CS M148 Project 2 - Binary Classification 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 (0-4) colored by flourosopy
- **thal:** 1 = normal; 2 = fixed defect; 3 = reversable defect

- **sick:** Indicates the presence of Heart disease (True = Disease; False = No disease)

`sick` is the label that you will be predicting.

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
# If you can run this cell without any errors, you're ready to go.
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 seaborn as sns # used for plot interactive graph.
from sklearn.model_selection import train_test_split, cross_val_score, GridSearcl
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 matplotlib import pyplot

import os
import itertools
import random

%matplotlib inline

random.seed(148)
```

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 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()
```

Part 1. Load the Data and Analyze

Let's load our dataset so we can work with it (correct the 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, the describe method, and the info method to display some of the rows so we can visualize the types of data fields we'll be working with.

In [5]:
```
# Your code here
# You may use separate cells if you'd like (one for `head`, one for `describe`, e
data.head(10)
```

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
5	57	1	0	140	192	0	1	148	0	0.4	1	0	1	False
6	56	0	1	140	294	0	0	153	0	1.3	1	0	2	False
7	44	1	1	120	263	0	1	173	0	0.0	2	0	3	False
8	52	1	2	172	199	1	1	162	0	0.5	2	0	3	False
9	57	1	2	150	168	0	1	174	0	1.6	2	0	2	False

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

Out[6]:

	age	sex	cp	trestbps	chol	fbs	restecg	tha
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.00
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.64
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.90
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.00
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.50
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.00
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.00
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.00

In [7]:
```
data.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    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  sick      303 non-null    bool   
dtypes: bool(1), float64(1), int64(12)
memory usage: 31.2 KB
```

Question 1.2

Sometimes data will be stored in different formats (e.g., string, date, boolean), but many learning methods work strictly on numeric inputs. Call the info method to determine the datafield type for each column. Are there any that are problemmatic and why?

In [8]:
```
data.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    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  sick      303 non-null    bool   
dtypes: bool(1), float64(1), int64(12)
memory usage: 31.2 KB
```

In [9]:
```
# Calling the data.info we see that the sick field is a bool value expressed in t
# For this we should conver those results to binary 0/1 to appropriately handle t
# This could be problematic since without a number representation we may not be c
# Statistic methods/functions that help us to draw correlations and other meaning
# However, by using number values we are able to more accurately interpolate resu
```

[Use this area to describe any fields you believe will be problemmatic and why] E.g., All the columns in our dataframe are numeric (either int or float), however our target variable 'sick' is a boolean and may need to be modified.

Question 1.3

Determine if we're dealing with any null values. If so, report which columns.

In [10]:
```
inc_rows = data[data.isnull().any(axis=1)].head()
```

In [11]:
```
inc_rows
```

Out[11]:

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	sick

In [12]:
```
# As Shown above there are no null values in the columns or rows. Also the earlie
# 303 non null count across the 14 columns.
```

Question 1.4

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 [13]: data
```

```
Out[13]:
     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
...  ...  ...  ...      ...   ...  ...      ...      ...    ...      ...    ...  ...  ...    ...
298   57    0   0       140   241    0        1      123      1      0.2      1   0     3   True
299   45    1   3       110   264    0        1      132      0      1.2      1   0     3   True
300   68    1   0       144   193    1        1      141      0      3.4      1   2     3   True
301   57    1   0       130   131    0        1      115      1      1.2      1   1     3   True
302   57    0   1       130   236    0        0      174      0      0.0      1   1     2   True

303 rows × 14 columns
```

```
In [14]: # Shown above we see that the sick value is listed in boolean values of T/F still
```

```
In [15]: data["sick"] = data["sick"].astype(int)
```

```
In [16]: data
```

```
Out[16]:
     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
```

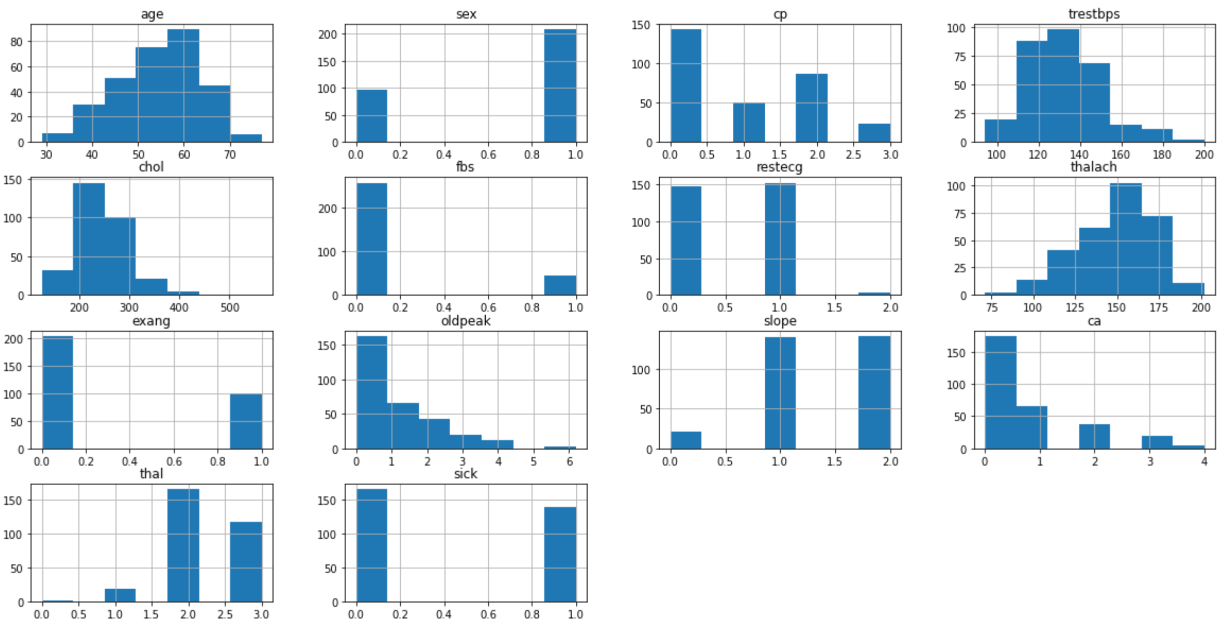
```
In [17]: # Now the values for the sick field are either 0 0r 1 where 0 sorresponds to Fals
```

Question 1.5

Now that we have a feel for the data-types for each of the variables, plot histograms of each field and attempt to get a feel of how each variable performs (for example, is it a binary, or limited selection, or does it follow a gradient)?

In [18]:
```
data.hist(bins = 7, figsize = (20,10))
```

Out[18]:
```
array([[<AxesSubplot:title={'center':'age'}>,
        <AxesSubplot:title={'center':'sex'}>,
        <AxesSubplot:title={'center':'cp'}>,
        <AxesSubplot:title={'center':'trestbps'}>],
       [<AxesSubplot:title={'center':'chol'}>,
        <AxesSubplot:title={'center':'fbs'}>,
        <AxesSubplot:title={'center':'restecg'}>,
        <AxesSubplot:title={'center':'thalach'}>],
       [<AxesSubplot:title={'center':'exang'}>,
        <AxesSubplot:title={'center':'oldpeak'}>,
        <AxesSubplot:title={'center':'slope'}>,
        <AxesSubplot:title={'center':'ca'}>],
       [<AxesSubplot:title={'center':'thal'}>,
        <AxesSubplot:title={'center':'sick'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```

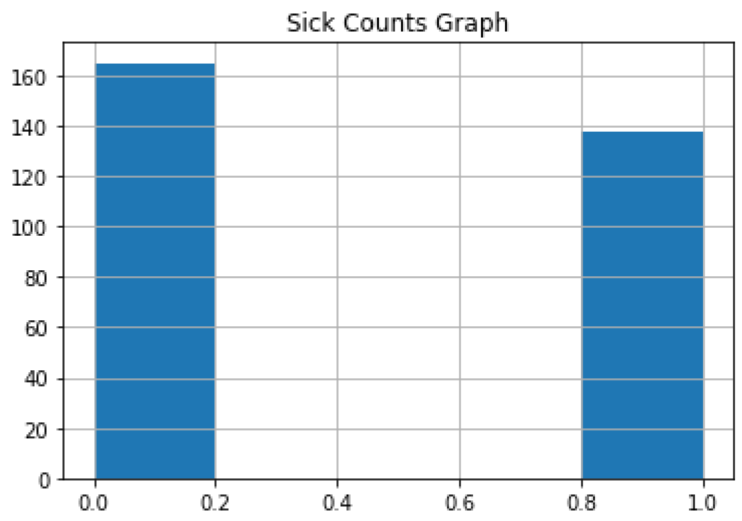


In [19]:
```
# Binary: sex, fibs, exang, sick
# Limited Selection: cp, restecg, slope, ca, thal
# Gradient: age, trestbps, chol, thalach, oldpeak
```

Question 1.6

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 [20]:
```
data["sick"].hist(bins=5); plt.title("Sick Counts Graph"); plt.show()
```



In [21]:
```
data["sick"].value_counts()
```

Out[21]:
```
0    165
1    138
Name: sick, dtype: int64
```

[Include description of findings here] E.g., As we can see, our sample contains xxx healthy individuals and yyy sick individuals, which reflects a [your conclusion here].

As we can see our sample contains 165 healthy (not sick) individuals and 138 sick individuals. The general closeness of these values seems to represent an equitable number that is sufficient for a balanced data set.

Question 1.7

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.

[You answer here] E.g., artificially inflating numbers to balance a dataset may result in xxx. Concurrently showing statistically uncommon events as likely may result in yyy.

Artificially inflating numbers to balance a dataset could result in incorrect statistics and conclusions that can be drawn from the data such as the correlations between different categories or inaccurate error estimations or a inadequate mean value within a category based on removed or artificailly inflated data.

Undersampling can occur by deleting or removing data from one class to try and match the numbers from another class. By removing data we are likely to sometimes discard useful information that could be important to identiying correlations or distingusihing key concepts among certain categories of data. An advantage is that this could improve processing time and make data computaiton quicker, but we are likely to lose important information.

Oversampling occurs when we generate attributes from observations in a minority class. This can generally be seen as a way to increment the size of rare samples. THere is an advantage of no loss of information, but this can cause overfitting and bias the data making some values seem more frequent then they should in the sample.

It is best to avoid both undersampling and oversampling when possible. We do not want to introduce bias by manipulating data such as duplicating a sample, removing relevant good data, or forcng values to change through programmer interference. The data should be left as correctly received as possible and not misrepresented if it will cause no computation issues (I.E. does not require pruning for syntax/code reasons, like how we changed bool to binary)

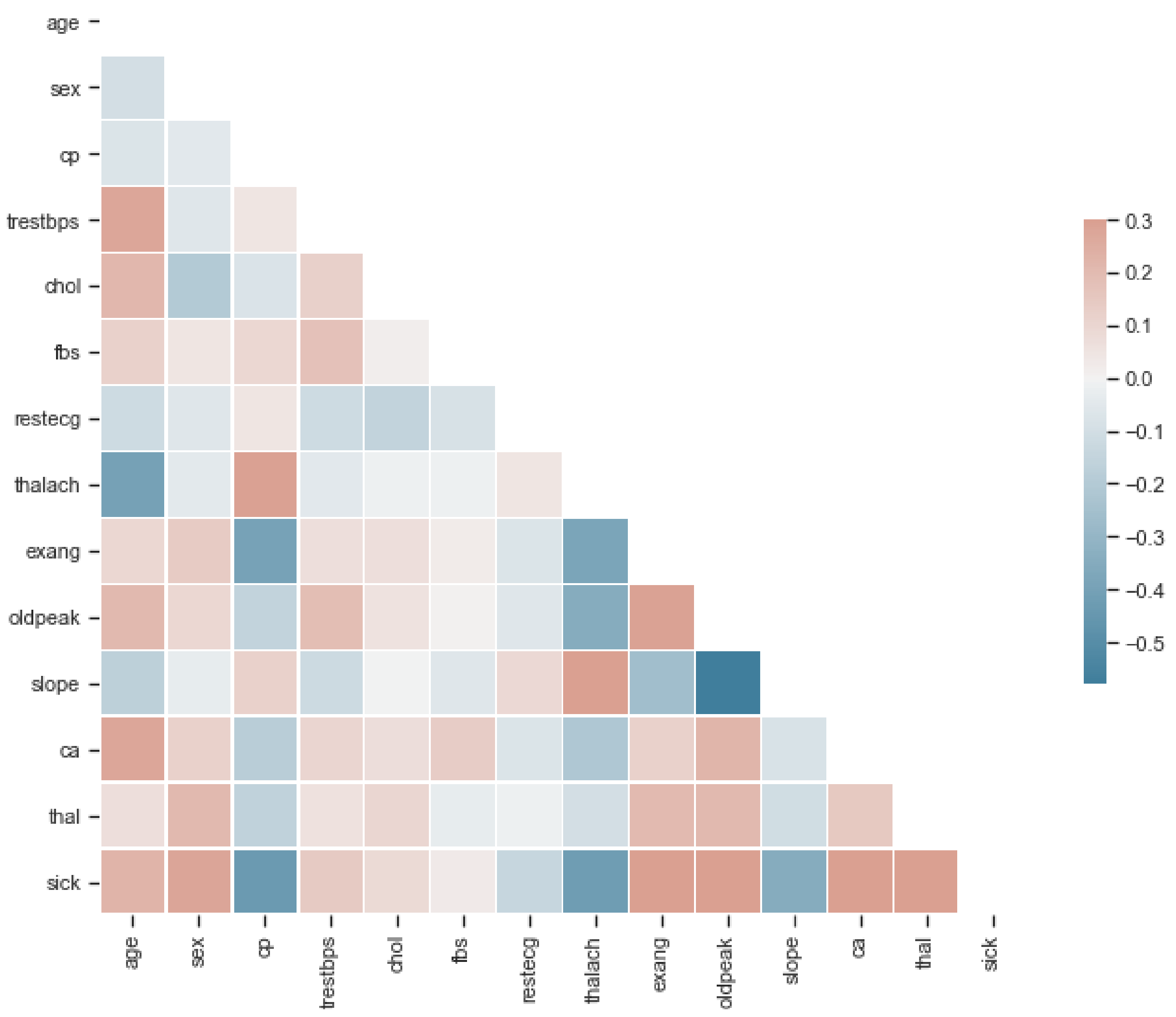
Question 1.8

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 (get some domain knowledge). Intuitively, why do you think some variables correlate more highly than others? Also discuss some pairs of variables that have very little correlation and why this might be the case for them.

Hint: one possible approach you can use the `sns.heatmap()` function to map the corr() method. Note: if your heatmap is not entirely visible, this link (https://github.com/mwaskom/seaborn/issues/1773#issuecomment-546466986) may be helpful in solving the issue.

```
In [22]: # SOurce
         # https://seaborn.pydata.org/examples/many_pairwise_correlations.html
         sns.set_theme(style="white")
         sns.set(style="ticks") # Set heat map
         corr = data.corr() # Set up correlation matrix
         mask = np.triu(np.ones_like(corr, dtype=bool))
         f, ax = plt.subplots(figsize=(11,9))
         cmap = sns.diverging_palette(230, 20, as_cmap=True)
         sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
                     square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

```
Out[22]: <AxesSubplot:>
```



In [23]:
```
corr
```
Out[23]:

	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	0.
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.044020	0.
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.295762	-0.
trestbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.046698	0.
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.009940	0.
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.008567	0.
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.044123	-0.
thalach	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.000000	-0.
exang	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.378812	1.
oldpeak	0.210013	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.344187	0.
slope	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.386784	-0.
ca	0.276326	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.213177	0.
thal	0.068001	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.096439	0.
sick	0.225439	0.280937	-0.433798	0.144931	0.085239	0.028046	-0.137230	-0.421741	0.

[Discuss correlations here] E.g., We find the the strongest direct correlation between the presence of exercise induced angina (also a binary), and depression induced by exercise relative to rest indicates a strong direct correlation. Both of these are understandable as heart failure under conditions of duress is a clear indication of heart disease. Conversely, maximum heart rate achieved is inversely correlated, likely as a healthy heart is unable to achieve a high heart rate.

In the above we see that most of the categories do not correlate strongly for the most part. But the most red and blue squares represent the highest magnitude for postive and negative correlations. Negative correlations include {(thalach, age), (sick,cp), (exang,cp), (thalac, age), ...}. The most notable negative correlation is with slope and oldpeak at about -.578 value. Some postive correlations include {(ca, age), (sick, sex), (sick, exang), (sick, oldpeak), ...}. From the positive correlations I found that (sick, exang) seemed to be a high value of about .436 correlation. Also the (sick, sex) was an interesting compatison with a corelation of .28 . Some near zero correlations include (thalac, col).

The near zero correlations for (thalac, col) could be explained by an academic journal which found that a healthy heart rate corresponded with decreases in cholesterol. "Effects of different exercise training intensities on lipoprotein cholesterol fractions in healthy middle-aged men". It would make sense then that a higher thalac should not correlate positively with a higher cholesterol level which is what we see in our data. However we do not see the two correlate negatively in a large way, but there is a minute slight negative correlation which supports the article and general best practices for exercising and lowering cholesterol. (slope, chol) are near zero because the type of slopeness seems to have very little to do with the cholesterum in a person. If this was not true, then I would expect more commercials or the medical industry to have been reccommending everyone to change workouts to either a positive or negative slope if there was a significant change to reduce cholesterol by doing so, but as show above there is a near zero correlation.

The positive correlation of (sick, exang) may be explained by what causes angina. Angina results from not enough oxygen rich blood reaches the heart. This can often lead to issues such as nausea and shortness of breath to name a few types of ailments one may endur with angina. https://www.mayoclinic.org/diseases-conditions/angina/symptoms-causes/syc-20369373 (https://www.mayoclinic.org/diseases-conditions/angina/symptoms-causes/syc-20369373) Because of the extra stress that the body would endure through having exercise incuded angina it would make sense that those affected are likely to feel sick after having such insufficient oxygen levels for a body that was exercising, and those who do not have angina would not be contributing to the odds of being sick as often post workout.

The negative corelation of (slope, oldpeak) demonstrates that depression received based on excersicing vs resting and the slope of the exercise iteself did not cause people to become depressed in a major way exercising with an increased slope. It is possible that the exercise itself is what is wanted to be avoided or that the slope plays a minor role in there excitement or depression in regards to exercising.

```
In [ ]:
```

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.

We're going to ask you to prepare 2 batches of data:

1. Raw numeric data that hasn't gone through any additional pre-processing.
2. 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 [24]:
```
labels = data['sick']
data = data.drop(['sick'], axis=1)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 13 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
dtypes: float64(1), int64(12)
memory usage: 30.9 KB
```

Question 2.2

Create your 'Raw' unprocessed training data by dividing your dataframe into training and testing cohorts, with your training cohort consisting of 85% 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 [33]:
```
from sklearn.model_selection import train_test_split
tst = 1 - .85
mst = 148
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=tst,
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(257, 13)
(46, 13)
(257,)
(46,)
```

Question 2.3

In lecture we learned about K-Nearest Neighbor. One thing we noted was because KNN rlies on Euclidean distance, they are highly sensitive to the relative magnitude of different features. Let's see that in action! Implement a KNN algorithm on our data (use `scikit-learn`) and report the results. For this initial implementation simply use the default settings. Report on the accuracy of the resulting model.

```
In [34]: KNN = KNeighborsClassifier()
         KNN.fit(X_train, y_train)
         print("KNN Fit Complete")
         print("KNN Test Results: ", KNN.score(X_test, y_test))
         print("With a value of nearly 80% I reason its a good model that will likely prec
```

```
KNN Fit Complete
KNN Test Results:  0.782608695652174
With a value of nearly 80% I reason its a good model that will likely predict a
n accurate label for most input
```

Question 2.4

Now implement a pipeline of your choice to transform the data. You can opt to handle null values and categoricals however you wish, however please scale your numeric features using standard scaler. Refer to Project 1 for a example pipeline that you can mimic.

```
In [35]: from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
         from sklearn.compose import ColumnTransformer
         data_num = data.drop(["sex","cp","fbs","restecg","exang","slope","ca","thal"], ax

         numerical_features = list(data_num)
         categorical_features = ["sex","cp","fbs","restecg","exang","slope","ca","thal"]
         print(numerical_features)
         print(categorical_features)
```

```
['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal']
```

```
In [36]: full_pipeline = ColumnTransformer([
                 ("num", StandardScaler(), numerical_features),
                 ("cat", OneHotEncoder(), categorical_features),
             ])
         prepared_data = full_pipeline.fit_transform(data)
```

Question 2.5

Now split your pipelined data into an 85/15 split and run the same KNN as you did previously. Report its accuracy, and discuss the implications of the different results you are obtaining.

```
In [38]: X_train, X_test, y_train, y_test = train_test_split(prepared_data, labels, test_s
         KNN = KNeighborsClassifier()
         KNN.fit(X_train, y_train)
         print(KNN.score(X_test, y_test))
```

```
0.8478260869565217
```

Post Pipelining the results of the KNN score are slightly higher for the given seed I chose (148) and the test train split ratio. The results went from about .782 to .847 . I believe that scaling the numerical features gave more weight to their meaning instead of possibly missing or undermining their value in the resulkts. Further, the categorical features were given hot encodings to also encapsulate the value that they should represent in the data. By adjusting for both of these types of features we are able to yield a higher score and likely beter capture meaningful results.

Question 2.6 Hyperparameter Optimization

As we saw in lecture, 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 letting `n` take on the values 1, 2, 3, 5, 7, 9, 10, 20, 50, 75, and 100. Run your model for each value and report the accuracy for each. Then, create a plot of accuracy versus `n_neighbors` and discuss how and why the accuracy changes as `n_neighbors` changes.

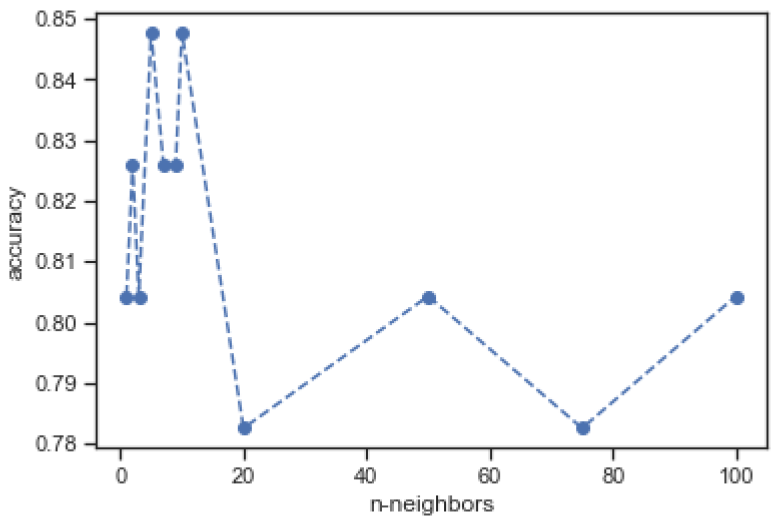
HINT: leverage Python's ability to loop to run through the array and generate results so that you don't need to manually code each iteration.

In [50]:
```
num_neighbors = [1, 2, 3, 5, 7, 9, 10, 20, 50, 75, 100]
knnscore = []
for i in range(len(num_neighbors)):
    KNN = KNeighborsClassifier(n_neighbors = num_neighbors[i])
    KNN.fit(X_train, y_train)
    knnscore.append(KNN.score(X_test, y_test))
    print(num_neighbors[i], ":", KNN.score(X_test, y_test))

plt.xlabel('n-neighbors'); plt.ylabel('accuracy')
plt.plot(num_neighbors, knnscore, marker ='o', linestyle='dashed')
```

```
1 : 0.8043478260869565
2 : 0.8260869565217391
3 : 0.8043478260869565
5 : 0.8478260869565217
7 : 0.8260869565217391
9 : 0.8260869565217391
10 : 0.8478260869565217
20 : 0.782608695652174
50 : 0.8043478260869565
75 : 0.782608695652174
100 : 0.8043478260869565
```

Out[50]:
```
[<matplotlib.lines.Line2D at 0x21689292160>]
```



The accuracy of the number of neighbors is given above for each score. Further we see that the best result from this set of numbers is at n=5 and n=10. What we notice in the accurracy is that the first couple numbers are not necessarily the highest, especially the first one. We also notice that generally as the number of neighbors becomes really large, then the accuracy goes down. This makes sense since using only one neighbor to group or place values for a data point would maybe not achieve great accuracy if that closest neighbor was not a good data point. Also that if we use 50 or 100 neighbors to categorize a data point then we might end up with clustered data such that a csv with only 1000 rows would really be chunked down by such a large neighbor comparison and may lose meaningful diversity in the results. In order to both make sure that data is represented well, accurate, and able to still be unique or diverse enough from all the data, then it is best to use enough neighbors that are not large in value but likely more then a couple.

Part 3. Additional Learning Methods

So we have a model that seems to work well. But let's see if we can do better! To do so we'll employ multiple learning methods and compare results. Throughout this part, use the data that was produced by your pipeline earlier.

Linear Decision Boundary Methods

We'll spend some time exploring logistic regression and SVM methods.

Question 3.1 Logistic Regression

Let's now try a classifier, we introduced in lecture, one 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. Implement a logistic regression classifier on your data with the default settings. Report accuracy, precision, recall, and F1 score and plot a confusion matrix.

In [56]:
```
log_r = LogisticRegression(random_state=mst)
log_r.fit(X_train, y_train)
y_pred = log_r.predict(X_test)
print("accuracy:", metrics.accuracy_score(y_test, y_pred))
print("precision:", metrics.precision_score(y_test, y_pred))
print("recall:", metrics.recall_score(y_test,y_pred))
print("f1:", metrics.f1_score(y_test, y_pred))
```

```
accuracy: 0.8260869565217391
precision: 0.8421052631578947
recall: 0.7619047619047619
f1: 0.8
```

Question 3.2

Discuss what each measure (accuracy, precision, recall, and F1 score) 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.

Accurracy identifies how much of the data is correctly classified. Generally the number of correct predicitions over data set. Precision measures how many of the predictions made were true positives. Generally the number of correct predictions over the number of predictions we made. Recall is the measure of positive cases that the classifer was able to correctly predict. Generally number of correct predictions over the number things with the predictions (i.e. predicting sick/everyone who is sick) Score measures precision and recall. It combines the two and gives a scaled value that is more then just arithmetic. It also can work for imbalanced data sets.

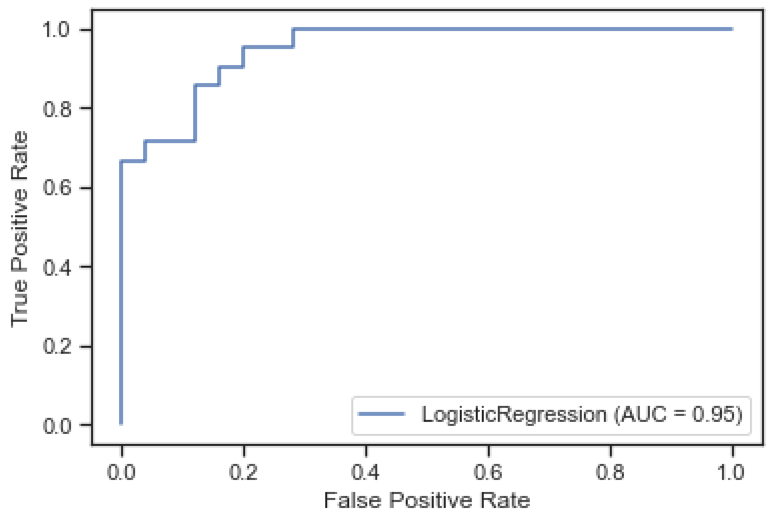
All of the values are above 75 which seems to be good. the f1 score uses both precision and recall and performs a weighted ratio adjustment and a sum as well as scaling by a factor of 2 to return the f1 score value. Because of this we see a result that is in between the values of precision and recall. Accuracy being about 82% implies that we were able to correctly classify a substantial amount of people that were sick. If being correct with our model is our goal then accuraccy is extremely important. However, one can be accurate, but also imprecise missing a target. Precision looks for the actual value of making correct predictions over all predicitions, such as how many people did we predict to be sick over all. Precision can reduce false positives in diagnosing patients. The recall is the lowest value we see above about .76, but is very powerful for minimizing missing positive cases such as predicting false negative. The issues of this are typically when there is a big cost wrongly classifying something as postive. So depending on our defintiion of "sick" this could be similar to the level of a cold or maybe it could be something that is terminal or life changing. This could be a very important metric for things like cancer or HIV. The F1 score combines recall and precision and can work for imbalanced sets. Because of its combination of earlier metrics it gives a useful overview of measuring for false rates and provides a well encapsulating score to analyze results of a study.

Question 3.3

Graph the ROC curve of the logistic regression model.

```
In [57]: metrics.plot_roc_curve(log_r, X_test, y_test)
```

```
Out[57]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x2168a9b3a90>
```



Question 3.4

Describe what an ROC curve is and what your ROC graph output is showing.

An Receiver Operating Characterstic curve is a graphical plot which illustrates the performance of a classifiaction model at all thresholds and particularly focuses on plotting the true positive rate against the false positive rate. Typically, if such a curve trends to the top-left of the plot, then it would be considered to have a great performance demonstrating high true positive rates and low false positive rates.

Our graph seems to show that the logistic regression model we used did a good ob at classifying this data. The AUC is .95 which s very big which implies that predictions and their values are well ranked and that the models predictions have a good quality in this (and possibly other) classifications.

Question 3.5

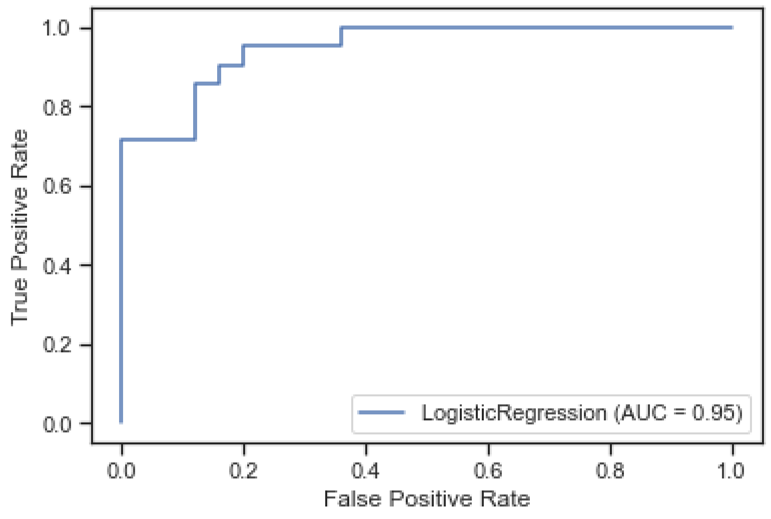
Let's tweak a few settings. Set your solver to `newton-cg` , your `max_iter=4` , and `penalty='none'` , and rerun your model. Report out the same metrics (the 4 + confusion matrix) as before. How do your results change?

In [59]:
```
iters = 4
log_r_2 = LogisticRegression(solver='newton-cg', max_iter=iters, penalty='none',
log_r_2.fit(X_train, y_train)
y_pred = log_r_2.predict(X_test)
print("accuracy:", metrics.accuracy_score(y_test, y_pred))
print("precision:", metrics.precision_score(y_test, y_pred))
print("recall:", metrics.recall_score(y_test,y_pred))
print("f1:", metrics.f1_score(y_test, y_pred))
metrics.plot_roc_curve(log_r_2, X_test, y_test)
```

```
accuracy: 0.8260869565217391
precision: 0.8421052631578947
recall: 0.7619047619047619
f1: 0.8
```

```
D:\Anaconda3\lib\site-packages\sklearn\utils\optimize.py:211: ConvergenceWarnin
g: newton-cg failed to converge. Increase the number of iterations.
  warnings.warn("newton-cg failed to converge. Increase the "
```

Out[59]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x2168aa6b190>
```



```
I do not notice any change from the current results to before. The values for
ech of the metrics seem consistent , the plots look identical, and the AUC is
still at .95
```

Question 3.6

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.

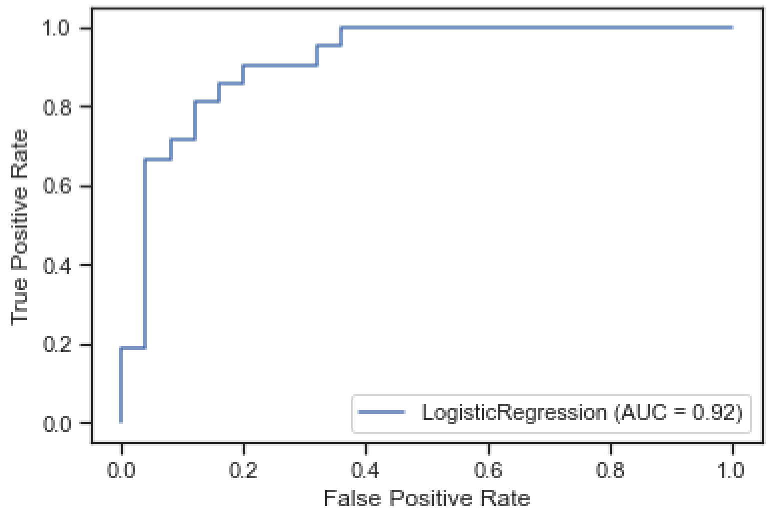
Note: if you did not get a warning, which might happen to those running this notebook in VSCode, please try running the following code, as described [here (https://github.com/microsoft/vscode-jupyter/issues/1312)](https://github.com/microsoft/vscode-jupyter/issues/1312):

```
import warnings
warnings.simplefilter(action="default")
```

In [61]:
```
# I am pretty sure there was no converging because of the error part asking us to
iters = 100
# RUnning 100 iterations
log_r_2 = LogisticRegression(solver='newton-cg', max_iter=iters, penalty='none',
log_r_2.fit(X_train, y_train)
y_pred = log_r_2.predict(X_test)
print("accuracy:", metrics.accuracy_score(y_test, y_pred))
print("precision:", metrics.precision_score(y_test, y_pred))
print("recall:", metrics.recall_score(y_test,y_pred))
print("f1:", metrics.f1_score(y_test, y_pred))
metrics.plot_roc_curve(log_r_2, X_test, y_test)
```

```
accuracy: 0.8260869565217391
precision: 0.8095238095238095
recall: 0.8095238095238095
f1: 0.8095238095238095
```

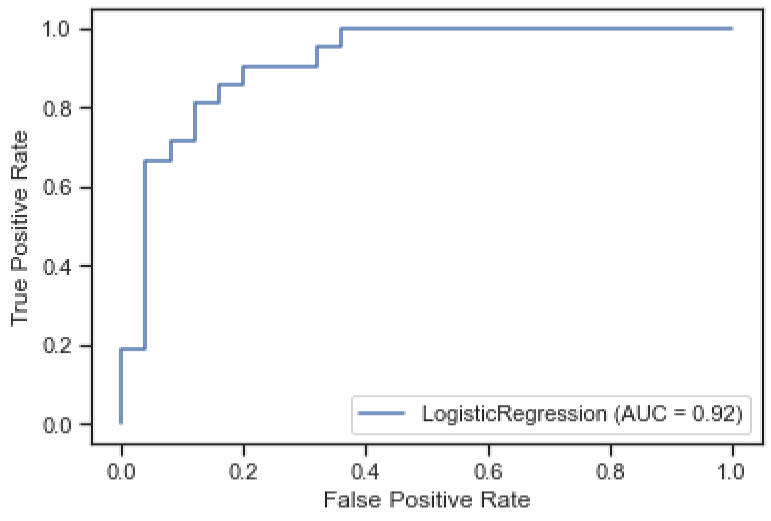
Out[61]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x2168ac68eb0>
```



In [63]:
```
iters = 10000
log_r_2 = LogisticRegression(solver='newton-cg', max_iter=iters, penalty='none',
log_r_2.fit(X_train, y_train)
y_pred = log_r_2.predict(X_test)
print("accuracy:", metrics.accuracy_score(y_test, y_pred))
print("precision:", metrics.precision_score(y_test, y_pred))
print("recall:", metrics.recall_score(y_test,y_pred))
print("f1:", metrics.f1_score(y_test, y_pred))
metrics.plot_roc_curve(log_r_2, X_test, y_test)
```

```
accuracy: 0.8260869565217391
precision: 0.8095238095238095
recall: 0.8095238095238095
f1: 0.8095238095238095
```

Out[63]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x2168ab27dc0>
```



Question 3.7

Explain what you changed and why this fixed the `ConvergenceWarning` problem. Are there any downsides of your fix? How might you have harmed the outcome instead? What other parameters you set may be playing a factor in affecting the results?

After inspecting the error code and reading the documentation I knew that the number of iterations was too low to perform the desired convergence. The accuraccy is near the same value, but precision decreased while recall increased. The result of these two movements also affected the f1 score which increased by almost 1 percentage point.

Because of the decrease in precision it is possible that our predictor will miss some false positives in diagnosing patients which can be a serious issue. The recall going up is good, but it seems like a metric that is constrained to seeing how well it predicted results based on the sample its comparing to. Maybe it would not be able to correctly predict some positive cases from a larger sample or other data.

One issue with increasing iterations is that we could be overfitting data. There is a hill, plateau, valley kind of min and max algorithm that is used in deep learning that looks for convergence as well (in another UCLA class). I believe the increase in iterations could make the data look for the next local minima/maxima, but may not find the global max/min and could be stuck looking before converging. Another idea is that the scalability would become difficult or even larger data sets if we are repeating processing or iterations over many gigabytes or terabytes of data.

We set penalty to zero also which giving penalty a value could help with shrinking coefficients of less contributing variables to 0 and leads to regularization. We also used newton-cg to solve this set, but there are other sets available as well as other random states to test the set in then the ones i used. There should be complexity, algorithmic, speed, and other possible differences associated with the other solvers and generally differ results.

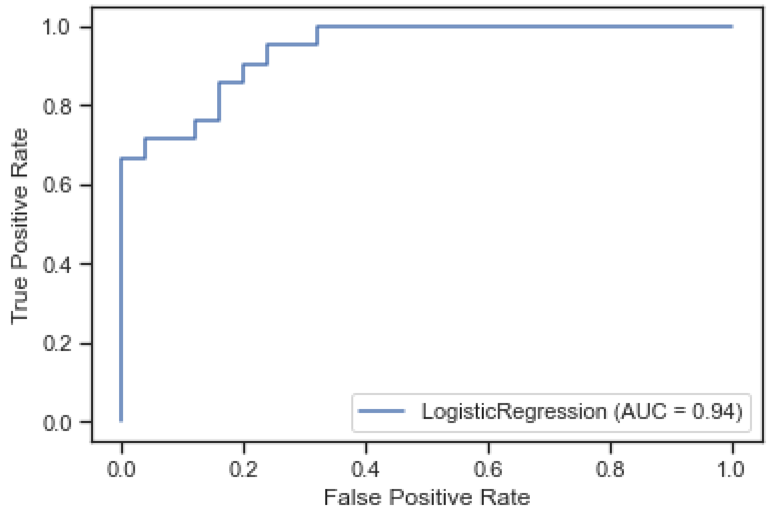
Question 3.8

Rerun your logistic classifier, but modify the `penalty='l1'` , `solver='liblinear'` and again report the results.

In [66]:
```python
log_r_3 = LogisticRegression(solver='liblinear', penalty='l1', random_state=mst)
log_r_3.fit(X_train, y_train)
y_pred = log_r_3.predict(X_test)
print("accuracy:", metrics.accuracy_score(y_test, y_pred))
print("precision:", metrics.precision_score(y_test, y_pred))
print("recall:", metrics.recall_score(y_test,y_pred))
print("f1:", metrics.f1_score(y_test, y_pred))
metrics.plot_roc_curve(log_r_3, X_test, y_test)
```

```
accuracy: 0.8260869565217391
precision: 0.8421052631578947
recall: 0.7619047619047619
f1: 0.8
```

Out[66]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x2168ac3dfa0>
```



Question 3.9

Explain what what the two solver approaches are, and why the model with `liblinear` and `l1` penalty likely produced the optimal outcome.

The former applied newton cg which uses a hessian matrix which implies that second derivatives are performed and could be very slow for large datasets. Also this is for multiclass sets. The liblinear is for large linear classification and uses coordinate descent. Liblinear is likely better because as the data increases it is probably quicker to compute and ma find the result in less iterations. Liblinear also does well in general with large data sets such as ours according to data science websites. "It performs pretty well with high dimensionality. It does have a number of drawbacks. It can get stuck, is unable to run in parallel, and can only solve multi-class logistic regression with one-vs.-rest."

Question 3.10

We also played around with different penalty terms (none, L1 etc.) Describe what the purpose of a penalty term is and the difference between L1 and L2 penalties.

The penalty term is meant to minimize the coefficients magnitude which can generally approach large values and cause bad modeling for a data set when performing a solvers such as newton-cg and liblinear. The loss functions add penalties which we generally have different options to choose from depending on the solver. L1's aproach is to force many values toward zero. It also tries to lower features by minimizing those useful and necessary to makin good predictions for the model.

"A regression model that uses L1 regularization technique is called Lasso Regression and model which uses L2 is called Ridge Regression. The key difference between these two is the penalty term. Ridge regression adds “ squared magnitude ” of coefficient as penalty term to the loss function."

Question 3.11 Support Vector Machine (SVM)

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 2-D space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

Implement an SVM classifier on your pipelined data (recommend using `scikit-learn`) For this implementation you can simply use the default settings, but set `probability=True` .

```
In [67]: SVM = SVC(probability=True, random_state=mst)
         SVM.fit(X_train, y_train)
```

```
Out[67]: SVC(probability=True, random_state=148)
```

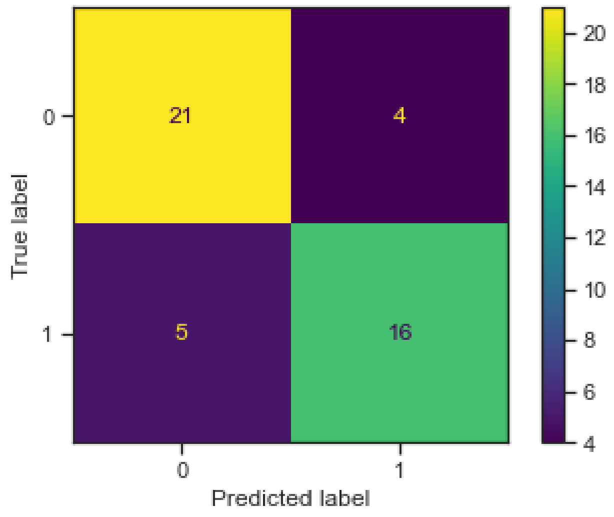
Question 3.12

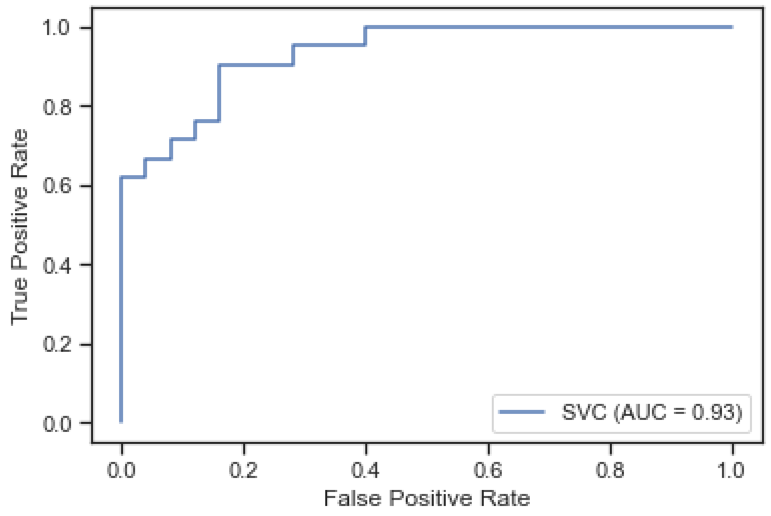
Report the accuracy, precision, recall, F1 Score, and confusion matrix of the resulting model.

```
In [68]: y_pred = SVM.predict(X_test)
         print("acc:", metrics.accuracy_score(y_test, y_pred))
         print("pre:", metrics.precision_score(y_test, y_pred))
         print("rec:", metrics.recall_score(y_test,y_pred))
         print("f1:", metrics.f1_score(y_test, y_pred))
         metrics.plot_confusion_matrix(SVM, X_test, y_test)
         metrics.plot_roc_curve(SVM, X_test, y_test)
```

```
acc: 0.8043478260869565
pre: 0.8
rec: 0.7619047619047619
f1: 0.7804878048780488
```

```
Out[68]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x21688f3ee20>
```





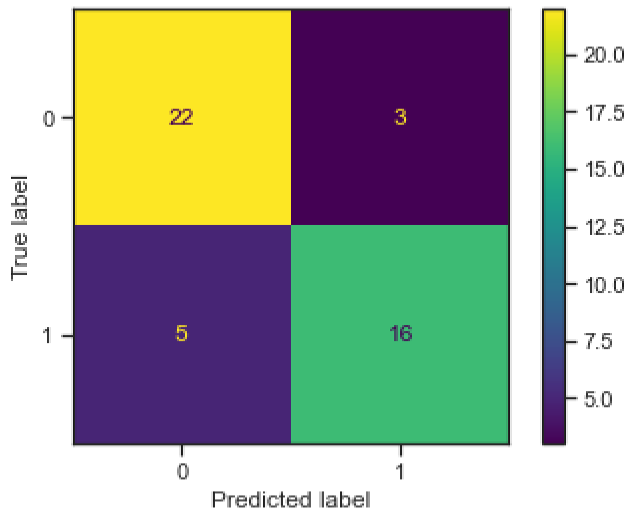
Question 3.13

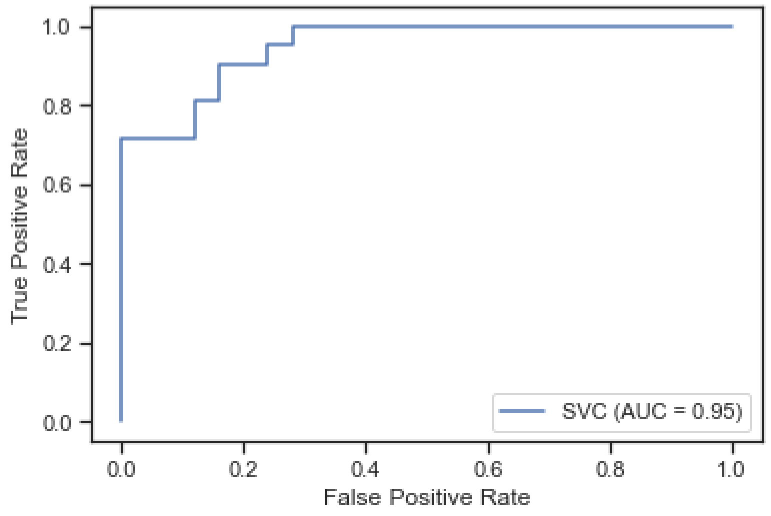
Rerun your SVM, but now modify your model parameter kernel to be `linear`. Again report your accuracy, precision, recall, F1 scores, and confusion matrix and plot the new ROC curve.

In [69]:
```
SVM2 = SVC(kernel='linear', probability=True, random_state=mst)
SVM2.fit(X_train, y_train)
y_pred = SVM2.predict(X_test)
print("acc:", metrics.accuracy_score(y_test, y_pred))
print("pre:", metrics.precision_score(y_test, y_pred))
print("rec:", metrics.recall_score(y_test,y_pred))
print("f1:", metrics.f1_score(y_test, y_pred))
metrics.plot_confusion_matrix(SVM2, X_test, y_test)
metrics.plot_roc_curve(SVM2, X_test, y_test)
```

```
acc: 0.8260869565217391
pre: 0.8421052631578947
rec: 0.7619047619047619
f1: 0.8
```

Out[69]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x2168aa7e6d0>
```





Question 3.14

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.

```
The kernel largely can determine where cutoff boundaries are vor seperating data
in such a classifier. By making it linear we are looking at data points that can
be seperated onto either side. THis is opposed to higher dimensionality
nonlinear kernels. A linear kernal can result in better scores and did so for
all values except our recall. This means that our data was better seperated into
the two buckets across the linear seperator rather then the multi dimensional
kernel space which likely creaed more buckets and did not achieve as high of
results.
```

Question 3.15

Both logistic regression and linear SVM are trying to classify data points using a linear decision boundary. How do they differ in how they try to find this boundary?

SVM is not as prone to outliers as it only cares about the points closest to the decision boundary. It changes its decision boundary depending on the placement of the new positive or negative events. Logistic Regression tries to maximize the conditional likelihood of the training data, it is highly prone to outliers. Standardization (as co-linearity checks) is also fundamental to make sure a features' weights do not dominate over the others. In essence LR uses a sigmoid function which will estimate and then categorize a data value to either a binary 0 or 1. Whereas SVM has a linear seperator which seeks to place itself such that is has a minimum distance between specific points and then categorizes based on this linear seperator on that dimensional plane.

Question 3.16

We also learned about linear regression in class. Why is linear regression not a suitable model for this classification task?

```
Because most of the data we have seen has had curves to it. This implies
logisitic is better fit. Further, linear regression require continious lines and
can have issue with data imbalance. Whereas logistic regression is for
classification problems, which predicts a probability range between 0 to 1. For
example, predict whether a customer will make a purchase or not. The regression
line is a sigmoid curve. Logistic seems better fit based on our graphs and these
reasons.
```

Statistical Classification Methods

Now we'll explore a statistical classification method, the naive Bayes classifier.

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. Bayes' theorem states the following relationship, given class variable C_k and dependent feature vector \bold$x = [x_1, x_2, \ldots, x_n]^T$,

$$P(C_k|\boldx) = \frac{P(C_k)P(\boldx|C_k)}{P(\boldx)}$$

Note for our purposes, there are 2 possible classes (sick or not sick), so k ranges from 1 to 2.

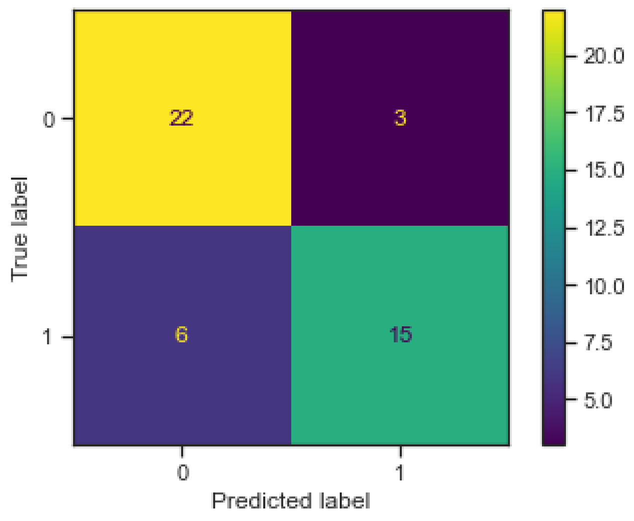
Question 3.17

Implement a naive Bayes Classifier on the pipelined data. Use the `GaussianNB` model. For this model, simply use the default parameters. Report out the number of mislabeled points that result (i.e., both the false positives and false negatives), along with the accuracy, precision, recall, F1 Score and confusion matrix. Also, plot an ROC curve.

In [71]:
```
from sklearn.naive_bayes import GaussianNB
NB = GaussianNB()
NB.fit(X_train, y_train)
y_pred = NB.predict(X_test)
print("acc:", metrics.accuracy_score(y_test, y_pred))
print("pre:", metrics.precision_score(y_test, y_pred))
print("rec:", metrics.recall_score(y_test,y_pred))
print("f1:", metrics.f1_score(y_test, y_pred))
metrics.plot_confusion_matrix(NB, X_test, y_test)
```

```
acc: 0.8043478260869565
pre: 0.8333333333333334
rec: 0.7142857142857143
f1: 0.7692307692307692
```

Out[71]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2168b0128b0
>
```



Question 3.18

Discuss the observed results. What assumptions about our data are we making here and why might those be inaccurate?

```
We are assuming that some features are completely isolated or independant of
each other. However, this is hardly ever true and there are usually some form of
confounding variables or features that slightly or majorly impact on another. A
bayesian network show how some nodes can branch off to another node and each is
done with some set of probabilty because of the dependencies children have on
parents in the network. The same ideas can apply here and may be connected.
```

Part 4: Cross Validation and Model Selection

You've sampled a number of different classification techniques, leveraging nearest neighbors, linear classifiers, and statistical classifiers. You've also tweaked with a few parameters for those models to optimize performance. Based on these experiments you should have settled on a particular model that performs most optimally on this dataset. Before our work is done though, we want to ensure that our results are not the result of the random sampling of our data we did with the train-test split. To check this, we will conduct a K-fold cross validation of our top 2 performing models, assess their cumulative performance across folds (report accuracy, precision, recall, and F1 score), and determine the best model for our particular data.

Question 4.1

Select your top 2 performing models and run a 10-Fold cross validation on both. Report your best performing model.

In [78]:
```python
from sklearn.model_selection import KFold
KF = KFold(n_splits=10, random_state=mst, shuffle=True)
log_r_3 = LogisticRegression(solver='liblinear', penalty='l1', random_state=24)
log_r_4 = LogisticRegression(solver='newton-cg', max_iter=100, penalty='l2', ranc
NB = GaussianNB()
log_r_3_score = cross_val_score(log_r_3, prepared_data, labels, cv=KF)
log_r_4_score = cross_val_score(log_r_4, prepared_data, labels, cv=KF)
NB_score = cross_val_score(NB, prepared_data, labels, cv=KF)
print("Logistic Regression liblinear: ", log_r_3_score.mean())
print("Logistic Regression newton-cg: ", log_r_4_score.mean())
print("Gaussian Bayes: ", NB_score.mean())
```

```
Logistic Regression liblinear:  0.8616129032258065
Logistic Regression newton-cg:  0.8682795698924732
Gaussian Bayes:  0.7519354838709678
```

Question 4.2

Discuss your results and why they differ slightly from what you got for the 2 models above.

```
This occurred because of K fold and cross validation. We were able to test
various folds of data and train the best model for the learning algorithm from
more samples. This will typically produce a more robust model which is proven
with the higher scores for our newton cg and liblinear models. The whole dataset
is randomly split into independent k-folds without replacement.
Steps from a site:
k-1 folds are used for the model training and one fold is used for performance
evaluation.
```

```
This procedure is repeated k times (iterations) so that we obtain k number of
performance estimates (e.g. MSE) for each iteration. Then we get the mean of k
number of performance estimates (e.g. MSE).
```

Question 4.3

Out of these 2 models, based on their scores for the 4 metrics, which one would you pick for this specific case of predicting if someone has heart disease or not?

```
Based on this predicament I would solely choose the model that had a good overal
record for each metric, but specifically in the recall metric. Among my models,
the victor appears to be newto cg which had the greated logistal regression mean
score of .86, but also had a recall score of about .81 which was nearly 2
percentage points higher then liblinear and even more higher then guassian.
Recall looks at actual positives which includes false negatives and true
positives. Also recall becomes extremely important when the cost/risk associated
with a false negative is high. Typically this is seen in finances or the health
industry. Heart Disease clearly meets this criteria and to be diagnosed negative
and walk away home happy without fixing a patient with heart disease asap is an
incredible mistake and issue. Thus recall is important here and newton cg
performed the best for my models and is my choice.
```