```python
# Common EDA and plotting libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Allowing plots to appear in the notebook
%matplotlib inline

## Models
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

## Model evaluators
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import precision_score, recall_score, f1_score
```

EDA

```python
df = pd.read_csv("heart-disease.csv")
df.shape
```

```
(303, 14)
```

```python
# Let's check the top 5 rows of our dataframe
df.head()
```

```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak
slope  \
0   63    1   3       145   233    1        0      150      0      2.3
0
1   37    1   2       130   250    0        1      187      0      3.5
0
2   41    0   1       130   204    0        0      172      0      1.4
2
3   56    1   1       120   236    0        1      178      0      0.8
2
4   57    0   0       120   354    0        1      163      1      0.6
2

   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
```

```python
# The top 10 rows
df.head(10)
```

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

   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
5   0     1       1
6   0     2       1
7   0     3       1
8   0     3       1
9   0     2       1
```

```python
#  The bottom 10 Rows
df.tail(10)
```

```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang
oldpeak  \
293   67    1   2       152   212    0        0      150      0
0.8
294   44    1   0       120   169    0        1      144      1
2.8
295   63    1   0       140   187    0        0      144      1
4.0
296   63    0   0       124   197    0        1      136      1
0.0
```

```
297   59    1   0       164   176    1         0        90       0
1.0
298   57    0   0       140   241    0         1       123       1
0.2
299   45    1   3       110   264    0         1       132       0
1.2
300   68    1   0       144   193    1         1       141       0
3.4
301   57    1   0       130   131    0         1       115       1
1.2
302   57    0   1       130   236    0         0       174       0
0.0

     slope  ca  thal  target
293      1   0     3       0
294      0   0     1       0
295      2   2     3       0
296      1   0     2       0
297      1   2     1       0
298      1   0     3       0
299      1   0     3       0
300      1   2     3       0
301      1   1     3       0
302      1   1     2       0
```

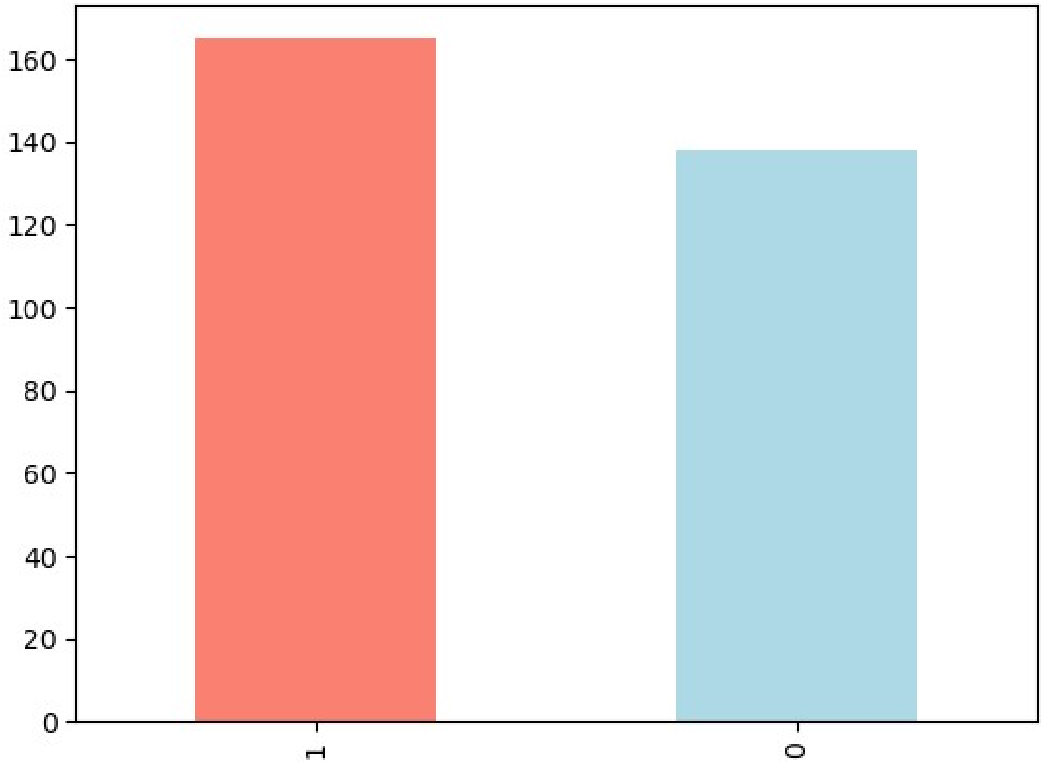
```
# Let's see how many positive (1) and negative (0) samples we have in
our dataframe
df.target.value_counts()
```

```
1    165
0    138
Name: target, dtype: int64
```

```
# Normalized value counts
df.target.value_counts(normalize=True)
```

```
1    0.544554
0    0.455446
Name: target, dtype: float64
```

```
# Plot the value counts with a bar graph
df.target.value_counts().plot(kind="bar", color=["salmon",
"lightblue"]);
```




```
# Dataframe Summary
df.info()
df.describe()
```

```
<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
```

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

          restecg     thalach       exang     oldpeak       slope  
ca  \
count  303.000000  303.000000  303.000000  303.000000  303.000000  
303.000000   
mean     0.528053  149.646865    0.326733    1.039604    1.399340    
0.729373   
std      0.525860   22.905161    0.469794    1.161075    0.616226    
1.022606   
min      0.000000   71.000000    0.000000    0.000000    0.000000    
0.000000   
25%      0.000000  133.500000    0.000000    0.000000    1.000000    
0.000000   
50%      1.000000  153.000000    0.000000    0.800000    1.000000    
0.000000   
75%      1.000000  166.000000    1.000000    1.600000    2.000000    
1.000000   
max      2.000000  202.000000    1.000000    6.200000    2.000000    
4.000000   

             thal      target  
count  303.000000  303.000000  
mean     2.313531    0.544554  
std      0.612277    0.498835  
min      0.000000    0.000000  
25%      2.000000    0.000000  
50%      2.000000    1.000000  
75%      3.000000    1.000000  
max      3.000000    1.000000  
```

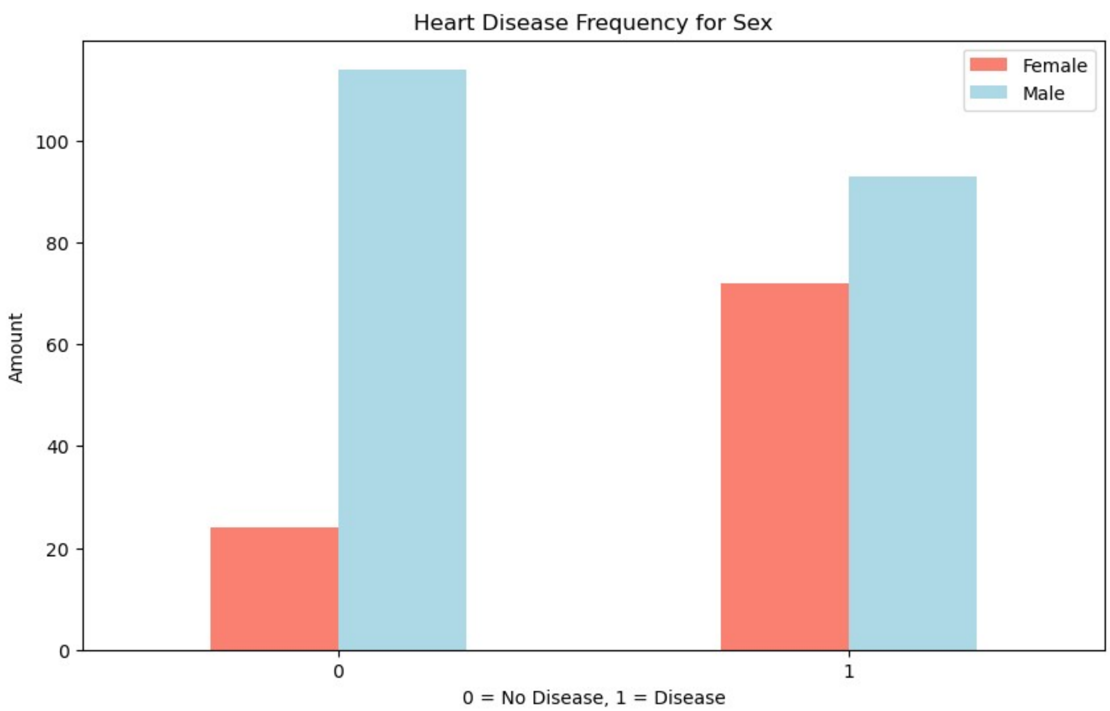
```python
# Gender(Sex) variable counts
df.sex.value_counts()
```

```
1    207
0     96
Name: sex, dtype: int64
```

```python
# Compare target column with sex column
pd.crosstab(df.target, df.sex)
```

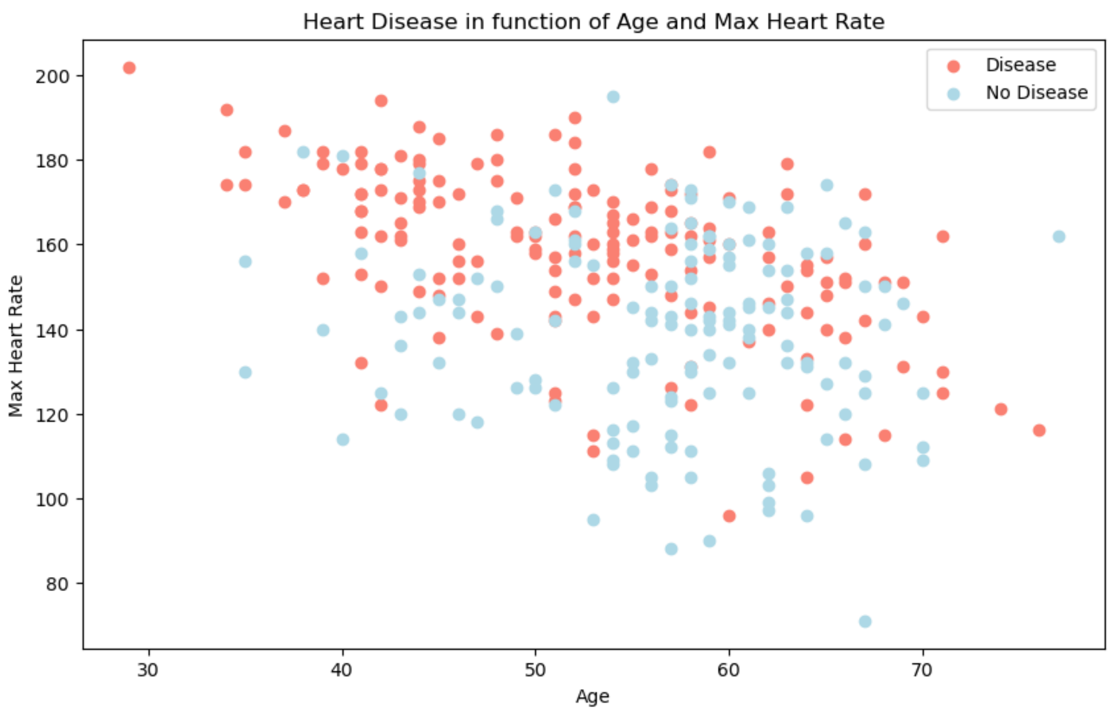
```
sex      0    1
target
0       24  114
1       72   93
```

```python
# Sex vs Target Plot
pd.crosstab(df.target, df.sex).plot(kind="bar", figsize=(10,6),
color=["salmon", "lightblue"])

# Plot attributes
plt.title("Heart Disease Frequency for Sex")
plt.xlabel("0 = No Disease, 1 = Disease")
plt.ylabel("Amount")
plt.legend(["Female", "Male"])
plt.xticks(rotation=0); # keep the labels on the x-axis vertical
```

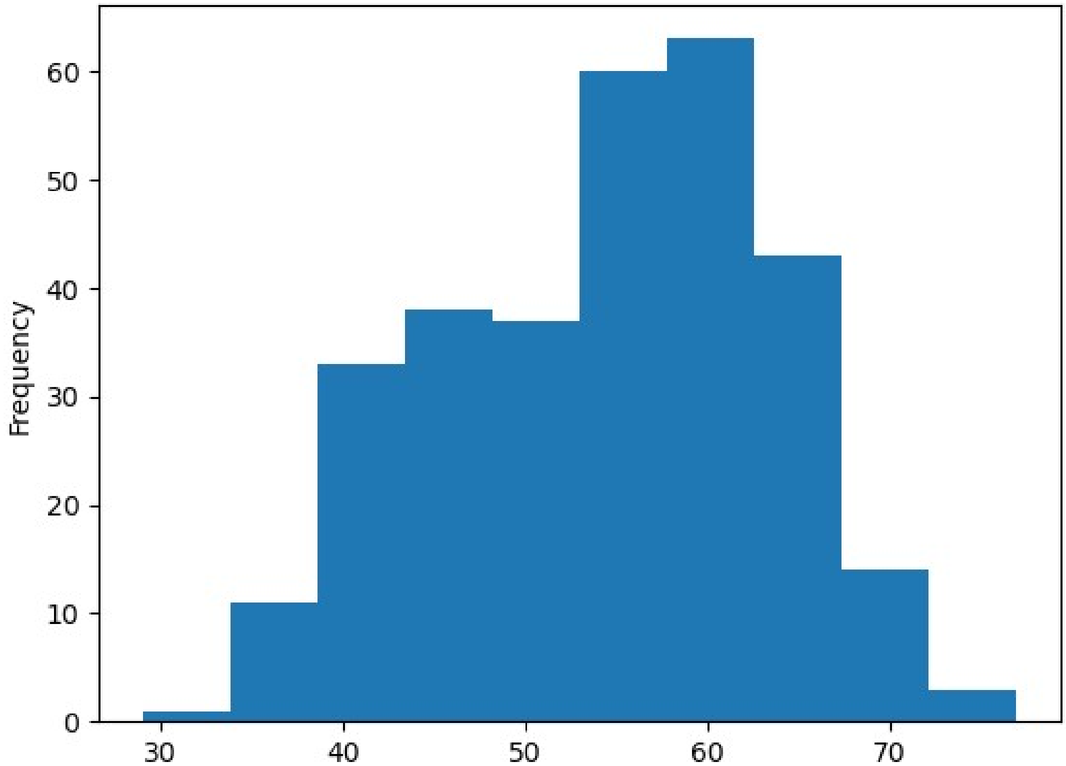



```python
# Age Vs Max Heart Rate Vs Target Plot
plt.figure(figsize=(10,6))

# Positve examples
plt.scatter(df.age[df.target==1],
            df.thalach[df.target==1],
```

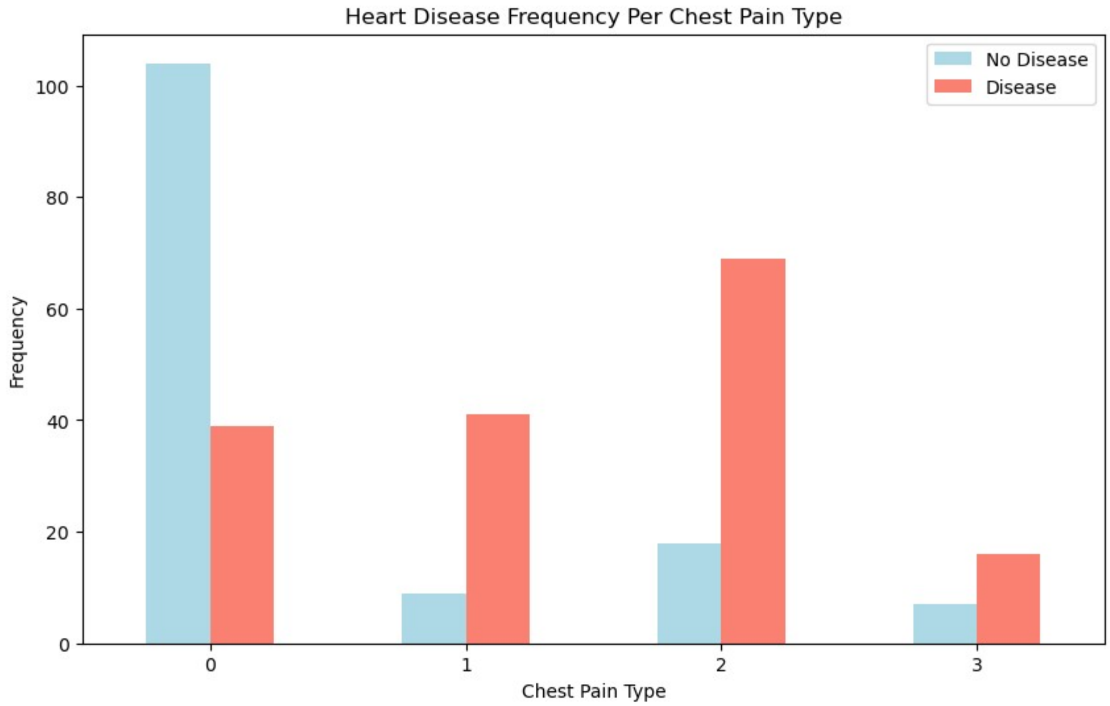
```
            c="salmon") # define it as a scatter figure

# Negative examples
plt.scatter(df.age[df.target==0],
            df.thalach[df.target==0],
            c="lightblue") # axis always come as (x, y)

# Plot attributes
plt.title("Heart Disease in function of Age and Max Heart Rate")
plt.xlabel("Age")
plt.legend(["Disease", "No Disease"])
plt.ylabel("Max Heart Rate");
```




```
# Age Histograms
df.age.plot.hist();
```




```python
# Chest pain(cp) vs Target
pd.crosstab(df.cp, df.target)

# Create a new crosstab and base plot
pd.crosstab(df.cp, df.target).plot(kind="bar",
                                   figsize=(10,6),
                                   color=["lightblue", "salmon"])

# Plot attributes
plt.title("Heart Disease Frequency Per Chest Pain Type")
plt.xlabel("Chest Pain Type")
plt.ylabel("Frequency")
plt.legend(["No Disease", "Disease"])
plt.xticks(rotation = 0);
```



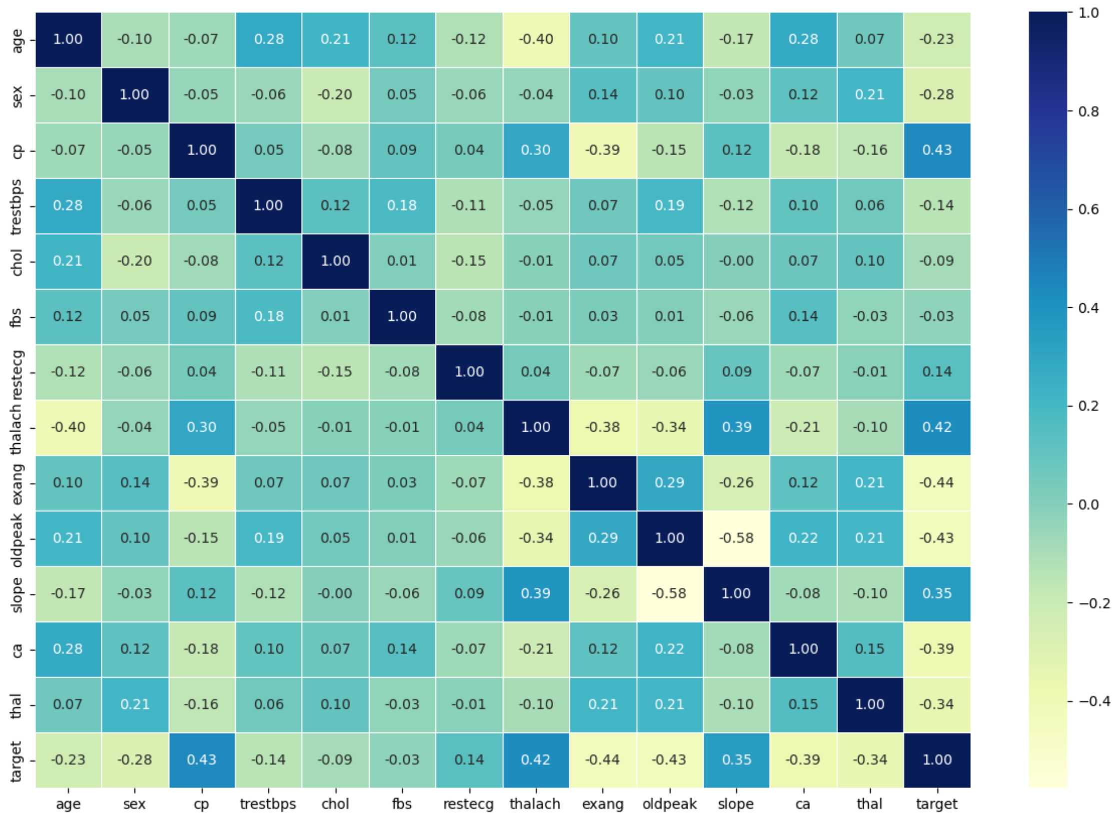

```python
# Find the correlation between each pair variables
corr_matrix = df.corr()
corr_matrix
```

```
               age       sex        cp  trestbps      chol
fbs  \
age       1.000000 -0.098447 -0.068653  0.279351  0.213678  0.121308

sex      -0.098447  1.000000 -0.049353 -0.056769 -0.197912  0.045032

cp       -0.068653 -0.049353  1.000000  0.047608 -0.076904  0.094444

trestbps  0.279351 -0.056769  0.047608  1.000000  0.123174  0.177531

chol      0.213678 -0.197912 -0.076904  0.123174  1.000000  0.013294

fbs       0.121308  0.045032  0.094444  0.177531  0.013294  1.000000

restecg  -0.116211 -0.058196  0.044421 -0.114103 -0.151040 -0.084189

thalach  -0.398522 -0.044020  0.295762 -0.046698 -0.009940 -0.008567

exang     0.096801  0.141664 -0.394280  0.067616  0.067023  0.025665

oldpeak   0.210013  0.096093 -0.149230  0.193216  0.053952  0.005747

slope    -0.168814 -0.030711  0.119717 -0.121475 -0.004038 -0.059894
```

```
ca        0.276326  0.118261 -0.181053  0.101389  0.070511  0.137979

thal      0.068001  0.210041 -0.161736  0.062210  0.098803 -0.032019

target   -0.225439 -0.280937  0.433798 -0.144931 -0.085239 -0.028046


           restecg   thalach     exang   oldpeak     slope
ca  \
age      -0.116211 -0.398522  0.096801  0.210013 -0.168814  0.276326

sex      -0.058196 -0.044020  0.141664  0.096093 -0.030711  0.118261

cp        0.044421  0.295762 -0.394280 -0.149230  0.119717 -0.181053

trestbps -0.114103 -0.046698  0.067616  0.193216 -0.121475  0.101389

chol     -0.151040 -0.009940  0.067023  0.053952 -0.004038  0.070511

fbs      -0.084189 -0.008567  0.025665  0.005747 -0.059894  0.137979

restecg   1.000000  0.044123 -0.070733 -0.058770  0.093045 -0.072042

thalach   0.044123  1.000000 -0.378812 -0.344187  0.386784 -0.213177

exang    -0.070733 -0.378812  1.000000  0.288223 -0.257748  0.115739

oldpeak  -0.058770 -0.344187  0.288223  1.000000 -0.577537  0.222682

slope     0.093045  0.386784 -0.257748 -0.577537  1.000000 -0.080155

ca       -0.072042 -0.213177  0.115739  0.222682 -0.080155  1.000000

thal     -0.011981 -0.096439  0.206754  0.210244 -0.104764  0.151832

target    0.137230  0.421741 -0.436757 -0.430696  0.345877 -0.391724


              thal    target
age       0.068001 -0.225439
sex       0.210041 -0.280937
cp       -0.161736  0.433798
trestbps  0.062210 -0.144931
chol      0.098803 -0.085239
fbs      -0.032019 -0.028046
restecg  -0.011981  0.137230
thalach  -0.096439  0.421741
exang     0.206754 -0.436757
```

```
oldpeak   0.210244 -0.430696
slope    -0.104764  0.345877
ca        0.151832 -0.391724
thal      1.000000 -0.344029
target   -0.344029  1.000000
```

```
# Correlation Heatmap
corr_matrix = df.corr()
plt.figure(figsize=(15, 10))
sns.heatmap(corr_matrix,
            annot=True,
            linewidths=0.5,
            fmt= ".2f",
            cmap="YlGnBu");
```




```
df = pd.read_csv("heart-disease.csv") # 'DataFrame' shortened to 'df'
df.shape # (rows, columns)
```

```
(303, 14)
```

Modelling

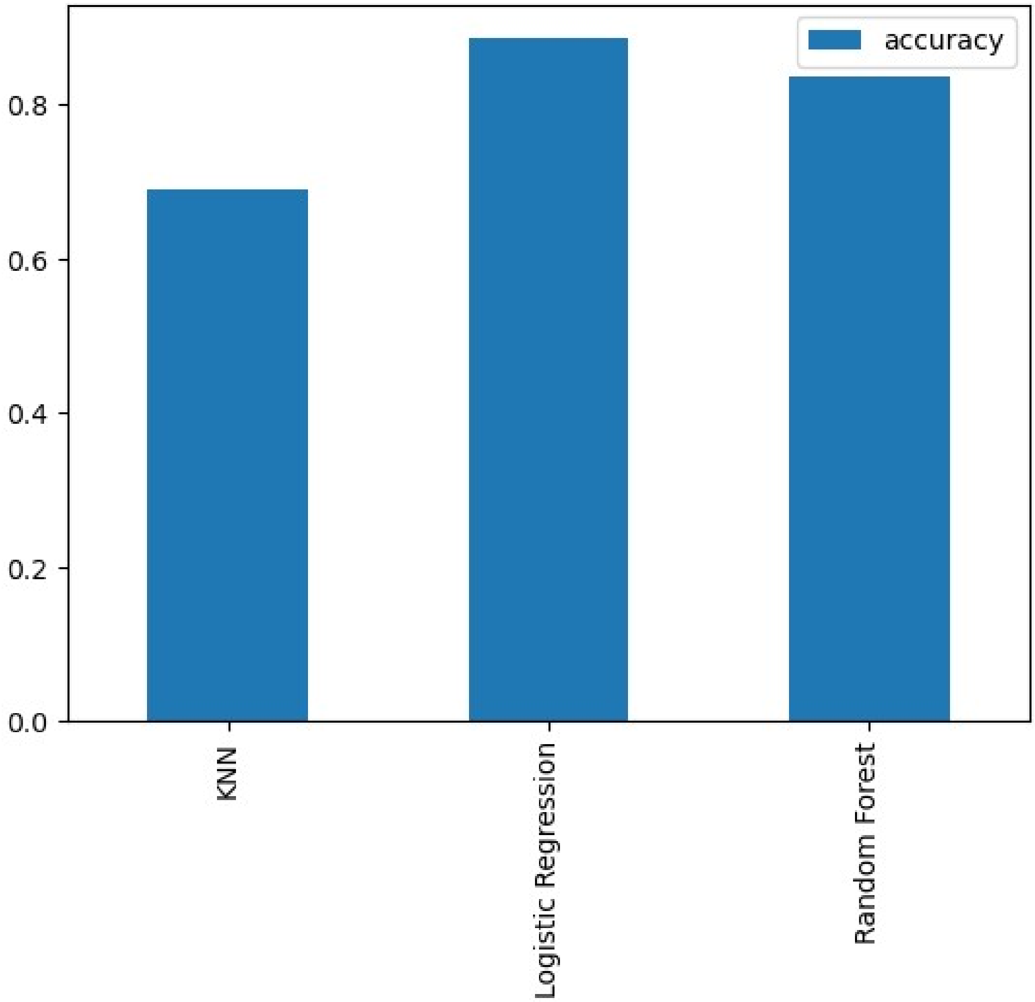
```
# Dataset processing
# Everything except target variable
X = df.drop("target", axis=1)
```

```
# Target variable
y = df.target.values

# Random seed for reproducibility
np.random.seed(42)

# Split into train & test set
X_train, X_test, y_train, y_test = train_test_split(X, # independent
variables
                                                    y, # dependent
variable
                                                    test_size = 0.2) #
percentage of data to use for test set

# Put models in a dictionary
models = {"KNN": KNeighborsClassifier(),
          "Logistic Regression": LogisticRegression(),
          "Random Forest": RandomForestClassifier()}

# Create function to fit and score models
def fit_and_score(models, X_train, X_test, y_train, y_test):
    """
    Fits and evaluates given machine learning models.
    models : a dict of different Scikit-Learn machine learning models
    X_train : training data
    X_test : testing data
    y_train : labels assosciated with training data
    y_test : labels assosciated with test data
    """
    # Random seed for reproducible results
    np.random.seed(42)
    # Make a list to keep model scores
    model_scores = {}
    # Loop through models
    for name, model in models.items():
        # Fit the model to the data
        model.fit(X_train, y_train)
        # Evaluate the model and append its score to model_scores
        model_scores[name] = model.score(X_test, y_test)
    return model_scores

# Model Score Comparision
model_scores = fit_and_score(models=models,
                             X_train=X_train,
                             X_test=X_test,
                             y_train=y_train,
                             y_test=y_test)
model_scores
```

```
C:\Users\zhizh\anaconda3\lib\site-packages\sklearn\linear_model\
_logistic.py:458: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
{'KNN': 0.6885245901639344,
 'Logistic Regression': 0.8852459016393442,
 'Random Forest': 0.8360655737704918}
```

```python
# Model Score Comparision Plot
model_compare = pd.DataFrame(model_scores, index=['accuracy'])
model_compare.T.plot.bar();
```



Logistic Regression have the best result

```
#Try improving the KNN model to produce better result
# Create a list of train scores
train_scores = []

# Create a list of test scores
test_scores = []

# Create a list of different values for n_neighbors
neighbors = range(1, 21) # 1 to 20

# Setup algorithm
knn = KNeighborsClassifier()

# Loop through different neighbors values
for i in neighbors:
    knn.set_params(n_neighbors = i) # set neighbors value

    # Fit the algorithm
```

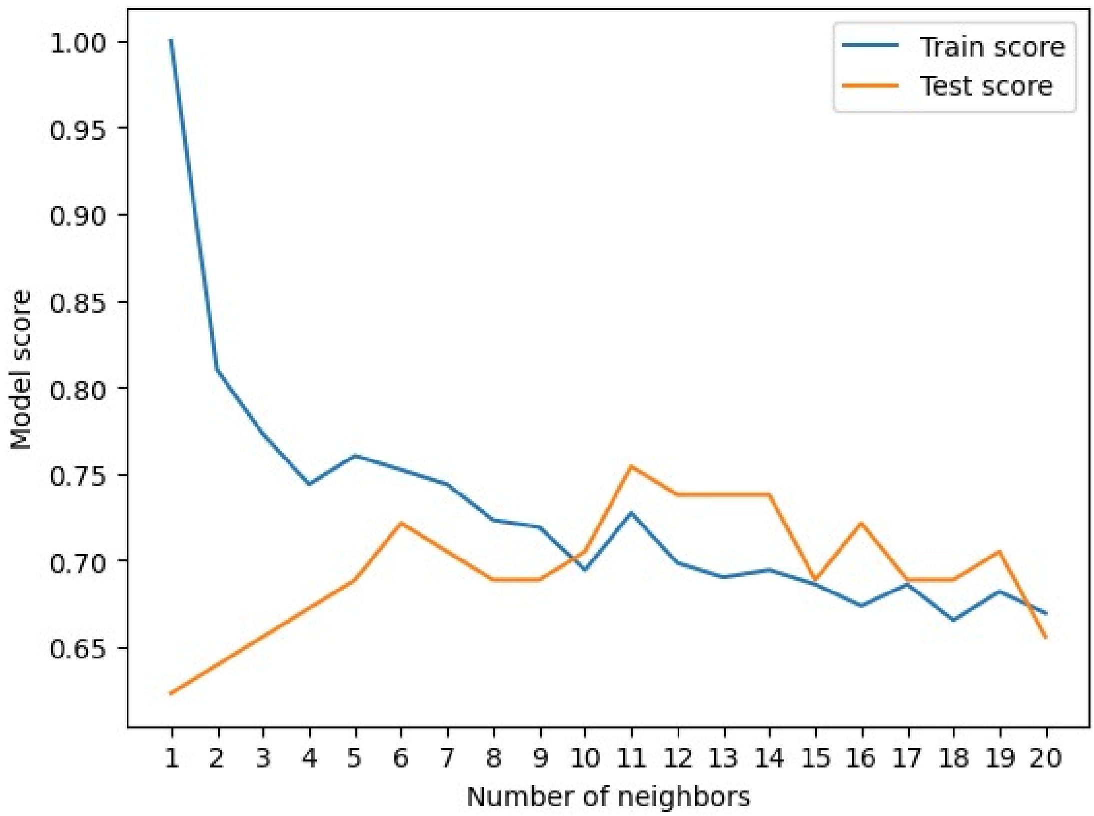
```python
    knn.fit(X_train, y_train)

    # Update the training scores
    train_scores.append(knn.score(X_train, y_train))

    # Update the test scores
    test_scores.append(knn.score(X_test, y_test))
```

```python
train_scores
```

```
[1.0,
 0.8099173553719008,
 0.7727272727272727,
 0.743801652892562,
 0.7603305785123967,
 0.7520661157024794,
 0.743801652892562,
 0.7231404958677686,
 0.71900826446281,
 0.6942148760330579,
 0.7272727272727273,
 0.6983471074380165,
 0.6900826446280992,
 0.6942148760330579,
 0.6859504132231405,
 0.6735537190082644,
 0.6859504132231405,
 0.6652892561983471,
 0.6818181818181818,
 0.6694214876033058]
```

```python
# Train Scores vs Test Scores Plot
plt.plot(neighbors, train_scores, label="Train score")
plt.plot(neighbors, test_scores, label="Test score")
plt.xticks(np.arange(1, 21, 1))
plt.xlabel("Number of neighbors")
plt.ylabel("Model score")
plt.legend()

print(f"Maximum KNN score on the test data: {max(test_scores)*100:.2f}
%")
```

```
Maximum KNN score on the test data: 75.41%
```




```python
# Tuning models with with RandomizedSearchCV
# Different LogisticRegression hyperparameters
log_reg_grid = {"C": np.logspace(-4, 4, 20),
                "solver": ["liblinear"]}

# Different RandomForestClassifier hyperparameters
rf_grid = {"n_estimators": np.arange(10, 1000, 50),
           "max_depth": [None, 3, 5, 10],
           "min_samples_split": np.arange(2, 20, 2),
           "min_samples_leaf": np.arange(1, 20, 2)}

# Setup random seed
np.random.seed(42)

# Setup random hyperparameter search for LogisticRegression
rs_log_reg = RandomizedSearchCV(LogisticRegression(),
                                param_distributions=log_reg_grid,
                                cv=5,
                                n_iter=20,
                                verbose=True)

# Fit random hyperparameter search model
rs_log_reg.fit(X_train, y_train);
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

```
rs_log_reg.best_params_
```

```
{'solver': 'liblinear', 'C': 0.23357214690901212}
```

```
rs_log_reg.score(X_test, y_test)
```

```
0.8852459016393442
```

```
# Setup random seed
np.random.seed(42)

# Setup random hyperparameter search for RandomForestClassifier
rs_rf = RandomizedSearchCV(RandomForestClassifier(),
                           param_distributions=rf_grid,
                           cv=5,
                           n_iter=20,
                           verbose=True)

# Fit random hyperparameter search model
rs_rf.fit(X_train, y_train);
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

```
# Find the best parameters
rs_rf.best_params_
```

```
{'n_estimators': 210,
 'min_samples_split': 4,
 'min_samples_leaf': 19,
 'max_depth': 3}
```

```
# Evaluate the randomized search random forest model
rs_rf.score(X_test, y_test)
```

```
0.8688524590163934
```

```
# Different LogisticRegression hyperparameters
log_reg_grid = {"C": np.logspace(-4, 4, 20),
                "solver": ["liblinear"]}

# Setup grid hyperparameter search for LogisticRegression
gs_log_reg = GridSearchCV(LogisticRegression(),
                          param_grid=log_reg_grid,
                          cv=5,
                          verbose=True)

# Fit grid hyperparameter search model
gs_log_reg.fit(X_train, y_train);
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

```python
# Check the best parameters
gs_log_reg.best_params_
```

```
{'C': 0.23357214690901212, 'solver': 'liblinear'}
```

```python
# Evaluate the model
gs_log_reg.score(X_test, y_test)
```

```
0.8852459016393442
```

```python
# Make preidctions on test data
y_preds = gs_log_reg.predict(X_test)
```

```python
y_preds
```

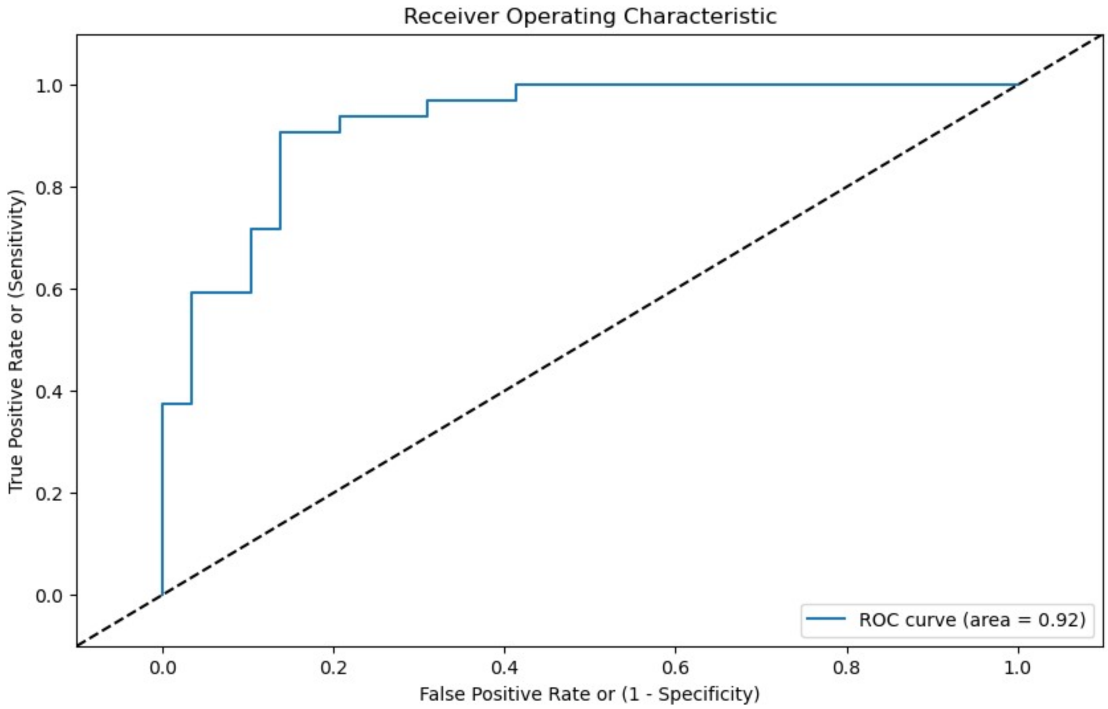
```
array([0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1,
0,
       0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
1,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0],
dtype=int64)
```

```python
y_test
```

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

```python
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Predict the probabilities of the positive class
y_probs = gs_log_reg.predict_proba(X_test)[:, 1]

# Compute the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_probs)

# Compute the AUC
auc = roc_auc_score(y_test, y_probs)

# Plot the ROC curve
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' % auc)
plt.plot([-.1, 1.1], [-.1, 1.1], 'k--')  # random predictions curve
plt.xlim([-0.1, 1.1])
plt.ylim([-0.1, 1.1])
plt.xlabel('False Positive Rate or (1 - Specificity)')
plt.ylabel('True Positive Rate or (Sensitivity)')
plt.title('Receiver Operating Characteristic')
```

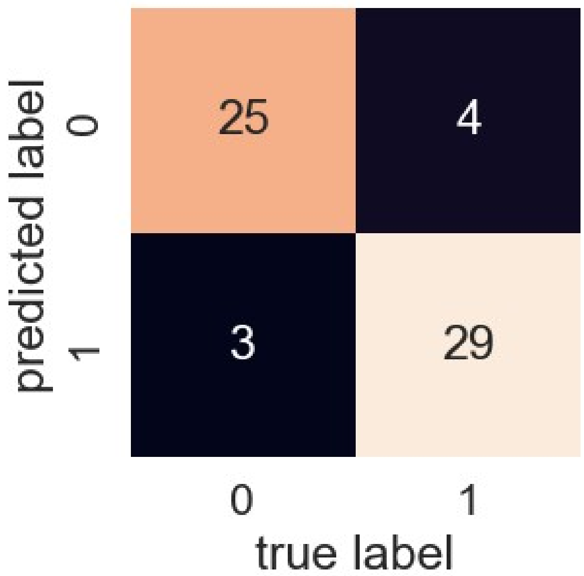
```
plt.legend(loc="lower right")
plt.show()
```




```
# Display confusion matrix
print(confusion_matrix(y_test, y_preds))
```

```
[[25  4]
 [ 3 29]]
```

```
# Import Seaborn
import seaborn as sns
sns.set(font_scale=1.5) # Increase font size

def plot_conf_mat(y_test, y_preds):
    """
    Plots a confusion matrix using Seaborn's heatmap().
    """
    fig, ax = plt.subplots(figsize=(3, 3))
    ax = sns.heatmap(confusion_matrix(y_test, y_preds),
                     annot=True, # Annotate the boxes
                     cbar=False)
    plt.xlabel("true label")
    plt.ylabel("predicted label")

plot_conf_mat(y_test, y_preds)
```




```
# Show classification report
print(classification_report(y_test, y_preds))

              precision    recall  f1-score   support

           0       0.89      0.86      0.88        29
           1       0.88      0.91      0.89        32

    accuracy                           0.89        61
   macro avg       0.89      0.88      0.88        61
weighted avg       0.89      0.89      0.89        61


# Check best hyperparameters
gs_log_reg.best_params_

{'C': 0.23357214690901212, 'solver': 'liblinear'}

# Import cross_val_score
from sklearn.model_selection import cross_val_score

# Instantiate best model with best hyperparameters (found with
GridSearchCV)
clf = LogisticRegression(C=0.23357214690901212,
                         solver="liblinear")

# Cross-validated accuracy score
cv_acc = cross_val_score(clf,
                         X,
                         y,
                         cv=5, # 5-fold cross-validation
```

```
                         scoring="accuracy") # accuracy as scoring
cv_acc
```

```
array([0.81967213, 0.90163934, 0.8852459 , 0.88333333, 0.75      ])
```

```
cv_acc = np.mean(cv_acc)
cv_acc
```

```
0.8479781420765027
```

```
# Cross-validated precision score
cv_precision = np.mean(cross_val_score(clf,
                                       X,
                                       y,
                                       cv=5, # 5-fold cross-validation
                                       scoring="precision")) #
precision as scoring
cv_precision
```

```
0.8215873015873015
```

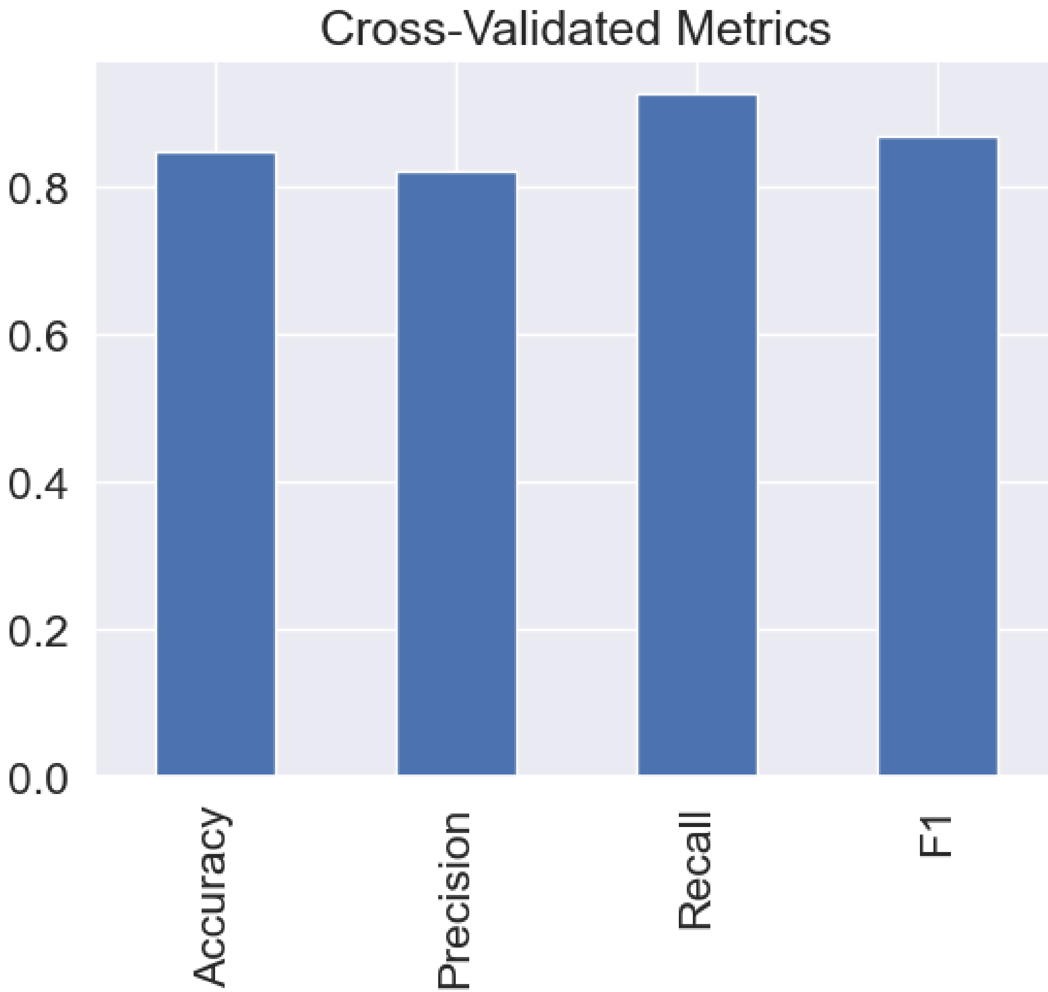
```
# Cross-validated recall score
cv_recall = np.mean(cross_val_score(clf,
                                    X,
                                    y,
                                    cv=5, # 5-fold cross-validation
                                    scoring="recall")) # recall as
scoring
cv_recall
```

```
0.9272727272727274
```

```
# Cross-validated F1 score
cv_f1 = np.mean(cross_val_score(clf,
                                X,
                                y,
                                cv=5, # 5-fold cross-validation
                                scoring="f1")) # f1 as scoring
cv_f1
```

```
0.8705403543192143
```

```
# Visualizing cross-validated metrics
cv_metrics = pd.DataFrame({"Accuracy": cv_acc,
                            "Precision": cv_precision,
                            "Recall": cv_recall,
                            "F1": cv_f1},
                          index=[0])
cv_metrics.T.plot.bar(title="Cross-Validated Metrics", legend=False);
```




```
# Fit an instance of LogisticRegression (taken from above)
clf.fit(X_train, y_train);

# Check coef_
clf.coef_
```

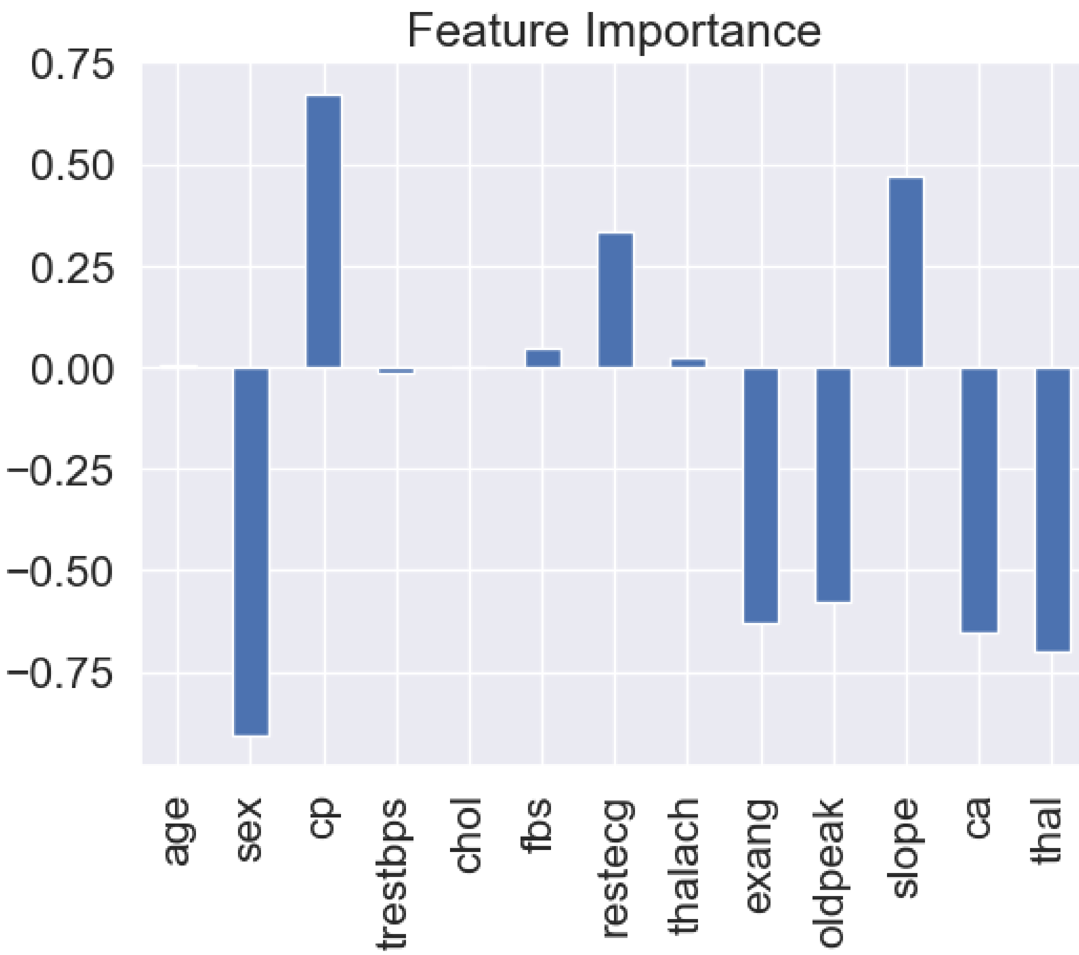
```
array([[ 0.00369922, -0.90424089,  0.67472826, -0.0116134 , -
0.00170364,
         0.04787688,  0.33490195,  0.02472938, -0.63120405, -
0.57590942,
         0.47095136, -0.65165348, -0.69984206]])
```

```
# Match features to columns
features_dict = dict(zip(df.columns, list(clf.coef_[0])))
features_dict
```

```
{'age': 0.0036992203516641483,
 'sex': -0.9042408930260735,
 'cp': 0.6747282624694215,
 'trestbps': -0.011613401789010375,
 'chol': -0.0017036441780094993,
```

```
 'fbs': 0.047876883382302414,
 'restecg': 0.3349019539205334,
 'thalach': 0.024729383396378347,
 'exang': -0.6312040510578483,
 'oldpeak': -0.5759094230155162,
 'slope': 0.47095135616471195,
 'ca': -0.6516534832909596,
 'thal': -0.6998420628111434}
```

```
# Visualize feature importance
features_df = pd.DataFrame(features_dict, index=[0])
features_df.T.plot.bar(title="Feature Importance", legend=False);
```




```
pd.crosstab(df["sex"], df["target"])
```

```
target    0   1
sex
0        24  72
1       114  93
```

```
# Contrast slope (positive coefficient) with target
pd.crosstab(df["slope"], df["target"])
```

target	0	1
slope		
0	12	9
1	91	49
2	35	107