Predicting Heart Disease ML

December 5, 2022

[ ]:

```

```

1 End to end ML Project for Heart Disease Prediction

1.1 1. Problem Definition

In this case, the problem we will be exploring is a binary classification.

Given clinical parameters about a patient, we can predict whether or not they have heart disease.

1.2 2. Data

The original data came from the Cleveland database from UCI Machine Learning Repository.

The original database contains 76 attributes, but here only 14 attributes will be used. Attributes (also called features) are the variables what we'll use to predict our target variable.

1.3 3. Evaluation

If we can reach 95% accuracy at predicting whether or not a patient has heart disease during the proof of concept, we'll pursure this project.

1.4 4. Features

Heart Disease Data Dictionary

The following are the features we'll use to predict our target variable (heart disease or no heart disease).

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)

Note: No personal identifiable information (PPI) can be found in the dataset.

[1]:
```
# Regular EDA and plotting libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# want the plots to appear in the notebook-->use either %matplotlib inline or␣
 ↪%matplotlib 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
from sklearn.metrics import plot_roc_curve
```

[2]:
```
df = pd.read_csv("heart-disease.csv")
df.shape
```

[2]:
```
(303, 14)
```

1.5 Data Exploration

[3]:
```
df.describe()
```

[3]:
```
              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
```

[4]:
```
df.head()
```

[4]:
```
   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
```

[5]:
```
df.count()
```

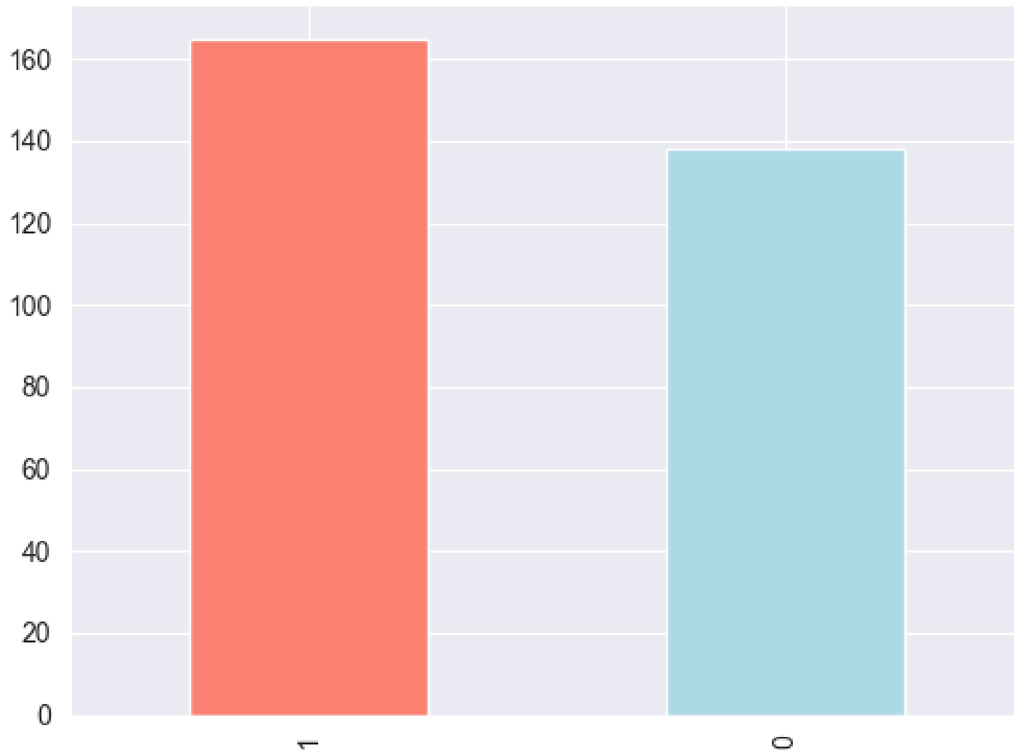
[5]:
```
age         303
sex         303
cp          303
trestbps    303
chol        303
fbs         303
restecg     303
thalach     303
exang       303
oldpeak     303
slope       303
ca          303
thal        303
target      303
dtype: int64
```

[6]:
```
df.target.value_counts()
df.target.value_counts(normalize=True)
```

[6]:
```
1    0.544554
0    0.455446
Name: target, dtype: float64
```

Since these two values are close to even, the target column can be considered balanced. An unbalanced target column, meaning some classes have far more samples, can be harder to model than a balanced set. Ideally, all of the target classes have the same number of samples.

[7]:
```
# Plot the value counts with a bar graph
df.target.value_counts().plot(kind="bar", color=["salmon", "lightblue"]);
```



Heart Disease Frequency according to Gender

compare two columns to each other –> pd.crosstab(column_1, column_2)

to gain an intuition about how the independent variables interact with dependent variables.

Let's compare the target column with the sex column.

For the target column, 1 = heart disease present, 0 = no heart disease. And for sex, 1 = male, 0 = female.

[8]:
```
df.sex.value_counts()
```

[8]:
```
1    207
0     96
Name: sex, dtype: int64
```

[9]:
```
# Compare target column with sex column
pd.crosstab(df.target, df.sex)
```

[9]:
```
sex      0    1
target
0       24  114
1       72   93
```

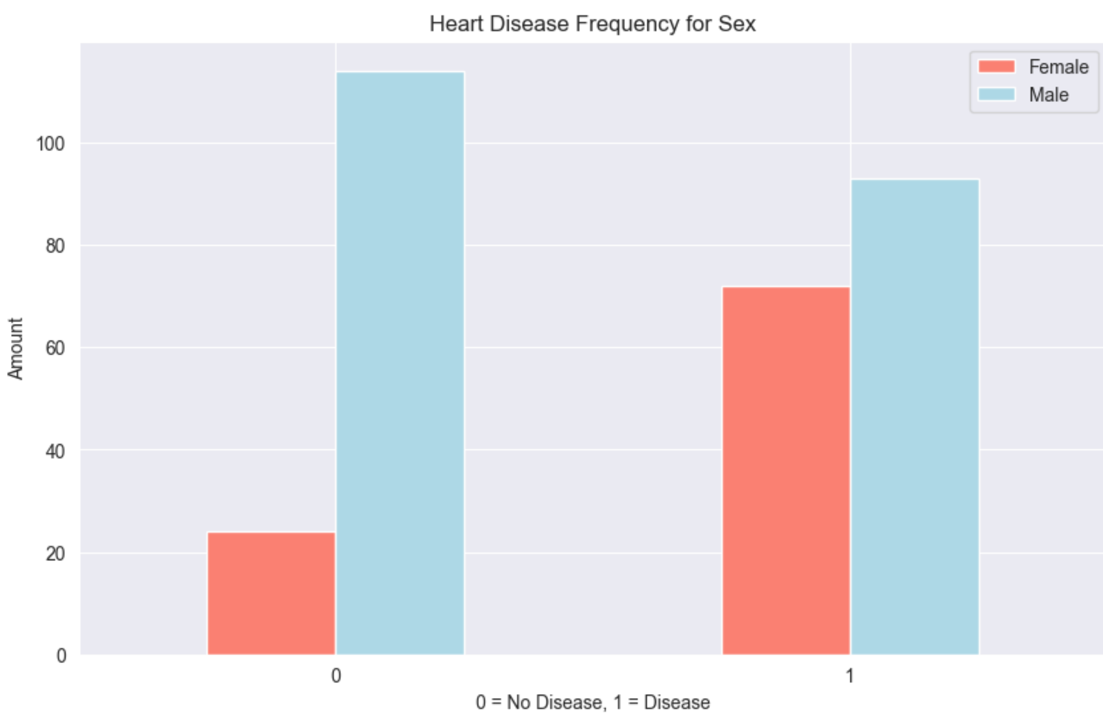
Since there are about 100 women and 72 of them have a postive value of heart disease being present, we might infer, based on this one variable if the participant is a woman, there's a 75% chance she has heart disease.

As for males, there's about 200 total with around half indicating a presence of heart disease. So we might predict, if the participant is male, 50% of the time he will have heart disease.

Averaging these two values, we can assume, based on no other parameters, if there's a person, there's a 62.5% chance they have heart disease.

This can be our very simple baseline.

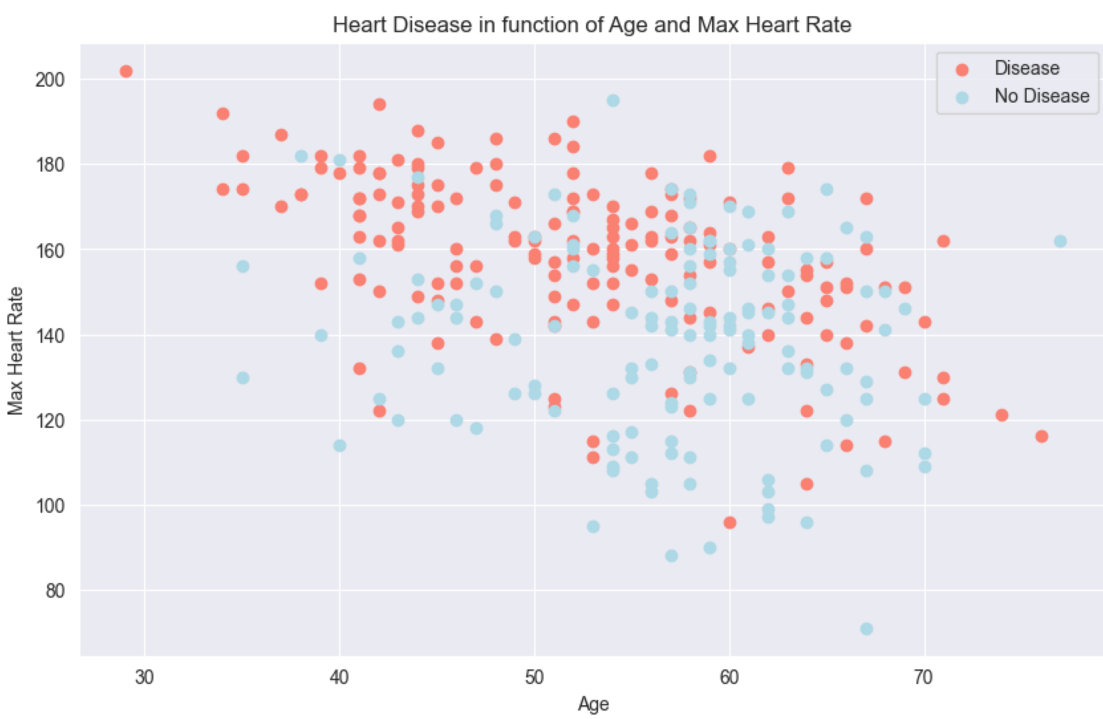
[10]:
```
# Create a plot
pd.crosstab(df.target, df.sex).plot(kind="bar", figsize=(10,6),␣
↪color=["salmon", "lightblue"])

# Add some attributes to it
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
```



1.6 Age vs Max Heart rate for Heart Disease

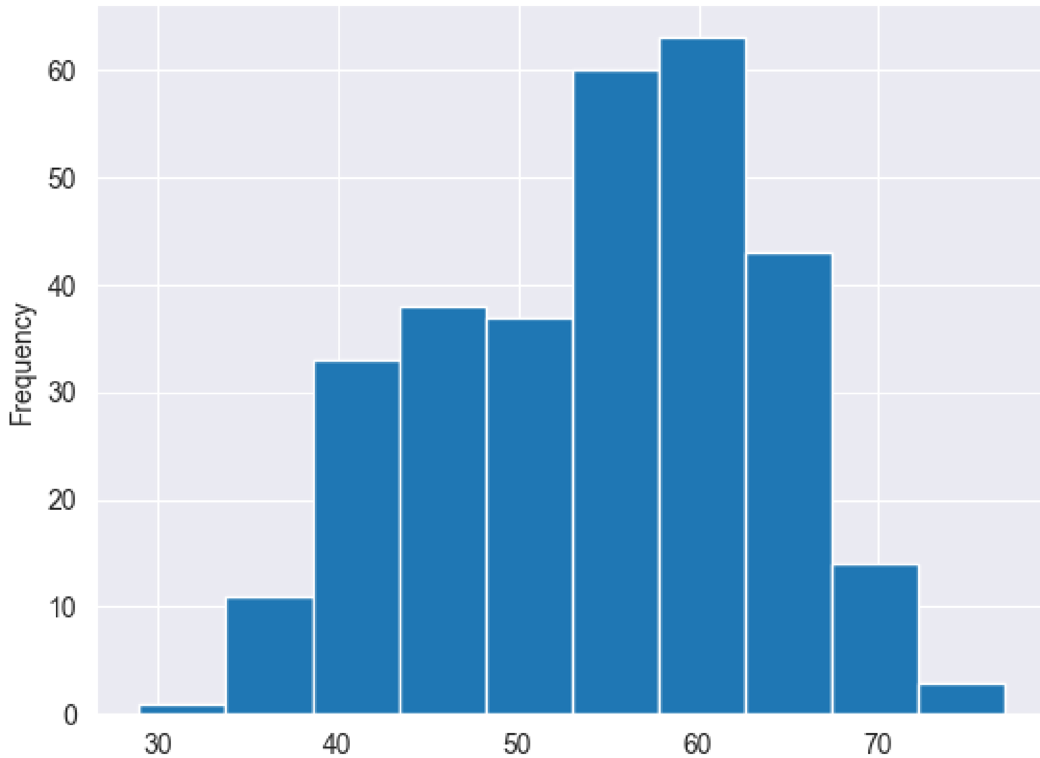
combining a couple of independent variables, such as, age and thalach (maximum heart rate) and then comparing them to the target variable heart disease. Because there are so many different values for age and thalach, we'll use a scatter plot.

[11]:
```
# Create another figure
plt.figure(figsize=(10,6))

# Start with positve examples
plt.scatter(df.age[df.target==1],
            df.thalach[df.target==1],
            c="salmon") # define it as a scatter figure

# Now for negative examples on the same plot, so we call plt again
plt.scatter(df.age[df.target==0],
            df.thalach[df.target==0],
            c="lightblue") # axis always come as (x, y)

# Add some helpful info
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");
```



It seems the younger someone is, the higher their max heart rate (dots are higher on the left of the graph) and the older someone is, the more green dots there are. But this may be because there are more dots all together on the right side of the graph (older participants).Both of these are observational of course.

[12]:
```
# Histograms are a great way to check the distribution of a variable
df.age.plot.hist();
```



1.7 Heart Disease Frequency per Chest Pain Type

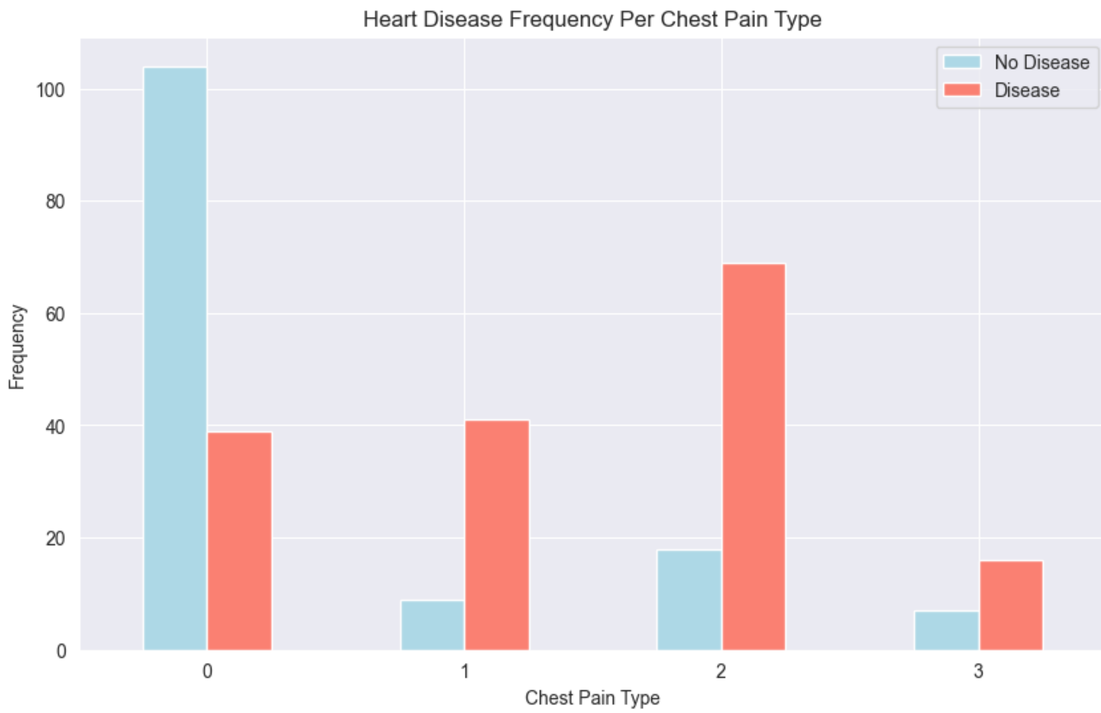
Let's try another independent variable. This time, cp (chest pain).

[13]:
```
pd.crosstab(df.cp, df.target)
```

[13]:
```
target    0   1
cp
0       104  39
1         9  41
2        18  69
3         7  16
```

[14]:
```
# Create a new crosstab and base plot
pd.crosstab(df.cp, df.target).plot(kind="bar",
```

```
                                    figsize=(10,6),
                                    color=["lightblue", "salmon"])

# Add attributes to the plot to make it more readable
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);
```



what the different levels of chest pain are.

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 It's interesting the atypical agina (value 1) states it's not related to the heart but seems to have a higher ratio of participants with heart disease than not.

According to PubMed, it seems even some medical professionals are confused by the term.

Today, 23 years later, "atypical chest pain" is still popular in medical circles. Its meaning, however, remains unclear. A few articles have the term in their title, but do not define or discuss it in their text. In other articles, the term refers to noncardiac causes of chest pain.

Although not conclusive, this graph above is a hint at the confusion of defintions being represented in data.

1.8 Correlation between independent variables

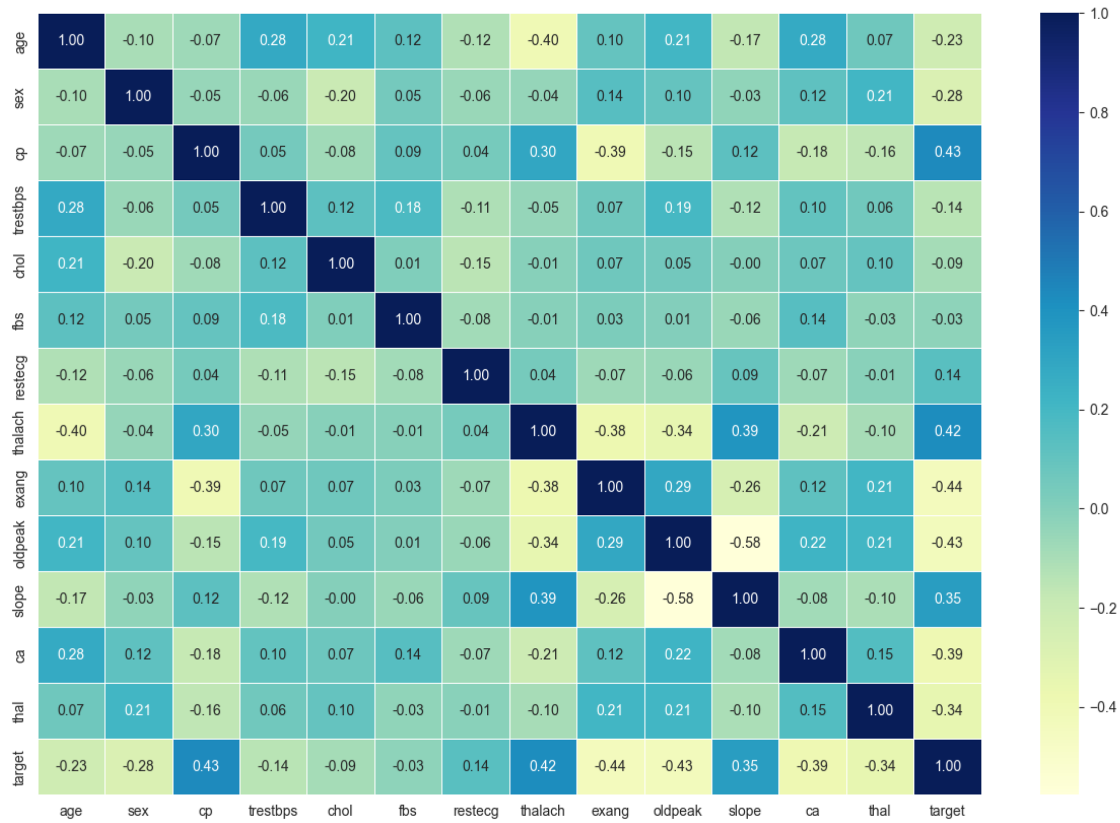
Comparing all of the independent variables in one hit.

[15]:
```python
# Find the correlation between independent variables
corr_matrix = df.corr()
corr_matrix
```

[15]:
```
               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
```

[16]:
```python
corr_matrix = df.corr()
plt.figure(figsize=(15, 10))
sns.heatmap(corr_matrix,
            annot=True,
            linewidths=0.5,
            fmt= ".2f",
            cmap="YlGnBu");
```



A higher positive value means a potential positive correlation (increase) and a higher negative value means a potential negative correlation (decrease).

The above exploratory data analysis (EDA) is to start building an intuitition of the dataset.

From EDA, aside from our basline estimate using sex, the rest of the data seems to be pretty distributed.

The next is model driven EDA, using machine learning models to drive the questions.

1.9 5. Modeling

[17]:
```
df.head()
```

[17]:
```
   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
```

[18]:
```
# Everything except target variable
X = df.drop("target", axis=1)

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

[19]:
```
# 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,
                                                    y,
                                                    test_size = 0.2) #␣
 ↪percentage of data to use for test set
```

[20]:
```
X_train.head()
```

[20]:
```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
132   42    1   1       120   295    0        1      162      0      0.0
202   58    1   0       150   270    0        0      111      1      0.8
196   46    1   2       150   231    0        1      147      0      3.6
75    55    0   1       135   250    0        0      161      0      1.4
176   60    1   0       117   230    1        1      160      1      1.4

     slope  ca  thal
132      2   0     2
```

```
202       2   0      3
196       1   0      2
75        1   0      2
176       2   2      3
```

[21]:
```python
y_train, len(y_train)
```

[21]:
```
(array([1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1,
        1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
        0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0,
        0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0,
        1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1,
        1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1,
        1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
        0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1,
        1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
        1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1]),
 242)
```

[22]:
```python
X_test.head()
```

[22]:
```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
179   57    1   0       150   276    0        0      112      1      0.6
228   59    1   3       170   288    0        0      159      0      0.2
111   57    1   2       150   126    1        1      173      0      0.2
246   56    0   0       134   409    0        0      150      1      1.9
60    71    0   2       110   265    1        0      130      0      0.0

     slope  ca  thal
179      1   1     1
228      1   0     3
111      2   1     3
246      1   2     3
60       2   1     2
```

1.9.1 Model choices

The following estimators will be used and their results will be compared

Logistic Regression - LogisticRegression()

K-Nearest Neighbors - KNeighboursClassifier()

RandomForest - RandomForestClassifier()

[23]:
```python
 # Put models in a dictionary
models = {"KNN": KNeighborsClassifier(),
          "Logistic Regression": LogisticRegression(),
```

```python
                 "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
```

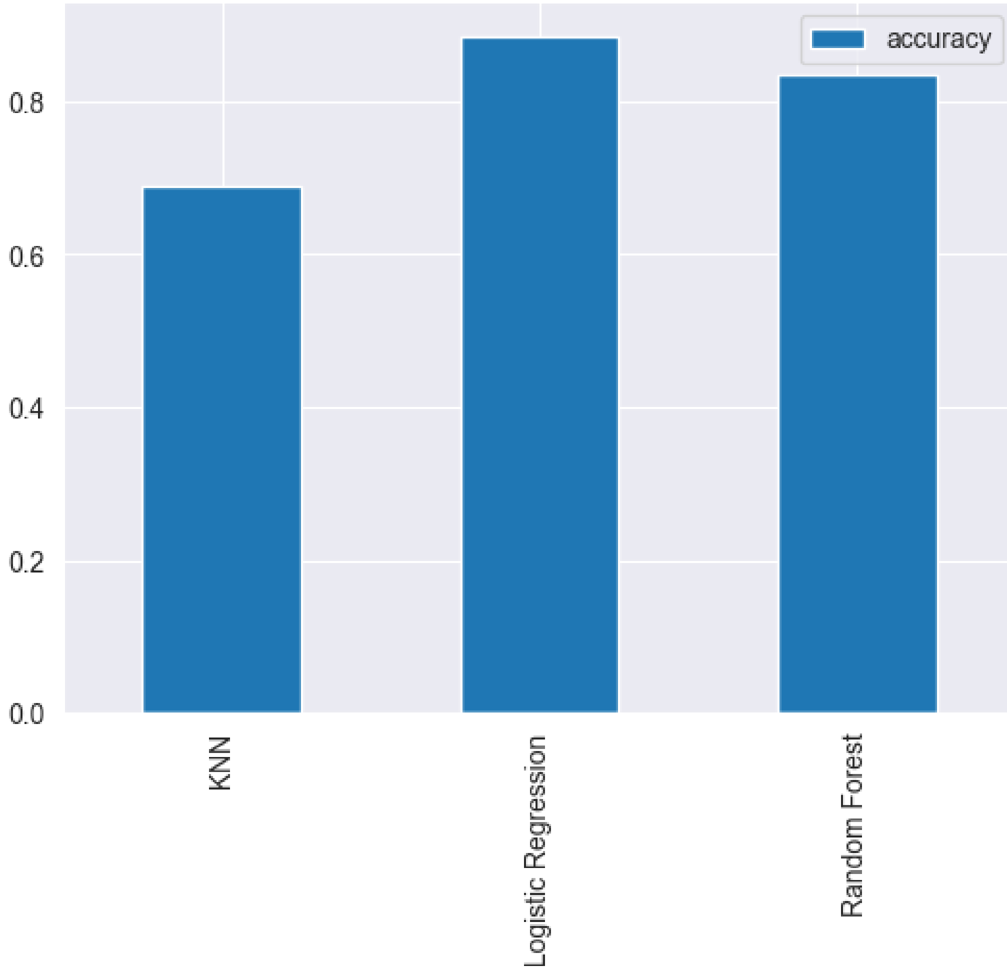
[24]:
```python
model_scores = fit_and_score(models=models,
                             X_train=X_train,
                             X_test=X_test,
                             y_train=y_train,
                             y_test=y_test)
model_scores
```

```
/Users/shiweiliu/opt/anaconda3/envs/heart_disease_prediction_env/lib/python3.10/
site-packages/sklearn/linear_model/_logistic.py:444: 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(
```

[24]:
```
{'KNN': 0.6885245901639344,
 'Logistic Regression': 0.8852459016393442,
 'Random Forest': 0.8360655737704918}
```

1.9.2 Model Comparison

[25]:
```
model_compare = pd.DataFrame(model_scores, index=['accuracy'])
model_compare.T.plot.bar();
```



1.9.3 Hyperparameter tuning and cross-validation

Tune KNeighborsClassifier (K-Nearest Neighbors or KNN) by hand. The default N is 5 (n_neigbors=5).

[26]:
```
train_scores,test_scores = [],[]

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

# Setup algorithm
knn = KNeighborsClassifier()
```

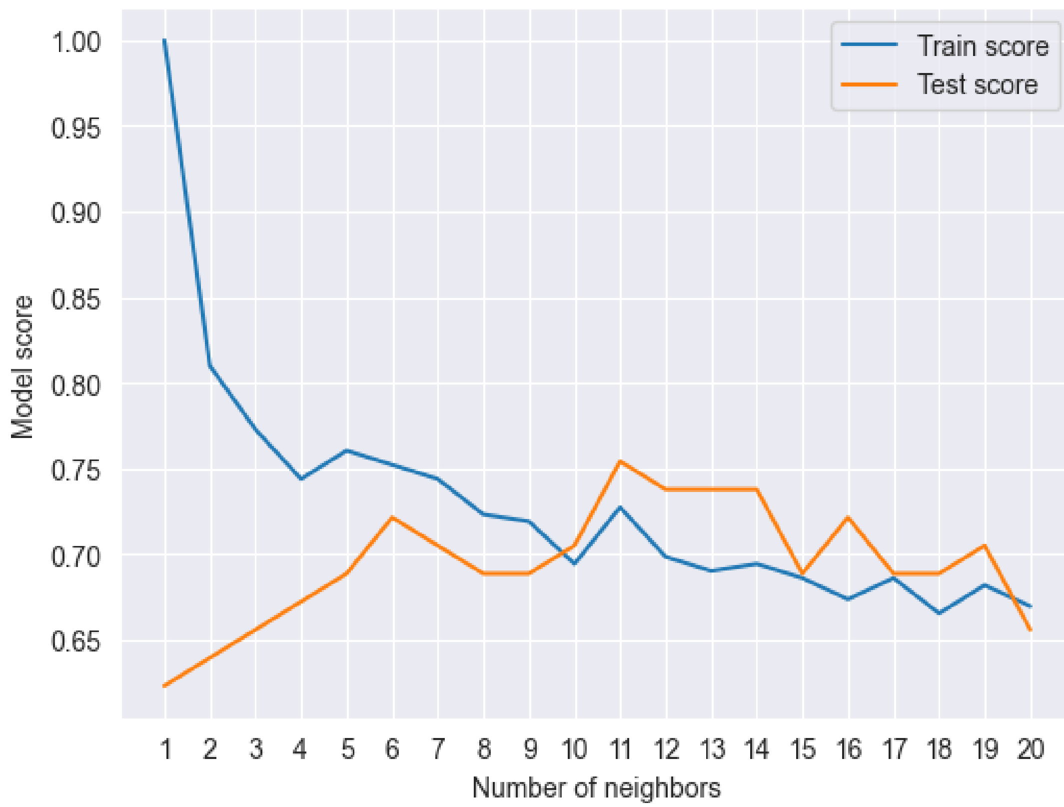
```python
# Loop through different neighbors values
for i in neighbors:
    knn.set_params(n_neighbors = i) # set neighbors value

    # Fit the algorithm
    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))
```

[27]:
```python
train_scores
```

[27]:
```
[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]
```

[28]:
```python
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%
```



1.9.4 Tuning models with RandomizedSearchCV

[29]:
```
# 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)}
```

[30]:
```
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
```

[31]:
```
rs_log_reg.best_params_
```

[31]:
```
{'solver': 'liblinear', 'C': 0.23357214690901212}
```

[32]:
```
rs_log_reg.score(X_test, y_test)
```

[32]:
```
0.8852459016393442
```

LogisticRegression is tunned by using RandomizedSearchCV; we'll do the same for RandomForestClassifier.

[33]:
```
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
```

[34]:
```
# Find the best parameters
rs_rf.best_params_
```

[34]:
```
{'n_estimators': 210,
 'min_samples_split': 4,
 'min_samples_leaf': 19,
 'max_depth': 3}
```

[35]:
```
# Evaluate the randomized search random forest model
rs_rf.score(X_test, y_test)
```

[35]:
```
0.8688524590163934
```

Tuning the hyperparameters for each model saw a slight performance boost in both the RandomForestClassifier and LogisticRegression.

Since LogisticRegression is pulling out in front, we'll try tuning it further with GridSearchCV.

1.9.5 Tuning a model with GridSearchCV

The difference between RandomizedSearchCV and GridSearchCV is where RandomizedSearchCV searches over a grid of hyperparameters performing n_iter combinations, GridSearchCV will test every single possible combination.

In short:

RandomizedSearchCV - tries n_iter combinations of hyperparameters and saves the best. GridSearchCV - tries every single combination of hyperparameters and saves the best.

[36]:
```
# 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
```

[37]:
```
# Check the best parameters
gs_log_reg.best_params_
```

[37]:
```
{'C': 0.23357214690901212, 'solver': 'liblinear'}
```

[38]:
```
# Evaluate the model
gs_log_reg.score(X_test, y_test)
```

[38]:
```
0.8852459016393442
```

we get the same results as before since our grid only has a maximum of 20 different hyperparameter combinations.

If there are a large amount of hyperparameters combinations in the grid, GridSearchCV may take a long time to try them all out. This is why it's a good idea to start with RandomizedSearchCV, try a certain amount of combinations and then use GridSearchCV to refine them.

1.9.6 Evaluating the classification model, beyond accuracy

Metrics to use:

ROC curve and AUC score - plot_roc_curve()

Confusion matrix - confusion_matrix()

Classification report - classification_report()

Precision - precision_score()

Recall - recall_score()

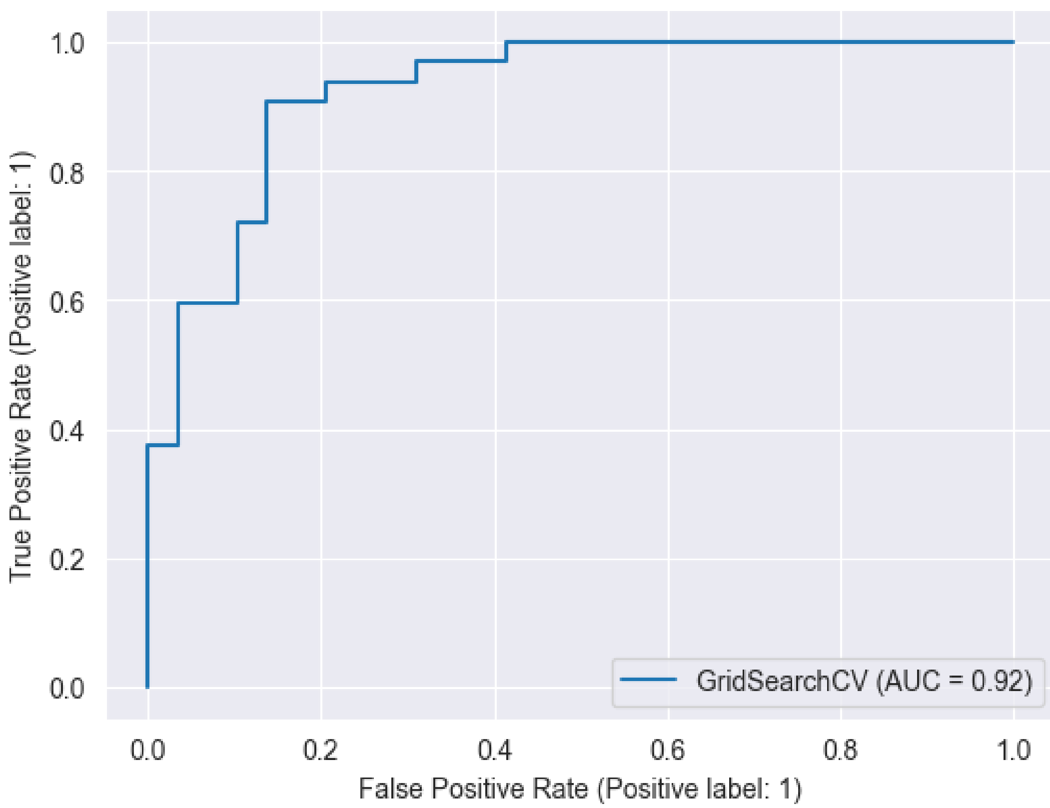
F1-score - f1_score()

[39]:
```python
# Make preidctions on test data
y_preds = gs_log_reg.predict(X_test)
```

1.9.7 ROC Curve and AUC Scores

[40]:
```python
# Import ROC curve function from metrics module
from sklearn.metrics import plot_roc_curve

# Plot ROC curve and calculate AUC metric
plot_roc_curve(gs_log_reg, X_test, y_test);
```

```
/Users/shiweiliu/opt/anaconda3/envs/heart_disease_prediction_env/lib/python3.10/
site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function
plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in
1.0 and will be removed in 1.2. Use one of the class methods:
:meth:`sklearn.metrics.RocCurveDisplay.from_predictions` or
:meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```

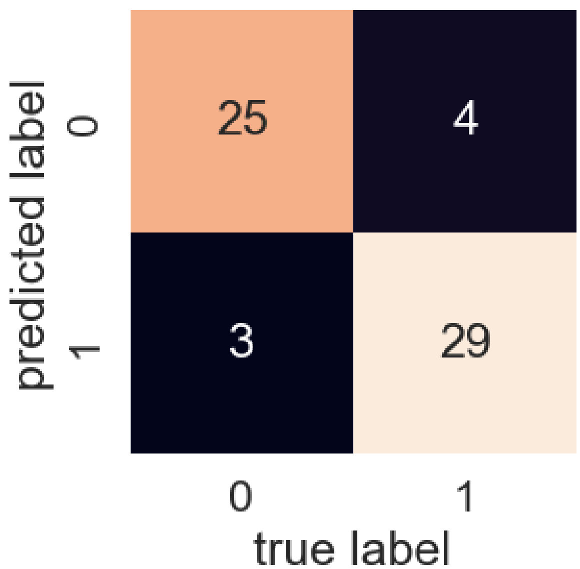


1.9.8 Confusion matrix

[41]:
```
# Display confusion matrix
print(confusion_matrix(y_test, y_preds))
```

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

[42]:
```
import seaborn as sns
sns.set(font_scale=1.5)

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)
```



1.9.9 Classification report

[43]:
```
#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
```

[44]:
```
# Check best hyperparameters
gs_log_reg.best_params_
```

[44]:
```
{'C': 0.23357214690901212, 'solver': 'liblinear'}
```

[45]:
```
# 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")
```

[46]:
```
# 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
```

[46]:
```
array([0.81967213, 0.90163934, 0.8852459 , 0.88333333, 0.75      ])
```

[47]:
```
#Since there are 5 metrics here, we'll take the average.
cv_acc = np.mean(cv_acc)
cv_acc
```

[47]:
```
0.8479781420765027
```

[48]:
```
# 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
```

[48]: 0.8215873015873015

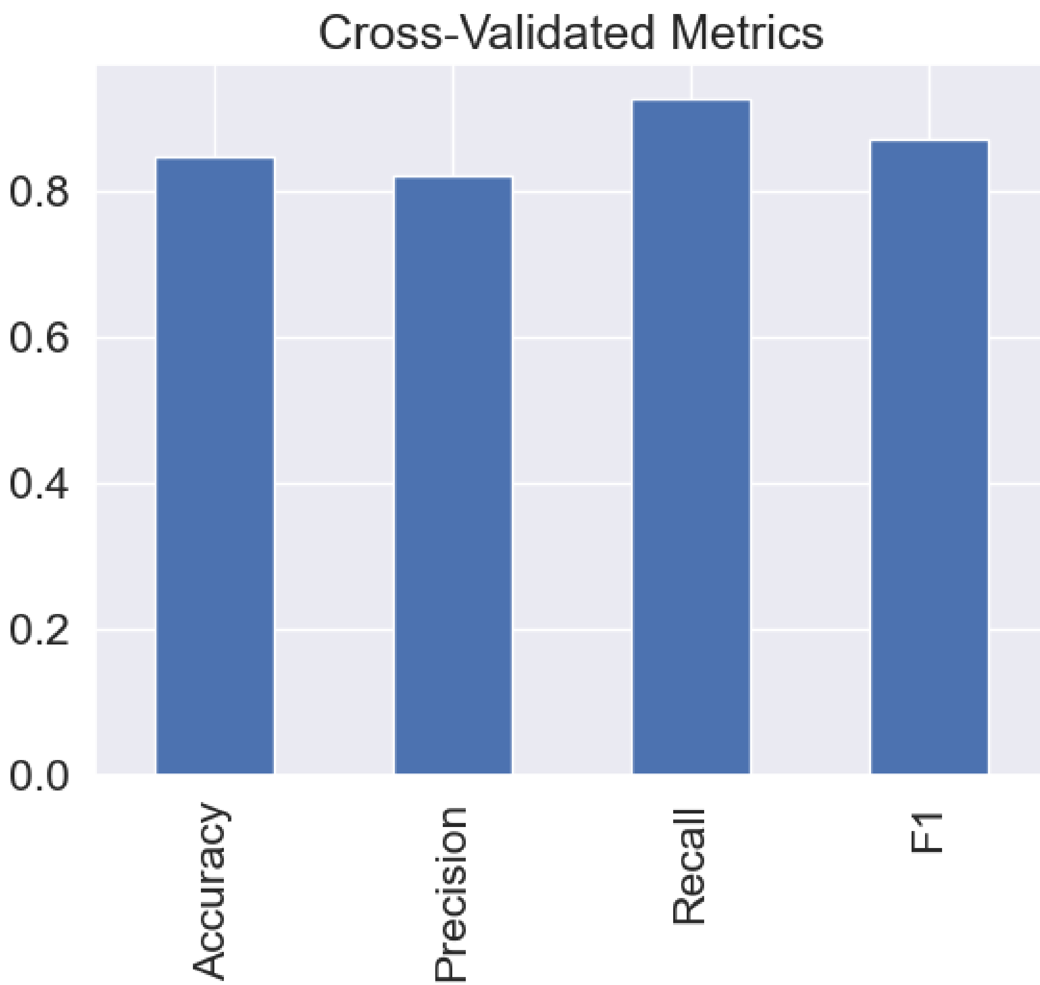
[49]:
```
# Cross-validated recall score
cv_recall = np.mean(cross_val_score(clf,
                                    X,
                                    y,
                                    cv=5, # 5-fold cross-validation
                                    scoring="recall")) # recall as scoring
cv_recall
```

[49]: 0.9272727272727274

[50]:
```
# 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
```

[50]: 0.8705403543192143

[51]:
```
# 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);
```



1.9.10 Feature importance

[52]:
```
# Fit an instance of LogisticRegression (taken from above)
clf.fit(X_train, y_train);
```

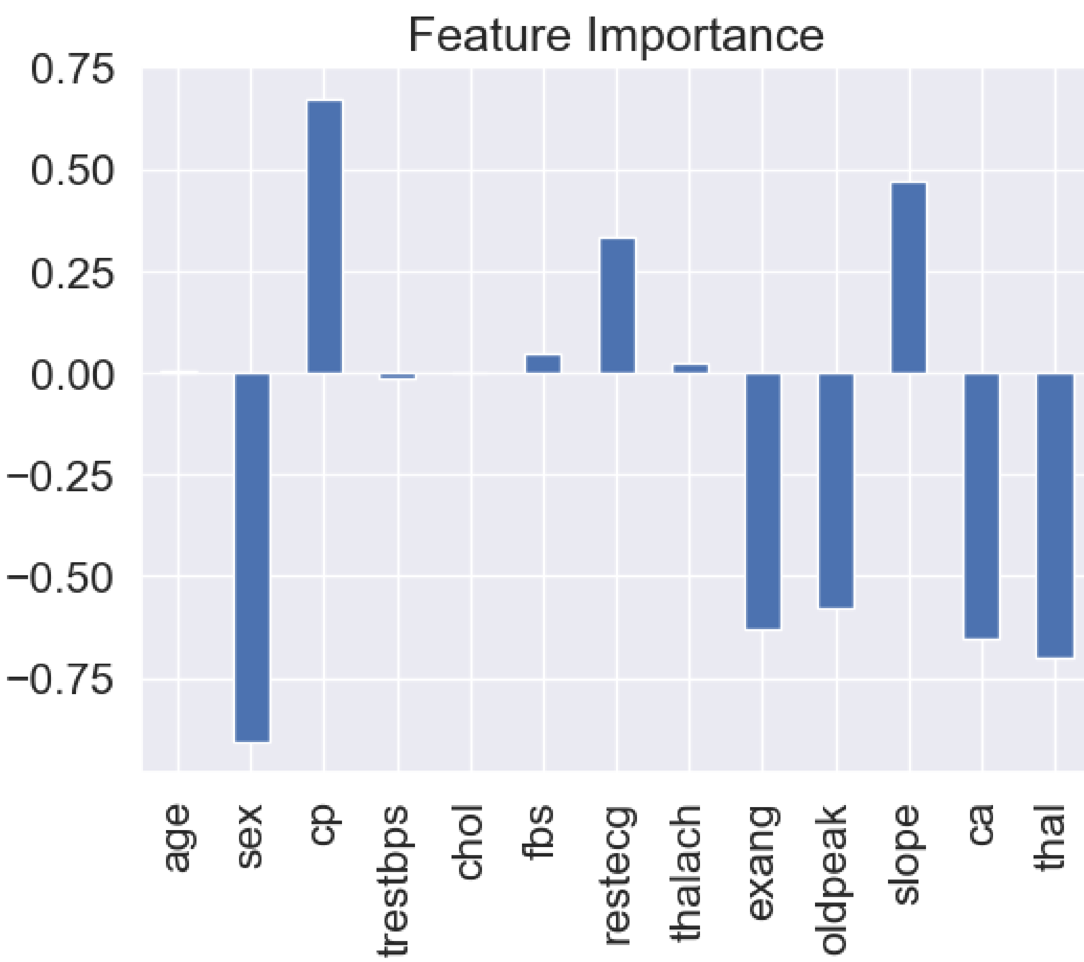
[53]:
```
# Check coef_
clf.coef_
```

[53]:
```
array([[ 0.00369922, -0.9042409 ,  0.67472826, -0.0116134 , -0.00170364,
         0.04787688,  0.33490198,  0.02472938, -0.63120406, -0.5759095 ,
         0.47095141, -0.65165348, -0.69984208]])
```

[54]:
```
# Match features to columns
features_dict = dict(zip(df.columns, list(clf.coef_[0])))
features_dict
```

[54]:
```
{'age': 0.003699220776580221,
 'sex': -0.9042409028785717,
 'cp': 0.6747282587404362,
 'trestbps': -0.011613401339975146,
 'chol': -0.0017036439067759743,
 'fbs': 0.047876881148997324,
 'restecg': 0.3349019815885189,
 'thalach': 0.024729382841083,
 'exang': -0.6312040612837573,
 'oldpeak': -0.5759095045469952,
 'slope': 0.4709514073081419,
 'ca': -0.6516534770577476,
 'thal': -0.6998420764664995}
```

[55]:
```python
# Visualize feature importance
features_df = pd.DataFrame(features_dict, index=[0])
features_df.T.plot.bar(title="Feature Importance", legend=False);
```



[ ]:

```

```

[ ]:

```

```

[55]:

```

```