 0.8509 (std: 0.0700)
Recall: 0.8548 (std: 0.0874)
F1: 0.8480 (std: 0.0471)
Roc_auc: 0.9033 (std: 0.0563)
```

ANN

Building the ANN

```
datasetann = pd.read_csv('/content/heart_disease_data.csv')
```

```
X3 = datasetann.iloc[:, :-1].values
y3 = datasetann.iloc[:, -1].values
```

Initializing the ANN

```
tf.random.set_seed(42)
```

```
X_train3, X_test3, y_train3, y_test3 = train_test_split(X3, y3, test_siz
```

```
ann = tf.keras.models.Sequential()
```

Adding the input layer and the first hidden layer

```
ann.add(tf.keras.layers.Dense(units=32, activation='relu'))
```

Adding the second hidden layer

```
ann.add(tf.keras.layers.Dense(units=32, activation='relu'))
```

```
#adding dropout layer
ann.add(tf.keras.layers.Dropout(0.5))
```

```
ann.add(tf.keras.layers.Dense(units=64, activation='relu'))
```

```
#adding another hidden layer
ann.add(tf.keras.layers.Dense(units=64, activation='relu'))
```

```
#adding another dropout layer
ann.add(tf.keras.layers.Dropout(0.5))
```

Adding the output layer

```
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Compiling the ANN

```
ann.compile(optimizer = 'adamW', loss = 'binary_crossentropy', metrics =
```

Training the ANN on the Training set

```
ann.fit(X_train3, y_train3, batch_size = 32, epochs = 300)
```

Show hidden output

Making the predictions and evaluating the model

Predicting the Test set results

```
y_pred3 = ann.predict(X_test3)
y_pred3 = (y_pred3 > 0.5)
# Extract NumPy arrays from Series before reshaping
print(np.concatenate((y_pred3.reshape(len(y_pred3),1), y_test.values.res
```

Show hidden output

Making the Confusion Matrix

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm3 = confusion_matrix(y_test3, y_pred3)
print(cm3)
print("Accuracy: {:.2f}%".format(accuracy_score(y_test3, y_pred3)*100))
accuracy_ann = accuracy_score(y_test3, y_pred3)
```

```
[[27  2]
 [ 8 24]]
Accuracy: 83.61%
```

```
from sklearn.metrics import precision_score
precision = precision_score(y_test3, y_pred3)
print("Precision:", precision)
```

```
Precision: 0.9230769230769231
```

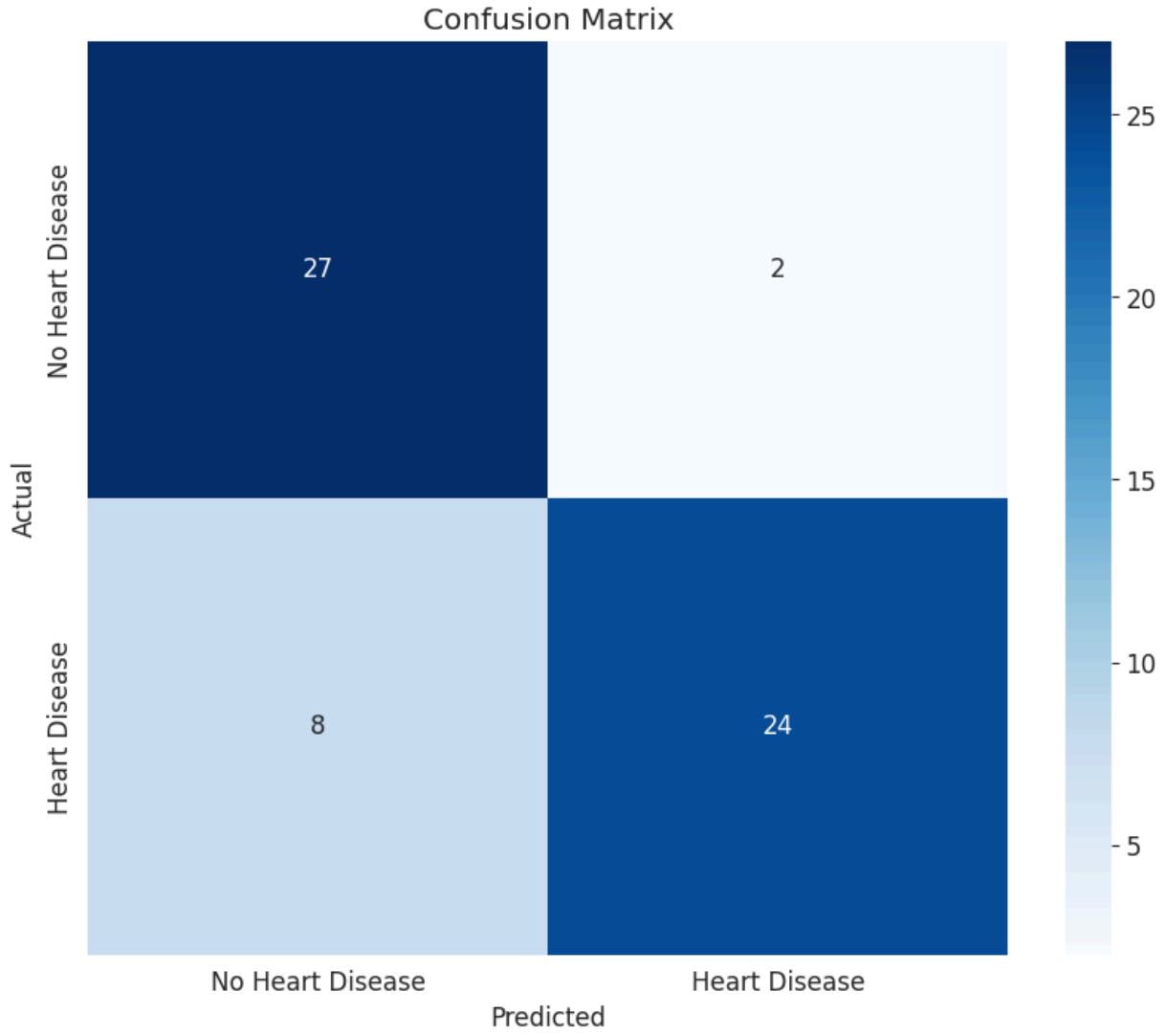
```
f1 = f1_score(y_test3, y_pred3)
print("F1 Score:", f1)
```

```
F1 Score: 0.8275862068965517
```

```
from sklearn.metrics import recall_score
recall = recall_score(y_test3, y_pred3)
print("Recall:", recall)
```

```
Recall: 0.75
```

```
import matplotlib.pyplot as plt
# Making the Confusion Matrix
cm3 = confusion_matrix(y_test3, y_pred3)

# Define the labels for the confusion matrix
labels = ['No Heart Disease', 'Heart Disease']

# Plot the confusion matrix with larger figure size and colorful visuali
plt.figure(figsize=(10, 8))
sns.heatmap(cm3, annot=True, fmt='d', cmap='Blues', xticklabels=labels,
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



Diagnostic tests are often sold, marketed, cited and used with sensitivity and specificity as the headline metrics. Sensitivity and specificity are defined as,

Sensitivity=TruePositives/(TruePositives+FalseNegatives)

Specificity=TrueNegatives/(TrueNegatives+FalsePositives)

Let's see what this model is giving,

```
# Sensitivity and Specificity
# Sensitivity (also called the true positive rate, the recall, or probab
# measures the proportion of positives that are correctly identified as

# Specificity (also called the true negative rate) measures the proporti
# (e.g., the percentage of healthy people who are correctly identified a
```

```
tn, fp, fn, tp = cm3.ravel()
sensitivity = tp/(tp+fn)
specificity = tn/(tn+fp)

print("Sensitivity: {:.2f}%".format(sensitivity*100))
print("Specificity: {:.2f}%".format(specificity*100))
```

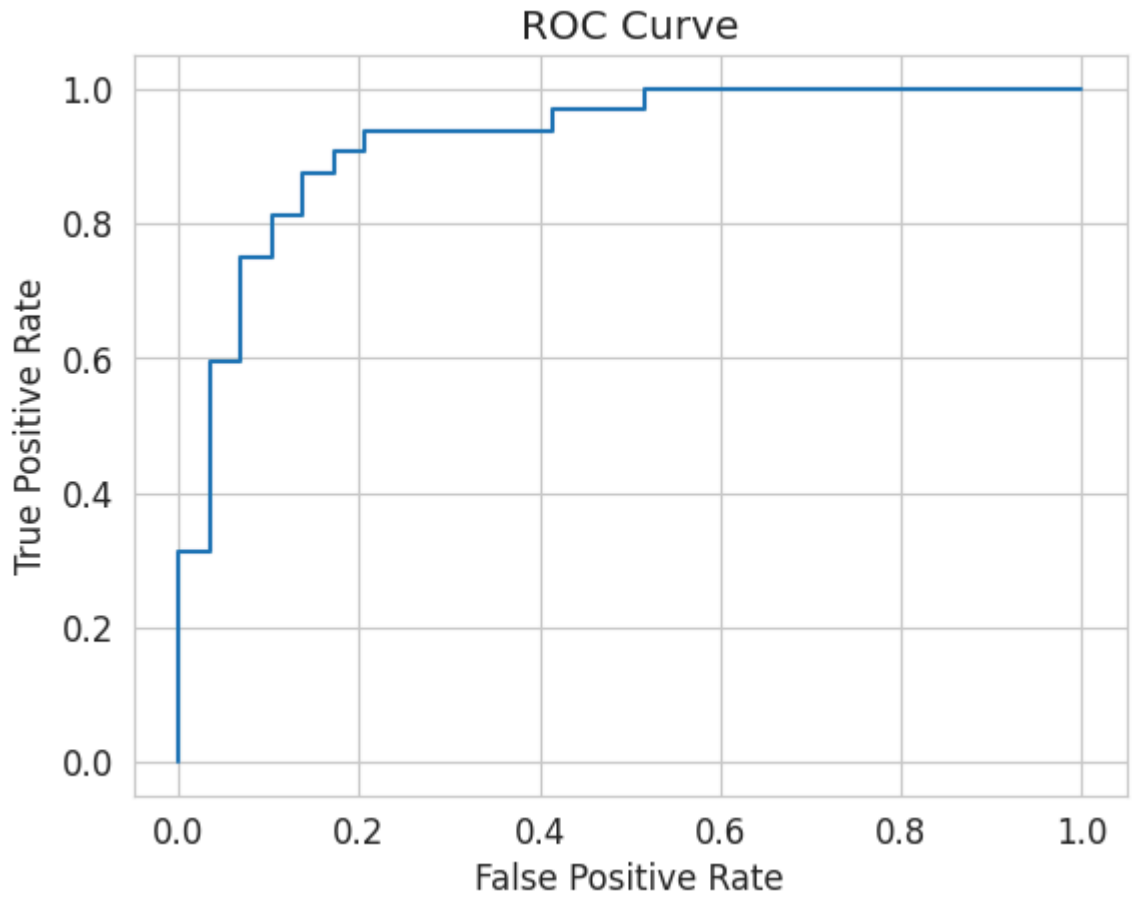
```
Sensitivity: 75.00%
Specificity: 93.10%
```

```
y_pred3_proba = ann.predict(X_test3)
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
```

Let's also check with a Receiver Operator Curve (ROC),

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve

#y_test (true labels) and y_pred_proba (predicted probabilities) from AN
fpr, tpr, thresholds = roc_curve(y_test3, y_pred3_proba)

# Plot the ROC curve
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
```



Double-click (or enter) to edit

Another common metric is the Area Under the Curve, or AUC. This is a convenient way to capture the performance of a model in a single number, although it's not without certain issues. As a rule of thumb, an AUC can be classed as follows,

0.90 - 1.00 = excellent

0.80 - 0.90 = good

0.70 - 0.80 = fair

0.60 - 0.70 = poor

0.50 - 0.60 = fail

Let's see what the above ROC gives us,

```
auc(fpr, tpr)
```

```
np.float64(0.9234913793103448)
```

```
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score, precision_score, recall_scor

# Define the cross-validation strategy
```

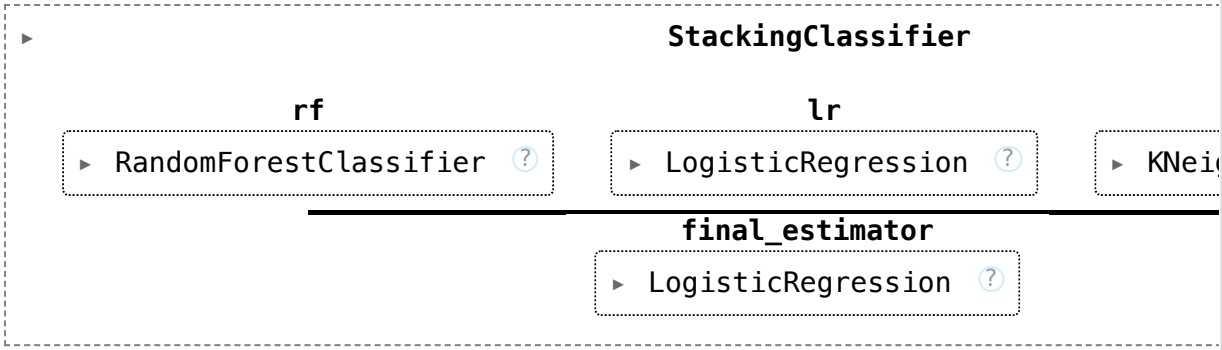
```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Define the metrics to evaluate
metrics = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']
results = {}

# Perform cross-validation for each metric
for metric in metrics:
  scores = []
  for train_index, test_index in cv.split(X3, y3):
    X_train3, X_test3 = X3[train_index], X3[test_index]
    y_train3, y_test3 = y3[train_index], y3[test_index]

    # Create and train the ANN model (same architecture as before)
    ann = tf.keras.models.Sequential()
    ann.add(tf.keras.layers.Dense(units=32, activation='relu'))
    ann.add(tf.keras.layers.Dense(units=32, activation='relu'))
    ann.add(tf.keras.layers.Dropout(0.5))
    ann.add(tf.keras.layers.Dense(units=64, activation='relu'))
    ann.add(tf.keras.layers.Dense(units=64, activation='relu'))
    ann.add(tf.keras.layers.Dropout(0.5))
    ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
    ann.compile(optimizer = 'adamW', loss = 'binary_crossentropy', metri
    ann.fit(X_train3, y_train3, batch_size = 32, epochs = 300, verbose=0

    # Make predictions
    y_pred3 = ann.predict(X_test3)
    y_pred3 = (y_pred3 > 0.5)

    # Calculate the score based on the current metric
    if metric == 'accuracy':
      score = accuracy_score(y_test3, y_pred3)
    elif metric == 'precision':
      score = precision_score(y_test3, y_pred3)
    elif metric == 'recall':
      score = recall_score(y_test3, y_pred3)
    elif metric == 'f1':
      score = f1_score(y_test3, y_pred3)
    elif metric == 'roc_auc':
      score = roc_auc_score(y_test3, y_pred3)

    scores.append(score)

  results[metric] = scores

# Print the cross-validation results
for metric in metrics:
  print(f'{metric.capitalize()}: {np.mean(results[metric]):.4f} (std: {n
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 225ms/step
WARNING:tensorflow:5 out of the last 7 calls to <function TensorFlowTrain
1/2 ━━━━━━━━━━━━━━━━━━━━ 0s 217ms/stepWARNING:tensorflow:6 out of the las
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 224ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 226ms/step
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 397ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 233ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 230ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 235ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 239ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 222ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 222ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 246ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 227ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 230ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 226ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 223ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 224ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 234ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 247ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 222ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 226ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 224ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 227ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 224ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 225ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 219ms/step
Accuracy: 0.7559 (std: 0.0357)
Precision: 0.7726 (std: 0.0518)
Recall: 0.8788 (std: 0.0507)
F1: 0.8443 (std: 0.0331)
Roc_auc: 0.7992 (std: 0.0326)
```

Stacking

```
import matplotlib.pyplot as plt
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Defining the base models
estimators = [
    ('rf', model),
    ('lr', logreg),
    ('knn', knn_classifier)
]

# Define the stacking classifier
stacking_model = StackingClassifier(
    estimators=estimators,
    final_estimator=LogisticRegression()
)

# Fit the stacking model
stacking_model.fit(X_train, y_train)
```




```
# Make predictions
y_pred = stacking_model.predict(X_test)
```

```
# Evaluate the stacking model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8852459016393442
```

```
from sklearn.metrics import precision_score

precision = precision_score(y_test, y_pred)
print("Precision:", precision)
```

```
Precision: 0.9310344827586207
```

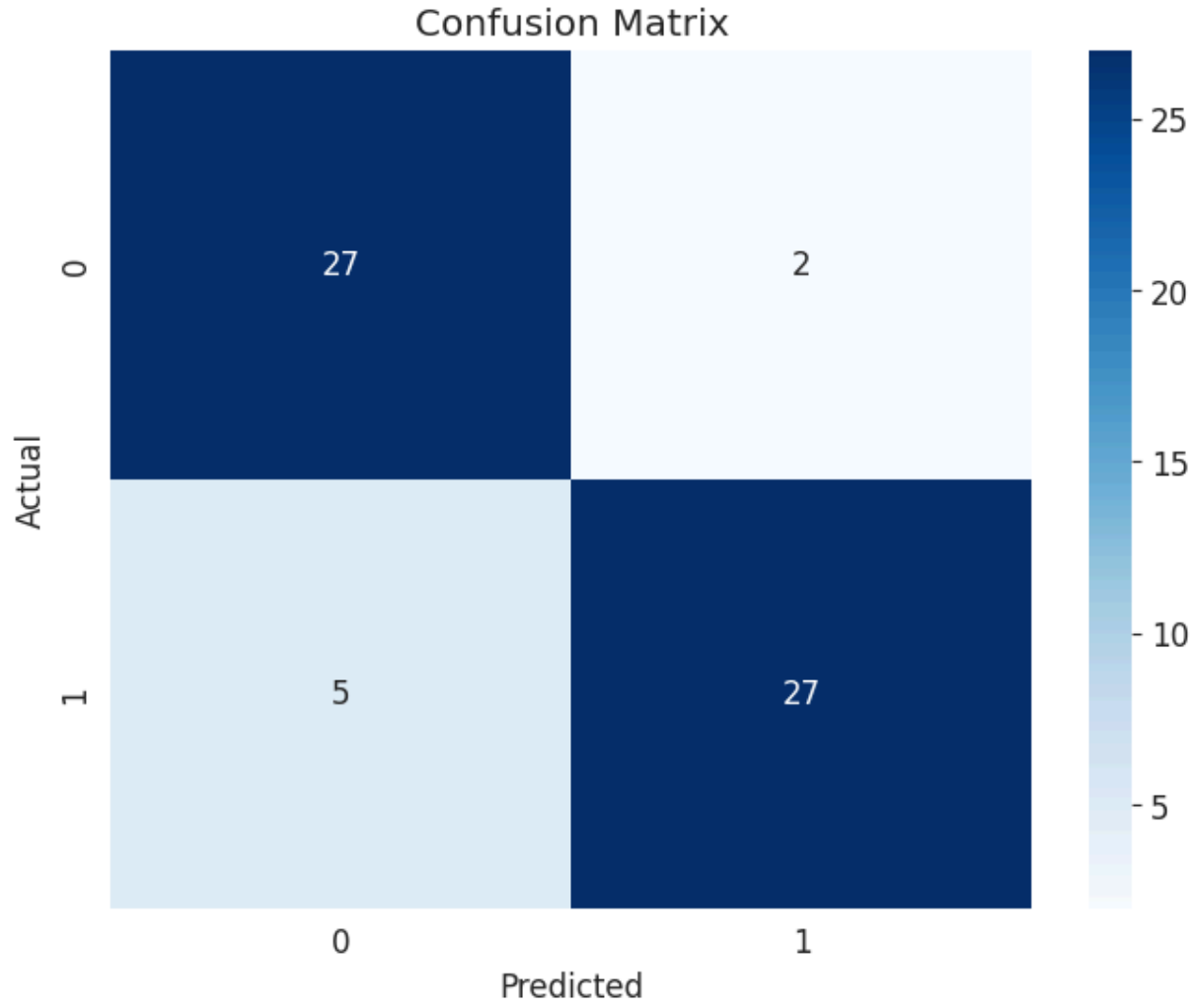
```
from sklearn.metrics import f1_score

f1 = f1_score(y_test, y_pred)
print("F1 Score:", f1)
```

```
F1 Score: 0.8852459016393442
```

```
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred)
print("Recall:", recall)
```

```
Recall: 0.84375
```

```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

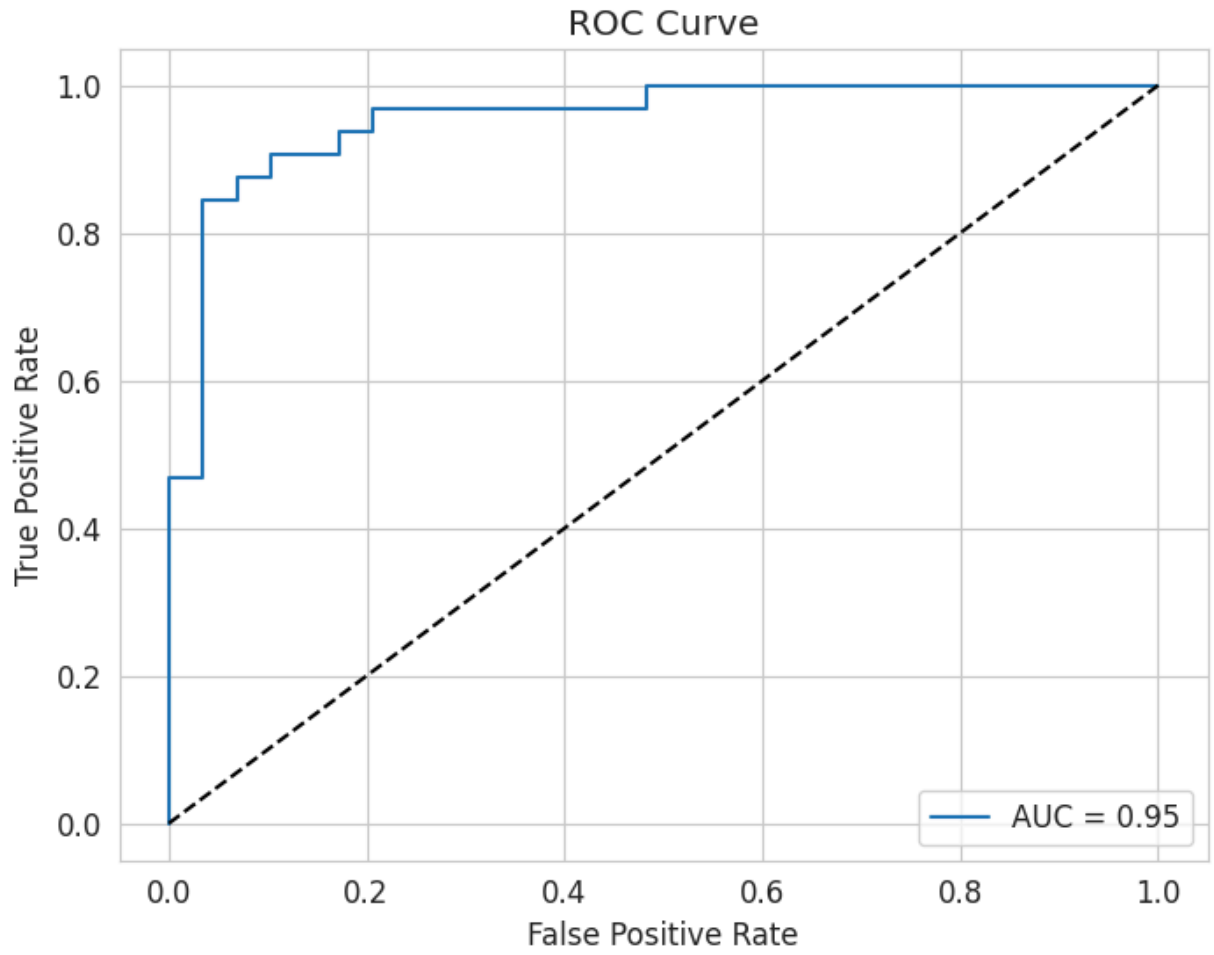



```
# Calculate sensitivity
sensitivity = cm[0, 0] / (cm[0, 0] + cm[1, 0])
print('Sensitivity:', sensitivity)
# Calculate specificity
specificity = cm[1, 1] / (cm[1, 1] + cm[0, 1])
print('Specificity:', specificity)
```

```
Sensitivity: 0.84375
Specificity: 0.9310344827586207
```

```
# ROC Curve and AUC
y_pred_proba = stacking_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
```

```
plt.show()
```




```
print(f'AUC: {roc_auc:.2f}')
```

```
AUC: 0.95
```

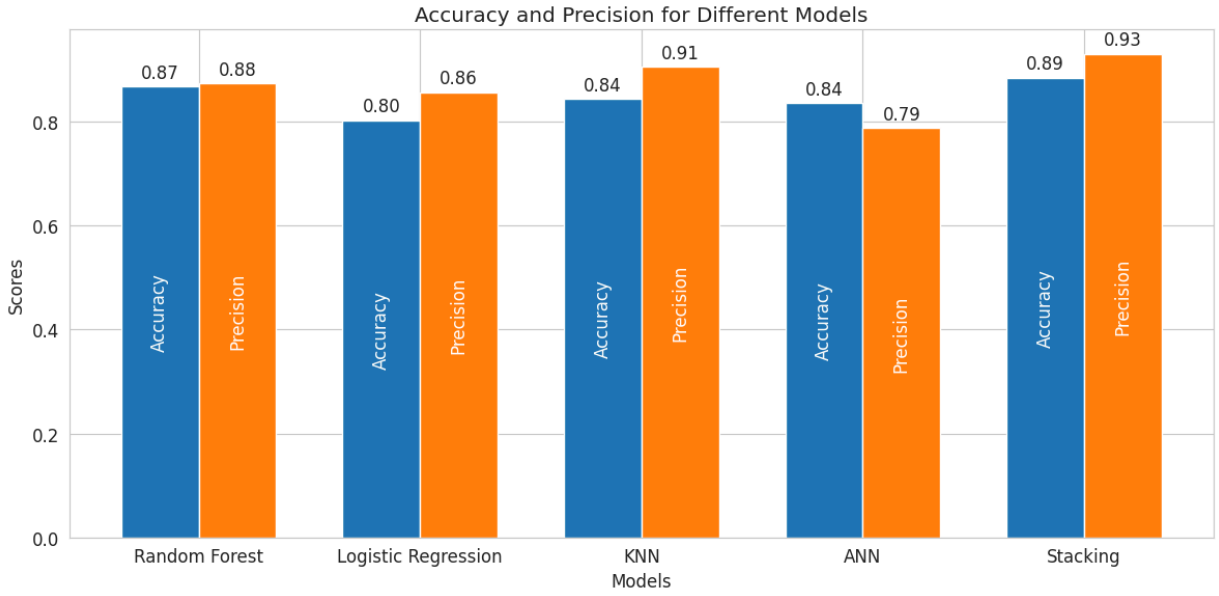
```
# Define the cross-validation strategy
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Define the metrics to evaluate
metrics = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']
results = {}

# Perform cross-validation for each metric
for metric in metrics:
  scores = cross_val_score(stacking_model, X, y, cv=cv, scoring=metric)
  results[metric] = scores

# Print the cross-validation results
for metric in metrics:
  print(f'{metric.capitalize()}: {results[metric].mean():.4f} (std: {res
```

```
Accuracy: 0.8450 (std: 0.0772)
Precision: 0.8457 (std: 0.0789)
Recall: 0.8909 (std: 0.0411)
F1: 0.8691 (std: 0.0570)
Roc_auc: 0.9041 (std: 0.0496)
```

Accuracy and precision values for each model

```
import matplotlib.pyplot as plt
import numpy as np
# Accuracy and precision values for each model (replace with your actual
accuracy_values = [accuracy_rf, accuracy_lr, accuracy_knn, accuracy_ann,
precision_values = [precision_score(y_test, y_pred_bin), precision_score

# Model names
model_names = ['Random Forest', 'Logistic Regression', 'KNN', 'ANN', 'St

# Create bar positions
bar_width = 0.35
index = np.arange(len(model_names))

# Create the bar graph
fig, ax = plt.subplots(figsize=(12, 6))
rects1 = ax.bar(index, accuracy_values, bar_width, label='Accuracy')
rects2 = ax.bar(index + bar_width, precision_values, bar_width, label='P

# Add labels, title, and legend
ax.set_xlabel('Models')
ax.set_ylabel('Scores')
ax.set_title('Accuracy and Precision for Different Models')
ax.set_xticks(index + bar_width / 2)
ax.set_xticklabels(model_names)
ax.legend()

# Remove the indicator box
ax.legend_.remove()

# Display values on top of bars horizontally and add text inside bars ve
def autolabel(rects, metric):
  for rect in rects:
    height = rect.get_height()
    ax.annotate(f'{height:.2f}',
                xy=(rect.get_x() + rect.get_width() / 2, height),
                xytext=(0, 3),  # 3 points vertical offset
                textcoords="offset points",
                ha='center', va='bottom')
    ax.text(rect.get_x() + rect.get_width() / 2, height / 2, metric, ha=

autolabel(rects1, 'Accuracy')
autolabel(rects2, 'Precision')

plt.tight_layout()
```

```
plt.show()
```



Testing the model

By giving a patient's features

```
##Stacking model testing
# new_sc = StandardScaler()
# X_train_scaled_for_stacking = new_sc.fit_transform(X_train) # This is

def get_user_input():
  """Gets user input for heart disease prediction features."""
  features = {
      "age": (20, 100),
      "sex": (0, 1),  # 0 for female, 1 for male
      "cp": (0, 3),  # Chest pain type (0-3)
      "trestbps": (90, 200),  # Resting blood pressure
      "chol": (100, 600),  # Serum cholesterol
      "fbs": (0, 1),  # Fasting blood sugar (0 or 1)
```

```
    "restecg": (0, 2),  # Resting electrocardiographic results (0-2)
    "thalach": (70, 200),  # Maximum heart rate achieved
    "exang": (0, 1),  # Exercise induced angina (0 or 1)
    "oldpeak": (0.0, 6.2),  # ST depression induced by exercise relati
    "slope": (0, 2),  # The slope of the peak exercise ST segment (0-2
    "ca": (0, 4),  # Number of major vessels (0, 1, 2, 3, or 4)
    "thal": (0, 3)  # Thalassemia (0, 1, 2, or 3)
}

user_input = {}
print("Please enter the patient's health information:")
for feature, (min_val, max_val) in features.items():
  while True:
    try:
      # For categorical features, ensure integer input and handle pote
      if feature in ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope',
          value = int(input(f"Enter {feature} ({min_val}-{max_val}): "
      else:
          value = float(input(f"Enter {feature} ({min_val}-{max_val}):

      if min_val <= value <= max_val:
        user_input[feature] = value
        break
      else:
        print(f"Invalid input for {feature}. Please enter a value with
    except ValueError:
      print(f"Invalid input for {feature}. Please enter a number.")

# Create a DataFrame from user input
user_df_initial = pd.DataFrame([user_input])
print("\n--- user_df after initial input ---")
display(user_df_initial)

# Separate numerical and categorical columns
numerical_cols = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
categorical_cols = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', '

user_df_numerical = user_df_initial[numerical_cols]

# Convert categorical columns to 'object' dtype for one-hot encoding
user_df_categorical = user_df_initial[categorical_cols].astype('object

# Apply one-hot encoding to the categorical features
user_df_categorical_encoded = pd.get_dummies(user_df_categorical, colu
print("\n--- user_df after one-hot encoding categorical ---")
display(user_df_categorical_encoded)

# Concatenate numerical and encoded categorical features
user_df_processed = pd.concat([user_df_numerical, user_df_categorical_
print("\n--- user_df after concatenating numerical and encoded categor
display(user_df_processed)


# Create a new DataFrame with all columns from X.columns and fill with
user_df = pd.DataFrame(0, index=[0], columns=X.columns)
```

```
    # Update the new DataFrame with values from the processed user input
    for col in user_df_processed.columns:
        if col in user_df.columns:
            user_df[col] = user_df_processed[col]

    # Calculate Risk Factor Composite Score for user input
    # Ensure the necessary columns exist before calculation
    if all(col in user_df.columns for col in ['age', 'trestbps', 'chol']):
        user_df['risk_factor_score'] = (user_df['age'] * 0.3 +
                                        user_df['trestbps'] * 0.3 +
                                        user_df['chol'] * 0.4)
    else:
        print("Warning: Could not calculate risk_factor_score. Required co
        user_df['risk_factor_score'] = 0 # Default value or handle appropr


    # Calculate Lifestyle-Adjusted Metric for user input
    # Check the exact column names after one-hot encoding
    user_df['lifestyle_metric'] = 0 # Initialize the metric

    if 'exercise_induced_angina_yes' in user_df.columns and user_df['exerc
        user_df['lifestyle_metric'] -= 0.5 # Assuming 'yes' is a negative

    # Check for both possible fasting blood sugar columns after one-hot en
    if 'fasting_blood_sugar_greater than 120mg/ml' in user_df.columns and
         user_df['lifestyle_metric'] -= 0.5 # Assuming 'greater than 120mg
    elif 'fasting_blood_sugar_lower than 120mg/ml' in user_df.columns and
         user_df['lifestyle_metric'] += 0.5 # Assuming 'lower than 120mg/m


    print("\n--- Columns of user_df before scaling ---")
    print(user_df.columns)
    print("\n--- Columns of X (training data) ---")
    print(X.columns)


    # Align the user input DataFrame columns with the training data column
    # This ensures the user input DataFrame has the same columns in the sa
    # user_df = user_df.reindex(columns=X.columns, fill_value=0) # Removed
    print("\n--- user_df before scaling ---") # Label updated
    display(user_df)


    # Scale the user input using the same scaler fitted on the stacking mo
    user_df_scaled = sc.transform(user_df)

    return user_df, user_input_scaled

user_df, user_input_scaled = get_user_input()

# Use the stacking model for prediction
prediction = stacking_model.predict(user_input_scaled)

if prediction > 0.5:
```

```
    print("The patient is likely to have heart disease.")
else:
    print("The patient is unlikely to have heart disease.")
```

```
Please enter the patient's health information:
Enter age (20-100): 40
Enter sex (0-1): 1
Enter cp (0-3): 1
Enter trestbps (90-200): 150
Enter chol (100-600): 180
Enter fbs (0-1): 1
Enter restecg (0-2): 1
Enter thalach (70-200): 170
Enter exang (0-1): 1
Enter oldpeak (0.0-6.2): 6
Enter slope (0-2): 1
Enter ca (0-4): 3
Enter thal (0-3): 2

--- user_df after initial input ---
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	sl
0	40.0	1	1	150.0	180.0	1	1	170.0	1	6.0	

```
--- user_df after one-hot encoding categorical ---
```

0

```
--- user_df after concatenating numerical and encoded categorical ---
```

	age	trestbps	chol	thalach	oldpeak
0	40.0	150.0	180.0	170.0	6.0

```
Warning: Could not calculate risk_factor_score. Required columns not fou

--- Columns of user_df before scaling ---
Index(['age', 'resting_blood_pressure', 'cholesterol',
       'max_heart_rate_achieved', 'st_depression', 'num_major_vessels',
       'sex_male', 'chest_pain_type_atypical angina',
       'chest_pain_type_non-anginal pain', 'chest_pain_type_typical angi
       'fasting_blood_sugar_lower than 120mg/ml',
       'rest_ecg_left ventricular hypertrophy', 'rest_ecg_normal',
       'exercise_induced_angina_yes', 'st_slope_flat', 'st_slope_upslopi
       'thalassemia_fixed defect', 'thalassemia_normal',
       'thalassemia_reversable defect', 'risk_factor_score',
       'lifestyle_metric'],
      dtype='object')

--- Columns of X (training data) ---
Index(['age', 'resting_blood_pressure', 'cholesterol',
       'max_heart_rate_achieved', 'st_depression', 'num_major_vessels',
       'sex_male', 'chest_pain_type_atypical angina',
       'chest_pain_type_non-anginal pain', 'chest_pain_type_typical angi
       'fasting_blood_sugar_lower than 120mg/ml',
       'rest_ecg_left ventricular hypertrophy', 'rest_ecg_normal',
       'exercise_induced_angina_yes', 'st_slope_flat', 'st_slope_upslopi
       'thalassemia_fixed defect', 'thalassemia_normal',
       'thalassemia_reversable defect', 'risk_factor_score',
       'lifestyle_metric'],
      dtype='object')
```

```
# Calculate Risk Factor Composite Score
# This is a simplified example, you might want to adjust the weights bas
```

```
dataset['risk_factor_score'] = (dataset['age'] * 0.3 +
                                dataset['resting_blood_pressure'] * 0.3
                                dataset['cholesterol'] * 0.4)

# Calculate Lifestyle-Adjusted Metric
# This is a simplified example, you might need to map categorical lifest
# Assuming 'exercise_induced_angina' and 'fasting_blood_sugar' are proxi
# You would typically need more direct measures of smoking, physical act
# Check the exact column names after one-hot encoding
if 'exercise_induced_angina_yes' in dataset.columns and 'fasting_blood_s
    dataset['lifestyle_metric'] = (dataset['exercise_induced_angina_yes'
                                   dataset['fasting_blood_sugar_greater
elif 'exercise_induced_angina_yes' in dataset.columns and 'fasting_blood
     dataset['lifestyle_metric'] = (dataset['exercise_induced_angina_yes
                                   dataset['fasting_blood_sugar_lower t
else:
    # Handle the case where neither column exists, perhaps by adding the
    print("Warning: Neither 'fasting_blood_sugar_greater than 120mg/ml'
    dataset['lifestyle_metric'] = 0 # Default value

# Select important features to display
# Start with a base list of important features
important_features_base = ['age', 'sex_male', 'chest_pain_type_typical a
                       'cholesterol', 'max_heart_rate_achieved',
                       'exercise_induced_angina_yes', 'st_depression', 'n
                       'risk_factor_score', 'lifestyle_metric', 'target']

# Dynamically add the relevant fasting blood sugar column if it exists
if 'fasting_blood_sugar_greater than 120mg/ml' in dataset.columns:
    important_features_base.insert(important_features_base.index('max_he
elif 'fasting_blood_sugar_lower than 120mg/ml' in dataset.columns:
    important_features_base.insert(important_features_base.index('max_he

# Dynamically add the relevant thalassemia columns if they exist
if 'thalassemia_fixed defect' in dataset.columns:
    important_features_base.insert(important_features_base.index('risk_f
```