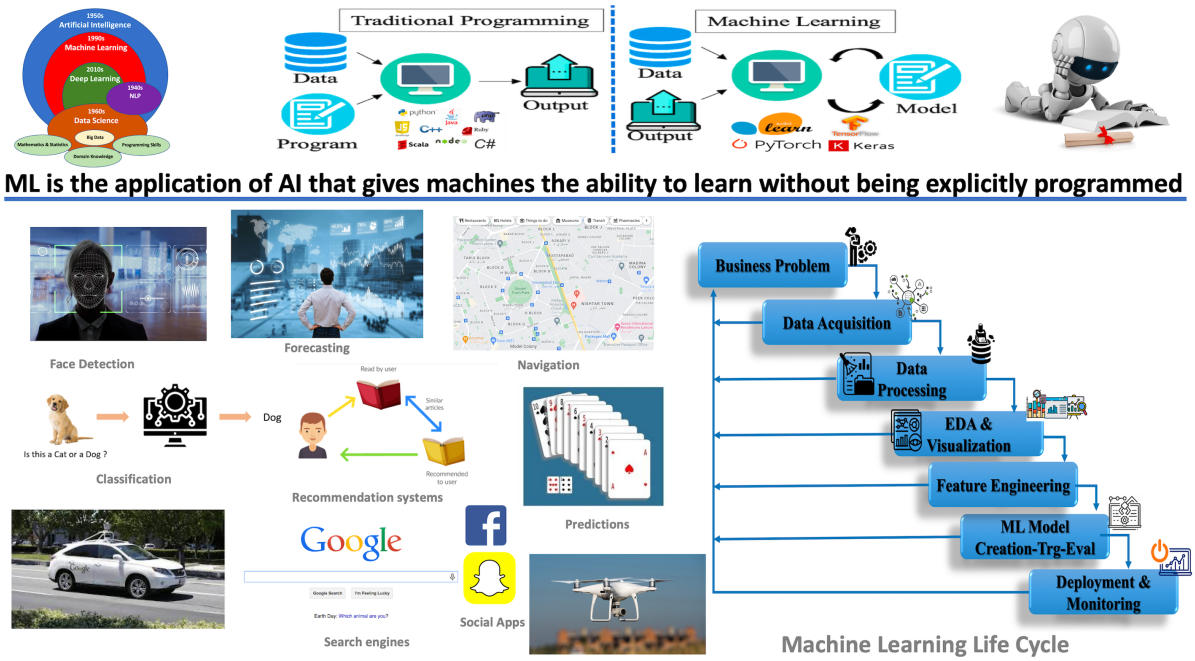
Department of Data Science

Course: Tools and Techniques for Data Science

Instructor: Muhammad Arif Butt, Ph.D.

Lecture 6.22 (Logistic Regression: Part-III)

Open in Colab
(https://colab.research.google.com/github/arifpucit/data-science/blob/master/Section-4-Mathematics-for-Data-Science/Lec-4.1(Descriptive-Statistics).ipynb)



Learning agenda of this notebook

- EDA of Heart Disease Dataset
- Training a Logistic Regression Model
- Evaluation Metrics of Classification Models
  - Confusion Matrix
  - Accuracy
  - Accuracy Paradox
  - Precision

    - Recall
    - F-1 Score
    - F-β Score
    - Precision Recall Tradeoff
    - ROC Curve and AUC Score
- Hyperparameters for Logistic Regression
- Task To Do (Assignment)

In [1]:
```
import numpy as np # np is short for numpy
import pandas as pd # pandas is so commonly used, it's shortened to pd
import matplotlib.pyplot as plt
import seaborn as sns # seaborn gets shortened to sns

%matplotlib inline

from sklearn.linear_model import LogisticRegression

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
```

1. Load Dataset

(i) Toy Datasets (`datasets.load_XXX()` methods)

https://scikit-learn.org/stable/datasets/toy_dataset.html#toy-datasets (https://scikit-learn.org/stable/datasets/toy_dataset.html#toy-datasets)

- Scikit-learn comes with a few small standard datasets that do not require to download any file from some external website.
- These datasets are useful to quickly illustrate the behavior of the various algorithms implemented in scikit-learn. They are however often too small to be representative of real world machine learning tasks.
- Don't be fooled by the word "toy". These datasets are powerful and serve as a strong starting point for learning ML
- These toy datasets are broadly categorised into two types:
- Regression
    - Boston house prices dataset
    - Diabetes dataset
    - Linnerud dataset (Multi-output Regression)
    -
- Classification
    - Iris plants dataset
    - Optical recognition of handwritten digits dataset
    - Wine recognition dataset
    - Breast cancer wisconsin (diagnostic) dataset¶

(ii) Real World Datasets (`datasets.fetch_XXX()` methods)

https://scikit-learn.org/stable/datasets/real_world.html (https://scikit-learn.org/stable/datasets/real_world.html)

- Scikit-learn provides tools to load larger datasets, downloading them if necessary.
- Regression
  - California Housing dataset
- Classification
  - The Olivetti faces dataset
  - The 20 newsgroups text dataset
  - The Labeled Faces in the Wild face recognition dataset
  - Forest covertypes
  - RCV1 dataset
  - Kddcup 99 dataset

(iii) Downloading Datasets from Public ML Repositories (`fetch_XXX()` and `fetch_openml()` methods)

- Kaggle: https://www.kaggle.com/datasets (https://www.kaggle.com/datasets)
- UCI ML Repository: https://archive.ics.uci.edu/ml/index.php (https://archive.ics.uci.edu/ml/index.php)
- OpenML Repository: https://www.openml.org/ (https://www.openml.org/)

(iv) Random Sample Generators

- In addition, scikit-learn includes various random sample generators that can be used to build artificial datasets of controlled size and complexity.
- These generators produce a matrix of features and corresponding discrete targets.
  - `make_regression()`
  - `make_classification()`
  - `make_moons()`
  - `make_circles()`
  - `make_blobs()`

UCI ML Repository: https://archive.ics.uci.edu/ml/datasets/heart+Disease (https://archive.ics.uci.edu/ml/datasets/heart+Disease)

In [2]:
```
from sklearn import datasets
bunch_object = datasets.fetch_openml(name='heart-disease', version=1)
bunch_object.keys()
```

Out[2]:
```
dict_keys(['data', 'target', 'frame', 'categories', 'feature_names',
'target_names', 'DESCR', 'details', 'url'])
```

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# We want our plots to appear in the notebook
%matplotlib inline

df = pd.DataFrame(bunch_object.data, columns=bunch_object.feature_names)
df
```

Out[3]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	tar
0	63.0	1.0	3.0	145.0	233.0	1.0	0.0	150.0	0.0	2.3	0.0	0.0	1.0	
1	37.0	1.0	2.0	130.0	250.0	0.0	1.0	187.0	0.0	3.5	0.0	0.0	2.0	
2	41.0	0.0	1.0	130.0	204.0	0.0	0.0	172.0	0.0	1.4	2.0	0.0	2.0	
3	56.0	1.0	1.0	120.0	236.0	0.0	1.0	178.0	0.0	0.8	2.0	0.0	2.0	
4	57.0	0.0	0.0	120.0	354.0	0.0	1.0	163.0	1.0	0.6	2.0	0.0	2.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
298	57.0	0.0	0.0	140.0	241.0	0.0	1.0	123.0	1.0	0.2	1.0	0.0	3.0	
299	45.0	1.0	3.0	110.0	264.0	0.0	1.0	132.0	0.0	1.2	1.0	0.0	3.0	
300	68.0	1.0	0.0	144.0	193.0	1.0	1.0	141.0	0.0	3.4	1.0	2.0	3.0	
301	57.0	1.0	0.0	130.0	131.0	0.0	1.0	115.0	1.0	1.2	1.0	1.0	3.0	
302	57.0	0.0	1.0	130.0	236.0	0.0	0.0	174.0	0.0	0.0	1.0	1.0	2.0	

303 rows × 14 columns

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.

2. Exploratory Data Analysis (EDA)

Since EDA has no real set methodolgy, the following is a short check list you might want to walk through:

1. What question(s) are you trying to solve (or prove wrong)?
2. What's missing from the data and how do you deal with it?
3. Are there any outliers and how to treat them?
4. What kind of data do you have and how do you treat different types?
5. How can you add, change or remove features to get more out of your data?
6. How your data is distributed and the correlation between different variables?

In [4]:
```
df.describe()
```

Out[4]:

	age	sex	cp	trestbps	chol	fbs	restecg	
count	303.000000	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	0.528053	149
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202

In [5]:
```
df['sex'].value_counts()
```

Out[5]:
```
1.0    207
0.0     96
Name: sex, dtype: int64
```

In [6]:
```
df['cp'].value_counts()
```

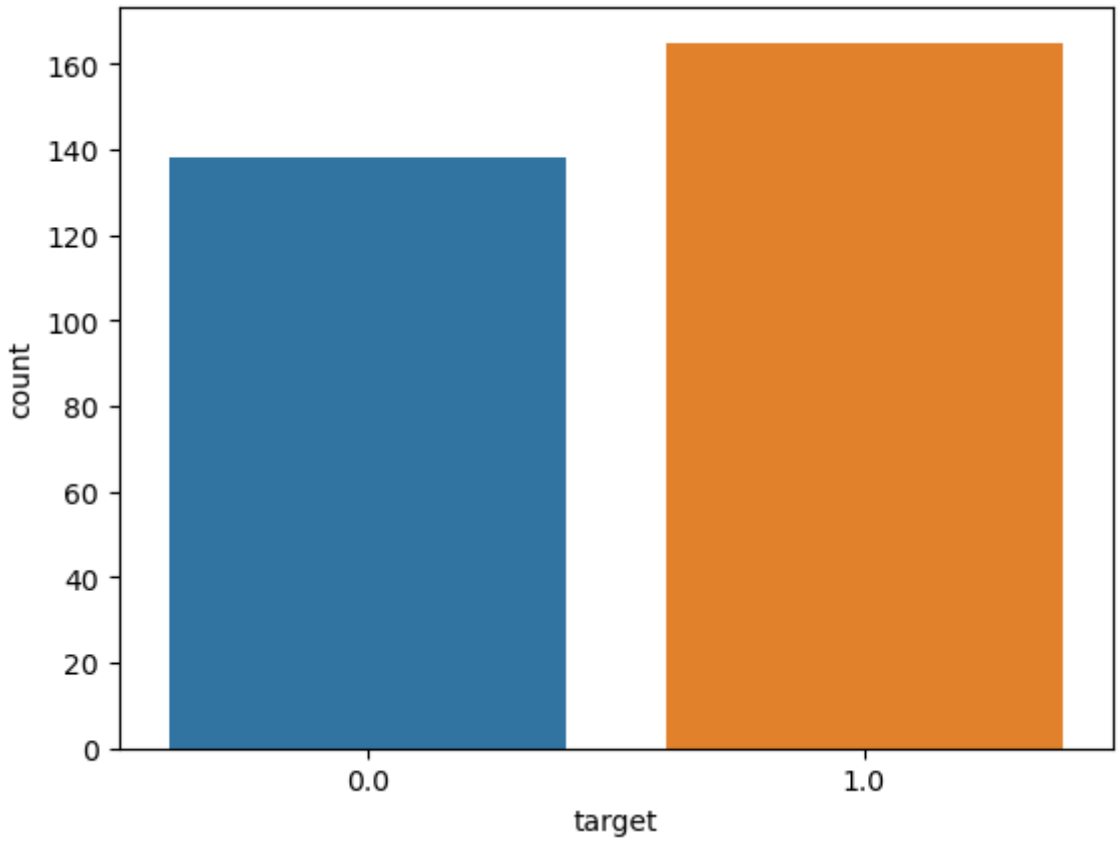
Out[6]:
```
0.0    143
2.0     87
1.0     50
3.0     23
Name: cp, dtype: int64
```

In [7]:
```
df['target'].value_counts()
```

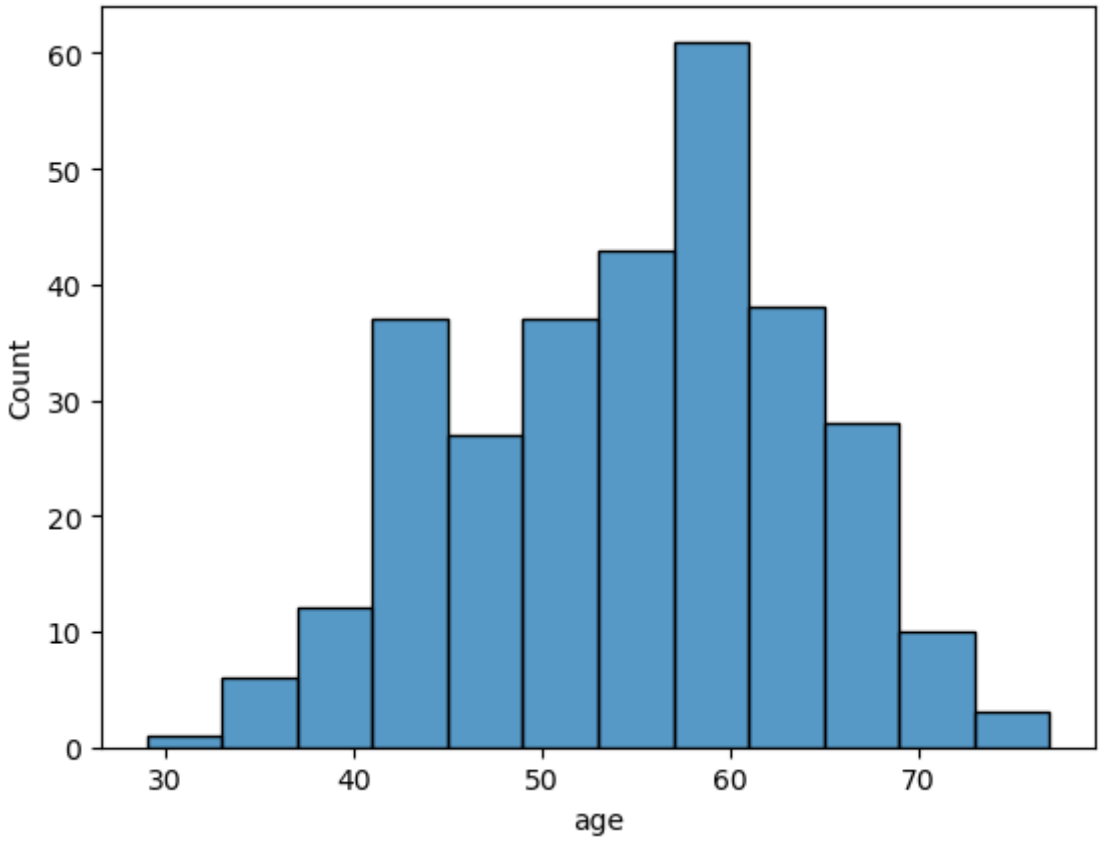
Out[7]:
```
1.0    165
0.0    138
Name: target, dtype: int64
```

In [8]:
```
sns.countplot(data=df, x='target')
```

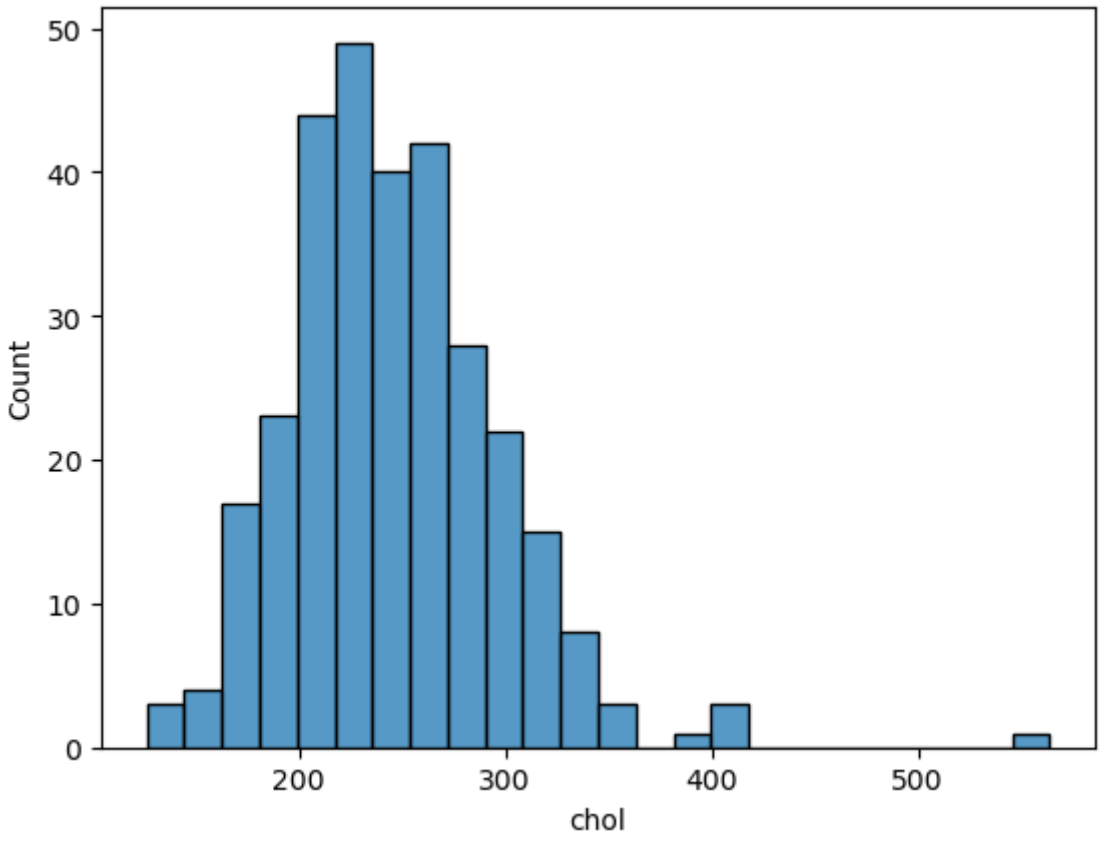
Out[8]:
```
<AxesSubplot:xlabel='target', ylabel='count'>
```



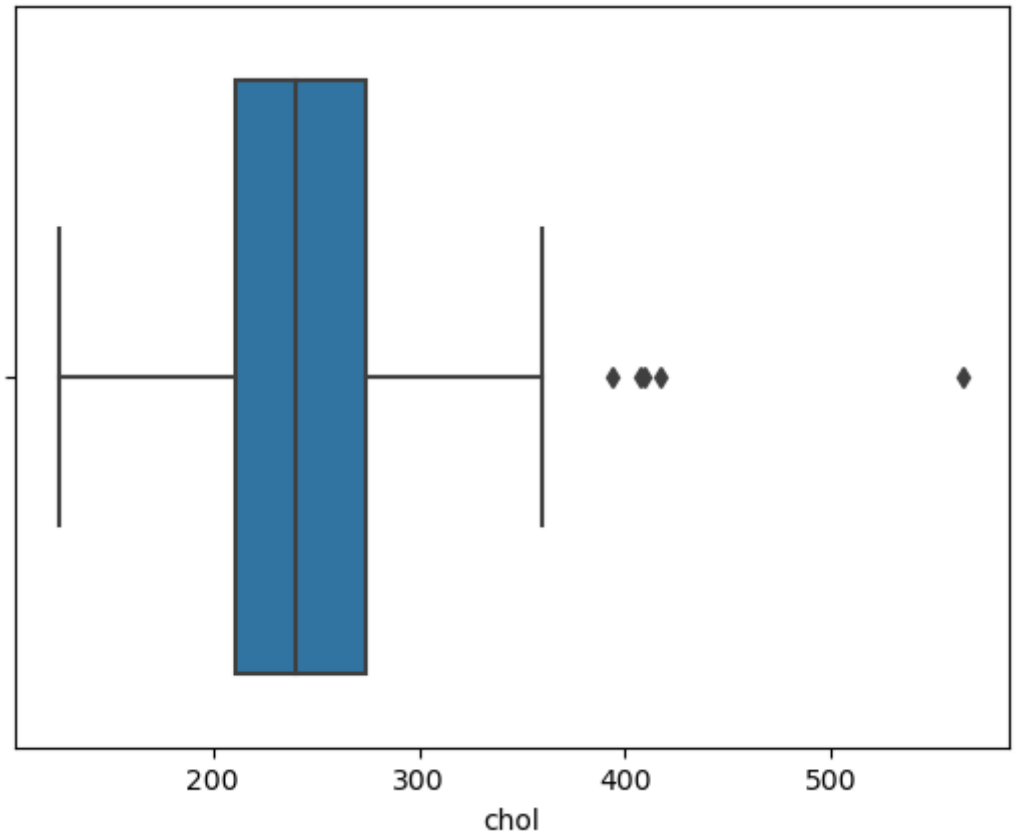
In [9]:
```
sns.histplot(data=df, x='age');
```



In [10]:
```
sns.histplot(data=df, x='chol');
```



In [11]:
```
sns.boxplot(data=df, x='chol');
```



In [12]:
```
df.info()
```

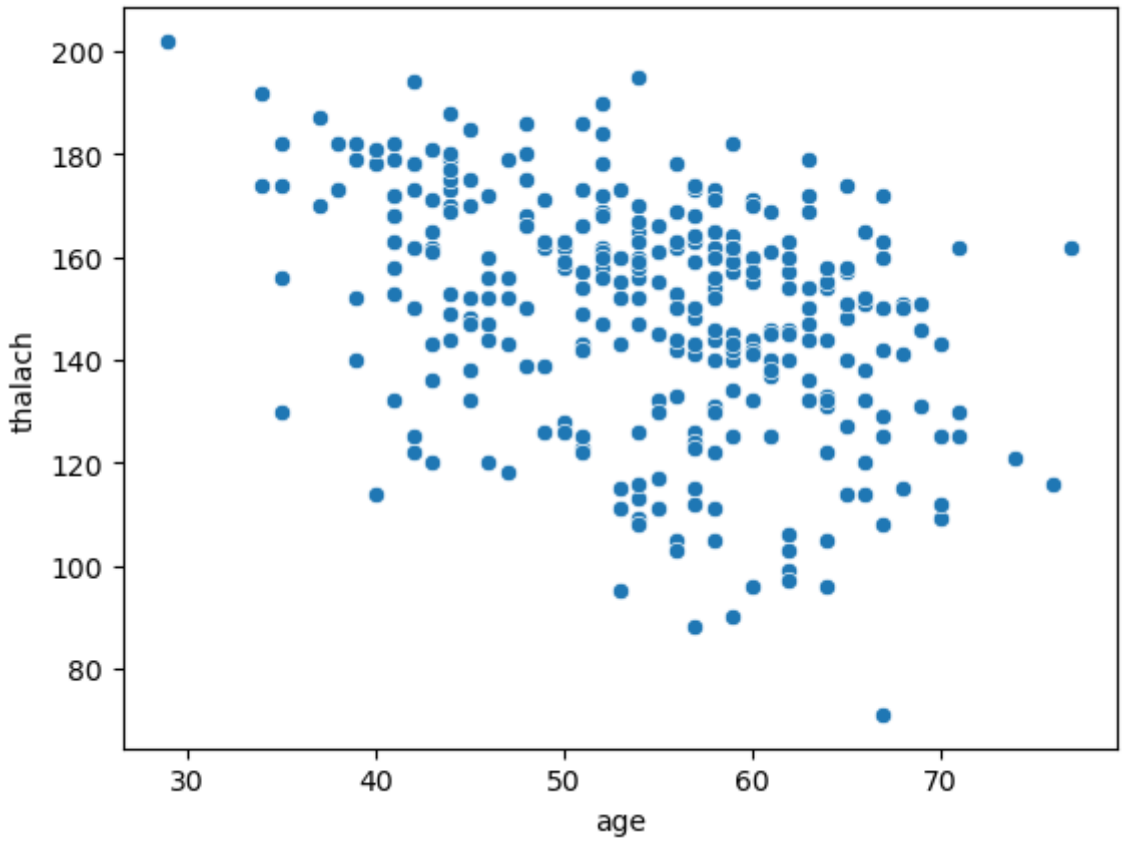
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    float64
 1   sex       303 non-null    float64
 2   cp        303 non-null    float64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

In [13]:
```
df.head()
```

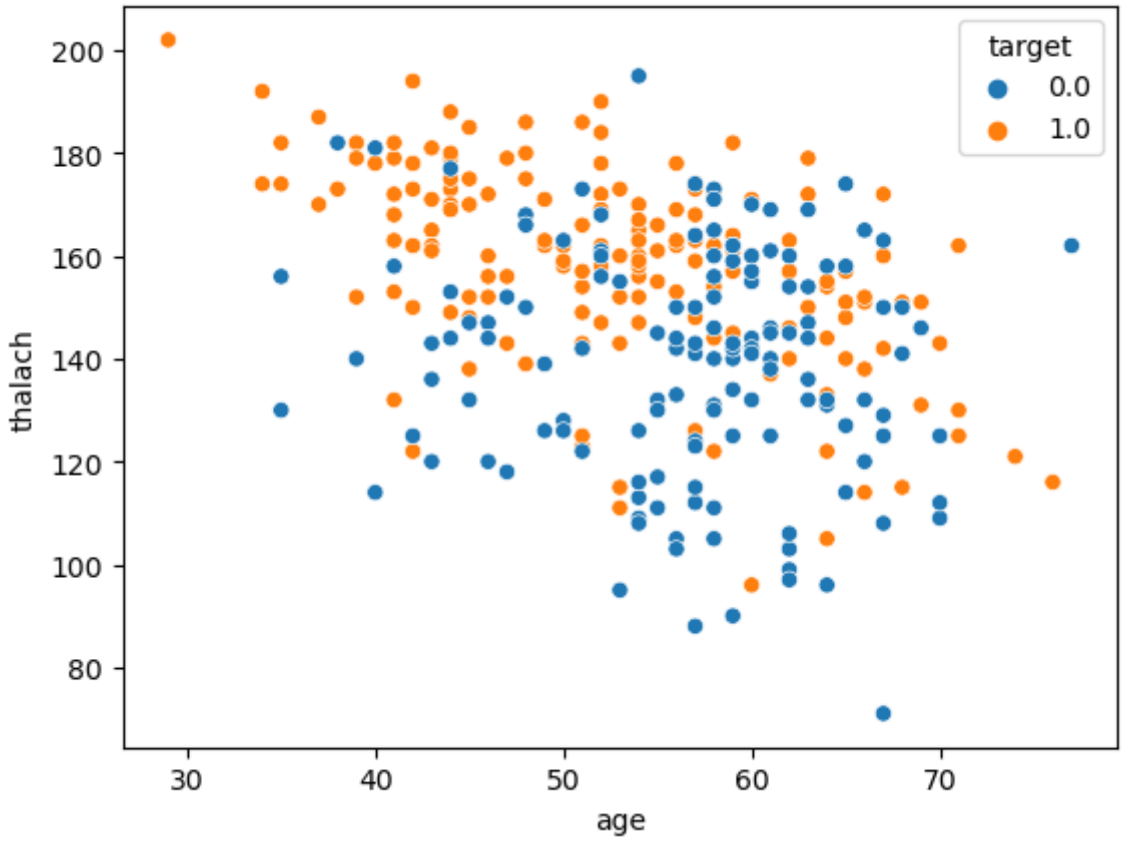
Out[13]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	targe
0	63.0	1.0	3.0	145.0	233.0	1.0	0.0	150.0	0.0	2.3	0.0	0.0	1.0	1.(
1	37.0	1.0	2.0	130.0	250.0	0.0	1.0	187.0	0.0	3.5	0.0	0.0	2.0	1.(
2	41.0	0.0	1.0	130.0	204.0	0.0	0.0	172.0	0.0	1.4	2.0	0.0	2.0	1.(
3	56.0	1.0	1.0	120.0	236.0	0.0	1.0	178.0	0.0	0.8	2.0	0.0	2.0	1.(
4	57.0	0.0	0.0	120.0	354.0	0.0	1.0	163.0	1.0	0.6	2.0	0.0	2.0	1.(

In [14]:
```
sns.scatterplot(data=df, x='age', y='thalach');
```



In [15]:
```
sns.scatterplot(data=df, x='age', y='thalach', hue='target');
```



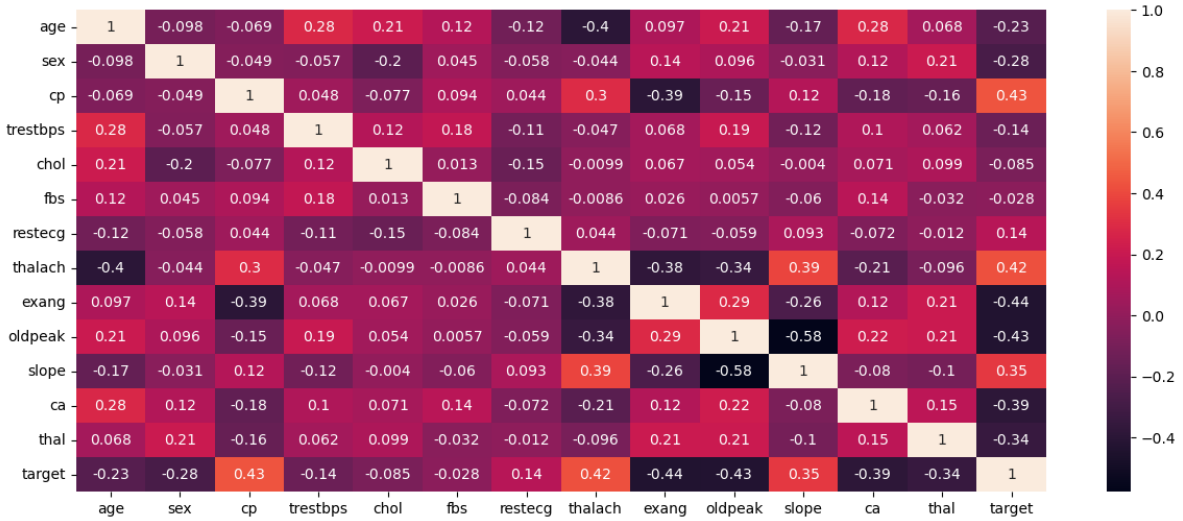
In [16]:
```
df.corr()
```

Out[16]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach
age	1.000000	-0.098447	-0.068653	0.279351	0.213678	0.121308	-0.116211	-0.398522
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.044020
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.295762
trestbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.046698
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.009940
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.008567
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.044123
thalach	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.000000
exang	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.378812
oldpeak	0.210013	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.344187
slope	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.386784
ca	0.276326	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.213177
thal	0.068001	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.096439
target	-0.225439	-0.280937	0.433798	-0.144931	-0.085239	-0.028046	0.137230	0.421741

In [17]:
```
plt.figure(figsize=(15,6))
sns.heatmap(df.corr(), annot=True)
```

Out[17]: <AxesSubplot:>



3. Model Training

Do a Train-Test Split:

In [18]:
```
df
```

Out[18]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	tar
0	63.0	1.0	3.0	145.0	233.0	1.0	0.0	150.0	0.0	2.3	0.0	0.0	1.0	
1	37.0	1.0	2.0	130.0	250.0	0.0	1.0	187.0	0.0	3.5	0.0	0.0	2.0	
2	41.0	0.0	1.0	130.0	204.0	0.0	0.0	172.0	0.0	1.4	2.0	0.0	2.0	
3	56.0	1.0	1.0	120.0	236.0	0.0	1.0	178.0	0.0	0.8	2.0	0.0	2.0	
4	57.0	0.0	0.0	120.0	354.0	0.0	1.0	163.0	1.0	0.6	2.0	0.0	2.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
298	57.0	0.0	0.0	140.0	241.0	0.0	1.0	123.0	1.0	0.2	1.0	0.0	3.0	
299	45.0	1.0	3.0	110.0	264.0	0.0	1.0	132.0	0.0	1.2	1.0	0.0	3.0	
300	68.0	1.0	0.0	144.0	193.0	1.0	1.0	141.0	0.0	3.4	1.0	2.0	3.0	
301	57.0	1.0	0.0	130.0	131.0	0.0	1.0	115.0	1.0	1.2	1.0	1.0	3.0	
302	57.0	0.0	1.0	130.0	236.0	0.0	0.0	174.0	0.0	0.0	1.0	1.0	2.0	

303 rows × 14 columns

In [19]:
```
from sklearn.model_selection import train_test_split

X = df.drop('target', axis=1)
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
len(X_train), len(y_train), len(X_test), len(y_test)
```

Out[19]:
```
(242, 242, 61, 61)
```

Scale the Data:

In [20]:
```
from sklearn.preprocessing import StandardScaler
scalar = StandardScaler()
scalar.fit(X_train)
scaled_X_train = scalar.transform(X_train)
scaled_X_test = scalar.transform(X_test)
len(scaled_X_train), len(scaled_X_test)
```

Out[20]:
```
(242, 61)
```

In [21]:
```
scaled_X_train
```

Out[21]:
```
array([[-1.33467373,  0.67015058, -0.95458269, ...,  1.04349839,
        -0.70898265, -0.48150478],
       [ 0.69176014, -1.49220195, -0.95458269, ..., -0.59628479,
        -0.70898265,  1.11471655],
       [ 0.80433979, -1.49220195, -0.95458269, ...,  1.04349839,
        -0.70898265, -0.48150478],
       ...,
       [ 0.69176014,  0.67015058,  0.93893379, ..., -0.59628479,
        -0.70898265, -0.48150478],
       [ 0.69176014,  0.67015058,  1.88569204, ..., -0.59628479,
         1.26313001, -0.48150478],
       [ 0.2414415 ,  0.67015058,  0.93893379, ...,  1.04349839,
         0.27707368,  1.11471655]])
```

In [22]:
```
scaled_X_train[1].mean()
```

Out[22]:
```
-0.012808298234685767
```

In [23]:
```
scaled_X_train[1].std()
```

Out[23]:
```
0.9146354242184234
```

Instentiate and train the `LogisticRegression()` model

In [24]:
```
from sklearn.linear_model import LogisticRegression

# Instentiate a model and fit it to training data with default hyperpara
model = LogisticRegression()
model.fit(scaled_X_train, y_train)
```

Out[24]:
```
LogisticRegression()
```

In [25]:
```
model.coef_
```

Out[25]:
```
array([[-0.03949332, -0.75234087,  0.75877113, -0.32921334, -0.1039029
6,
         0.03871837,  0.30406442,  0.63910927, -0.29446201, -0.6386831
4,
         0.22677787, -0.73164988, -0.52449106]])
```

In [26]:
```
model.intercept_
```

Out[26]:
```
array([0.17833406])
```

In [27]:
```
# Display probabilities of all 61 tes t points
probs = model.predict_proba(scaled_X_test)
probs
```

Out[27]:
```
array([[0.00815874, 0.99184126],
       [0.11904725, 0.88095275],
       [0.02394178, 0.97605822],
       [0.74461485, 0.25538515],
       [0.68620514, 0.31379486],
       [0.9626643 , 0.0373357 ],
       [0.33800871, 0.66199129],
       [0.03318872, 0.96681128],
       [0.68017285, 0.31982715],
       [0.07128913, 0.92871087],
       [0.1785114 , 0.8214886 ],
       [0.9907996 , 0.0092004 ],
       [0.9103587 , 0.0896413 ],
       [0.25066682, 0.74933318],
       [0.04469425, 0.95530575],
       [0.48843381, 0.51156619],
       [0.11135776, 0.88864224],
       [0.30970563, 0.69029437],
       [0.66644289, 0.33355711],
       [0.99622629, 0.00377371],
       [0.00864235, 0.99135765],
       [0.9794142 , 0.0205858 ],
       [0.41305354, 0.58694646],
       [0.92827737, 0.07172263],
       [0.98612426, 0.01387574],
       [0.26777933, 0.73222067],
       [0.99233132, 0.00766868],
       [0.1145202 , 0.8854798 ],
       [0.35361946, 0.64638054],
       [0.99413058, 0.00586942],
       [0.46920163, 0.53079837],
       [0.05538591, 0.94461409],
       [0.02855026, 0.97144974],
       [0.00310133, 0.99689867],
       [0.12845962, 0.87154038],
       [0.02348819, 0.97651181],
       [0.45813923, 0.54186077],
       [0.99308552, 0.00691448],
       [0.07806954, 0.92193046],
       [0.08498948, 0.91501052],
       [0.00872771, 0.99127229],
       [0.0425679 , 0.9574321 ],
       [0.94649972, 0.05350028],
       [0.81404903, 0.18595097],
       [0.21630377, 0.78369623],
       [0.92753576, 0.07246424],
       [0.08666053, 0.91333947],
       [0.30546225, 0.69453775],
       [0.50637838, 0.49362162],
       [0.70970727, 0.29029273],
       [0.68159018, 0.31840982],
       [0.95983464, 0.04016536],
       [0.28815894, 0.71184106],
       [0.98921981, 0.01078019],
       [0.22428716, 0.77571284],
       [0.04796412, 0.95203588],
       [0.12789335, 0.87210665],
       [0.37468455, 0.62531545],
       [0.35360505, 0.64639495],
```

```
       [0.06516502, 0.93483498],
       [0.00647783, 0.99352217]])
```

In [28]:
```
# Ignore Class 0 and consider class 1 only, i.e., the probability of bel
y_prob = probs[:,1]
y_prob
```

Out[28]:
```
array([0.99184126, 0.88095275, 0.97605822, 0.25538515, 0.31379486,
       0.0373357 , 0.66199129, 0.96681128, 0.31982715, 0.92871087,
       0.8214886 , 0.0092004 , 0.0896413 , 0.74933318, 0.95530575,
       0.51156619, 0.88864224, 0.69029437, 0.33355711, 0.00377371,
       0.99135765, 0.0205858 , 0.58694646, 0.07172263, 0.01387574,
       0.73222067, 0.00766868, 0.8854798 , 0.64638054, 0.00586942,
       0.53079837, 0.94461409, 0.97144974, 0.99689867, 0.87154038,
       0.97651181, 0.54186077, 0.00691448, 0.92193046, 0.91501052,
       0.99127229, 0.9574321 , 0.05350028, 0.18595097, 0.78369623,
       0.07246424, 0.91333947, 0.69453775, 0.49362162, 0.29029273,
       0.31840982, 0.04016536, 0.71184106, 0.01078019, 0.77571284,
       0.95203588, 0.87210665, 0.62531545, 0.64639495, 0.93483498,
       0.99352217])
```

In [29]:
```
# Threshold value of Probability
# Let us map all probabilities >= 0.5 to a discrete value of 1 and else
output = np.array([1.0 if p>=0.5 else 0.0 for p in y_prob])
output
```

Out[29]:
```
array([1., 1., 1., 0., 0., 0., 1., 1., 0., 1., 1., 0., 0., 1., 1., 1.,
       1.,
       1., 0., 0., 1., 0., 1., 0., 0., 1., 0., 1., 1., 0., 1., 1., 1.,
       1.,
       1., 1., 1., 0., 1., 1., 1., 1., 0., 0., 1., 0., 1., 1., 0., 0.,
       0.,
       0., 1., 0., 1., 1., 1., 1., 1., 1., 1.])
```

In [30]:
```
# Sklearn uses a default threshold value of 0.5 and outputs a discrete v
y_pred = model.predict(scaled_X_test)
y_pred
```

Out[30]:
```
array([1., 1., 1., 0., 0., 0., 1., 1., 0., 1., 1., 0., 0., 1., 1., 1.,
       1.,
       1., 0., 0., 1., 0., 1., 0., 0., 1., 0., 1., 1., 0., 1., 1., 1.,
       1.,
       1., 1., 1., 0., 1., 1., 1., 1., 0., 0., 1., 0., 1., 1., 0., 0.,
       0.,
       0., 1., 0., 1., 1., 1., 1., 1., 1., 1.])
```

- By `increasing` the threshold from 0.5 `False Negative` rate increases
- By `decreasing` the threshold from 0.5 `False Positive` rate increases

In case of two input features (previous sessions), we also visually plotted the decision boundry which was a line. But for this dataset (having thirteen input features) the decision boundry will be a hyperplane, which is difficult to visualize.

4. Evaluation Metrics for Classification Models

Evaluation allows us to check how well our trained model is performing on data that has never been used for training

In [31]:
```
from sklearn.metrics import SCORERS
sorted(SCORERS)
```

Out[31]:
```
['accuracy',
 'adjusted_mutual_info_score',
 'adjusted_rand_score',
 'average_precision',
 'balanced_accuracy',
 'completeness_score',
 'explained_variance',
 'f1',
 'f1_macro',
 'f1_micro',
 'f1_samples',
 'f1_weighted',
 'fowlkes_mallows_score',
 'homogeneity_score',
 'jaccard',
 'jaccard_macro',
 'jaccard_micro',
 'jaccard_samples',
 'jaccard_weighted',
 'max_error',
 'mutual_info_score',
 'neg_brier_score',
 'neg_log_loss',
 'neg_mean_absolute_error',
 'neg_mean_absolute_percentage_error',
 'neg_mean_gamma_deviance',
 'neg_mean_poisson_deviance',
 'neg_mean_squared_error',
 'neg_mean_squared_log_error',
 'neg_median_absolute_error',
 'neg_root_mean_squared_error',
 'normalized_mutual_info_score',
 'precision',
 'precision_macro',
 'precision_micro',
 'precision_samples',
 'precision_weighted',
 'r2',
 'rand_score',
 'recall',
 'recall_macro',
 'recall_micro',
 'recall_samples',
 'recall_weighted',
 'roc_auc',
 'roc_auc_ovo',
 'roc_auc_ovo_weighted',
 'roc_auc_ovr',
 'roc_auc_ovr_weighted',
 'top_k_accuracy',
 'v_measure_score']
```

In [32]:
```
y_test.values
```

Out[32]:
```
array([1., 1., 1., 0., 0., 0., 0., 1., 0., 1., 1., 0., 0., 1., 1., 1.,
1.,
       1., 0., 0., 1., 0., 1., 0., 0., 1., 0., 1., 0., 0., 0., 1., 1.,
1.,
       1., 1., 0., 0., 0., 1., 1., 1., 0., 0., 1., 0., 0., 0., 1., 0.,
0.,
       0., 1., 0., 1., 1., 1., 1., 0., 1., 1.])
```

In [33]:
```
y_pred = model.predict(scaled_X_test)
y_pred
```

Out[33]:
```
array([1., 1., 1., 0., 0., 0., 1., 1., 0., 1., 1., 0., 0., 1., 1., 1.,
1.,
       1., 0., 0., 1., 0., 1., 0., 0., 1., 0., 1., 1., 0., 1., 1., 1.,
1.,
       1., 1., 1., 0., 1., 1., 1., 1., 0., 0., 1., 0., 1., 1., 0., 0.,
0.,
       0., 1., 0., 1., 1., 1., 1., 1., 1., 1.])
```

By comparing the Ground Labels (y) and Predicted Labels $(\hat{y})$ one can calculate the `Accuracy` of the model:

$$Accuracy = \frac{\text{Count of correct answers of Classifier}}{\text{Count of all Qs asked from Classifier}}$$

	Positive Class	Negative Class
Heart Disease	Heart Patient	Healthy
Email	SPAM	HAM
Tumor	Malignant	Benign
Transaction	Fraud	Not Fraud
Disease	Yes	No
Binary	1	0

Hello TP, TN, FP, FN

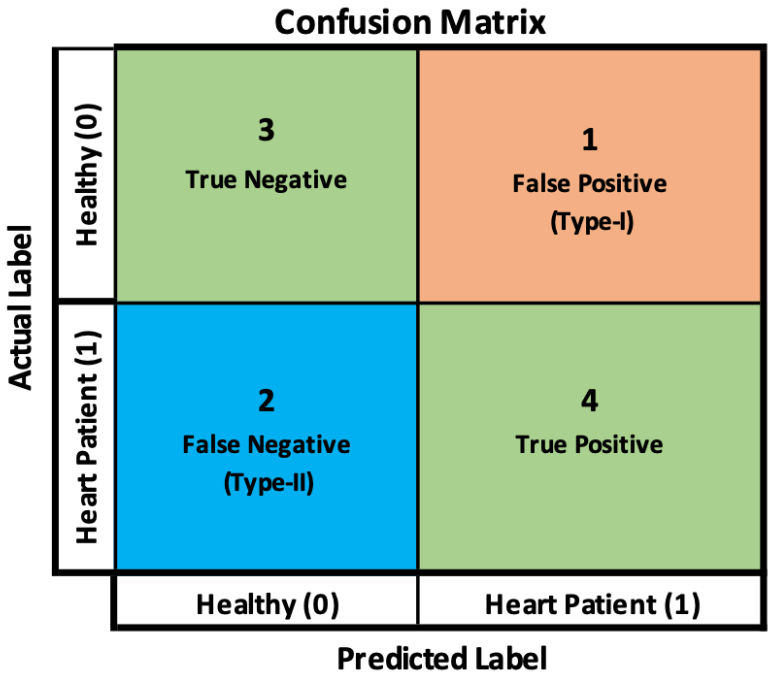
x1	x2	---	y	y_prob	y_pred	Remarks
-	-	-	0	[0.744, 0.255]	0	True Negative (TN)
-	-	-	0	[0.488, 0.511]	1	False Positive (FP) Type-I Error
-	-	-	0	[0.962 , 0.037]	0	True Negative (TN)
-	-	-	1	[0.084, 0.915]	1	True Positive (TP)
-	-	-	1	[0.928, 0.0717]	0	False Negative (FN) Type-II Error
-	-	-	0	[0.979 , 0.020]	0	True Negative (TN)
-	-	-	1	[0.2881, 0.7118]	1	True Positive (TP)
-	-	-	1	[0.946, 0.0535]	0	False Negative (FN) Type-II Error
-	-	-	1	[0.374, 0.625]	1	True Positive (TP)
-	-	-	1	[0.0064, 0.993]	1	True Positive (TP)

- **True Positive (TP):** Number of data points with `y=1` and are classified as $\hat{y} = 1$
- **True Negative (TN):** Number of data points with `y=0` and are classified as $\hat{y} = 0$
- **False Positive (FP):** Number of data points with `y=0` and are classified as $\hat{y} = 1$
- **False Negative (FN):** Number of data points with `y=1` and are classified as $\hat{y} = 0$

a. Confusion Matrix

- `confusion_matrix()` (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html)
- `confusionmatrixdisplay()` (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html)

A Confusion Matrix also called contingency matrix is a $n \times n$ matrix used to define the performance of classification algorithms (where n is the number of classes being predicted)

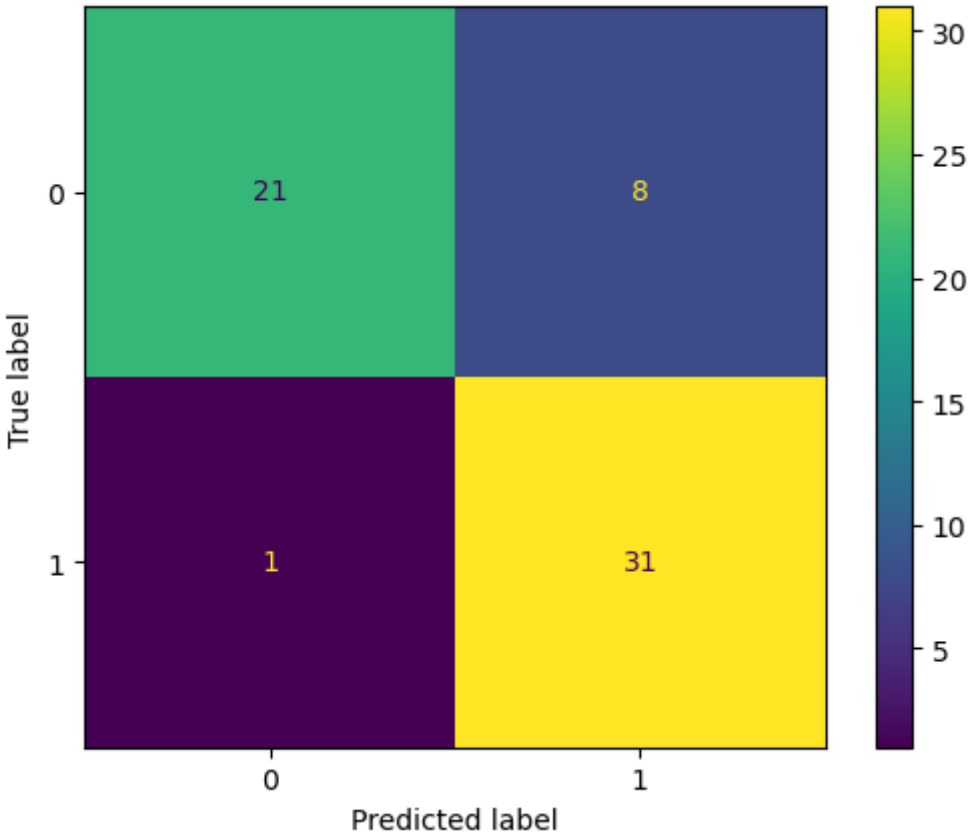


The Confusion Matrix is not confusing for humans

In [34]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

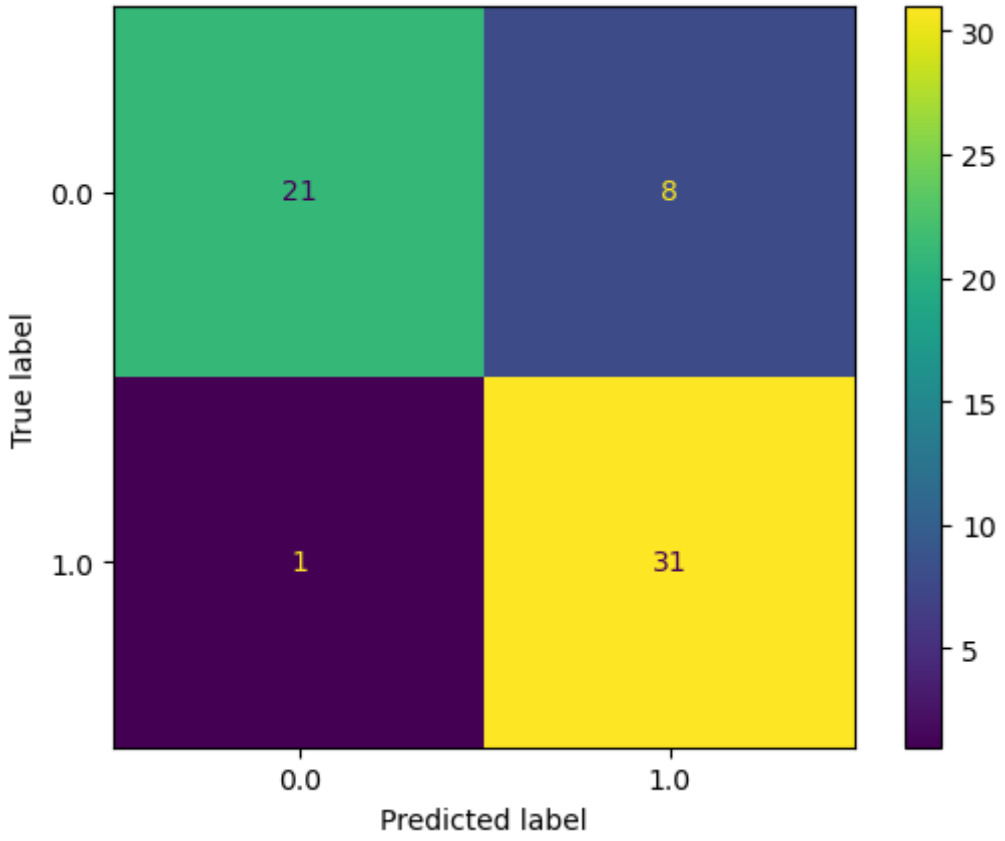
Out[34]:
```
array([[21,  8],
       [ 1, 31]])
```

In [35]:
```
from sklearn.metrics import ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred))
disp.plot();
```

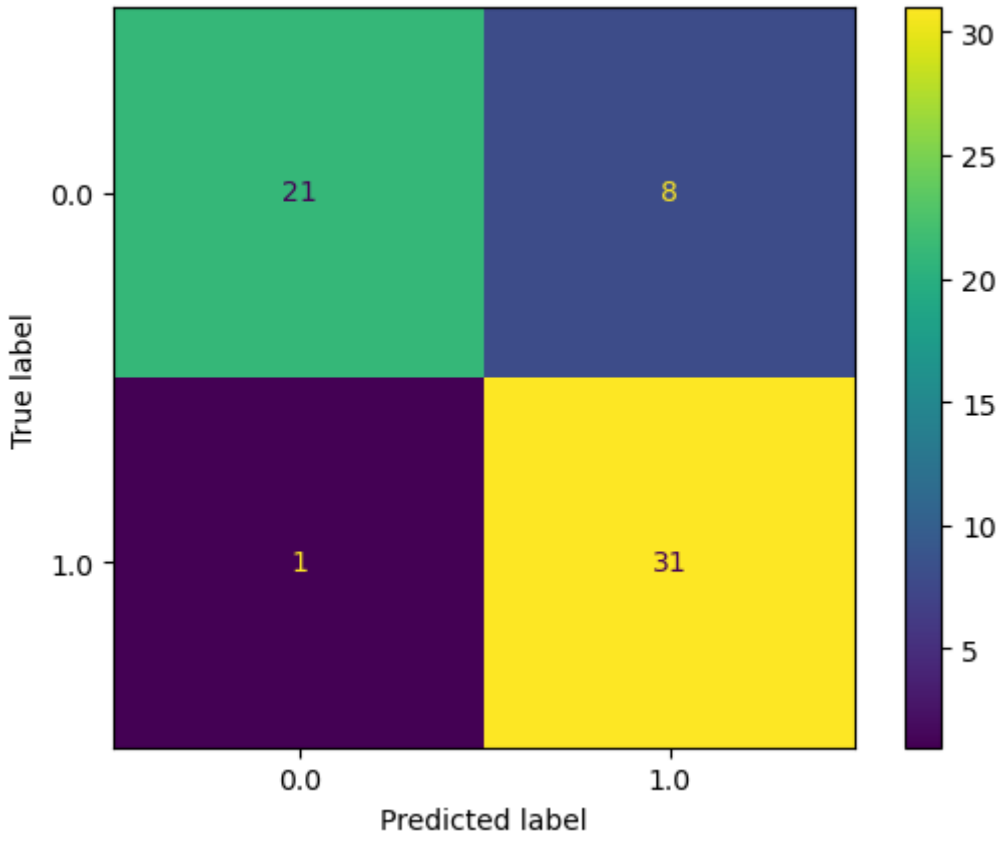


`ConfusionMatrixDisplay()` method has gone obsolete in Sklearn's version 1.2 onwards. So you should prefer using either the `ConfusionMatrixDisplay.from_predictions()` method or the `ConfusionMatrixDisplay.from_estimator()` method

In [36]:
```
ConfusionMatrixDisplay.from_predictions(y_test, y_pred);
```

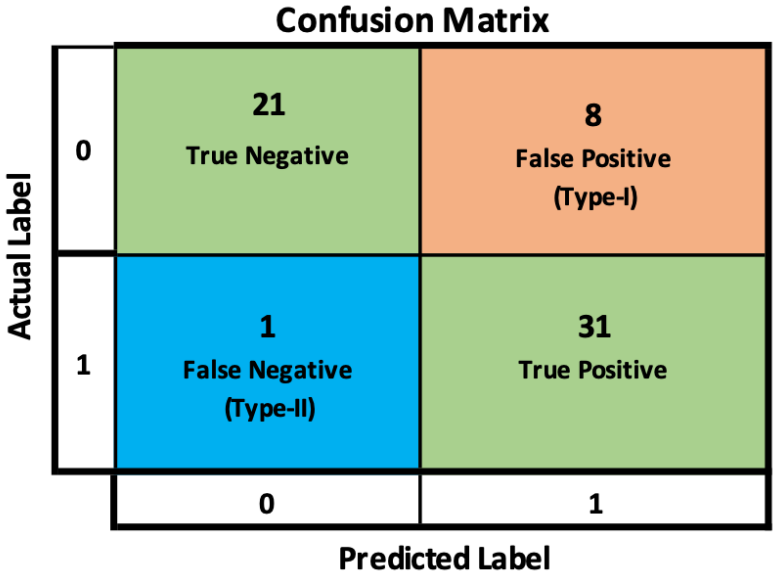


In [37]:
```
ConfusionMatrixDisplay.from_estimator(model, scaled_X_test, y_test);
```



b. Accuracy

accuracy_score() (https://scikit-



learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html#sklearn.metrics.accura

- **How frequently the model predicted correctly?**

$$Accuracy = \frac{\text{Count of correct answers of Classifier}}{\text{Count of all Qs asked from Classifier}}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Accuracy = \frac{31 + 21}{31 + 21 + 8 + 1} = \frac{52}{61} = 0.852$$

- **Accuracy score will be zero, if TP=TN=0 and Accuracy will be one, if FN=FP=0**

In [38]:
```
confusion_matrix(y_test,y_pred)
```

Out[38]:
```
array([[21,  8],
       [ 1, 31]])
```

In [39]:
```
np.reshape(confusion_matrix(y_test,y_pred), (-1))
```

Out[39]:
```
array([21,  8,  1, 31])
```

In [40]:
```
tn,fp,fn,tp = np.reshape(confusion_matrix(y_test,y_pred), (-1))
accuracy = (tp+tn)/(tp+tn+fp+fn)
print("Accuracy= ", accuracy)
```

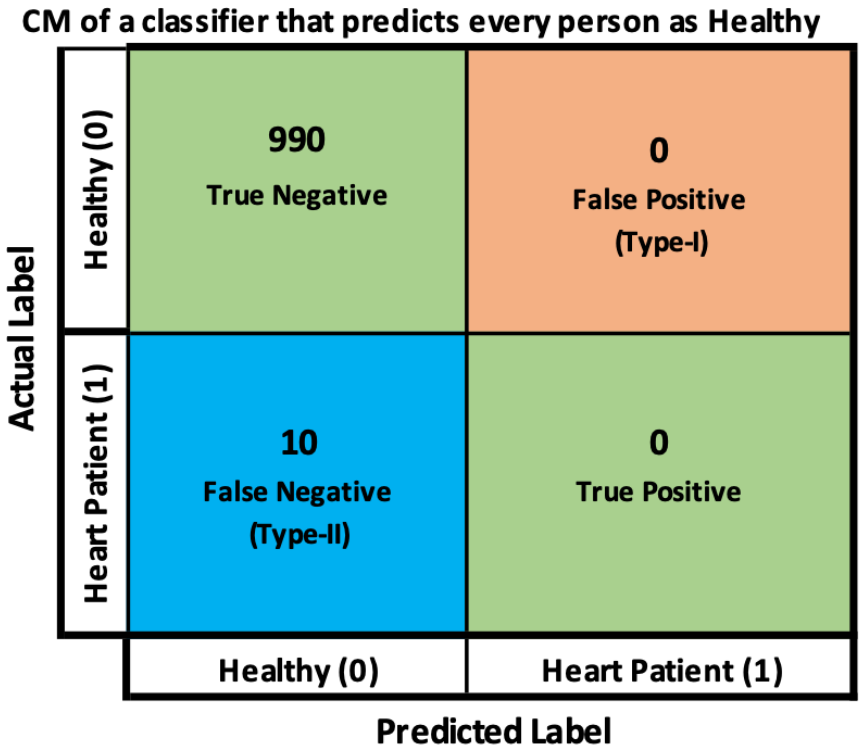
```
Accuracy=  0.8524590163934426
```

In [41]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

Out[41]:
```
0.8524590163934426
```

The Accuracy Paradox

- Accuracy does not prove to be a good evaluation metric in case of imbalance data sets



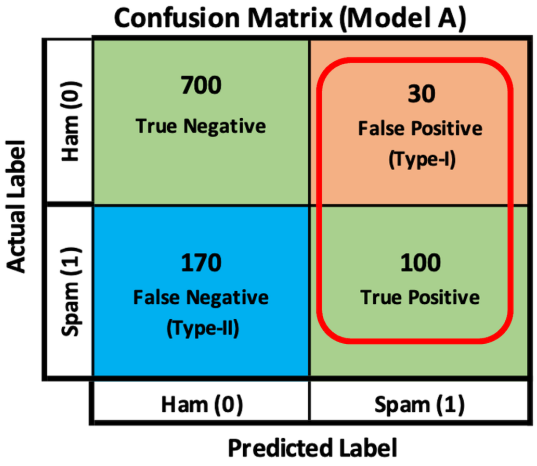
$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{0 + 990}{0 + 990 + 0 + 10} = 0.95$$

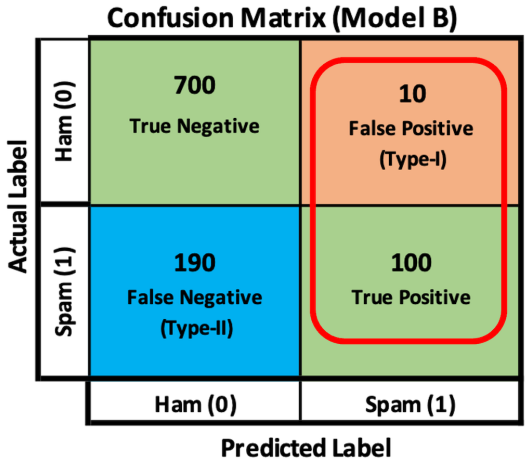
- A classifier that always predicts the majority class, in a highly

c. Precision

`precision_score()` (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html)

- **Out of all the times model predicted Positive, how many times was it correct?**
- **Precision evaluates a model based on False Positive (Type-I Error)**





`Accuracy = 800/1000 = 0.8`

`Accuracy = 800/1000 = 0.8`

`FP of Model A > FP of Model B`

`FN of Model A < FN of Model B`

$$\text{Precision} = \frac{\text{True Positives}}{\text{Predicted Positives}} = \frac{TP}{TP + FP}$$

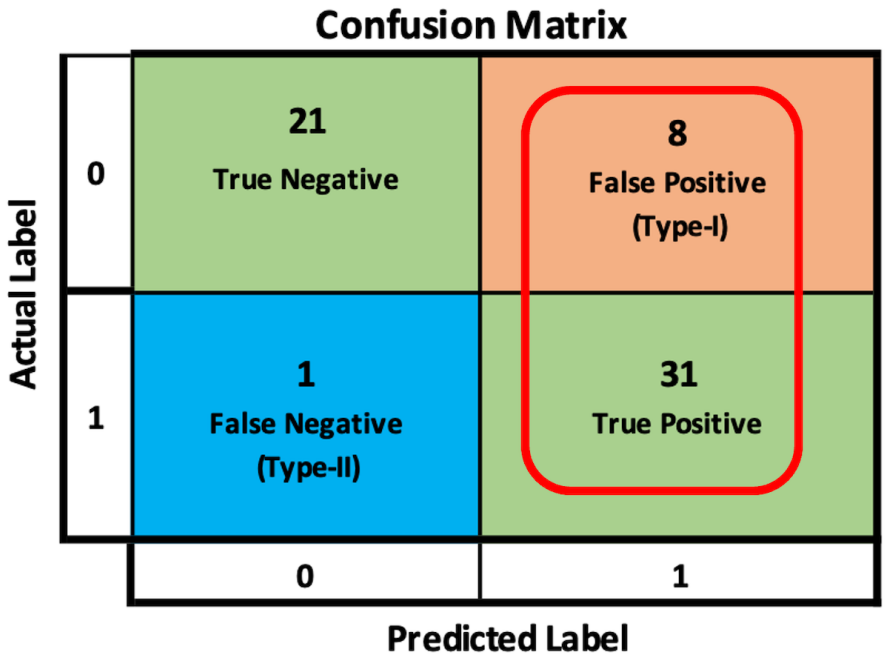
$$\text{Precision for Model A} = \frac{100}{100 + 130} = \frac{100}{130} = 0.769$$

$$\text{Precision for Model B} = \frac{100}{100 + 10} = \frac{100}{110} = 0.909$$

- **Since Precision score of Model B is greater than Model A, therefore, we select Model B**

So in case if you want to miniminze False Positives (Type-I Error), you use

Precision Score for Heart Disease Dataset



$$\text{Precision} = \frac{TP}{TP + FP} = \frac{31}{31 + 8} = \frac{31}{39} = 0.7948$$

- **Precision score will be zero, if TP=0 and Precision will be one, if FP=0**

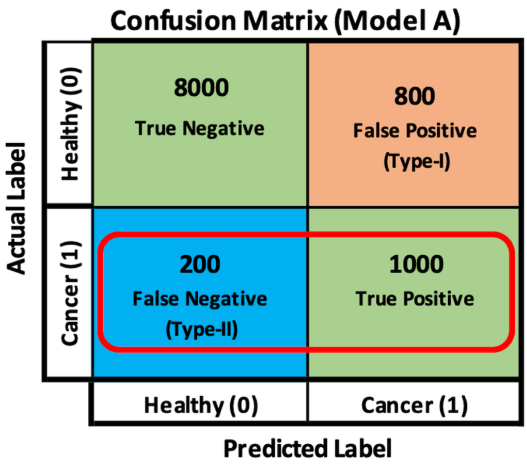
In [42]:
```
from sklearn.metrics import precision_score
precision_score(y_test, y_pred)
```

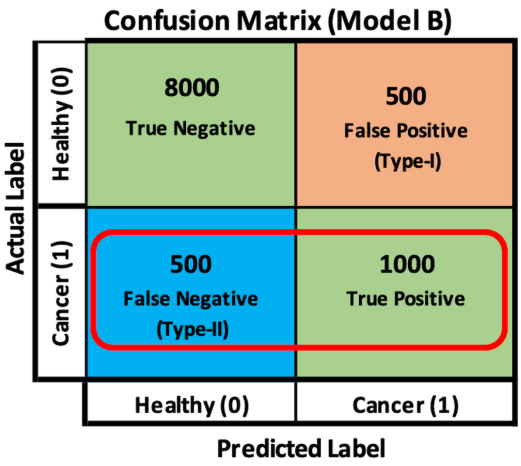
Out[42]:
```
0.7948717948717948
```

d. Recall (a.k.a Sensitivity)

`recall_score()` (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html)

- **Out of all the times actual class was Positive, how many times the model was correct?**
- **Recall evaluates a model based on False Negative (Type-II Error)**





`Accuracy = 9000/10000 = 0.9`

`Accuracy = 9000/10000 = 0.9`

`FP of Model A > FP of Model B`

`FN of Model A < FN of Model B`

$$\text{Recall} = \frac{\text{True Positives}}{\text{Real Positives}} = \frac{TP}{TP + FN}$$

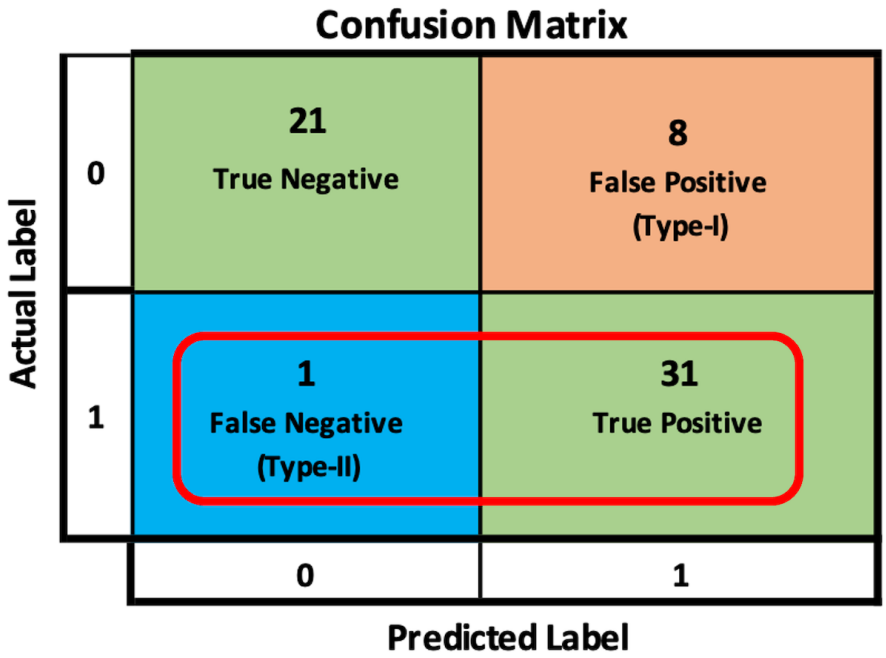
$$\text{Recall for Model A} = \frac{1000}{1000 + 200} = \frac{1000}{1200} = 0.83$$

$$\text{Recall for Model B} = \frac{1000}{1000 + 500} = \frac{1000}{1500} = 0.666$$

- **Since Recall score of Model A is greater than Model B, therefore, we select Model A**

So in case if you want to miniminze False Negatives (Type-II Error), you use Recall as evaluation metric of your model.

Recall Score for Heart Disease Dataset



$$\text{Recall} = \frac{TP}{TP + FN} = \frac{31}{31 + 1} = \frac{31}{32} = 0.968$$

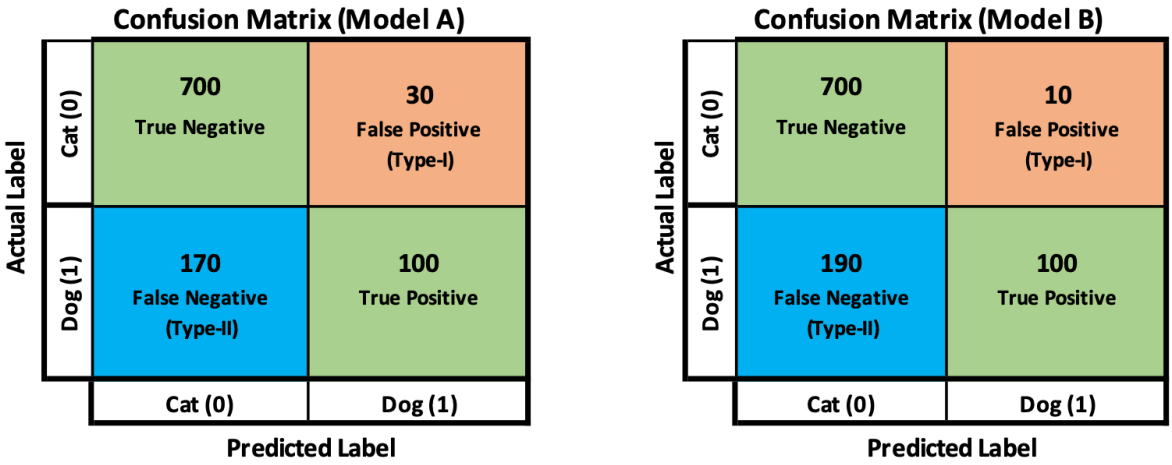
- **Recall score will be zero, if TP=0 and Recall will be one, if FN=0**

In [43]:
```
from sklearn.metrics import recall_score
recall_score(y_test, y_pred)
```

Out[43]: 0.96875

A Real 100 $ Question

To classify cats/dogs, should we go for a high precision model or a high recall model?



Accuracy = 800/1000 = 0.8

FP of Model A > FP of Model B

Accuracy = 800/1000 = 0.8

FN of Model A < FN of Model B

So in case if you want to miniminze False Negatives, you use Recall as evaluation metric of your model.

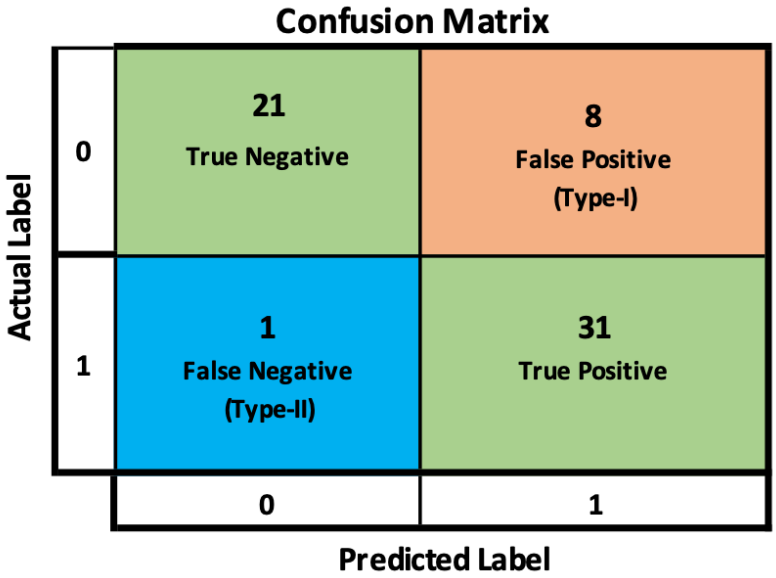
e. F-1 Score

- `f1_score()` (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html)
- `classification_report()` (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html)
- Precision and Recall alone do not solve the Accuracy Paradox. So, we combine both these metrics to create a new metric called F1 Score, which is harmonic mean of precision and recall.

- We use harmonic mean instead of arithmetic mean because harmonic mean punishes extreme values more. OR in simple words, the harmonic means goes to zero if either of the recall or precision ends up being zero

$$F1 = \frac{2}{\frac{1}{P} + \frac{1}{R}}$$

$$F1 = \frac{2PR}{P + R}$$



$$F1 = \frac{2 * 0.79487 * 0.96875}{0.79487 + 0.96875} = \frac{1.54}{1.76362} = 0.873$$

- F1 score will be high, if **both** precision and recall are high.
- F1 score will be low, if **any** of precision or recall are low.

In [44]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_scor
print("Accuracy: ", accuracy_score(y_test, y_pred))
print("Precision: ", precision_score(y_test, y_pred))
print("Recall: ", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
```

```
Accuracy:  0.8524590163934426
Precision:  0.7948717948717948
Recall:  0.96875
F1 Score: 0.8732394366197183
```

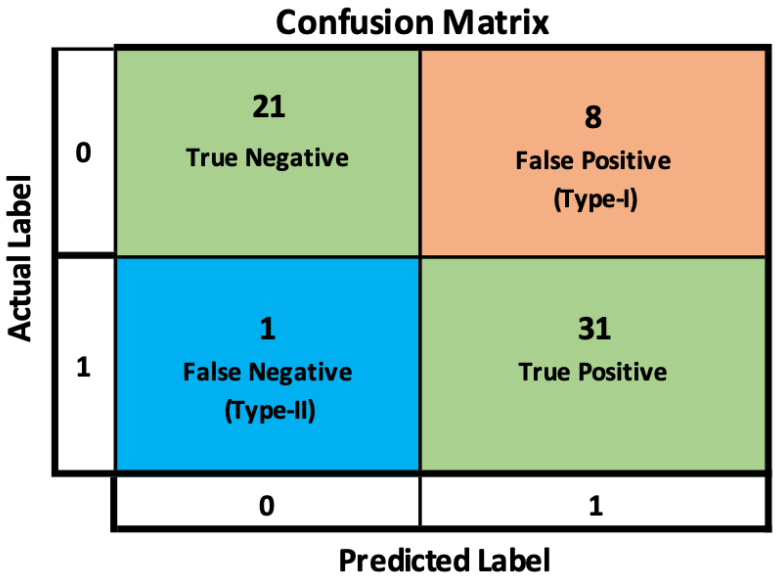
In [45]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.95      0.72      0.82        29
         1.0       0.79      0.97      0.87        32

    accuracy                           0.85        61
   macro avg       0.87      0.85      0.85        61
weighted avg       0.87      0.85      0.85        61
```

f. F-β Score

- The F-β score is the weighted harmonic mean of precision and recall.

- We can choose to favor precision or recall by using an interpolation weight α:



$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R}$$

$where \quad \beta = \frac{1-\alpha}{\alpha}$

- If β=1, then F-β is the same as F1 Score
- β -> 0 considers only `precision` , β -> +∞ considers only `recall`

In [46]:
```
from sklearn.metrics import fbeta_score
print("F-β Score (β=1): ", fbeta_score(y_test, y_pred,beta=1))
print("F-β Score (β=0.5): ", fbeta_score(y_test, y_pred,beta=0.5))
print("F-β Score (β=10): ", fbeta_score(y_test, y_pred,beta=10))
```

```
F-β Score (β=1):  0.8732394366197183
F-β Score (β=0.5):  0.8244680851063829
F-β Score (β=10):  0.9666563754245135
```

Summary of what we have done so far


Actual Total Negatives
Predicted Label
Predicted Total Positives

g. Precision-Recall Tradeoff

If you increase precision, it will reduce recall and vice versa.

So by changing the threshold value one can change the positive and negative predictions

In [47]:
```
probs = model.predict_proba(scaled_X_test)
probs
```

Out[47]:
```
array([[0.00815874, 0.99184126],
       [0.11904725, 0.88095275],
       [0.02394178, 0.97605822],
       [0.74461485, 0.25538515],
       [0.68620514, 0.31379486],
       [0.9626643 , 0.0373357 ],
       [0.33800871, 0.66199129],
       [0.03318872, 0.96681128],
       [0.68017285, 0.31982715],
       [0.07128913, 0.92871087],
       [0.1785114 , 0.8214886 ],
       [0.9907996 , 0.0092004 ],
       [0.9103587 , 0.0896413 ],
       [0.25066682, 0.74933318],
       [0.04469425, 0.95530575],
       [0.48843381, 0.51156619],
       [0.11135776, 0.88864224],
       [0.30970563, 0.69029437],
       [0.66644289, 0.33355711],
       [0.99622629, 0.00377371],
       [0.00864235, 0.99135765],
       [0.9794142 , 0.0205858 ],
       [0.41305354, 0.58694646],
       [0.92827737, 0.07172263],
       [0.98612426, 0.01387574],
       [0.26777933, 0.73222067],
       [0.99233132, 0.00766868],
       [0.1145202 , 0.8854798 ],
       [0.35361946, 0.64638054],
       [0.99413058, 0.00586942],
       [0.46920163, 0.53079837],
       [0.05538591, 0.94461409],
       [0.02855026, 0.97144974],
       [0.00310133, 0.99689867],
       [0.12845962, 0.87154038],
       [0.02348819, 0.97651181],
       [0.45813923, 0.54186077],
       [0.99308552, 0.00691448],
       [0.07806954, 0.92193046],
       [0.08498948, 0.91501052],
       [0.00872771, 0.99127229],
       [0.0425679 , 0.9574321 ],
       [0.94649972, 0.05350028],
       [0.81404903, 0.18595097],
       [0.21630377, 0.78369623],
       [0.92753576, 0.07246424],
       [0.08666053, 0.91333947],
       [0.30546225, 0.69453775],
       [0.50637838, 0.49362162],
       [0.70970727, 0.29029273],
       [0.68159018, 0.31840982],
       [0.95983464, 0.04016536],
       [0.28815894, 0.71184106],
       [0.98921981, 0.01078019],
       [0.22428716, 0.77571284],
       [0.04796412, 0.95203588],
       [0.12789335, 0.87210665],
       [0.37468455, 0.62531545],
       [0.35360505, 0.64639495],
```

```
       [0.06516502, 0.93483498],
       [0.00647783, 0.99352217]])
```

In [48]:
```
y_prob = probs[:,1]
y_prob
```

Out[48]:
```
array([0.99184126, 0.88095275, 0.97605822, 0.25538515, 0.31379486,
       0.0373357 , 0.66199129, 0.96681128, 0.31982715, 0.92871087,
       0.8214886 , 0.0092004 , 0.0896413 , 0.74933318, 0.95530575,
       0.51156619, 0.88864224, 0.69029437, 0.33355711, 0.00377371,
       0.99135765, 0.0205858 , 0.58694646, 0.07172263, 0.01387574,
       0.73222067, 0.00766868, 0.8854798 , 0.64638054, 0.00586942,
       0.53079837, 0.94461409, 0.97144974, 0.99689867, 0.87154038,
       0.97651181, 0.54186077, 0.00691448, 0.92193046, 0.91501052,
       0.99127229, 0.9574321 , 0.05350028, 0.18595097, 0.78369623,
       0.07246424, 0.91333947, 0.69453775, 0.49362162, 0.29029273,
       0.31840982, 0.04016536, 0.71184106, 0.01078019, 0.77571284,
       0.95203588, 0.87210665, 0.62531545, 0.64639495, 0.93483498,
       0.99352217])
```

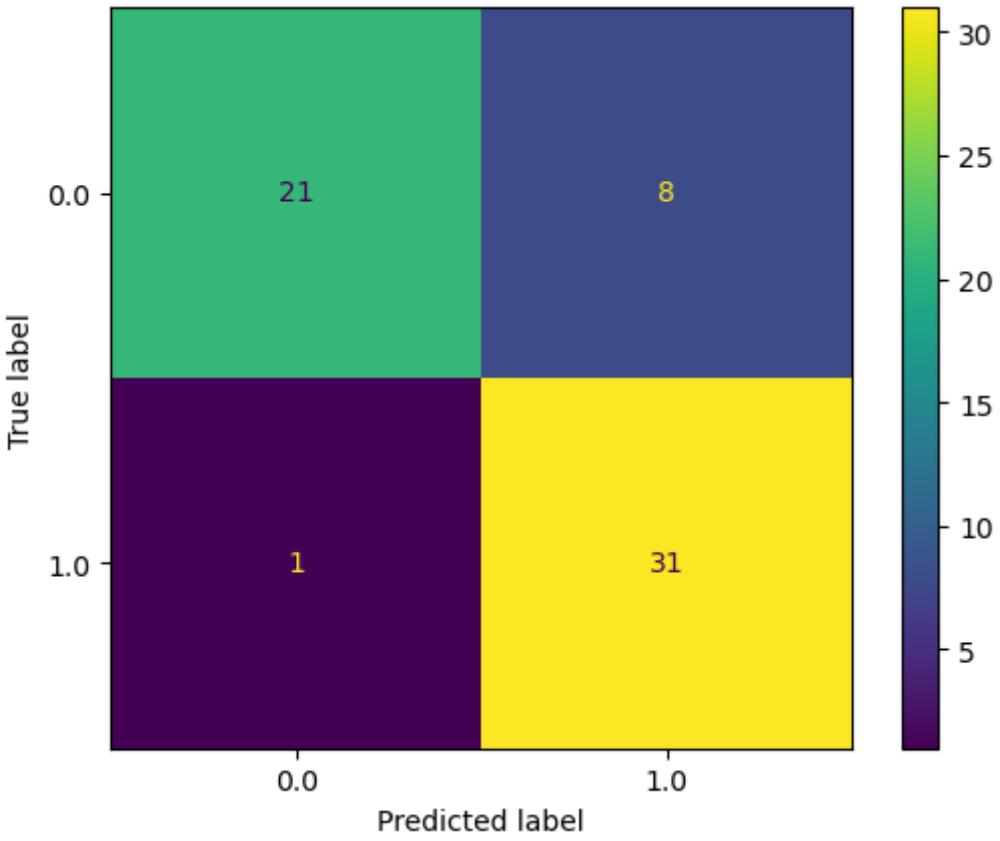
CM for Default Threshold of 0.5:

In [49]:
```
y_pred1 = np.array([1.0 if p>=0.5 else 0.0 for p in y_prob])
y_pred1
```

Out[49]:
```
array([1., 1., 1., 0., 0., 0., 1., 1., 0., 1., 1., 0., 0., 1., 1., 1.,
1.,
       1., 0., 0., 1., 0., 1., 0., 0., 1., 0., 1., 1., 0., 1., 1., 1.,
1.,
       1., 1., 1., 0., 1., 1., 1., 1., 0., 0., 1., 0., 1., 1., 0., 0.,
0.,
       0., 1., 0., 1., 1., 1., 1., 1., 1., 1.])
```

In [50]:
```
#disp = ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred1))
#disp.plot();
ConfusionMatrixDisplay.from_predictions(y_test, y_pred1);
print("Precision Score:", precision_score(y_test, y_pred1))
print("Recall Score:", recall_score(y_test, y_pred1))
```

```
Precision Score: 0.7948717948717948
Recall Score: 0.96875
```



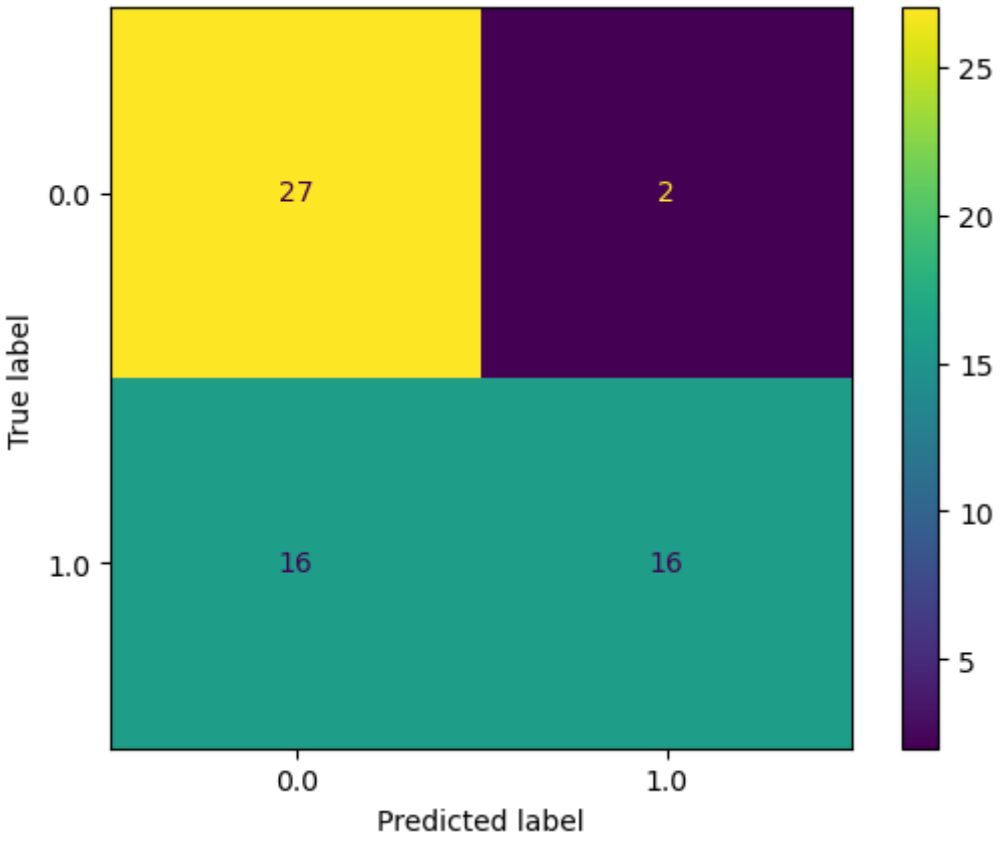
CM for High Precision and Low Recall Model (Minimize Type-I Error (FP):

In [51]:
```
y_pred2 = np.array([1 if p >= 0.9 else 0 for p in y_prob])
y_pred2
```

Out[51]:
```
array([1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0,
0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1])
```

In [52]:
```
#disp = ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred2))
#disp.plot();
ConfusionMatrixDisplay.from_predictions(y_test, y_pred2);
print("Precision Score:", precision_score(y_test, y_pred2))
print("Recall Score:", recall_score(y_test, y_pred2))
```

```
Precision Score: 0.8888888888888888
Recall Score: 0.5
```



Note:

- `FP` are `reduced` from 8 to 2, while `FN` are `increased` from 1 to 16
- `Precision` is `increased` from 0.79 to 0.88, while `Recall` is `reduced` from 0.96 to 0.5

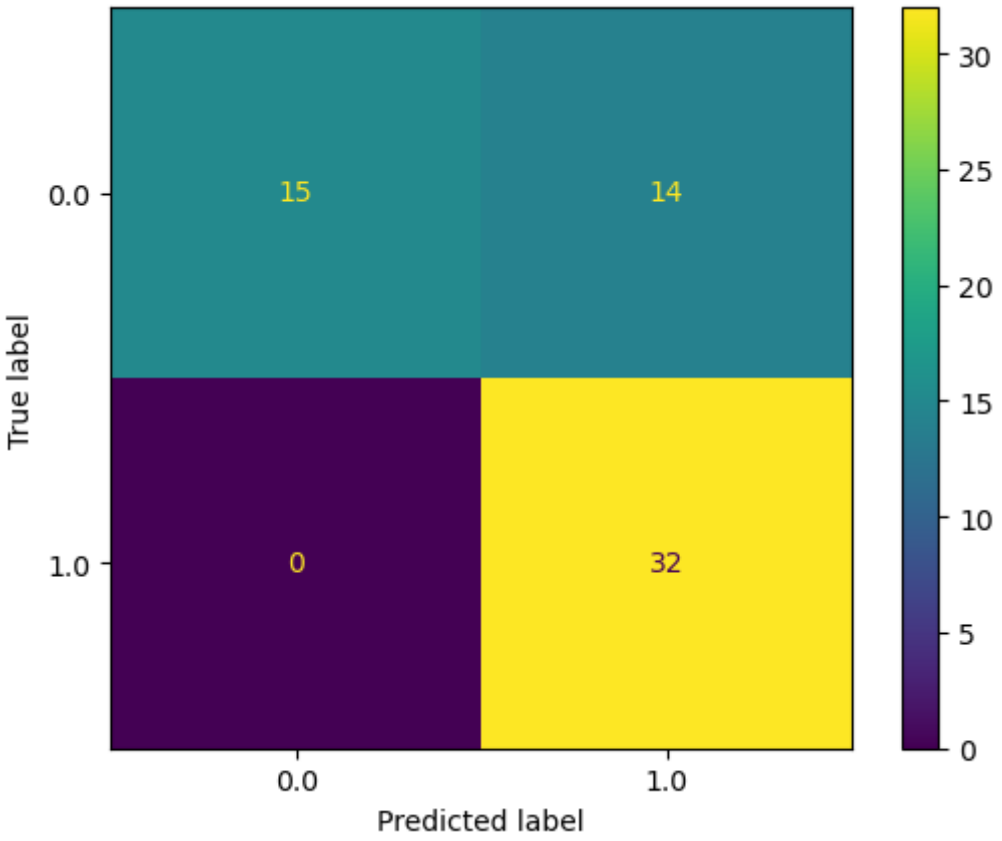
CM for High Recall and Low Precision Model (Minimize Type-II Error (FN):

In [53]:
```
y_pred3 = np.array([1 if p >= 0.2 else 0 for p in y_prob])
y_pred3
```

Out[53]:
```
array([1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1,
0,
       1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
0,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1])
```

In [54]:
```
#disp = ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred3))
#disp.plot();
ConfusionMatrixDisplay.from_predictions(y_test, y_pred3);
print("Precision Score:", precision_score(y_test, y_pred3))
print("Recall Score:", recall_score(y_test, y_pred3))
```

```
Precision Score: 0.6956521739130435
Recall Score: 1.0
```



Note:

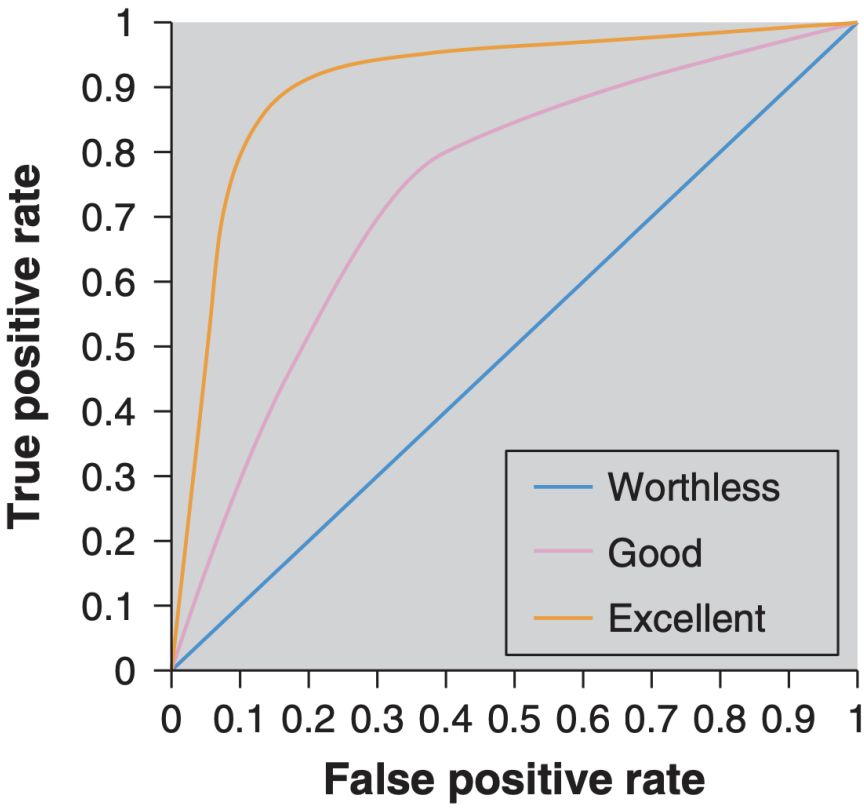
- `FP` are `increased` from 8 to 14, while `FN` are `decreased` from 1 to 0
- `Recall` is `increased` from 0.96 to 1.0, while `Precision` is `reduced` from 0.79 to 0.69

h. ROC Curve and AUC Score

Receiver Operator Characteristic (ROC) Curve is a plot between the False Positive rate (FPR) along x-axis versus the True Positive rate (TPR) along y-axis at different probability threshold values between 0 and 1

The Area under the ROC curve represents the goodness of the model or AUC Score

False Positive Rate (1-Specificity): The probability that a `negative sample` is `incorrectly` predicted in the positive class



$$FPR = \frac{FP}{FP + TN}$$

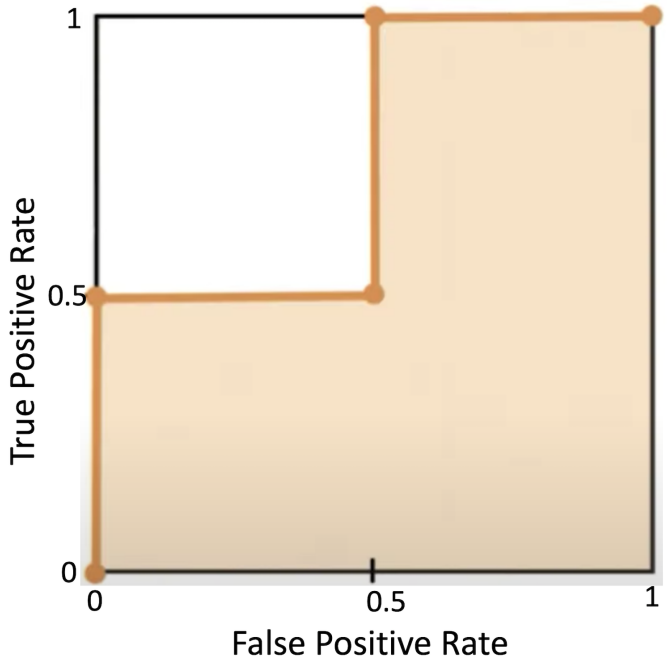
True Positive Rate (Sensitivity): The probability that a `positive sample` is `correctly` predicted in the positive class

$$TPR = \frac{TP}{TP + FN}$$

- An AUC score of 1 means the classifier can perfectly distinguish between all the Positive and the Negative class points
- An AUC score of 0.5 means that the classifier is not working.
- An AUC value of 0 shows that the classifier predicts all Negatives as Positives and vice versa

Example Data:

y	h(x)	0.3 threshold	0.5 threshold	0.6 threshold
0	0.3	0 (TN)	0 (TN)	0 (TN)
1	0.5	1 (TP)	0 (**FN**)	0 (**FN**)
0	0.6	1 (**FP**)	1 (**FP**)	0 (TN)
1	0.9	1 (TP)	1 (TP)	1 (TP)
TPR: $\frac{TP}{TP+FN}$		$\frac{2}{2+0} = 1.0$	$\frac{1}{1+1} = 0.5$	$\frac{1}{1+1} = 0.5$
FPR: $\frac{FP}{FP+TN}$		$\frac{1}{1+1} = 0.5$	$\frac{1}{1+1} = 0.5$	$\frac{0}{0+2} = 0$



ROC-AUC Score for Heart Disease Dataset

Model Probabilities

In [55]:
```
model_probs = model.predict_proba(scaled_X_test)[:,1]
model_probs
```

Out[55]:
```
array([0.99184126, 0.88095275, 0.97605822, 0.25538515, 0.31379486,
       0.0373357 , 0.66199129, 0.96681128, 0.31982715, 0.92871087,
       0.8214886 , 0.0092004 , 0.0896413 , 0.74933318, 0.95530575,
       0.51156619, 0.88864224, 0.69029437, 0.33355711, 0.00377371,
       0.99135765, 0.0205858 , 0.58694646, 0.07172263, 0.01387574,
       0.73222067, 0.00766868, 0.8854798 , 0.64638054, 0.00586942,
       0.53079837, 0.94461409, 0.97144974, 0.99689867, 0.87154038,
       0.97651181, 0.54186077, 0.00691448, 0.92193046, 0.91501052,
       0.99127229, 0.9574321 , 0.05350028, 0.18595097, 0.78369623,
       0.07246424, 0.91333947, 0.69453775, 0.49362162, 0.29029273,
       0.31840982, 0.04016536, 0.71184106, 0.01078019, 0.77571284,
       0.95203588, 0.87210665, 0.62531545, 0.64639495, 0.93483498,
       0.99352217])
```

In [56]:
```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_true=y_test,  y_score=model_probs)

print("FPR: ", fpr)
print("\nTPR: ", tpr)
print("\nThresholds: ", thresholds)
```

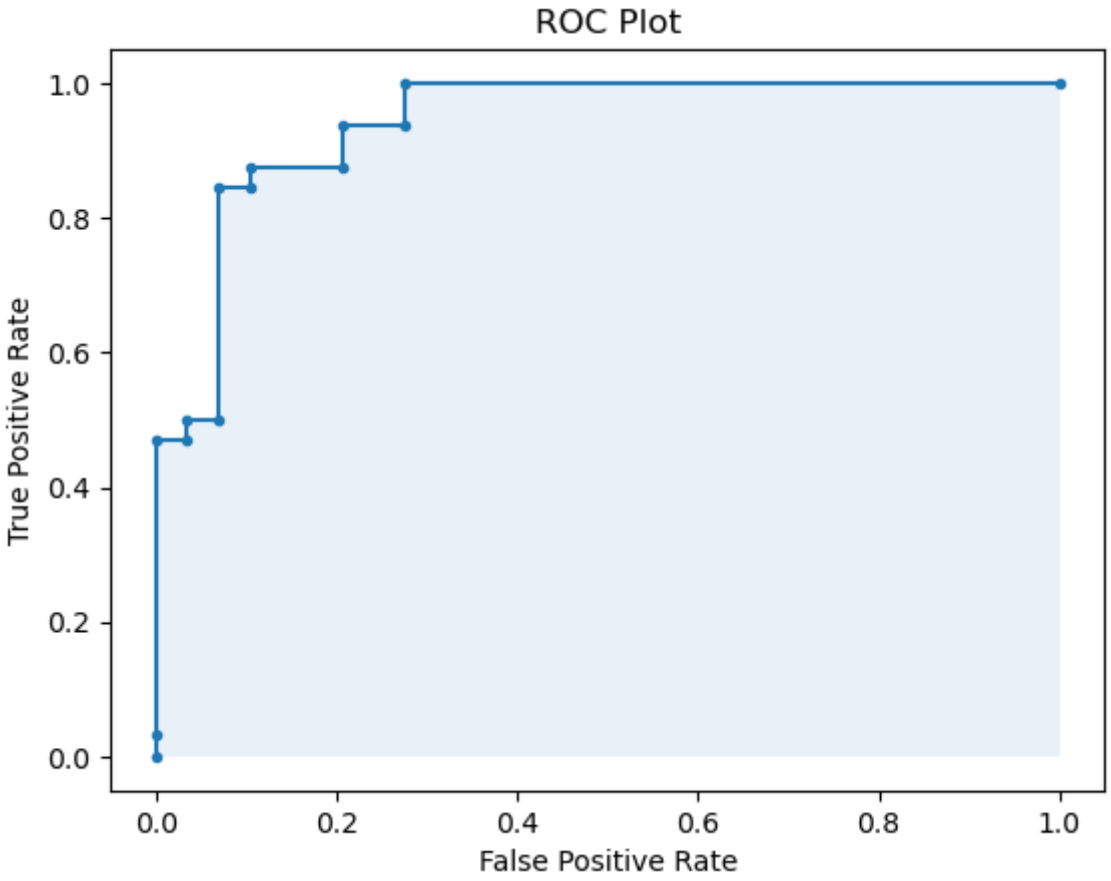
```
FPR:  [0.         0.         0.         0.03448276 0.03448276 0.068965
52
 0.06896552 0.10344828 0.10344828 0.20689655 0.20689655 0.27586207
 0.27586207 1.        ]

TPR:  [0.      0.03125 0.46875 0.46875 0.5     0.5     0.84375 0.84375
0.875
 0.875   0.9375  0.9375  1.      1.     ]

Thresholds:  [1.99689867 0.99689867 0.92871087 0.92193046 0.91501052
0.91333947
 0.71184106 0.69453775 0.69029437 0.64638054 0.58694646 0.53079837
 0.49362162 0.00377371]
```

In [57]:
```
# zip() is used to combine two or more iterables into a single iterable
pd.DataFrame(zip(thresholds, fpr, tpr), columns=["Threshold", "FPR", "TP
```

Out[57]:

	Threshold	FPR	TPR
0	1.996899	0.000000	0.00000
1	0.996899	0.000000	0.03125
2	0.928711	0.000000	0.46875
3	0.921930	0.034483	0.46875
4	0.915011	0.034483	0.50000
5	0.913339	0.068966	0.50000
6	0.711841	0.068966	0.84375
7	0.694538	0.103448	0.84375
8	0.690294	0.103448	0.87500
9	0.646381	0.206897	0.87500
10	0.586946	0.206897	0.93750
11	0.530798	0.275862	0.93750
12	0.493622	0.275862	1.00000
13	0.003774	1.000000	1.00000

In [58]:
```
plt.plot(fpr, tpr, marker='.')
plt.fill_between(fpr, tpr, 0, alpha=0.1)
plt.title('ROC Plot')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```



> The more that the curve hugs the top left corner of the plot, the better the model does at classifying the data into categories.

In [59]:
```
# To quantify, we can calculate the AUC, which tells us how much of the
# The closer AUC is to 1, the better the model.
#A model with an AUC equal to 0.5 is no better than a model that makes r
from sklearn.metrics import roc_auc_score
model_auc=  roc_auc_score(y_test, model_probs)
print("Model AUC Score: ", model_auc)
```

```
Model AUC Score:  0.9418103448275862
```

In [ ]:

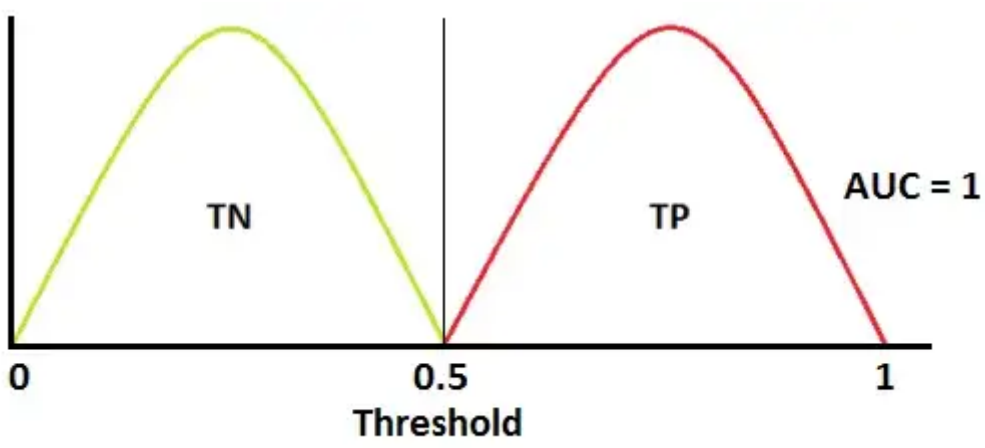
How to speculate about the performance of the model?

- An excellent model has AUC near to the `1` which means it has a good measure of separability. A poor model has an AUC near `0` which means it has the worst measure of separability. In fact, it means it is reciprocating the result. It is predicting 0s as 1s and 1s as

0s. And when AUC is `0.5`, it means the model has no class separation capacity whatsoever.
- As we know, ROC is a curve of probability. So let's plot the distributions of those probabilities. **Note:** `Red` distribution curve is of the positive class (patients with disease) and the `green` distribution curve is of the negative class(patients with no disease).

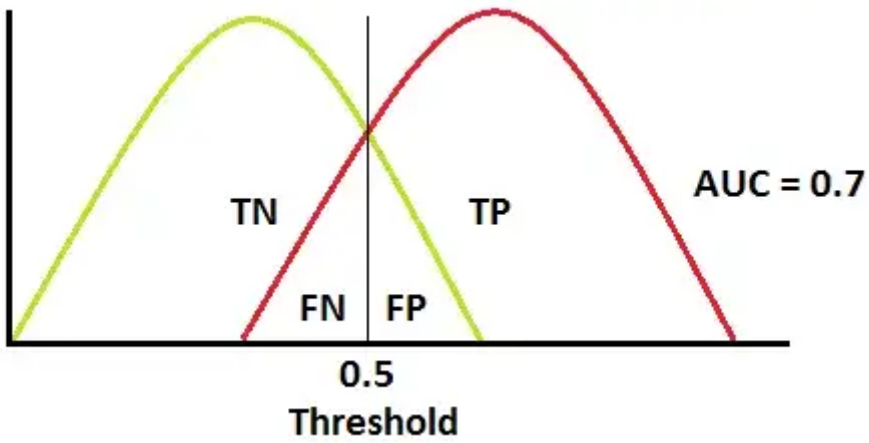
Case 1:

- This is an ideal situation. When two curves don't overlap at all means model has an ideal measure of separability. It is perfectly able to distinguish between positive class and negative class.



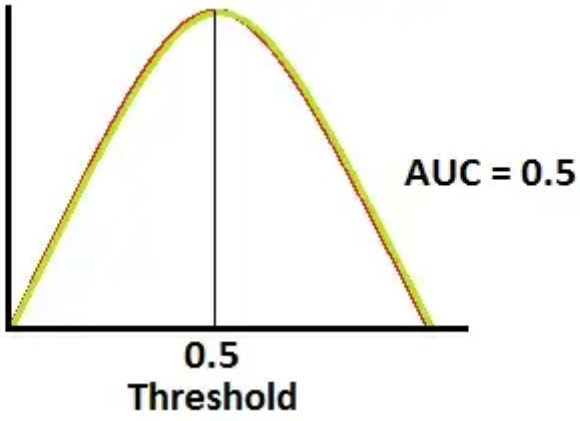
Case 2:

- When two distributions overlap, we introduce type 1 and type 2 errors. Depending upon the threshold, we can minimize or maximize them. When AUC is 0.7, it means there is a 70% chance that the model will be able to distinguish between positive class and negative class.



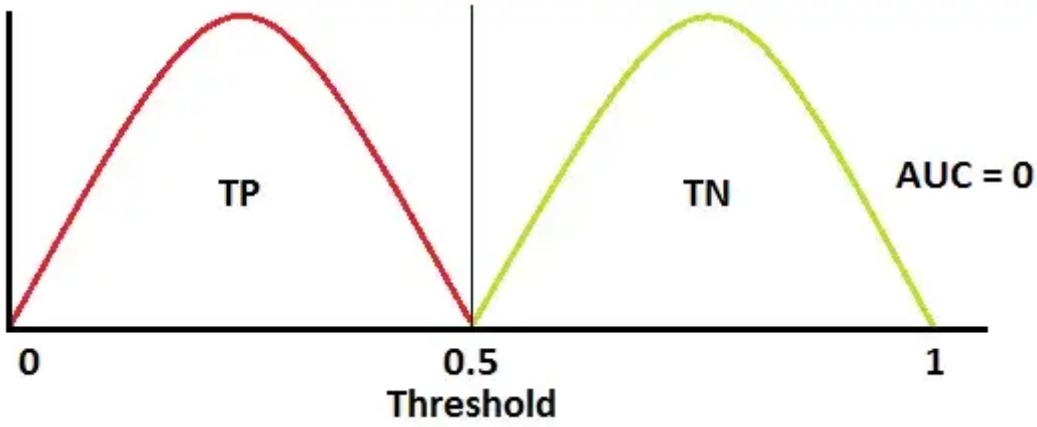
Case 3:

- This is the worst situation. When AUC is approximately 0.5, the model has no discrimination capacity to distinguish between positive class and negative class.



Case 4:

- When AUC is approximately 0, the model is actually reciprocating the classes. It means the model is predicting a negative class as a positive class and vice versa.



5. Hyperparameters for Logistic Regression

- `LogisticRegression()` (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)

In [60]:
```
model = LogisticRegression()
model.get_params()
```

Out[60]:
```
{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 100,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': None,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

Hyperparameter Tuning Techniques:

- `By Hand:` Select the hyperparameters values based on intuition/experience/guessing, train the model with the hyperparameters, and score on the validation data. Repeat process until you run out of patience or are satisfied with the results.
- `GridSearchCV:` Set up a grid of hyperparameter values and for each combination, train a model and score on the validation data. In this approach, every single combination of hyperparameters values is tried which can be very inefficient!
- `RandomizedSearchCV:` Set up a grid of hyperparameter values and select random combinations to train the model and score. The number of search iterations is set based on time/resources.

Solver Parameters

The solver parameter in logistic regression is used to specify the algorithm that will be used to find the optimal set of coefficients for the model. The five options for the solver parameter are `newton-cg` , `lbfgs` , `liblinear` , `sag` , and `saga` . The choice of solver will depend on the size and characteristics of your dataset, and the desired trade-off between speed and accuracy. `newton-cg` , `lbfgs` and `sag` are more suited for larger datasets, while `liblinear` is more efficient for smaller datasets. `saga` is a solver that combines the benefits of `sag` and `lbfgs` and is generally recommended.

newton-cg: The Newton-Conjugate Gradient (Newton-CG) solver is an optimization algorithm used for finding the minimum of a function. It is a combination of Newton's method and the Conjugate Gradient method, and it is particularly well-suited for solving large-scale optimization problems in logistic regression.

lbfgs: The Limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) solver is an optimization algorithm used for finding the minimum of a function. LBFGS is particularly well-suited for problems with a large number of parameters, such as logistic regression, as it can handle a large number of features and still converge quickly.

liblinear: The liblinear solver is an optimization algorithm used for solving linear classification problems, such as logistic regression. It is based on a linear support vector machine (SVM) and is implemented in the C++ library LIBLINEAR.

saga: The SAGA (Stochastic Average Gradient Descent) solver is an optimization algorithm used for solving large-scale optimization problems, such as logistic regression. It is an extension of the standard stochastic gradient descent (SGD) algorithm and can handle both L1 and L2 regularization.

sag: The Stochastic Average Gradient (SAG) solver is an optimization algorithm used for solving large-scale optimization problems, such as logistic regression. It is a variant of the SAGA algorithm and also uses a mini-batch approach to update the parameters of the model by computing the gradient of the loss function with respect to the parameters using a small random subset of the training data.

Supported Penalties by Different Solvers are:

- `lbfgs` --> ['l2', None]
- `liblinear` --> ['l1', 'l2']
- `newton-cg` --> ['l2', None]
- `newton-cholesky` --> ['l2', None]
- `sag` --> ['l2', None]

- `saga` --> ['elasticnet', 'l1', 'l2', None]

Task to Do: Logistic Regression (Titanic Dataset)

In [61]:
```
from sklearn.datasets import fetch_openml
titanic = datasets.fetch_openml(name='titanic', version=1, as_frame=True
df = pd.DataFrame(titanic.data, columns=titanic.feature_names)
df['target'] = titanic.target
df
```

Out[61]:

	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	b
0	1.0	Allen, Miss. Elisabeth Walton	female	29.0000	0.0	0.0	24160	211.3375	B5	S	
1	1.0	Allison, Master. Hudson Trevor	male	0.9167	1.0	2.0	113781	151.5500	C22 C26	S	
2	1.0	Allison, Miss. Helen Loraine	female	2.0000	1.0	2.0	113781	151.5500	C22 C26	S	N
3	1.0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1.0	2.0	113781	151.5500	C22 C26	S	N
4	1.0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1.0	2.0	113781	151.5500	C22 C26	S	N
...	...	...	...	...	...	...	...	...	...	...	
1304	3.0	Zabour, Miss. Hileni	female	14.5000	1.0	0.0	2665	14.4542	None	C	N
1305	3.0	Zabour, Miss. Thamine	female	NaN	1.0	0.0	2665	14.4542	None	C	N
1306	3.0	Zakarian, Mr. Mapriededer	male	26.5000	0.0	0.0	2656	7.2250	None	C	N
1307	3.0	Zakarian, Mr. Ortin	male	27.0000	0.0	0.0	2670	7.2250	None	C	N
1308	3.0	Zimmerman, Mr. Leo	male	29.0000	0.0	0.0	315082	7.8750	None	S	N

1309 rows × 14 columns

VARIABLE DESCRIPTIONS

`Pclass` -> Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd).
`survival` -> Survival (0 = No; 1 = Yes)
`name` -> Name
`sex` -> Sex

`age` -> Age
`sibsp` -> Number of Siblings/Spouses Aboard
`parch` -> Number of Parents/Children Aboard
`ticket` -> Ticket Number
`fare` -> Passenger Fare (British pound)
`cabin` -> Cabin
`embarked` -> Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)
`boat` -> Lifeboat
`body` -> Body Identification Number
`home.dest` -> Home/Destination

1. Data Preprocessing and EDA
    - Detect and handle outliers
    - Detect and impute missing values
    - Encode categorical features
    - Feature scaling
    - Feature Engineering (Drop unnecessary features)
2. Train Test Split, Cross Validation
3. Training of Logistic Regression Non-pipelined and Pipelined Model
4. Model Evaluation
5. Polynomial Features introduction (Optional)
6. Hyperparameter Tuning using GridSearchCV to get the best evaluation results
7. Saving the Model
8. Model Deployment using Streamlit or Flask

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: