Predicting heart disease using machine learning

Objective

Given details of patients, can we predict whether or not they have heart disease?

Data

Data is collected from : https://archive.ics.uci.edu/ml/datasets/heart+Disease

Features

There are following features on our data:

Create data dictionary

1. **age** - age in years
2. **sex** - (1 = male; 0 = female)
3. **cp** - chest pain type
   - 0: Typical angina: chest pain related decrease blood supply to the heart
   - 1: Atypical angina: chest pain not related to heart
   - 2: Non-anginal pain: typically esophageal spasms (non heart related)
   - 3: Asymptomatic: chest pain not showing signs of disease
4. **trestbps** - resting blood pressure (in mm Hg on admission to the hospital) anything above 130-140 is typically cause for concern
5. **chol** - serum cholestoral in mg/dl
   - serum = LDL + HDL + .2 * triglycerides
   - above 200 is cause for concern
6. **fbs** - (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
   - '>126' mg/dL signals diabetes
7. **restecg** - resting electrocardiographic results
   - 0: Nothing to note
   - 1: ST-T Wave abnormality
     - can range from mild symptoms to severe problems
     - signals non-normal heart beat
   - 2: Possible or definite left ventricular hypertrophy
     - Enlarged heart's main pumping chamber
8. **thalach** - maximum heart rate achieved
9. **exang** - exercise induced angina (1 = yes; 0 = no)
10. **oldpeak** - ST depression induced by exercise relative to rest looks at stress of heart during excercise unhealthy heart will stress more
11. **slope** - the slope of the peak exercise ST segment
    - 0: Upsloping: better heart rate with excercise (uncommon)

– 1: Flatsloping: minimal change (typical healthy heart)
– 2: Downslopins: signs of unhealthy heart

12. **ca** - number of major vessels (0-3) colored by flourosopy
    – colored vessel means the doctor can see the blood passing through
    – the more blood movement the better (no clots)
13. **thal** - thalium stress result
    – 1,3: normal
    – 6: fixed defect: used to be defect but ok now
    – 7: reversable defect: no proper blood movement when excercising
14. **target** - have disease or not (1=yes, 0=no) (= the predicted attribute)

Import packages

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

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

from sklearn.model_selection import train_test_split, cross_val_score,
RandomizedSearchCV, GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import RocCurveDisplay, ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
```

Import Data

```
file_path = '/kaggle/input/heart-disease-dataset/heart.csv'
df = pd.read_csv(file_path)
df.head()
```

```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak
slope  \
0   52    1   0       125   212    0        1      168      0      1.0
2
1   53    1   0       140   203    1        0      155      1      3.1
0
2   70    1   0       145   174    0        1      125      1      2.6
0
3   61    1   0       148   203    0        1      161      0      0.0
2
4   62    0   0       138   294    1        1      106      0      1.9
1
```

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

Exploratory Data Analysis

```
df.shape
```

```
(1025, 14)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1025 non-null   int64
 1   sex       1025 non-null   int64
 2   cp        1025 non-null   int64
 3   trestbps  1025 non-null   int64
 4   chol      1025 non-null   int64
 5   fbs       1025 non-null   int64
 6   restecg   1025 non-null   int64
 7   thalach   1025 non-null   int64
 8   exang     1025 non-null   int64
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64
 11  ca        1025 non-null   int64
 12  thal      1025 non-null   int64
 13  target    1025 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

Are there any missing values?

```
df.isna().sum()
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
```

```
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

Inference : No missing value

```
df.describe()
```

```
                 age          sex           cp     trestbps         chol  
\
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.00000   

mean     54.434146     0.695610     0.942439   131.611707   246.00000   

std       9.072290     0.460373     1.029641    17.516718    51.59251   

min      29.000000     0.000000     0.000000    94.000000   126.00000   

25%      48.000000     0.000000     0.000000   120.000000   211.00000   

50%      56.000000     1.000000     1.000000   130.000000   240.00000   

75%      61.000000     1.000000     2.000000   140.000000   275.00000   

max      77.000000     1.000000     3.000000   200.000000   564.00000   


                 fbs      restecg      thalach        exang      oldpeak  
\
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.000000   

mean      0.149268     0.529756   149.114146     0.336585     1.071512   

std       0.356527     0.527878    23.005724     0.472772     1.175053   

min       0.000000     0.000000    71.000000     0.000000     0.000000   

25%       0.000000     0.000000   132.000000     0.000000     0.000000   

50%       0.000000     1.000000   152.000000     0.000000     0.800000   

75%       0.000000     1.000000   166.000000     1.000000     1.800000   

max       1.000000     2.000000   202.000000     1.000000     6.200000   


               slope           ca         thal       target  
count  1025.000000  1025.000000  1025.000000  1025.000000  
```

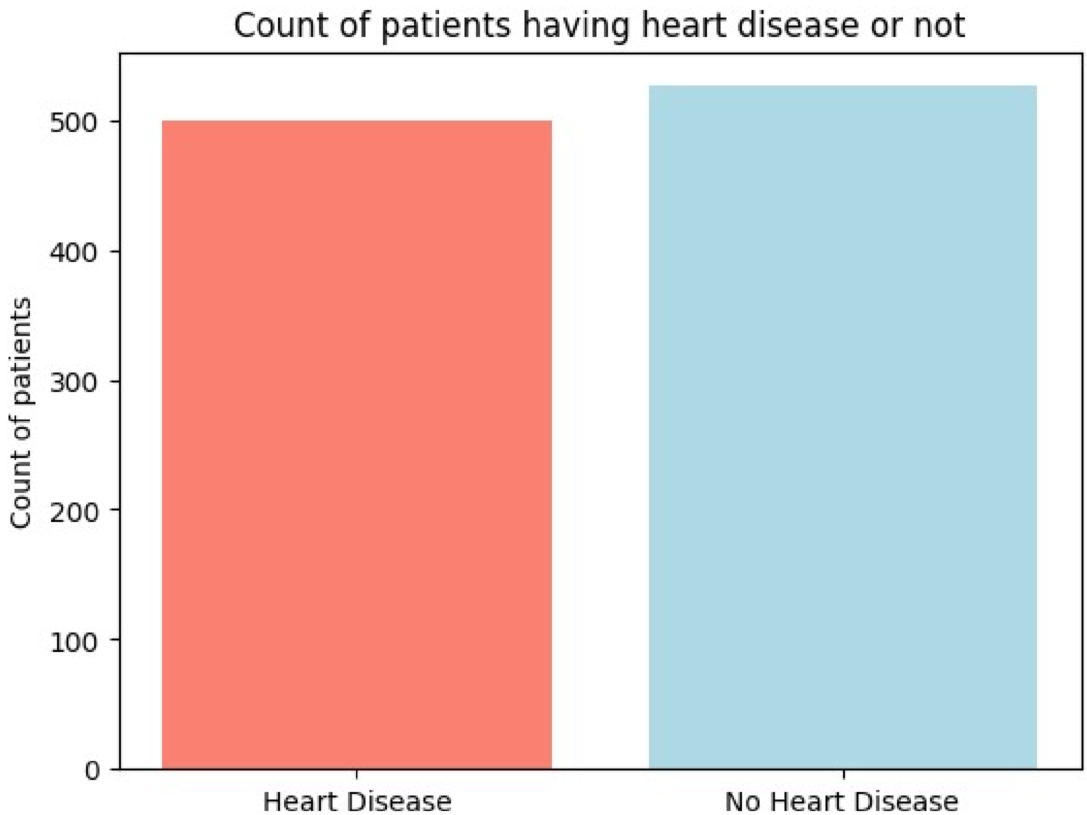
```
mean       1.385366     0.754146     2.323902     0.513171
std        0.617755     1.030798     0.620660     0.500070
min        0.000000     0.000000     0.000000     0.000000
25%        1.000000     0.000000     2.000000     0.000000
50%        1.000000     0.000000     2.000000     1.000000
75%        2.000000     1.000000     3.000000     1.000000
max        2.000000     4.000000     3.000000     1.000000
```

Count of having heart disease or not

```
df['target'].value_counts()
```

```
target
1    526
0    499
Name: count, dtype: int64
```

```
fig, ax = plt.subplots()

x = ["Heart Disease", "No Heart Disease"]
counts = [df['target'].value_counts()[0], df['target'].value_counts()
[1]]

ax.bar(x, counts, color=['salmon', 'lightblue'])
ax.set_title("Count of patients having heart disease or not")
ax.set_ylabel("Count of patients");
```



Count of Heart Disease sample = 165 count of No Heart Disease sample = 138

Inference : Dataset is balanced as number of samples of each class are roughly equal.

Heart disease counts according to sex

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

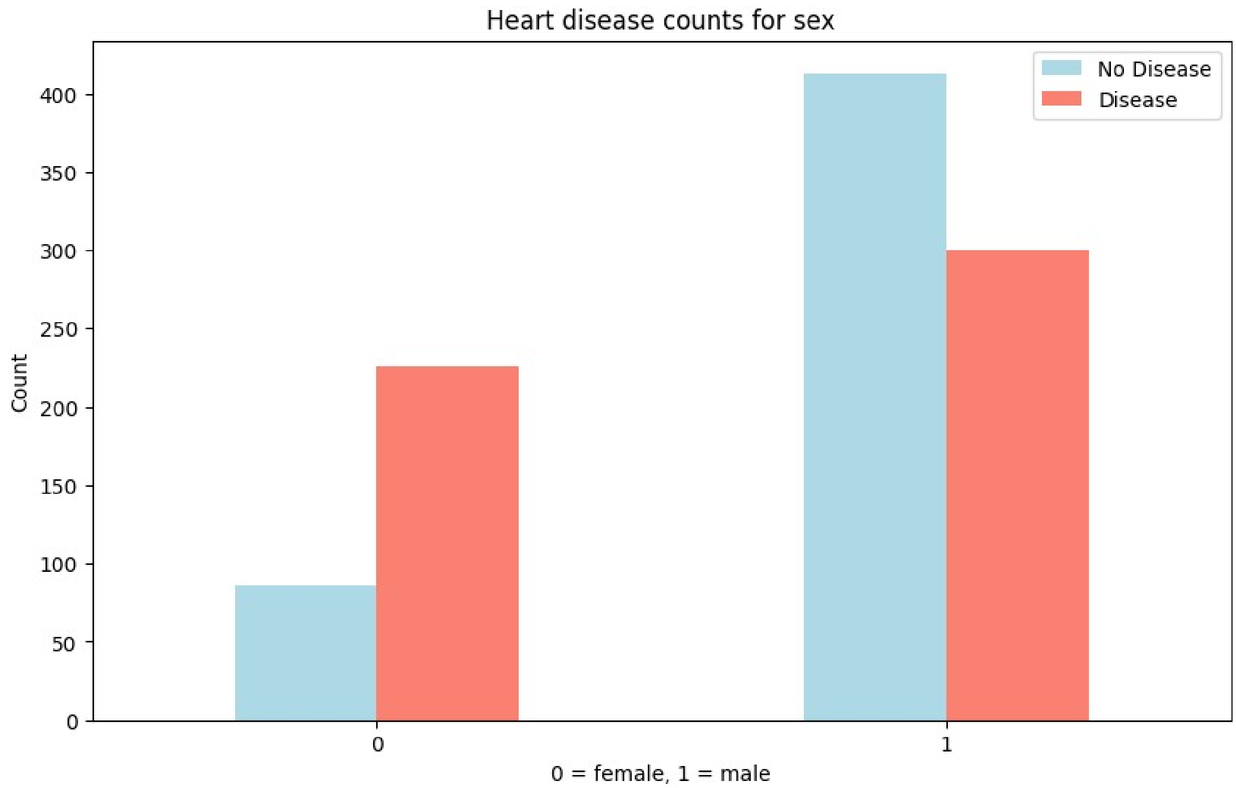
target	0	1
sex		
0	86	226
1	413	300

Inference : Above table shows that 75% chance of heart disease in female and 45% chance of heart disease in male.

We can deduce the same conclusion by drawing graph.

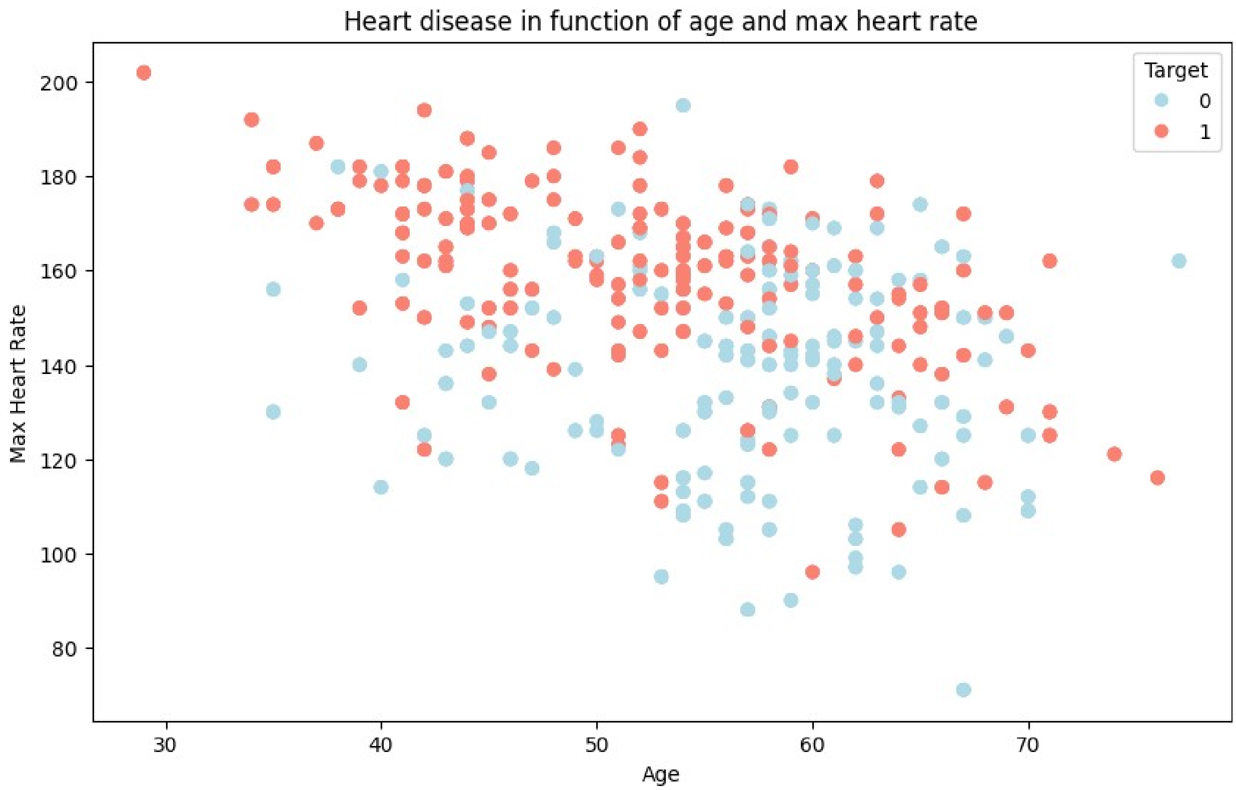
```
pd.crosstab(df['sex'], df['target']).plot(kind='bar',
color=['lightblue', 'salmon'], figsize=(10,6))

plt.title("Heart disease counts for sex")
plt.xlabel('0 = female, 1 = male')
```

```
plt.ylabel('Count')
plt.legend(['No Disease', 'Disease'])
plt.xticks(rotation=0);
```



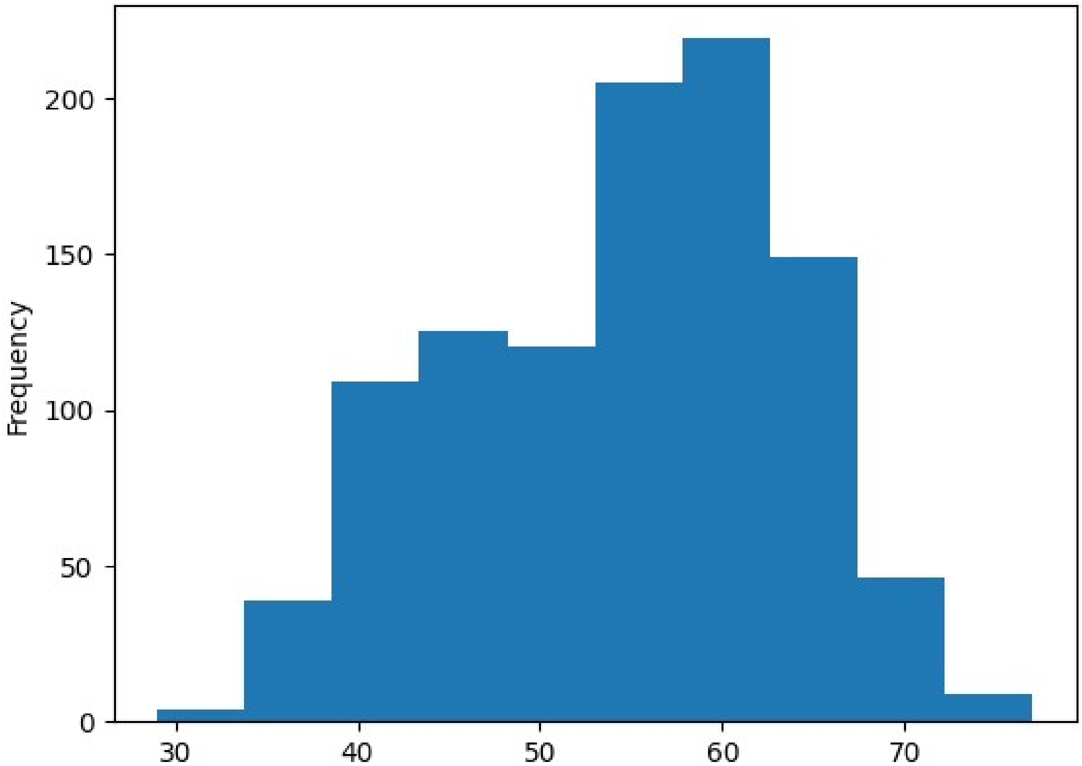
Age vs. Max heart Rate for Heart Disease

```
import matplotlib.colors as mcolors
fig, ax = plt.subplots(figsize=(10,6))

cmap = mcolors.ListedColormap(['lightblue', 'salmon'])
scatter = ax.scatter(df['age'], df['thalach'], c=df['target'],
cmap=cmap)
ax.set_title("Heart disease in function of age and max heart rate")
ax.set_xlabel("Age")
ax.set_ylabel("Max Heart Rate")
ax.legend(*scatter.legend_elements(), title="Target");
```



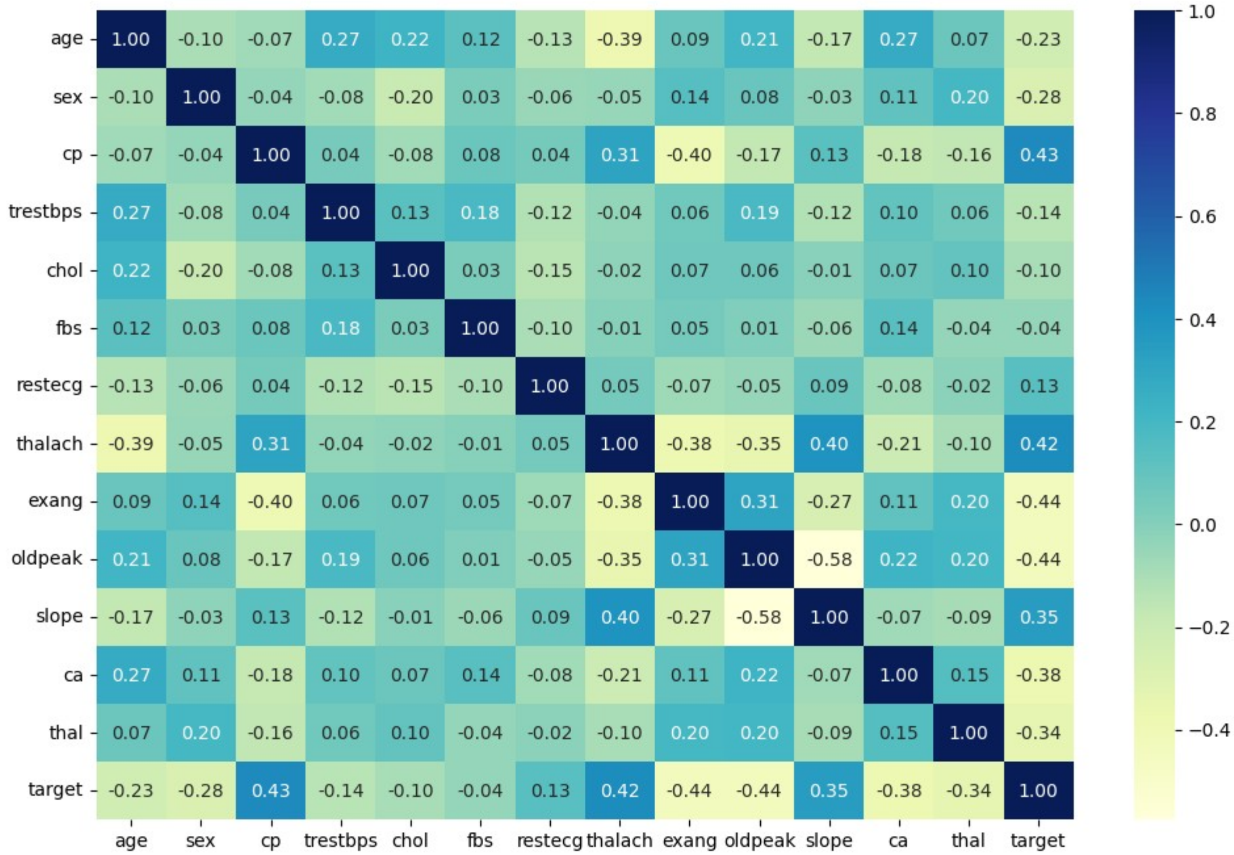
Inference : In younger age groups, heart disease is more commonly found in individuals with higher heart rates, while in older age groups, heart disease is prevalent across all heart rates.

```
# Distribution of age using histogram.
df['age'].plot(kind='hist');
```



Correlation

```
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(numeric_only=True), annot=True, fmt='.2f',
cmap="YlGnBu");
```



Inference : There is no such strong correlation.

Modelling

```
df.head()

   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak
slope  \
0   52    1   0       125   212    0        1      168      0      1.0
2
1   53    1   0       140   203    1        0      155      1      3.1
0
2   70    1   0       145   174    0        1      125      1      2.6
0
3   61    1   0       148   203    0        1      161      0      0.0
2
4   62    0   0       138   294    1        1      106      0      1.9
1

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

```
3   1     3       0
4   3     2       0

# Split the data into X and y
X = df.drop('target', axis = 1)
y = df['target']

X

      age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang
oldpeak  \
0      52    1   0       125   212    0        1      168      0
1.0
1      53    1   0       140   203    1        0      155      1
3.1
2      70    1   0       145   174    0        1      125      1
2.6
3      61    1   0       148   203    0        1      161      0
0.0
4      62    0   0       138   294    1        1      106      0
1.9
...   ...  ...  ..       ...   ...  ...      ...      ...    ...
...
1020   59    1   1       140   221    0        1      164      1
0.0
1021   60    1   0       125   258    0        0      141      1
2.8
1022   47    1   0       110   275    0        0      118      1
1.0
1023   50    0   0       110   254    0        0      159      0
0.0
1024   54    1   0       120   188    0        1      113      0
1.4

      slope  ca  thal
0         2   2     3
1         0   0     3
2         0   0     3
3         2   1     3
4         1   3     2
...     ...  ..   ...
1020      2   0     2
1021      1   1     3
1022      1   1     2
1023      2   0     2
1024      1   1     3

[1025 rows x 13 columns]

y
```

```
0       0
1       0
2       0
3       0
4       0
       ..
1020    1
1021    0
1022    0
1023    1
1024    0
Name: target, Length: 1025, dtype: int64
```

```
np.random.seed(42)

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)

X_train
```

```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang
oldpeak  \
835   49    1   2       118   149    0        0      126      0
0.8
137   64    0   0       180   325    0        1      154      1
0.0
534   54    0   2       108   267    0        0      167      0
0.0
495   59    1   0       135   234    0        1      161      0
0.5
244   51    1   2       125   245    1        0      166      0
2.4
..   ...  ...  ..       ...   ...  ...      ...      ...    ...       .
..
700   41    1   2       130   214    0        0      168      0
2.0
71    61    1   0       140   207    0        0      138      1
1.9
106   51    1   0       140   299    0        1      173      1
1.6
270   43    1   0       110   211    0        1      161      0
0.0
860   52    1   0       112   230    0        1      160      0
0.0

     slope  ca  thal
835      2   3     2
137      2   0     2
534      2   0     2
495      1   0     3
244      1   0     2
```

```
..      ...  ..   ...
700       1   0     2
71        2   1     3
106       2   0     3
270       2   0     3
860       2   1     2

[820 rows x 13 columns]
```

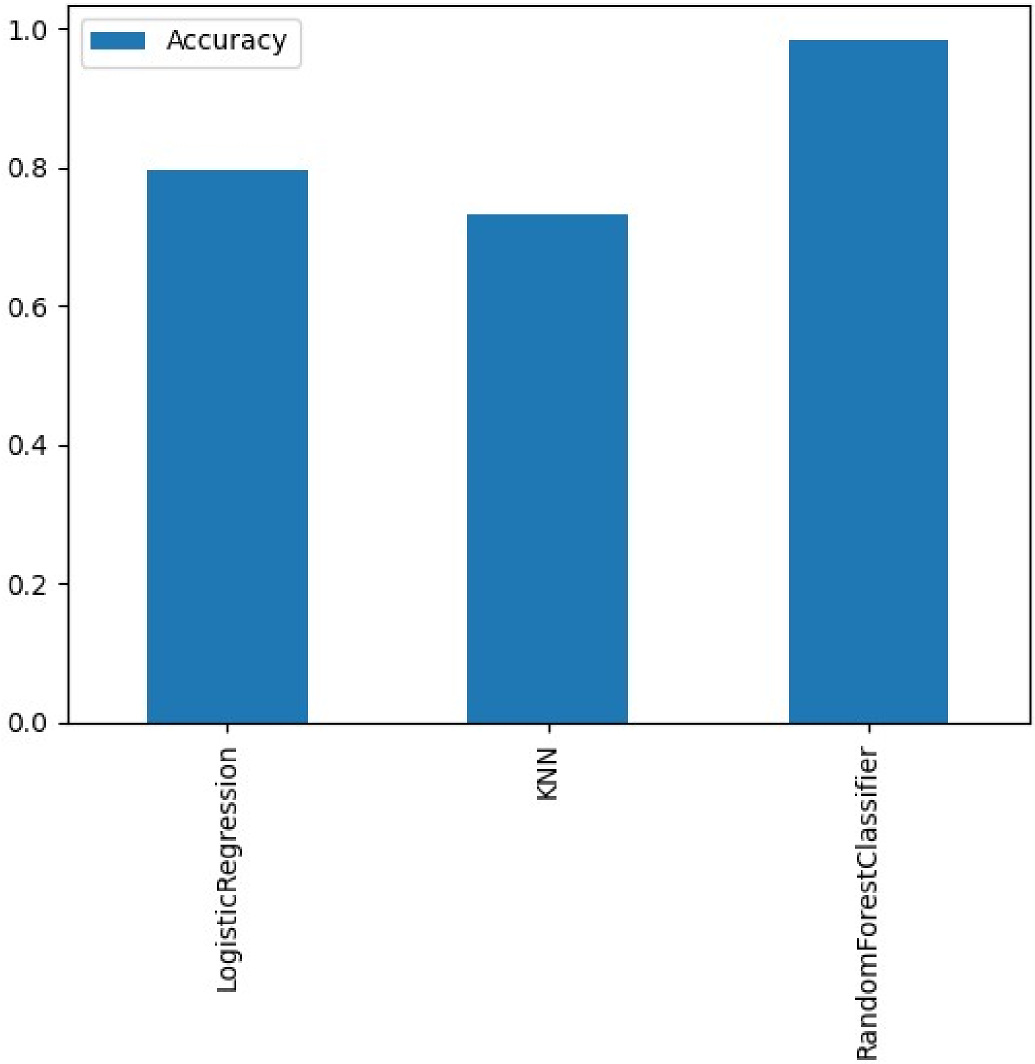
As we know this problem is of classification type. Hence we will use these three models:

- Logistic Regression
- K-Nearest Neighbours Classifier
- Random Forest Classifier

```
def fit_and_score(models, X_train, X_test, y_train, y_test):
  """
    Fit and train models simultaneously
  """
  models_score = {}

  for key, values in models.items():
    model = values
    model.fit(X_train, y_train)
    models_score[key] = model.score(X_test, y_test)

  return models_score

models = {
    "LogisticRegression": LogisticRegression(max_iter=1000),
    "KNN": KNeighborsClassifier(),
    "RandomForestClassifier": RandomForestClassifier()
}

models_score = fit_and_score(models, X_train, X_test, y_train, y_test)
models_score

{'LogisticRegression': 0.7951219512195122,
 'KNN': 0.7317073170731707,
 'RandomForestClassifier': 0.9853658536585366}
```

Model Comparision

```
model_compare = pd.DataFrame(models_score, index=["Accuracy"])
model_compare.T.plot(kind='bar')

<Axes: >
```



Inference : Here we can see that LogisticRegression and RandomForestClassifier are doing well as compared to KNN.

Hence, we will move forward with LogisticRegression and RandomForestClassifier models. First create a function to store the different score metrics for different model for future use.

```
def pred_metrics(model, X_test, y_true):

  # accuracy = cross_val_score(model, X, y,
scoring='accuracy').mean()
  # precision = cross_val_score(model, X, y,
scoring='precision').mean()
  # recall = cross_val_score(model, X, y, scoring='recall').mean()
  # f1 = cross_val_score(model, X, y, scoring='f1').mean()
  y_preds = model.predict(X_test)
  accuracy = accuracy_score(y_true, y_preds)
  precision = precision_score(y_true, y_preds)
```

```
  recall = recall_score(y_true, y_preds)
  f1 = f1_score(y_true, y_preds)

  metric_dict = {
      "Accuracy": round(accuracy,8),
      "Precision": round(precision,8),
      "Recall": round(recall,8),
      "F1": round(f1,8),
  }

  return metric_dict
```

Base Logisitc Regression performance

```
lr_base_model_metrics = pred_metrics(models['LogisticRegression'],
X_test, y_test)
lr_base_model_metrics
```

```
{'Accuracy': 0.79512195,
 'Precision': 0.75630252,
 'Recall': 0.87378641,
 'F1': 0.81081081}
```

Base Random Forest Classifier performance

```
rfc_base_model_metrics =
pred_metrics(models['RandomForestClassifier'], X_test, y_test)
rfc_base_model_metrics
```

```
{'Accuracy': 0.98536585,
 'Precision': 1.0,
 'Recall': 0.97087379,
 'F1': 0.98522167}
```

Hyperparameter Tuning

Hyperparamter Tuning using RandomizedSearchCV

```
# Hyperparameter grid for LogisiticRegression
lr_rs_grid = {
    "solver" : ["liblinear", 'lbfgs', 'newton-cg'],
    "C" : np.logspace(-4, 4, num=20, base=10),
    "max_iter" : [1000, 1500, 2000]
}

# Hyperparameter grid for RandomForestClassifier
rfc_rs_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)
}
```

Logistic Regression

```
lr_rs_model = RandomizedSearchCV(models['LogisticRegression'],
lr_rs_grid, cv=5, n_iter=20)
lr_rs_model.fit(X_train, y_train)
```

```
/opt/conda/lib/python3.10/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(
```

```
RandomizedSearchCV(cv=5, estimator=LogisticRegression(max_iter=1000),
n_iter=20,
                   param_distributions={'C': array([1.00000000e-04,
2.63665090e-04, 6.95192796e-04, 1.83298071e-03,
       4.83293024e-03, 1.27427499e-02, 3.35981829e-02, 8.85866790e-02,
       2.33572147e-01, 6.15848211e-01, 1.62377674e+00, 4.28133240e+00,
       1.12883789e+01, 2.97635144e+01, 7.84759970e+01, 2.06913808e+02,
       5.45559478e+02, 1.43844989e+03, 3.79269019e+03,
1.00000000e+04]),
                                        'max_iter': [1000, 1500,
2000],
                                        'solver': ['liblinear',
'lbfgs',
                                                   'newton-cg']})
```

```
lr_rs_model.best_params_
```

```
{'solver': 'liblinear', 'max_iter': 1500, 'C': 1.623776739188721}
```

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

```
0.7853658536585366
```

```
lr_rs_model.best_estimator_
```

```
LogisticRegression(C=1.623776739188721, max_iter=1500,
solver='liblinear')
```

```
lr_rs_model_metrics = pred_metrics(lr_rs_model.best_estimator_,
X_test, y_test)
lr_rs_model_metrics
```

```
{'Accuracy': 0.78536585,
 'Precision': 0.74380165,
 'Recall': 0.87378641,
 'F1': 0.80357143}
```

Random Forest Classifier

```
rfc_rs_model = RandomizedSearchCV(models['RandomForestClassifier'],
rfc_rs_grid, cv=5, n_iter=20)
rfc_rs_model.fit(X_train, y_train)
```

```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(),
n_iter=20,
                   param_distributions={'max_depth': [None, 3, 5, 10],
                                        'min_samples_leaf': array([ 1,
3,  5,  7,  9, 11, 13, 15, 17, 19]),
                                        'min_samples_split':
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18]),
                                        'n_estimators': array([ 10,
60, 110, 160, 210, 260, 310, 360, 410, 460, 510, 560, 610,
       660, 710, 760, 810, 860, 910, 960])})
```

```
rfc_rs_model.best_params_
```

```
{'n_estimators': 960,
 'min_samples_split': 4,
 'min_samples_leaf': 3,
 'max_depth': 10}
```

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

```
0.9560975609756097
```

```
rfc_rs_model_metrics = pred_metrics(rfc_rs_model.best_estimator_,
X_test, y_test)
rfc_rs_model_metrics
```

```
{'Accuracy': 0.95609756,
 'Precision': 0.95192308,
 'Recall': 0.96116505,
 'F1': 0.95652174}
```

```
lr_rs_model_metrics
```

```
{'Accuracy': 0.78536585,
 'Precision': 0.74380165,
```

```
 'Recall': 0.87378641,
 'F1': 0.80357143}
```

Compare all four models

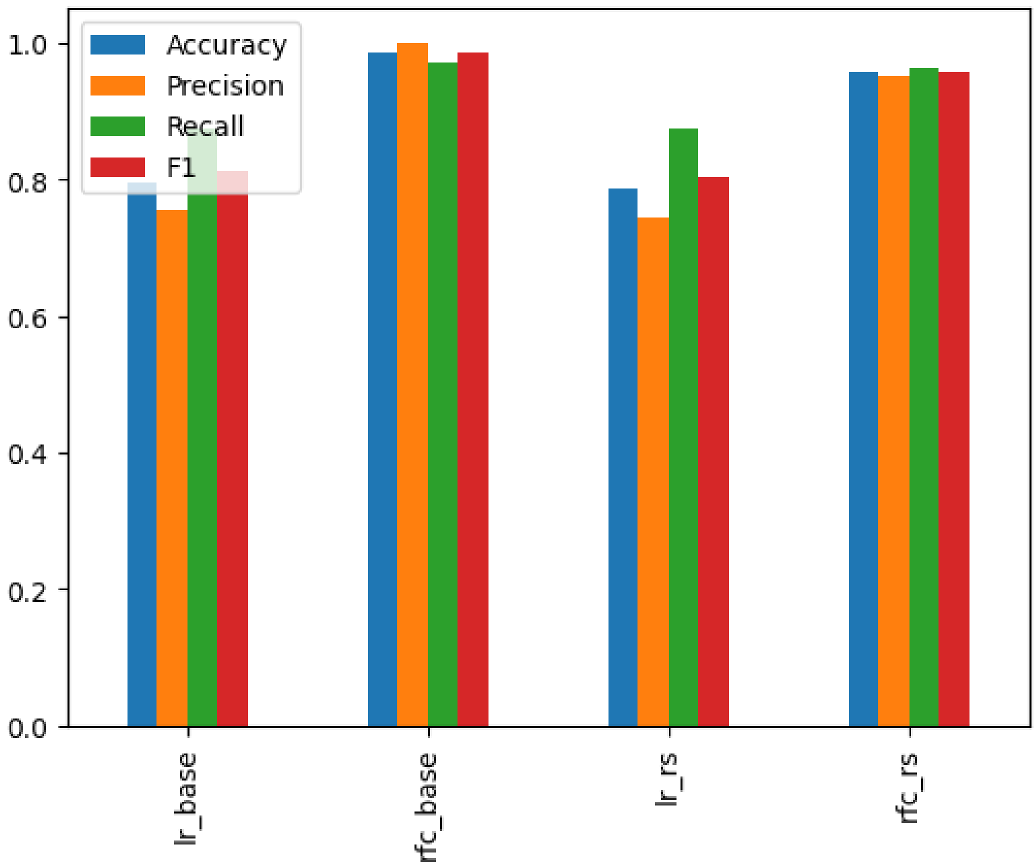
(Base logisticRegression, Base RandomForestClassifier, RandomizedLogisisticRegression and RandomizedRandomForesClassifier)

```
compare_base_and_rs_metrics = pd.DataFrame({
    "lr_base": lr_base_model_metrics,
    "rfc_base": rfc_base_model_metrics,
    "lr_rs": lr_rs_model_metrics,
    "rfc_rs": rfc_rs_model_metrics
})

compare_base_and_rs_metrics
```

	lr_base	rfc_base	lr_rs	rfc_rs
Accuracy	0.795122	0.985366	0.785366	0.956098
Precision	0.756303	1.000000	0.743802	0.951923
Recall	0.873786	0.970874	0.873786	0.961165
F1	0.810811	0.985222	0.803571	0.956522

```
compare_base_and_rs_metrics.T.plot.bar()
```

```
<Axes: >
```



As we can see that lr_rs(logisticRegression from randomizedSearchCV) has more accuracy(If we emphasis more on accuracy than other metric) work better than other three.

Hence, We will move forward with LogisticRegression and apply GridSearchCV on it.

GridSearchCV

Apply GridSearchCV on LogisticRegression

```
# Hyperparameter grid for LogisiticRegression(for GridSearchCV)
lr_rs_grid = {
    "solver" : ["liblinear", 'newton-cg'],
    "C" : np.logspace(-5, 5, num=30),
    "max_iter" : [1000, 1500, 2000, 2500]
}

lr_gs_model = GridSearchCV(models['LogisticRegression'], lr_rs_grid,
cv=5)
lr_gs_model.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=LogisticRegression(max_iter=1000),
             param_grid={'C': array([1.00000000e-05, 2.21221629e-05,
4.89390092e-05, 1.08263673e-04,
       2.39502662e-04, 5.29831691e-04, 1.17210230e-03, 2.59294380e-03,
```

```
       5.73615251e-03, 1.26896100e-02, 2.80721620e-02, 6.21016942e-02,
       1.37382380e-01, 3.03919538e-01, 6.72335754e-01, 1.48735211e+00,
       3.29034456e+00, 7.27895384e+00, 1.61026203e+01, 3.56224789e+01,
       7.88046282e+01, 1.74332882e+02, 3.85662042e+02, 8.53167852e+02,
       1.88739182e+03, 4.17531894e+03, 9.23670857e+03, 2.04335972e+04,
       4.52035366e+04, 1.00000000e+05]),
                         'max_iter': [1000, 1500, 2000, 2500],
                         'solver': ['liblinear', 'newton-cg']})
```

```
lr_gs_model.best_params_
```

```
{'C': 1.4873521072935119, 'max_iter': 1000, 'solver': 'liblinear'}
```

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

```
0.7853658536585366
```

```
lr_gs_model_metrics = pred_metrics(lr_gs_model.best_estimator_,
X_test, y_test)
lr_gs_model_metrics
```

```
{'Accuracy': 0.78536585,
 'Precision': 0.74380165,
 'Recall': 0.87378641,
 'F1': 0.80357143}
```

```
compare_base_and_rs_metrics
```

	lr_base	rfc_base	lr_rs	rfc_rs
Accuracy	0.795122	0.985366	0.785366	0.956098
Precision	0.756303	1.000000	0.743802	0.951923
Recall	0.873786	0.970874	0.873786	0.961165
F1	0.810811	0.985222	0.803571	0.956522

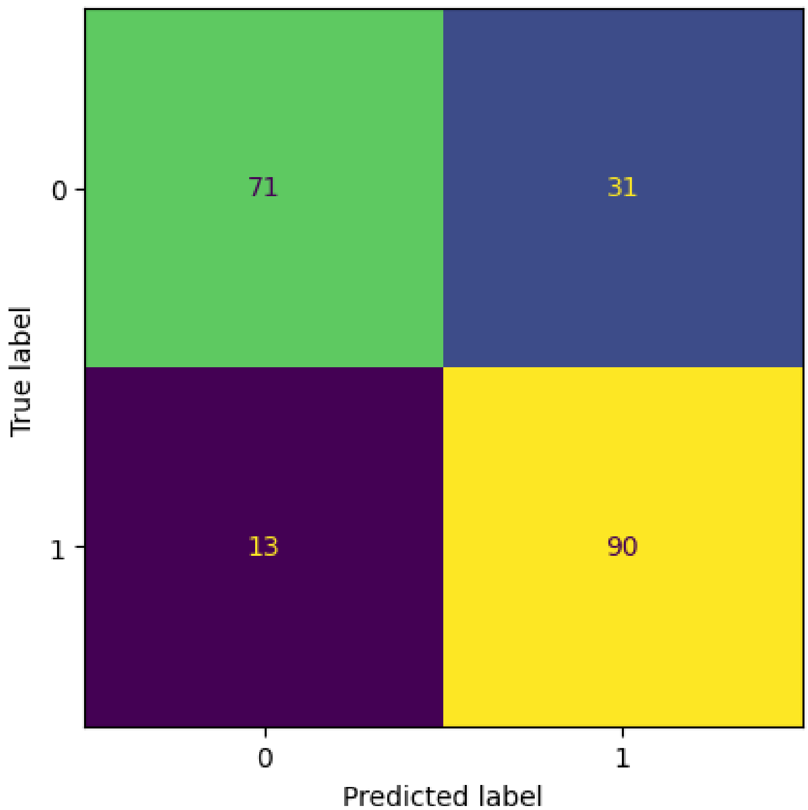
After comparing logisticRegression base model, RandomizedSearch logisticRegression model and GridSearch logisticRegression we found that that is not such differnce. Hence we can move forward with any of three model

Model Evaluation

- Accuracy
- Area under ROC curve
- Confusion matrix
- Classification report

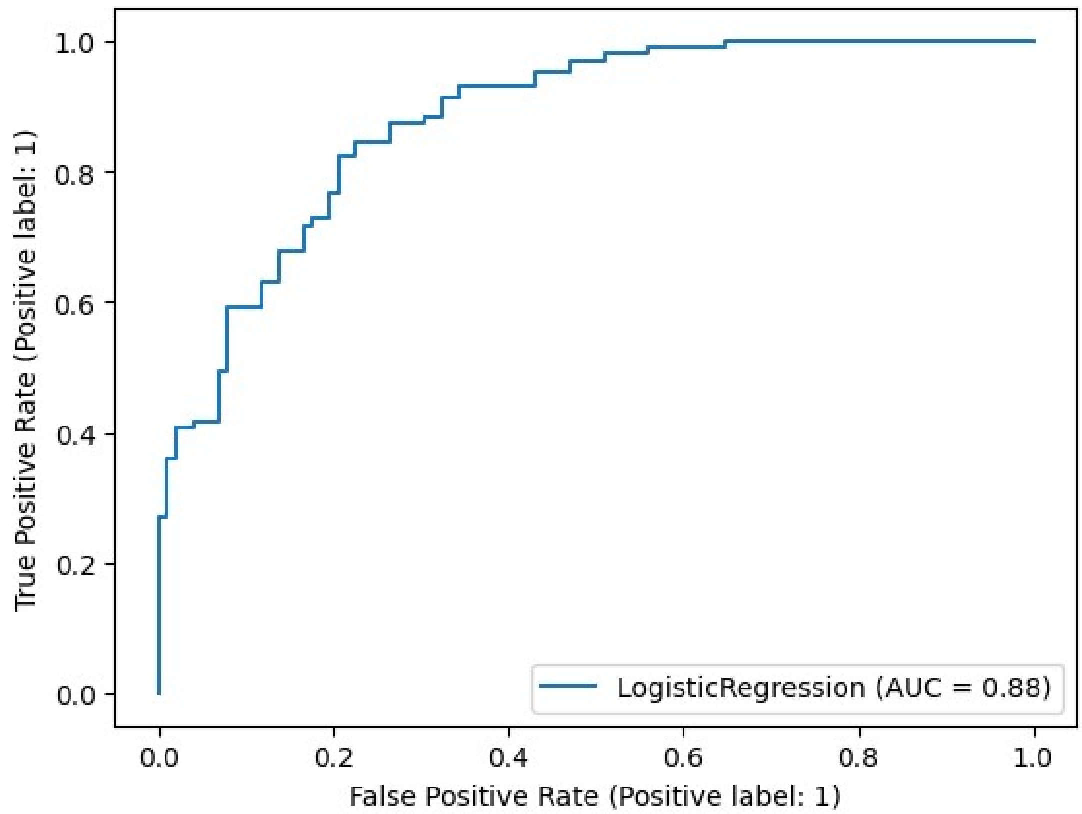
```
lr_final_model = lr_rs_model.best_estimator_
lr_final_model_preds = lr_final_model.predict(X_test)
```

```
ConfusionMatrixDisplay.from_estimator(lr_final_model, X_test,
y_test,colorbar=False);
```



ROC and AUC

```
RocCurveDisplay.from_estimator(lr_final_model, X_test, y_test);
```



Classification Report

```
print(classification_report(y_test,lr_final_model_preds))
```

```
              precision    recall  f1-score   support

           0       0.85      0.70      0.76       102
           1       0.74      0.87      0.80       103

    accuracy                           0.79       205
   macro avg       0.79      0.78      0.78       205
weighted avg       0.79      0.79      0.78       205
```

Evaluation metrics using cross-validation

Accuracy

```
np.random.seed(42)
lr_final_model_cv_acc = cross_val_score(lr_final_model, X, y, cv=5,
scoring='accuracy').mean()
print(f'The cross-validated accuracy is
{lr_final_model_cv_acc*100:.2f}')
```

```
The cross-validated accuracy is 84.78
```

Precision

```
np.random.seed(42)
lr_final_model_cv_precision = cross_val_score(lr_final_model, X, y,
cv=5, scoring='precision').mean()
print(f'The cross-validated precision is
{lr_final_model_cv_precision:.2f}')
```

```
The cross-validated precision is 0.82
```

Recall

```
np.random.seed(42)
lr_final_model_cv_recall = cross_val_score(lr_final_model, X, y, cv=5,
scoring='recall').mean()
print(f'The cross-validated recall is {lr_final_model_cv_recall:.2f}')
```

```
The cross-validated recall is 0.90
```

F1

```
np.random.seed(42)
lr_final_model_cv_f1 = cross_val_score(lr_final_model, X, y, cv=5,
scoring='f1').mean()
print(f'The cross-validated F1 is {lr_final_model_cv_f1:.2f}')
```

```
The cross-validated F1 is 0.86
```

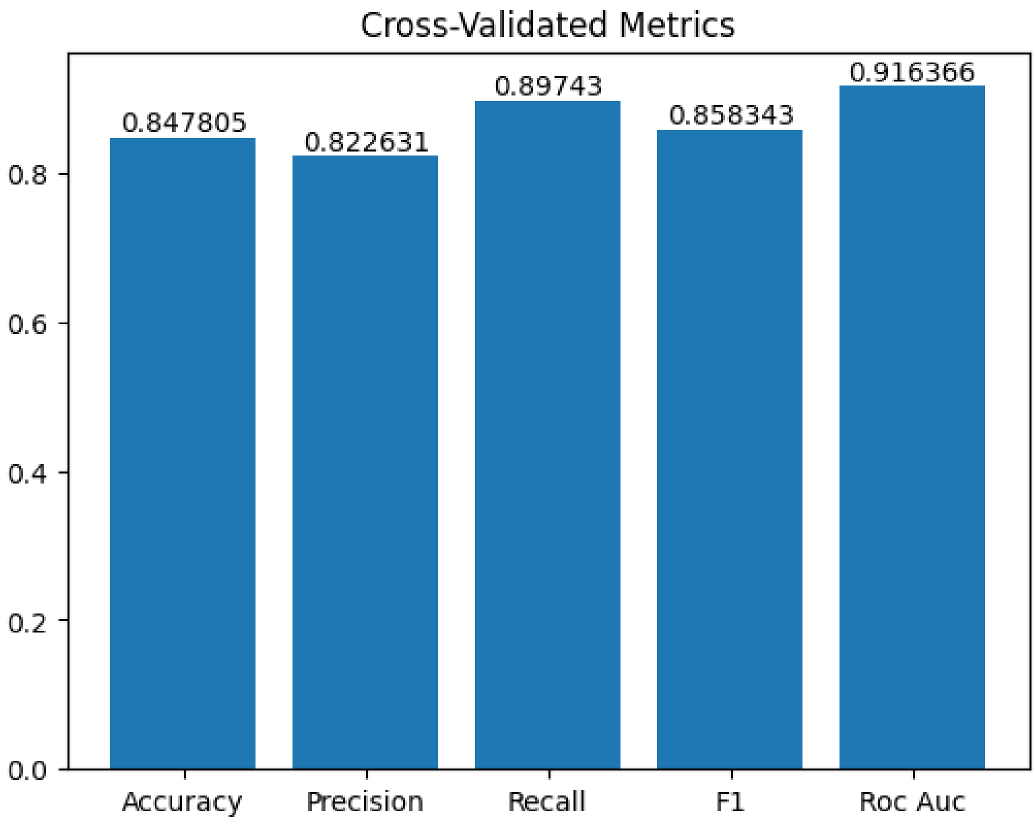
AUC

```
np.random.seed(42)
lr_final_model_cv_auc = cross_val_score(lr_final_model, X, y, cv=5,
scoring='roc_auc').mean()
print(f'The cross-validated AUC score is is
{lr_final_model_cv_auc:.2f}')
```

```
The cross-validated AUC score is is 0.92
```

Visualize cross-validation score

```
metric_name = ['Accuracy', 'Precision', 'Recall', 'F1', 'Roc Auc']
metric_value = [lr_final_model_cv_acc, lr_final_model_cv_precision,
lr_final_model_cv_recall, lr_final_model_cv_f1, lr_final_model_cv_auc]

bar_container = plt.bar(metric_name, metric_value)
plt.bar_label(bar_container)
plt.title("Cross-Validated Metrics");
```



Inference:

- **Accuracy = 84.8 % => It means model correctly predicted the outcome(both heart disease and no heart disease) in 84.8% of the case.**
- **Precision = 82.1% => It means that out of prediction of heart dieseas, 82.8% of them truly have cancer.**
- **Recall = 92.7% => It means out of all the patients have heart disease, our model predicts 92% of them having heart disease.**

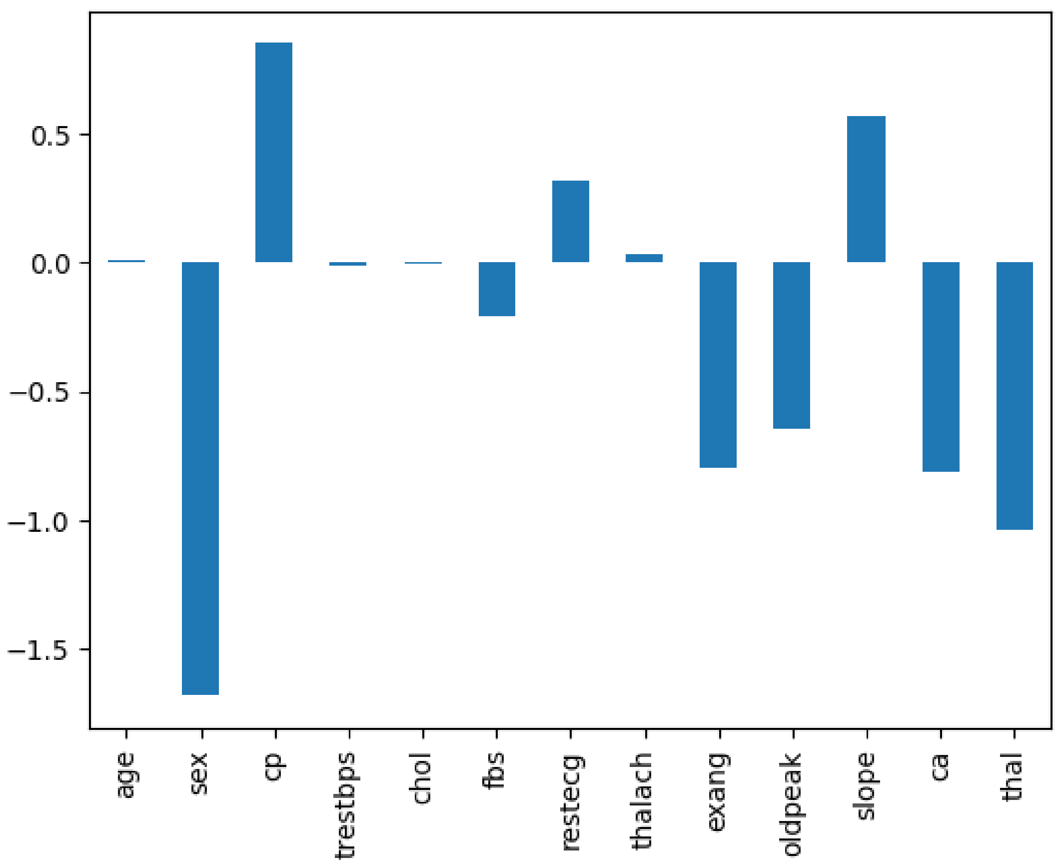
Here, It is excellent at identifying patients with heart disease (92.7% recall) and fairly accurate in predicting heart disease for those who truly have it (82.1% precision).

It depend upon buisness task that we emphasis more on which on metrics(recall or precision).

Feature Importance

```
lr_final_model.coef_

array([[ 0.01245689, -1.68544402,  0.85198764, -0.01578096, -
0.00828761,
        -0.20903191,  0.32008044,  0.03461353, -0.79648686, -
0.65056533,
         0.56750662, -0.81568847, -1.04166391]])
```

```
df.columns

Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg',
'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')

feature_dict = dict(zip(df.columns, lr_final_model.coef_[0]))
feature_df = pd.DataFrame(feature_dict, index=[0])
feature_df

        age       sex        cp  trestbps      chol       fbs   restecg
\
0  0.012457 -1.685444  0.851988 -0.015781 -0.008288 -0.209032   0.32008


    thalach     exang   oldpeak     slope        ca      thal
0  0.034614 -0.796487 -0.650565  0.567507 -0.815688 -1.041664

feature_df.T.plot.bar(legend=False);
```



Larger the value of coeffecient, the more it contributes to dicision making. If value is negative then there is negative correlation and vice-versa for positive value.

Inference:

1. `sex` feature has larger negative value implies it has strong negative correlation.

It means that value of sex decrease(i.e 0) there is more chance of having a heart disease. We have seen earlier that female have 75% of change of heart. Hence women have more chance of heart disease.

1. `slope` feature has larger positive value implies it has positive correlation

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

target	0	1
slope		
0	46	28
1	324	158
2	129	340

As we can notice form the table is that as slope increase chance of haveing heart disease also increases.(same logic goes with `cp`, `exang` and `thal` etc)

Still there is lot of thing we can do

- Try different classification model(like SVM, XGboost)
- Try more hyper paramter
- Ask for more data samples.