CS188 Project 2 - Binary Classification Comparative Methods

For this project we're going to attempt a binary classification of a dataset using multiple methods and compare results.

Our goals for this project will be to introduce you to several of the most common classification techniques, how to perform them and tweek parameters to optimize outcomes, how to produce and interpret results, and compare performance. You will be asked to analyze your findings and provide explanations for observed performance.

Specifically you will be asked to classify whether a **patient is suffering from heart disease** based on a host of potential medical factors.

DEFINITIONS

Binary Classification: In this case a complex dataset has an added 'target' label with one of two options. Your learning algorithm will try to assign one of these labels to the data.

Supervised Learning: This data is fully supervised, which means it's been fully labeled and we can trust the veracity of the labeling.

Background: The Dataset

For this exercise we will be using a subset of the UCI Heart Disease dataset, leveraging the fourteen most commonly used attributes. All identifying information about the patient has been scrubbed.

The dataset includes 14 columns. The information provided by each column is as follows:

- **age:** Age in years
- **sex:** (1 = male; 0 = female)
- **cp:** Chest pain type (0 = asymptomatic; 1 = atypical angina; 2 = non-anginal pain; 3 = typical angina)
- **trestbps:** Resting blood pressure (in mm Hg on admission to the hospital)
- **cholserum:** Cholestoral in mg/dl
- **fbs** Fasting blood sugar > 120 mg/dl (1 = true; 0 = false)
- **restecg:** Resting electrocardiographic results (0= showing probable or definite left ventricular hypertrophy by Estes' criteria; 1 = normal; 2 = having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV))
- **thalach:** Maximum heart rate achieved
- **exang:** Exercise induced angina (1 = yes; 0 = no)
- **oldpeakST:** Depression induced by exercise relative to rest
- **slope:** The slope of the peak exercise ST segment (0 = downsloping; 1 = flat; 2 = upsloping)
- **ca:** Number of major vessels (0-3) colored by flourosopy
- **thal:** 1 = normal; 2 = fixed defect; 7 = reversable defect
- **Sick:** Indicates the presence of Heart disease (True = Disease; False = No disease)

Loading Essentials and Helper Functions

In [1]:
```
#Here are a set of libraries we imported to complete this assignment.
#Feel free to use these or equivalent libraries for your implementation
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # this is used for the plot the graph
import os
import seaborn as sns # used for plot interactive graph.
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix
import sklearn.metrics.cluster as smc
from sklearn.model_selection import KFold


from matplotlib import pyplot
import itertools

%matplotlib inline
import random

random.seed(42)
```

In [2]:
```
# Helper function allowing you to export a graph
def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

In [3]:
```
# Helper function that allows you to draw nicely formatted confusion matrices
def draw_confusion_matrix(y, yhat, classes):
    '''
        Draws a confusion matrix for the given target and predictions
        Adapted from scikit-learn and discussion example.
    '''
    plt.cla()
    plt.clf()
    matrix = confusion_matrix(y, yhat)
    plt.imshow(matrix, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title("Confusion Matrix")
    plt.colorbar()
    num_classes = len(classes)
    plt.xticks(np.arange(num_classes), classes, rotation=90)
    plt.yticks(np.arange(num_classes), classes)

    fmt = 'd'
    thresh = matrix.max() / 2.
    for i, j in itertools.product(range(matrix.shape[0]), range(matrix.shape[1])):
        plt.text(j, i, format(matrix[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if matrix[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
    plt.show()
```

[20 Points] Part 1. Load the Data and Analyze

Let's first load our dataset so we'll be able to work with it. (correct the relative path if your notebook is in a different directory than the csv file.)

In [4]:
```
data = pd.read_csv("heartdisease.csv")
```

Question 1.1 Now that our data is loaded, let's take a closer look at the dataset we're working with. Use the head method to display some of the rows so we can visualize the types of data fields we'll be working with, then use the describe method, along with any additional methods you'd like to call to better help you understand what you're working with and what issues you might face.

In [5]:
```
data.head()
```

Out[5]:

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

In [6]:
```
data.describe()
```

Out[6]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang
count	303.000000	303.000000	303.000000	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	0.528053	149.646865	0.326733
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000

In [7]:
```
data.dtypes
```

Out[7]:
```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
sick           bool
dtype: object
```

In [8]:
```
data.isnull().sum()
```

Out[8]:
```
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
sick        0
dtype: int64
```

Question 1.2 Discuss your data preprocessing strategy. Are their any datafield types that are problemmatic and why? Will there be any null values you will have to impute and how do you intend to do so? Finally, for your numeric and categorical features, what if any, additional preprocessing steps will you take on those data elements?

The data looks pretty good to start off with, as there is not much needed to do for preprocessing. Most of the data types are numerical, and thus we can use these to fit our model. The only one that is not numerical is the 'sick' label, which is a boolean data type. This is very easy to convert to numerical as we can denote 'True' as '1' and 'False' as '0'. Even categorical data has been converted to numerical values, which is what we need. The data also does not contain any missing data, and so we do not need to impute any data. Finally, we should standardize the data for each feature.

Question 1.3 Before we begin our analysis we need to fix the field(s) that will be problematic. Specifically convert our boolean sick variable into a binary numeric target variable (values of either '0' or '1'), and then drop the original sick datafield from the dataframe.

In [9]:
```
data.sick = data.sick.astype(int)
data
```

Out[9]:

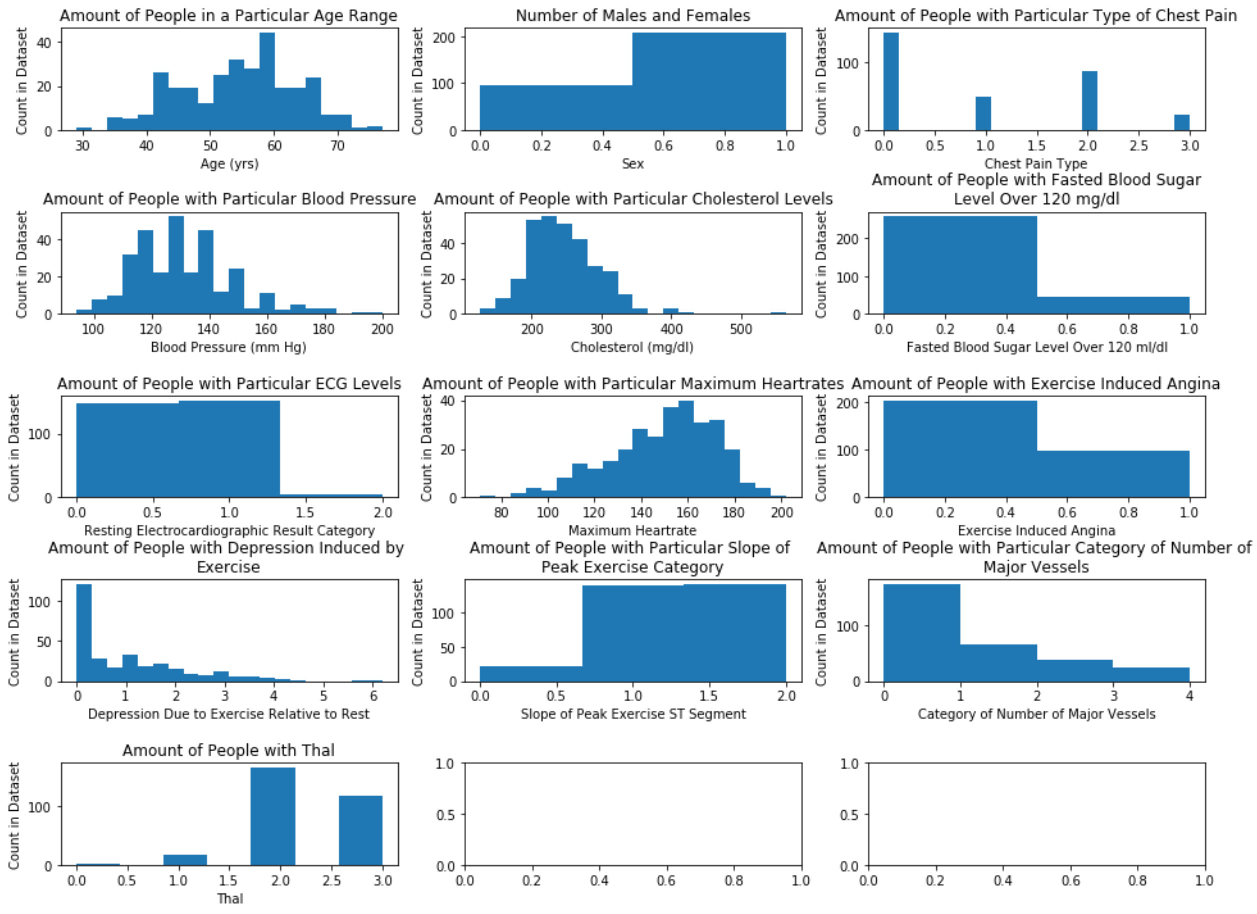
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	sick
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	0
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	0
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	0
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	0
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	1
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	1
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	1
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	1
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	1

303 rows × 14 columns

Question 1.4 Now that we have a feel for the data-types for each of the variables, plot histograms of each field and attempt to ascertain how each variable performs (is it a binary, or limited selection, or does it follow a gradient? (Note: No need to describe each variable, but pick out a few you wish to highlight)

In [10]:
```
f, a = plt.subplots(5, 3)

a[0,0].hist(data["age"], bins=20)
a[0,0].set_xlabel("Age (yrs)")
a[0,0].set_ylabel("Count in Dataset")
a[0,0].set_title("Amount of People in a Particular Age Range")

a[0,1].hist(data["sex"], bins=2)
a[0,1].set_xlabel("Sex")
a[0,1].set_ylabel("Count in Dataset")
a[0,1].set_title("Number of Males and Females")

a[0, 2].hist(data["cp"], bins=20)
a[0, 2].set_xlabel("Chest Pain Type")
a[0, 2].set_ylabel("Count in Dataset")
a[0, 2].set_title("Amount of People with Particular Type of Chest Pain")

a[1, 0].hist(data["trestbps"], bins=20)
a[1, 0].set_xlabel("Blood Pressure (mm Hg)")
a[1, 0].set_ylabel("Count in Dataset")
a[1, 0].set_title("Amount of People with Particular Blood Pressure")

a[1, 1].hist(data["chol"], bins=20)
a[1, 1].set_xlabel("Cholesterol (mg/dl)")
a[1, 1].set_ylabel("Count in Dataset")
a[1, 1].set_title("Amount of People with Particular Cholesterol Levels")

a[1, 2].hist(data["fbs"], bins=2)
a[1, 2].set_xlabel("Fasted Blood Sugar Level Over 120 ml/dl")
a[1, 2].set_ylabel("Count in Dataset")
a[1, 2].set_title("Amount of People with Fasted Blood Sugar\nLevel Over 120 mg/dl")

a[2, 0].hist(data["restecg"], bins=3)
a[2, 0].set_xlabel("Resting Electrocardiographic Result Category")
a[2, 0].set_ylabel("Count in Dataset")
a[2, 0].set_title("Amount of People with Particular ECG Levels")

a[2, 1].hist(data["thalach"], bins=20)
a[2, 1].set_xlabel("Maximum Heartrate")
a[2, 1].set_ylabel("Count in Dataset")
a[2, 1].set_title("Amount of People with Particular Maximum Heartrates")

a[2, 2].hist(data["exang"], bins=2)
a[2, 2].set_xlabel("Exercise Induced Angina")
a[2, 2].set_ylabel("Count in Dataset")
a[2, 2].set_title("Amount of People with Exercise Induced Angina")

a[3, 0].hist(data["oldpeak"], bins=20)
a[3, 0].set_xlabel("Depression Due to Exercise Relative to Rest")
a[3, 0].set_ylabel("Count in Dataset")
a[3, 0].set_title("Amount of People with Depression Induced by \nExercise")

a[3, 1].hist(data["slope"], bins=3)
a[3, 1].set_xlabel("Slope of Peak Exercise ST Segment")
a[3, 1].set_ylabel("Count in Dataset")
a[3, 1].set_title("Amount of People with Particular Slope of \nPeak Exercise Category")

a[3, 2].hist(data["ca"], bins=4)
a[3, 2].set_xlabel("Category of Number of Major Vessels")
a[3, 2].set_ylabel("Count in Dataset")
a[3, 2].set_title("Amount of People with Particular Category of Number of \nMajor Vessels")

a[4, 0].hist(data["thal"], bins=7)
a[4, 0].set_xlabel("Thal")
a[4, 0].set_ylabel("Count in Dataset")
a[4, 0].set_title("Amount of People with Thal")
```

```
f.set_figheight(12)
f.set_figwidth(16)
plt.subplots_adjust(hspace=0.8)
plt.show()
```



Some observations we observe from the dataset

-Age: The age variable follows a gradient, and it is approxiamtely normally distributed with the mid-50s being the mean.

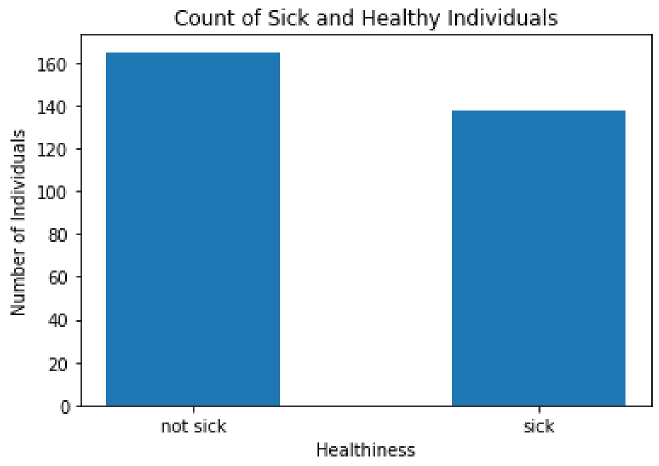
-Sex: The sex variable is a binary variable. We can see that there is around twice as many female records than male records.

-Type of chest pain: The type of chest pain is neither a binary variable nor does it follow a gradient. It is a categorical variable.

Question 1.5 We also want to make sure we are dealing with a balanced dataset. In this case, we want to confirm whether or not we have an equitable number of sick and healthy individuals to ensure that our classifier will have a sufficiently balanced dataset to adequately classify the two. Plot a histogram specifically of the sick target, and conduct a count of the number of sick and healthy individuals and report on the results:

In [11]:
```python
count_sick = data[data["sick"] == 1].shape[0]
count_not_sick = data[data["sick"] == 0].shape[0]
y = [count_not_sick, count_sick]
plt.bar(x=["not sick", "sick"], height=y, width=0.5)
plt.title("Count of Sick and Healthy Individuals")
plt.xlabel("Healthiness")
plt.ylabel("Number of Individuals")
print("Number of sick people: {}".format(count_sick))
print("Number of healthy people: {}".format(count_not_sick))
```

```
Number of sick people: 138
Number of healthy people: 165
```



In this dataset, there are 138 sick individuals and 165 healthy individuals. There is an approximately equal number of individuals in both categories, so the dataset is balanced, and we can use this dataset to adequately classify the two.

Question 1.6 Balanced datasets are important to ensure that classifiers train adequately and don't overfit, however arbitrary balancing of a dataset might introduce its own issues. Discuss some of the problems that might arise by artificially balancing a dataset.

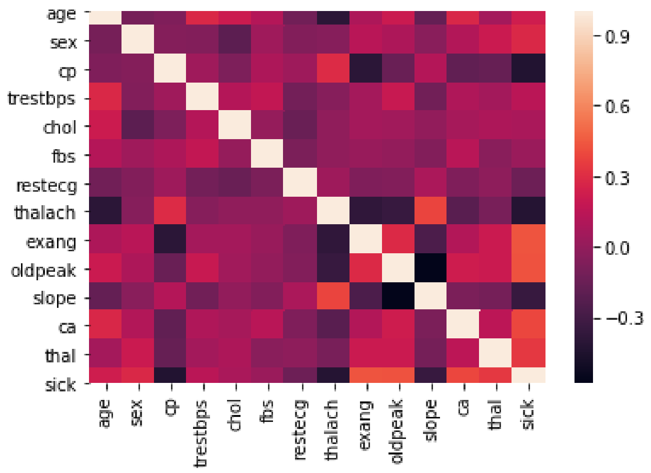
Artificially balancing a dataset comes in many flavors, and there are strengths and limitations to each method. First, we can downsample the category that has more samples so that all the categories end up having the same quantity. The disadvantage of this method is that it is essentially throwing out real data that could be used. Another method is to synthetically generate data for the category that has fewer samples. The disadvantage to this method is that the synthetically generated data depends fully on the model that is used to generate the data. Thus, if the model is not strong, then the generated data will not be representative of the real data, and thus the final model that uses the synthetic data will not predict well, because it is overfitted. Also, the synthetically generated data may not generate corner cases.

Question 1.9 Now that we have our dataframe prepared let's start analyzing our data. For this next question let's look at the correlations of our variables to our target value. First, map out the correlations between the values, and then discuss the relationships you observe. Do some research on the variables to understand why they may relate to

the observed corellations. Intuitively, why do you think some variables correlate more highly than others (hint: one possible approach you can use the sns heatmap function to map the corr() method)?

```
In [12]: import seaborn as sns
         sns.heatmap(data.corr(method="pearson"))
```

```
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x2174896b588>
```



POSITIVE CORRELATION

There is a positive correlation between maximum heartrate achieved (thalach) and slope of the peak exercise ST segment (slope). Slope of the peak exercise ST segment refers to the activity inside the heart. If the slope is decreasing, this means that blood was flowing back into the heart when it was trying to pump blood out. Alternatively, if the slope is increasing, blood was flowing out constantly when the heart tries to pump blood out. This is linked to maximum heartrate, because those whose hearts are allowing blood to flow back in, probably have weaker hearts and are thus more likely to have a lower maximum heartrate.

There is a positive correlation between a patient being sick (sick) and their number of major vessels colored by fluoroscopy (ca). Fluorscopy is the process of watching movement of structures inside the body. In this case, we are looking at the blood vessels. More seen blood vessels probably correlates to an individual's blood vessels clotting, thus resulting in sickness.

There is a positive correlation between exercise induced angina (exang) and sick (sick). This is expected, as exercise induced angina is a type of chest pain, which correlates to sickness.

There is a positive correlation between depression induced by exercise relative to rest (oldpeak) and sick (sick). This is abnormal, because people usually don't get depressed when they exercise. Thus, this behavior is positively correlated with sickness.

NEGATIVE CORRELATION

There is a negative correlation between the slope of the peak exercise ST segment (slope) and depression induced by exercise relative to rest (oldpeak). As mentioned earlier, a downsloping ST segment is unhealthy. Thus, we see a relationship between downsloping segments and depression induced by exercise.

There is a negative correlation between the slope of the peak exercise ST segment (slope) and sick (sick). Once again, a downsloping ST segment is unhealthy, which corresponds to a, individual more likely to be sick.

There is a negative correlation between maximum heartrate achieved (thalach) and sick (sick). If the individual is healthy, the maximum heartrate is higher, and vice versa.

[30 Points] Part 2. Prepare the Data

Before running our various learning methods, we need to do some additional prep to finalize our data. Specifically you'll have to cut the classification target from the data that will be used to classify, and then you'll have to divide the dataset into training and testing cohorts.

Specifically, we're going to ask you to prepare 2 batches of data: 1. Will simply be the raw numeric data that hasn't gone through any additional pre-processing. The other, will be data that you pipeline using your own selected methods. We will then feed both of these datasets into a classifier to showcase just how important this step can be!

Question 2.1 Save the target column as a separate array and then drop it from the dataframe.

In [13]:
```
if "sick" in data.columns:
    target = np.array(data["sick"])
    data = data.drop(columns=["sick"])
```

Question 2.2 First Create your 'Raw' unprocessed training data by dividing your dataframe into training and testing cohorts, with your training cohort consisting of 70% of your total dataframe (hint: use the train_test_split method) Output the resulting shapes of your training and testing samples to confirm that your split was successful.

In [14]:
```
raw_X_train, raw_X_test, raw_y_train, raw_y_test = train_test_split(data, target, test_size=0.3,
print("The training features set has {} rows and {} columns.".format(raw_X_train.shape[0], raw_X_
print("The test features set has {} rows and {} columns.".format(raw_X_test.shape[0], raw_X_test.
print("The training targets set has {} rows and 1 column.".format(raw_y_train.shape[0]))
print("The test targets set has {} rows and 1 column.".format(raw_y_test.shape[0]))
```

```
The training features set has 212 rows and 13 columns.
The test features set has 91 rows and 13 columns.
The training targets set has 212 rows and 1 column.
The test targets set has 91 rows and 1 column.
```

Question 2.3 Now create a pipeline to conduct any additional preparation of the data you would like. Output the resulting array to ensure it was processed correctly.

In [15]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

# categorical: sex, cp, restecg, slope, thal
# boolean: fbs, exang
# numerical: age, trestbps, chol, thalach, oldpeak, ca

# Make dataframe with categorical features and numerical features
cat_columns = []
data_num = data.copy(deep=True)

num_columns = list(data_num.columns)
if "sex" in num_columns:
    data_num.drop(["sex"], axis=1, inplace=True)
    cat_columns.append("sex")

if "cp" in num_columns:
    data_num.drop(["cp"], axis=1, inplace=True)
    cat_columns.append("cp")

if "restecg" in num_columns:
    data_num.drop(["restecg"], axis=1, inplace=True)
    cat_columns.append("restecg")

if "slope" in num_columns:
    data_num.drop(["slope"], axis=1, inplace=True)
    cat_columns.append("slope")

if "thal" in num_columns:
    data_num.drop(["thal"], axis=1, inplace=True)
    cat_columns.append("thal")

if "fbs" in num_columns:
    data_num.drop(["fbs"], axis=1, inplace=True)

if "exang" in num_columns:
    data_num.drop(["exang"], axis=1, inplace=True)

num_pipeline = Pipeline([
    ('std_scaler', StandardScaler())
])

num_columns = list(data_num)

full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_columns),
    ("cat", OneHotEncoder(categories="auto"), cat_columns),
])

data_prepared = full_pipeline.fit_transform(data)
data_prepared
```

Out[15]:
```
array([[ 0.9521966 ,  0.76395577, -0.25633371, ...,  1.        ,
         0.        ,  0.        ],
       [-1.91531289, -0.09273778,  0.07219949, ...,  0.        ,
         1.        ,  0.        ],
       [-1.47415758, -0.09273778, -0.81677269, ...,  0.        ,
         1.        ,  0.        ],
       ...,
       [ 1.50364073,  0.70684287, -1.029353  , ...,  0.        ,
         0.        ,  1.        ],
       [ 0.29046364, -0.09273778, -2.2275329 , ...,  0.        ,
         0.        ,  1.        ],
       [ 0.29046364, -0.09273778, -0.19835726, ...,  0.        ,
         1.        ,  0.        ]])
```

Question 2.4 Now create a separate, processed training data set by dividing your processed dataframe into training and testing cohorts, using the same settings as Q2.2 (REMEMBER TO USE DIFFERENT TRAINING AND TESTING VARIABLES SO AS NOT TO OVERWRITE YOUR PREVIOUS DATA). Output the resulting shapes of your training and testing samples to confirm that your split was successful, and describe what differences there are between your two training datasets.

In [16]:
```
X_train, X_test, y_train, y_test = train_test_split(data_prepared, target, test_size=0.3, random_
print("The training features set has {} rows and {} columns.".format(X_train.shape[0], X_train.sh
print("The test features set has {} rows and {} columns.".format(X_test.shape[0], X_test.shape[1]
print("The training targets set has {} rows and 1 column.".format(y_train.shape[0]))
print("The test targets set has {} rows and 1 column.".format(y_test.shape[0]))
```

```
The training features set has 212 rows and 22 columns.
The test features set has 91 rows and 22 columns.
The training targets set has 212 rows and 1 column.
The test targets set has 91 rows and 1 column.
```

The prepared dataset has extra columns. This is because we fed the unprocessed data through the pipeline, which includes a OneHotEncoder. This process separates categorical features into boolean values. Say a categorical feature has three possible values. Then, the encoder would replace the original column with three columns, each with the name of the value. Then, they would be set to 0 or 1 accordingly.

[50 Points] Part 3. Learning Methods

We're finally ready to actually begin classifying our data. To do so we'll employ multiple learning methods and compare result.

Linear Decision Boundary Methods

SVM (Support Vector Machine)

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimentional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

Question 3.1.1 Implement a Support Vector Machine classifier on your RAW dataset. Review the [SVM Documentation (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html)](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html) for how to implement a model. For this implementation you can simply use the default settings, but set probability = True.

In [17]:
```
# SVM
svm_raw_clf = SVC(probability=True, gamma="scale")
svm_raw_clf.fit(raw_X_train, raw_y_train)
```

Out[17]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=True, random_state=None, shrinking=True, tol=0.001,
    verbose=False)
```

Question 3.1.2 Report the accuracy, precision, recall, F1 Score, and confusion matrix

of the resulting model.

In [18]:
```
from sklearn import metrics
from sklearn.linear_model import LogisticRegression

svm_raw_preds = svm_raw_clf.predict(raw_X_test)

print("Accuracy: {}".format(metrics.accuracy_score(raw_y_test, svm_raw_preds)))
print("Precision: {}".format(metrics.precision_score(raw_y_test, svm_raw_preds)))
print("Recall: {}".format(metrics.recall_score(raw_y_test, svm_raw_preds)))
print("F1: {}".format(metrics.f1_score(raw_y_test, svm_raw_preds)))
print("Confusion Matrix: \n{}".format(metrics.confusion_matrix(raw_y_test, svm_raw_preds)))
```

```
Accuracy: 0.7032967032967034
Precision: 0.7916666666666666
Recall: 0.4634146341463415
F1: 0.5846153846153846
Confusion Matrix:
[[45  5]
 [22 19]]
```

Question 3.1.3 Discuss what each measure is reporting, why they are different, and why are each of these measures is significant. Explore why we might choose to evaluate the performance of differing models differently based on these factors. Try to give some specific examples of scenarios in which you might value one of these measures over the others.

Accuracy is a simple measure of the percentage of correct classifications or predictions. We use this method to evaluate the performance of a model when neither false negatives nor false positives are more costly than the other. For example, a classifier that predict whether an individual is male or female would depend on accuracy the best. Both false negatives and false positives are equally costly, in this situation.

Precision is the measure of the ratio of true positives to the sum of true positives and false positives. This means that we are looking at the percentage of samples that we classified as true that were correct. We use this method to evaluate the performance of a model, when we want to get a high accuracy when only looking at the true classifications, ignoring all of the false classifications. For example, security techniques such as facial recognition would depend highly on precision. This is because we want to minimize the false positive rate, or in other words, minimize the rate at which the security system fails and grants access to someone who is not authorized. On the other hand, we won't care if we have a high rate of false negatives, or if it doesn't grant access to someone who is authorized, because there may be alternate security methods, like a password.

Recall is the measure of the ratio of true positives to the sum of true positives and false negatives. This means that we are looking at the percentage of samples that are actually true that were classified true. We use this method to evaluate the performance of a model, when we want to get a high accuracy when looking at samples that actually have the true classification. For example, this project (classifying an individual as sick or not) would depend highly on recall score. We care in particular about how we well can identify sick individuals, but we don't care as much about a person who is healthy and incorrectly classified as sick (false positives), because some other tests will reveal that they are healthy.

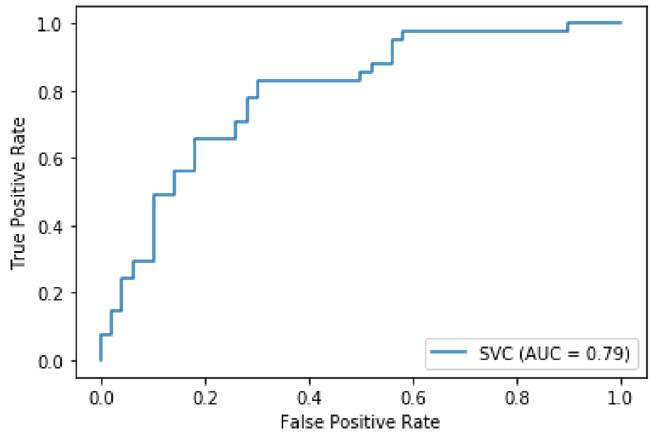
F1 is the weighted average of the recall score and the precision score. It is defined as two times the product of the recall and precision scores divided by the sum of the recall and precision scores. We use this metric typically for the same reasons as accuracy - when neither classifications are more costly than the other. However, F1 score provides a better metric than accuracy, because it takes into consideration the balance between the classification rates. For example, consider a fraud detection system. The accuracy may be good, meaning that it identifies true positives and true negatives, in total, very well. However, if we just look at the recall score, we may see that it is much lower than the precision. This means that the false negative rates are very high and that fraud detection is actually missing a lot of the fraud that is happening. The accuracy score was high because there was an imbalance in the score - the high precision score was covering up the low recall score. This does not mean that we should just look at recall score though. If the precision was low (false positive rates were high), it would be frustrating for the user trying to make a transaction if the model was constantly accusing him or her of fraud, when it actually isn't. Thus, F1 score is better metric to evaluate a model than accuracy, as it takes into consideration the imbalance of the precision and recall scores.

The confusion matrix simply represents the count of each type of classification that we made in our test set. The values of the matrix, from left to right, top to bottom, represent true positives, false positives, false negatives and true negatives.

Question 3.1.4 Plot a Receiver Operating Characteristic curve, or ROC curve, and describe what it is and what the results indicate

```
In [19]: metrics.plot_roc_curve(svm_raw_clf, raw_X_test, raw_y_test)
```

```
Out[19]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x21748a84488>
```



The ROC curve, or Receiver Operating Characteristics curve, is a graph that relates the false positive rate against the true positive rate. The false positive rate is defined as the number of false positives divided by the sum of true positives and false positives. In other words, we can ask the question: out of all samples we classified as positive, regardless of it being correct or incorrect, how many did we classify incorrectly? The true positive rate is defined as the number of true positives divided by the sum of true positives and false negatives. In other words, we can ask the question: out of all samples that had a true label of positive, how many did we correctly classify? This is the same as recall score. These two values relate in that if we have a high true positive rate, it must mean that we also have some falsely classified positive samples. Thus, we will also get a high false positive rate, and vice versa. However, we can measure how well the model performs by measuring the area under the curve. If the area under the curve is higher, we are better at correctly classifying samples.

Question 3.1.5 Rerun, using the exact same settings, only this time use your processed data as inputs.

```
In [20]: svm_clf = SVC(probability=True, gamma="scale")
         svm_clf.fit(X_train, y_train)
```

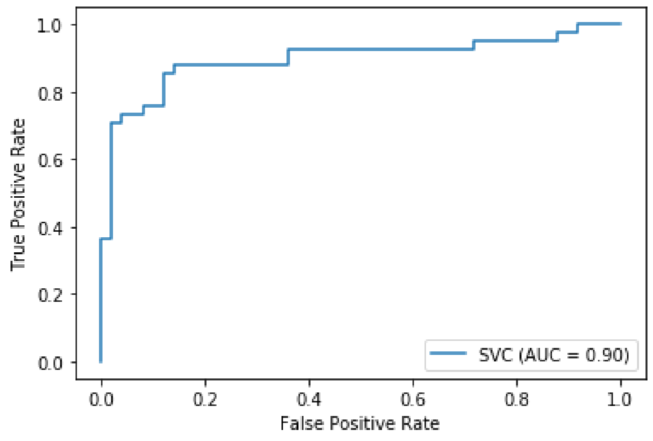
```
Out[20]: SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
             decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
             max_iter=-1, probability=True, random_state=None, shrinking=True, tol=0.001,
             verbose=False)
```

Question 3.1.6 Report the accuracy, precision, recall, F1 Score, confusion matrix, and plot the ROC Curve of the resulting model.

In [21]:
```
svm_preds = svm_clf.predict(X_test)

print("Accuracy: {}".format(metrics.accuracy_score(y_test, svm_preds)))
print("Precision: {}".format(metrics.precision_score(y_test, svm_preds)))
print("Recall: {}".format(metrics.recall_score(y_test, svm_preds)))
print("F1: {}".format(metrics.f1_score(y_test, svm_preds)))
print("Confusion Matrix: \n{}".format(metrics.confusion_matrix(y_test, svm_preds)))

metrics.plot_roc_curve(svm_clf, X_test, y_test)
```

```
Accuracy: 0.8681318681318682
Precision: 0.8372093023255814
Recall: 0.8780487804878049
F1: 0.8571428571428572
Confusion Matrix: 
[[43  7]
 [ 5 36]]
```

Out[21]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x21748b35308>
```



Question 3.1.7 Hopefully you've noticed a dramatic change in performance. Discuss why you think your new data has had such a dramatic impact.

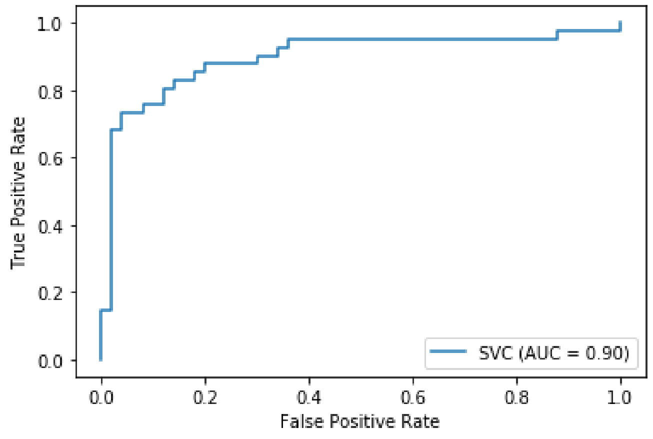
As mentioned in 2.4, our OneHotEncoder added extra columns to our dataframe that represent each value of the cateogical features. In our unprocessed dataset, we left the columns in their unencoded form, and thus the algorithm was trying to find some sort of linear relationship between the values, when it really didn't exist. After OneHotEncoding, the algorithm was able to use the new boolean columns as a better predictor for the label. Thus, our new model has a much improved performance.

Question 3.1.8 Rerun your SVM, but now modify your model parameter kernel to equal 'linear'. Again report your Accuracy, Precision, Recall, F1 scores, and Confusion matrix and plot the new ROC curve.

In [22]:
```python
# SVM
svm_linear_clf = SVC(kernel="linear", probability=True, gamma="scale")
svm_linear_clf.fit(X_train, y_train)

svm_linear_preds = svm_linear_clf.predict(X_test)

print("Accuracy: {}".format(metrics.accuracy_score(y_test, svm_linear_preds)))
print("Precision: {}".format(metrics.precision_score(y_test, svm_linear_preds)))
print("Recall: {}".format(metrics.recall_score(y_test, svm_linear_preds)))
print("F1: {}".format(metrics.f1_score(y_test, svm_linear_preds)))
print("Confusion Matrix: \n{}".format(metrics.confusion_matrix(y_test, svm_linear_preds)))

metrics.plot_roc_curve(svm_linear_clf, X_test, y_test)
```

```
Accuracy: 0.8241758241758241
Precision: 0.7906976744186046
Recall: 0.8292682926829268
F1: 0.8095238095238095
Confusion Matrix: 
[[41  9]
 [ 7 34]]
```

Out[22]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x21748bb3f88>
```



Question 3.1.9 Explain the what the new results you've achieved mean. Read the documentation to understand what you've changed about your model and explain why changing that input parameter might impact the results in the manner you've observed.

By changing the kernel type to linear, we change the way that the model is trained by adjusting the way it comes up with the decision boundaries. The default parameter is "rbf", which stands for Radial Basis Function. This means that the model's decision boundaries are determined by radial functions, or those that are based on a polar coordinate system. Boundaries of this type will be more curved and rounded. In contrast, the "linear" parameter means that the model's decision boundaries are linear or simply straight lines. We can see that a linear kernel type actually performs worse, as each of its evaluation metrics were lower than that of the "rbf" kernel type. However, each dataset is different, and different kernel types may perform better on different datasets.

Logistic Regression

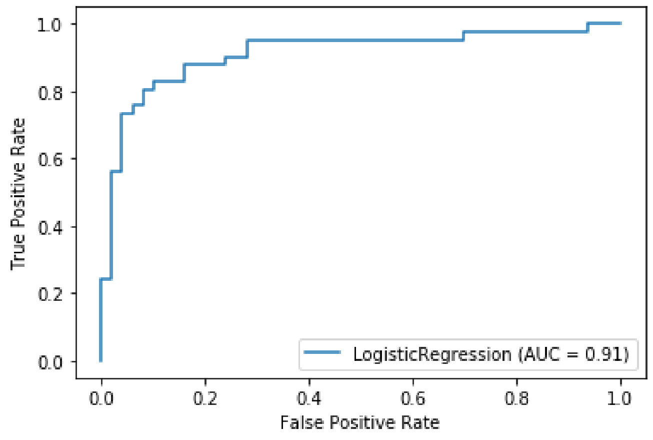
Knowing that we're dealing with a linearly configured dataset, let's now try another classifier that's well known for handling linear models: Logistic Regression. Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable.

Question 3.2.1 Implement a Logistical Regression Classifier. Review the [Logistical Regression Documentation (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) for how to implement the model. For this initial model set the solver = 'sag' and max_iter= 10). Report on the same four metrics as the SVM and graph the resulting ROC curve.

In [23]:
```python
# Logistic Regression
log_reg_clf = LogisticRegression(random_state=42, solver="sag", max_iter=10)
log_reg_clf.fit(X_train, y_train)

log_reg_preds = log_reg_clf.predict(X_test)

print("Accuracy: {}".format(metrics.accuracy_score(y_test, log_reg_preds)))
print("Precision: {}".format(metrics.precision_score(y_test, log_reg_preds)))
print("Recall: {}".format(metrics.recall_score(y_test, log_reg_preds)))
print("F1: {}".format(metrics.f1_score(y_test, log_reg_preds)))
print("Confusion Matrix: \n{}".format(metrics.confusion_matrix(y_test, log_reg_preds)))

metrics.plot_roc_curve(log_reg_clf, X_test, y_test)
```

```
Accuracy: 0.8461538461538461
Precision: 0.813953488372093
Recall: 0.8536585365853658
F1: 0.8333333333333333
Confusion Matrix: 
[[42  8]
 [ 6 35]]

C:\Users\shado\Anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning: 
The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

Out[23]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x21748c16988>
```

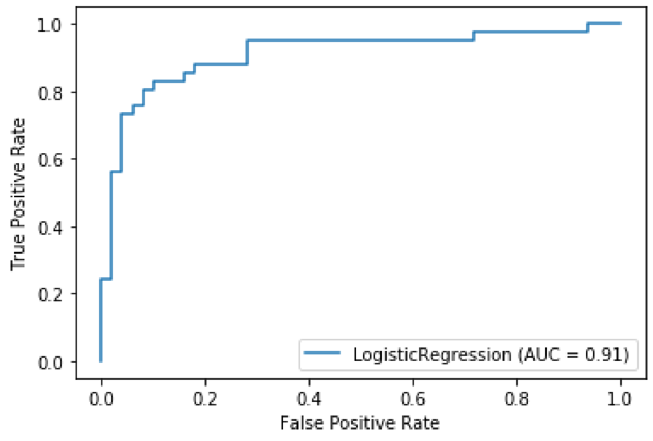


Question 3.2.2 Did you notice that when you ran the previous model you got the following warning: "ConvergenceWarning: The max_iter was reached which means the coef_ did not converge". Check the documentation and see if you can implement a fix for this problem, and again report your results.

In [24]:
```python
# Logistic Regression

new_log_reg_clf = LogisticRegression(random_state=42, solver="sag", max_iter=100)
new_log_reg_clf.fit(X_train, y_train)

new_log_reg_preds = new_log_reg_clf.predict(X_test)

print("Accuracy: {}".format(metrics.accuracy_score(y_test, new_log_reg_preds)))
print("Precision: {}".format(metrics.precision_score(y_test, new_log_reg_preds)))
print("Recall: {}".format(metrics.recall_score(y_test, new_log_reg_preds)))
print("F1: {}".format(metrics.f1_score(y_test, new_log_reg_preds)))
print("Confusion Matrix: \n{}".format(metrics.confusion_matrix(y_test, new_log_reg_preds)))

metrics.plot_roc_curve(new_log_reg_clf, X_test, y_test)
```

```
Accuracy: 0.8351648351648352
Precision: 0.8095238095238095
Recall: 0.8292682926829268
F1: 0.8192771084337348
Confusion Matrix:
[[42  8]
 [ 7 34]]
```

Out[24]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x21748c56048>
```



Question 3.2.3 Explain what you changed, and why that produced an improved outcome.

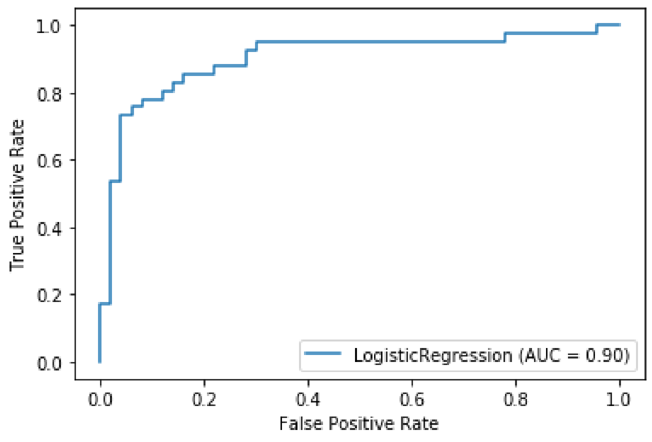
I changed the "max_iter" parameter of the Logistic Regression model from 10 to 100. This means that the learning algorithm was able to find a selection of weights and bias that minimized the cost function. The cost function indicates the error between the predictions and the actual true labels. In the case of logistic regression, the cost function is the negative log likelihood function. In theory, increasing the maximum iterations parameter can only help the model, as it is more likely to converge and find the correct paramters. However, we actually do not see a significant improvement in performance (actually, it regresses slightly). This is probably because this particular dataset is able to converge quickly, and even after 10 iterations, the algorithm was very close to converging. Thus, we don't see a significant improvement in performance, even after updating the maximum iterations parameter.

Question 3.2.4 Rerun your logistic classifier, but modify the penalty = 'none', solver='sag' and again report the results.

In [25]:
```python
# Logistic Regression
new2_log_reg_clf = LogisticRegression(random_state=42, solver="sag", penalty="none", max_iter=100
new2_log_reg_clf.fit(X_train, y_train)

new2_log_reg_preds = new2_log_reg_clf.predict(X_test)

print("Accuracy: {}".format(metrics.accuracy_score(y_test, new2_log_reg_preds)))
print("Precision: {}".format(metrics.precision_score(y_test, new2_log_reg_preds)))
print("Recall: {}".format(metrics.recall_score(y_test, new2_log_reg_preds)))
print("F1: {}".format(metrics.f1_score(y_test, new2_log_reg_preds)))
print("Confusion Matrix: \n{}".format(metrics.confusion_matrix(y_test, new2_log_reg_preds)))

metrics.plot_roc_curve(new2_log_reg_clf, X_test, y_test)
```

```
Accuracy: 0.8461538461538461
Precision: 0.8292682926829268
Recall: 0.8292682926829268
F1: 0.8292682926829268
Confusion Matrix:
[[43  7]
 [ 7 34]]


C:\Users\shado\Anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:330: ConvergenceWarning:
The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

Out[25]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x21748cb6dc8>
```



Question 3.2.5 Explain what what the penalty parameter is doing in this function, what the solver method is, and why this combination likely produced a more optimal outcome.

The penalty parameter sets the way in which the logistic regression computes the regularization for the model. A regularization is an extra value that is added to the cost function, and its purpose is to lower the weights of the model, thus leading to a simpler model and the reduction of overfitting. We can specify the penalty to be computed using an L1 norm, an L2 norm, or an elasticnet. We can also set it to none, where the model will not use a regularization term at all.

The solver is the algorithm by which the Logistic Regression algorithm uses to find the minimum cost. It begins with an initial set of weights and it updates these weights per iteration, until it reaches a point of convergence, which is the point of minimum cost. When solver is set to "sag", it means that the algorithm used is called Stochastic Average Gradient Descent. This algorithm picks one point in the dataset and applies the weights to that point in order to get a prediction. If it is incorrect, the algorithm will update the weights. A similar method called Stochastic Gradient Descent uses just a single point for comparison per iteration. Its modified version, Stochastic Average Gradient Descent, incorporates results from previous iterations, so that it can converge faster than Stochastic Gradient Descent.

Question 3.2.6 Both logistic regression and linear SVM are trying to classify data points using a linear decision boundary, then what’s the difference between their ways to find this boundary?

Logistic Regression finds the decision boundary by finding a set of weights and minimizing the cost, over a number of iterations, using an algorithm known as gradient decsent. Once the weights are found, they can be generalized to the hyperplane/decision boundary.

In contrast, SVM finds the closest two data points and draws support vectors that run through these lines. The support vectors are parallel to each other, and SVM tries to maximize the distance between them. The boundary that is found is the one that lies directly in the middle of the two support vectors.

Clustering Approaches

Let us now try a different approach to classification using a clustering algorithm. Specifically, we're going to be using K-Nearest Neighbor, one of the most popular clustering approaches.

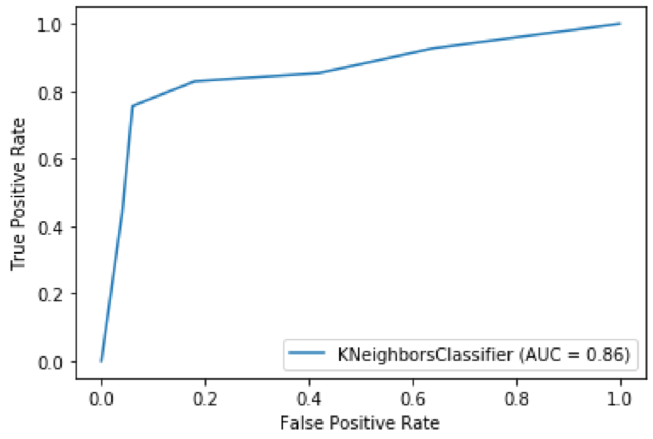
K-Nearest Neighbor

Question 3.3.1 Implement a K-Nearest Neighbor algorithm on our data and report the results. For this initial implementation simply use the default settings. Refer to the KNN Documentation (https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html) for details on implementation. Report on the accuracy of the resulting model.

In [26]:
```
# k-Nearest Neighbors algorithm

knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_train)

knn_preds = knn_clf.predict(X_test)

print("Accuracy: {}".format(metrics.accuracy_score(y_test, knn_preds)))
print("Precision: {}".format(metrics.precision_score(y_test, knn_preds)))
print("Recall: {}".format(metrics.recall_score(y_test, knn_preds)))
print("F1: {}".format(metrics.f1_score(y_test, knn_preds)))
print("Confusion Matrix: \n{}".format(metrics.confusion_matrix(y_test, knn_preds)))

metrics.plot_roc_curve(knn_clf, X_test, y_test)
```

```
Accuracy: 0.8241758241758241
Precision: 0.7906976744186046
Recall: 0.8292682926829268
F1: 0.8095238095238095
Confusion Matrix:
[[41  9]
 [ 7 34]]
```

Out[26]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x21749131248>
```



Question 3.3.2 For clustering algorithms, we use different measures to determine the effectiveness of the model. Specifically here, we're interested in the Homogeneity Score, Completeness Score, V-Measure, Adjusted Rand Score, and Adjusted Mutual Information. Calculate each score (hint review the SKlearn Metrics Clustering documentation for how to implement).

In [27]:
```
print("Homogeneity score: {}".format(metrics.homogeneity_score(y_test, knn_preds)))
print("Completeness score: {}".format(metrics.completeness_score(y_test, knn_preds)))
print("V-Measure score: {}".format(metrics.v_measure_score(y_test, knn_preds)))
print("Adjusted Rand Score score: {}".format(metrics.adjusted_rand_score(y_test, knn_preds)))
print("Adjusted Mutual Information score: {}".format(metrics.adjusted_mutual_info_score(y_test, k
```

```
Homogeneity score: 0.32939901704632263
Completeness score: 0.32778522042452185
V-Measure score: 0.32859013729750264
Adjusted Rand Score score: 0.41391364708155476
Adjusted Mutual Information score: 0.3231072567137206
```

Question 3.3.3 Explain what each score means and interpret the results for this particular model.

Homogeneity score refers to how much of a particular cluster is of the same label. It returns a value between 0 and 1, where 0 represents an inhomogeneous cluster (there is the presence of both labels in the cluster) and 1 represents a homogeneous cluster (all samples in the cluster are of the same label).

Completeness score refers to all of the data with the same label and whether or not they are in the same cluster. It returns a value between 0 and 1, where 1 represents the fact that all data with the same labels are within the same cluster, and 0 represents the opposite.

V-Measure score refers to the harmonic mean between the homogeneity score and the completeness score. It is defined by the equation: ((1 + beta) * homogeneity * completeness)/(beta * homogeneity + completeness). In this case, beta refers to the ratio of weight attributed to homogeneity compared to completeness. If beta is greater than 1, completeness is weighted more strongly, and vice versa.

Adjusted Rand score refers to the rand index score that is then adjusted to fit between 0 and 1. Rand Index is a measure of similarity between two clusterings by considering all pairs of possible samples and counting whether they are assigned to the same or different clusters in the predicted and true clusterings. Then, this rand index is fed into this equation for normalization: (RI - Expected_RI)/(max(RI) - Expected_RI). A score of 1 represents when clusterings are identical, but a score of 0 represents that they were labeled randomly and independently of the number of clusters and samples.

Adjusted Mutual Information score refers to an adjustment to the mutual information score to account for chance. The mutual information score represents a measure of the mutual dependence between two random variables. The adjusted mutual score accoutns for the fact that the mutual information score is generally higher for two clusterings with a larger number of clusters, even though there may not actually be more information shared. It is defined by the equation: (MI(U, V) - E(MI(U, V)))/(avg(H(U), H(V)) - E(MI(U, V))) where U and V are two clusterings, H(.) is the entroy of a random variable, and E(.) is the expectation of a random variable. It returns a value of 1 when the two clusters or sets are identical and a value of 0 when the clusters are independently labeled.

As we're beginning to see, the input parameters for your model can dramatically impact the performance of the model. How do you know which settings to choose? Studying the models and studying your datasets are critical as they can help you anticipate which models and settings are likely to produce optimal results. However sometimes that isn't enough, and a brute force method is necessary to determine which parameters to use. For this next question we'll attempt to optimize a parameter using a brute force approach.

Question 3.3.4 Parameter Optimization. The KNN Algorithm includes an n_neighbors attribute that specifies how many neighbors to use when developing the cluster. (The default value is 5, which is what your previous model used.) Lets now try n values of: 1, 2, 3, 5, 10, 20, 50, and 100. Run your model for each value and report the 6 measures (5 clustering specific plus accuracy) for each. Report on which n value produces the best accuracy and V-Measure. (HINT leverage python's ability to loop to run through the array and generate results without needing to manually code each iteration).

In [28]:
```
n_values = [1, 2, 3, 5, 10, 20, 50, 100]

for n in n_values:
    new_knn_clf = KNeighborsClassifier(n_neighbors=n)
    new_knn_clf.fit(X_train, y_train)

    new_knn_preds = new_knn_clf.predict(X_test)

    print("Scores for KNN with {} Neighbors".format(n))
    print("Accuracy: {}".format(metrics.accuracy_score(y_test, new_knn_preds)))
    print("Homogeneity score: {}".format(metrics.homogeneity_score(y_test, new_knn_preds)))
    print("Completeness score: {}".format(metrics.completeness_score(y_test, new_knn_preds)))
    print("V-Measure score: {}".format(metrics.v_measure_score(y_test, new_knn_preds)))
    print("Adjusted Rand Score score: {}".format(metrics.adjusted_rand_score(y_test, new_knn_pred
    print("Adjusted Mutual Information score: {}".format(metrics.adjusted_mutual_info_score(y_tes
    print('\n')
```

```
Scores for KNN with 1 Neighbors
Accuracy: 0.7582417582417582
Homogeneity score: 0.20985098431317886
Completeness score: 0.20853140645344054
V-Measure score: 0.20918911440934143
Adjusted Rand Score score: 0.2585989950396032
Adjusted Mutual Information score: 0.20273603619785738


Scores for KNN with 2 Neighbors
Accuracy: 0.7472527472527473
Homogeneity score: 0.17861037960004397
Completeness score: 0.18315005279865285
V-Measure score: 0.1808517323867137
Adjusted Rand Score score: 0.23617238233242102
Adjusted Mutual Information score: 0.17405400512227162


Scores for KNN with 3 Neighbors
Accuracy: 0.7802197802197802
Homogeneity score: 0.2496425020846802
Completeness score: 0.2480727085491445
V-Measure score: 0.24885512975288338
Adjusted Rand Score score: 0.30646282803441144
Adjusted Mutual Information score: 0.24272572930559527


Scores for KNN with 5 Neighbors
Accuracy: 0.8241758241758241
Homogeneity score: 0.32939901704632263
Completeness score: 0.32778522042452185
V-Measure score: 0.32859013729750264
Adjusted Rand Score score: 0.41391364708155476
Adjusted Mutual Information score: 0.3231072567137206


Scores for KNN with 10 Neighbors
Accuracy: 0.8241758241758241
Homogeneity score: 0.3258940510936594
Completeness score: 0.3258940510936594
V-Measure score: 0.3258940510936594
Adjusted Rand Score score: 0.41391854016339674
Adjusted Mutual Information score: 0.3203745872108415


Scores for KNN with 20 Neighbors
Accuracy: 0.8461538461538461
Homogeneity score: 0.37620925388444065
Completeness score: 0.3791512192913599
```

```
V-Measure score: 0.3776745074289339
Adjusted Rand Score score: 0.4735103958323643
Adjusted Mutual Information score: 0.37255766188986056


Scores for KNN with 50 Neighbors
Accuracy: 0.8571428571428571
Homogeneity score: 0.40435266727056474
Completeness score: 0.40578245127841794
V-Measure score: 0.4050662975822981
Adjusted Rand Score score: 0.5047641197201996
Adjusted Mutual Information score: 0.4001858365952612


Scores for KNN with 100 Neighbors
Accuracy: 0.8571428571428571
Homogeneity score: 0.40813676966794804
Completeness score: 0.41851022925510817
V-Measure score: 0.4132584120276255
Adjusted Rand Score score: 0.5047818386733216
Adjusted Mutual Information score: 0.4083893187908626
```

The n-value that produces the best results is 100. It results in the highest accuracy at around 0.857 or 85.7%. It seems as though we are generalizing quite well when we look at the nearest 100 neighbors in order to classify a point, which is around half of the entire training set. The accuracy is quite good, as we are classifying around 85% of the data correctly. The lower n values seem to be overfitting the data and is not generalized enough. Thus, their scores are much lower than that of 100 neighbors.

Question 3.3.5 When are clustering algorithms most effective, and what do you think explains the comparative results we achieved?

In [29]:
```python
print("Scores for KNN with {} Neighbors".format(100))
print("Accuracy: {}".format(metrics.accuracy_score(y_test, new_knn_preds)))
print("Precision: {}".format(metrics.precision_score(y_test, new_knn_preds)))
print("Recall: {}".format(metrics.recall_score(y_test, new_knn_preds)))
print("F1: {}".format(metrics.f1_score(y_test, new_knn_preds)))
```

```
Scores for KNN with 100 Neighbors
Accuracy: 0.8571428571428571
Precision: 0.8888888888888888
Recall: 0.7804878048780488
F1: 0.8311688311688312
```

Clustering algorithms are most effective when there are no associated labels with the dataset, or in other words, we are performing unsupervised training. This allows us to identify different groups without actually knowing what their true labels are. We can do this just by looking at the similarities between the samples' feature values.

In comparing different classification algorithms, we explored the performances of three main classification algorithms: SVM, logistic regression and K-Nearest Neighbors. Each of these algorithms have their own strengths and weaknesses. In our case, we focus more on the recall score and the F1 score. As mentioned above, recall score is more important in our case, because we don't want to misclassify diseased patients as healthy. Additionally, F1 score is also important because it provides a more general and balanced score compared to accuracy.

SVM achieved an F1 score of 0.8571 and a recall score of 0.8780. SVM has both the highest F1 and recall score for our particular use case, and thus it can be concluded that it performed the best. SVM performs well there is a clear separation between two classes.

Logistic regression achieved an F1 score and a recall score of 0.8293. Logistic regression had a slightly lower F1 score but a higher recall score than KNN. Logistic regression is beneficial if we want to find out the probability that a sample belongs to a particular class, instead of just a guess at what the label is.

KNN achieved an F1 score of 0.8311 and a recall score of 0.7804. KNN achieved the worst results out of all our algorithms, but that does not mean that it is the worst algorithm. Its strengths are apparent when there is a lot of overlap between feature values of data points. Additionally, it does not require a training step, as it simply just needs to find the nearest neighbors within the existing dataset. That being said, it must carry around the entire dataset, which may be expensive or have high memory consumption. Finally, the KNN algorithm yields complex decision boundaries.