Project 2

Graded

Student

ARNAV KRISHNAKUMAR MARDA

Total Points

98 / 100 pts

Question 1

Load the Data and Analyze **25** / 25 pts

1.1 **Load data** **1** / 1 pt

- ✔ **– 0 pts** Correct
- **– 1 pt** not readable

1.2 **Descriptive statistics** **3** / 3 pts

- ✔ **– 0 pts** Correct
- **– 1 pt** missing .head() call
- **– 1 pt** missing .describe() call
- **– 0.5 pts** missing result from .head()
- **– 0.5 pts** missing result from .describe()
- **– 3 pts** not readable

1.3 **Info analysis** **2** / 2 pts

- ✔ **– 0 pts** Correct
- **– 1.5 pts** missing .info() call
- **– 0.5 pts** missing output
- **– 2 pts** not readable

1.4 **Label numeric conversion** **3** / 3 pts

- ✔ **– 0 pts** Correct
- **– 2 pts** An attempt was made, but incorrect
- **– 3 pts** not readable

1.5 **Histograms** **3** / 3 pts

- **– 0 pts** Correct
- **– 0 pts** partial missing graph, make sure all plots in pdf are readable next time
- ✔ **– 0 pts** not mentioning how each variable performs
- **– 0 pts** not mentioning how each variable performs for some variables
- **– 0.5 pts** missing plot
- **– 3 pts** not readable

1.6 **Label balance analysis** **5** / 5 pts

- ✔ **– 0 pts** Correct
- **– 2.5 pts** Partially correct: no histogram
- **– 2.5 pts** Partially correct: no count
- **– 5 pts** not readable

1.7 **Correlation analysis** **8** / 8 pts

- ✔ **– 0 pts** Correct
- **– 0 pts** Correlation plot with `sick` only
- **– 5 pts** Incorrect plot
- **– 2 pts** Insufficient analysis
- **– 3 pts** No/wrong analysis
- **– 8 pts** not readable

Question 2

Raw and Pipelined KNN Model **28** / 30 pts

2.1 **Split data** **5** / 5 pts

✔ **– 0 pts** Correct

– 1 pt Printing of shapes was not done.

– 1.5 pts 70% of the data was not split into the train data.

– 2.5 pts The sick axis was removed and in place was not used and neither it was stored in a new variable.

2.2 **Run raw KNN model** **5** / 5 pts

✔ **– 0 pts** Correct

– 5 pts Not attempted

2.3 **Implement pipeline** **13** / 15 pts

– 0 pts Correct

– 1 pt Fit_tranform was used on test data instead of transform

✔ **– 2 pts** Fit transform should only be used on train and transform should be used on test

– 15 pts Not attempted

– 1.5 pts Splitting had to be 0.2 fraction as testing data.

– 3 pts Didn't transform test data

– 3 pts Didn't fit_transform train data

2.4 **Run pipelined KNN model** **5** / 5 pts

✔ **– 0 pts** Correct

– 2 pts Trained again on the same model that was used to train on non pipelined data

– 1.5 pts Model was not run and comparison was not made.

– 5 pts Not attempted

– 1 pt Incomplete comparison

Question 3

Additional Learning Methods **45** / 45 pts

3.1 **Run basic Logistic Regression** **4** / 4 pts

✔ **– 0 pts** Correct

– 1 pt Not supposed to train Logistic Regression on test dataset

– 1 pt No code shown

– 4 pts Answer not visible

– 0 pts Incorrect

3.2 **Metrics analysis** **5** / 5 pts

✔ **– 0 pts** Correct

– 2.5 pts Missing some metrics

– 5 pts No answer

– 5 pts Answer not visible

3.3 **SAG Logistic Regression** **4** / 4 pts

✔ **– 0 pts** Correct

– 4 pts Answer not visible

– 1 pt Supposed to keep max iteration - 10

3.4 **SAG model fix** **5** / 5 pts

✔ **– 0 pts** Correct. **Even if they fixed the warning in a different way.**

– 5 pts Answer not visible

– 2 pts No explanation

3.5 **Liblinear Logistic model and analysis** **5** / 5 pts

✔ **– 0 pts** Correct

– 5 pts Answer not visible

– 2 pts No explanation

3.6 **Implement basic SVM** **4** / 4 pts

✔ **– 0 pts** Correct

– 1 pt Probability is not set to True

– 4 pts Assignment not readable / Unattempted

3.7 **Plot Confusion Matrix** **4** / 4 pts

- ✔ **– 0 pts** Correct
- **– 1 pt** Incorrect metrics/Confusion matrix because label and predictions are interchanged
- **– 2 pts** No confusion matrix
- **– 4 pts** Unattempted / Assignment not readable

3.8 **Plot ROC Curve** **4** / 4 pts

- ✔ **– 0 pts** Correct. As long as they plot the correct metric and the explanation makes sense.
- **– 1 pt** Plot on prediction labels instead of prediction probabilities
- **– 4 pts** Unattempted/Unreadable
- **– 2 pts** Missing/Incorrect ROC plot

3.9 **Linear SVM and analysis** **5** / 5 pts

- ✔ **– 0 pts** Correct
- **– 1 pt** Rbf kernel not mentioned
- **– 5 pts** Unattempted/Assignment unreadable
- **– 2 pts** No explanation
- **– 1 pt** Missing ROC curve
- **– 1 pt** Missing confusion matrix

3.10 **Linear decision boundary analysis** **5** / 5 pts

- ✔ **– 0 pts** Correct
- **– 1 pt** Partially correct
- **– 3 pts** Incorrect explanation
- **– 5 pts** No attempt/Answer not readable

No questions assigned to the following page.

CS148_Project_2_Winter24_To_Do

February 23, 2024

0.1 24W-COM SCI-M148 Project 2 - Binary Classification Comparative Methods

Name: Arnav Marda

UID: 405772661

0.1.1 Submission Guidelines

1. Please fill in your name and UID above.
2. Please submit a **PDF printout** of your Jupyter Notebook to **Gradescope**. If you have any trouble accessing Gradescope, please let a TA know ASAP.
3. As the PDF can get long, please tag the respective sections to ensure the readers know where to look.

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.

0.2 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

No questions assigned to the following page.

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)

0.3 Loading Essentials and Helper Functions

[26]:
```
#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
```

Questions assigned to the following page: 1.1 and 1.2

```
%matplotlib inline

import random

random.seed(42)
```

0.4 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.)

[27]:
```
data = pd.read_csv('heartdisease.csv')
```

0.4.1 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.

[28]:
```
data.head()
```

[28]:

	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

[29]:
```
data.describe()
```

[29]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca
count	303.000000	303.000000	303.000000	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	1.039604	1.399340	0.729373
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000

Questions assigned to the following page: 1.2 and 1.3

```
          count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000
          mean     0.528053  149.646865    0.326733    1.039604    1.399340    0.729373
          std      0.525860   22.905161    0.469794    1.161075    0.616226    1.022606
          min      0.000000   71.000000    0.000000    0.000000    0.000000    0.000000
          25%      0.000000  133.500000    0.000000    0.000000    1.000000    0.000000
          50%      1.000000  153.000000    0.000000    0.800000    1.000000    0.000000
          75%      1.000000  166.000000    1.000000    1.600000    2.000000    1.000000
          max      2.000000  202.000000    1.000000    6.200000    2.000000    4.000000

                      thal
          count  303.000000
          mean     2.313531
          std      0.612277
          min      0.000000
          25%      2.000000
          50%      2.000000
          75%      3.000000
          max      3.000000
```

[30]:
```
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
```

Questions assigned to the following page: 1.4 and 1.5

0.4.2 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. (hint: try label encoder or .astype()

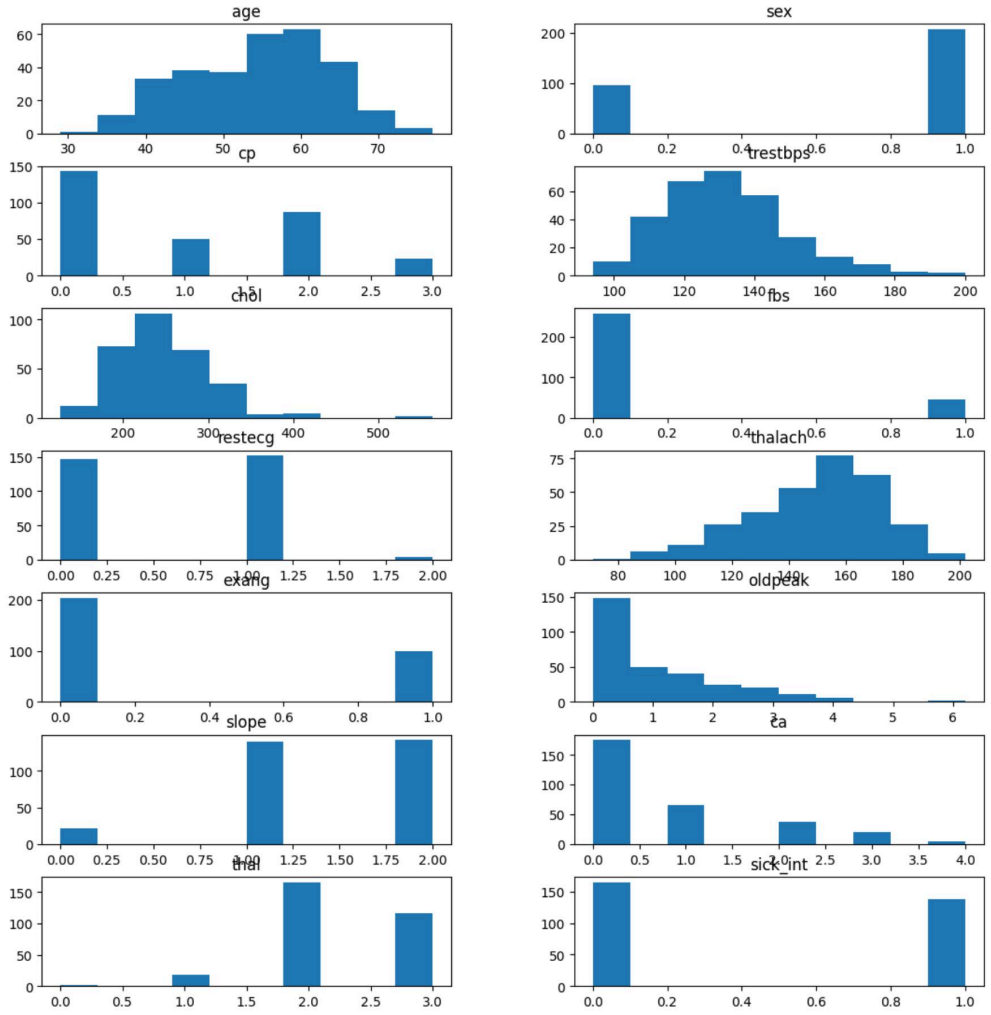
[31]:
```
data['sick_int'] = data["sick"].astype(int)
data.drop('sick', axis=1, inplace=True)
```

0.4.3 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?

[32]:
```
fig, axis = plt.subplots(7, 2, figsize=(12, 12))
fig.tight_layout(pad=5.0)
data.hist(ax=axis, grid=False)
```

[32]:
```
array([[<Axes: title={'center': 'age'}>, <Axes: title={'center': 'sex'}>],
       [<Axes: title={'center': 'cp'}>,
        <Axes: title={'center': 'trestbps'}>],
       [<Axes: title={'center': 'chol'}>,
        <Axes: title={'center': 'fbs'}>],
       [<Axes: title={'center': 'restecg'}>,
        <Axes: title={'center': 'thalach'}>],
       [<Axes: title={'center': 'exang'}>,
        <Axes: title={'center': 'oldpeak'}>],
       [<Axes: title={'center': 'slope'}>,
        <Axes: title={'center': 'ca'}>],
       [<Axes: title={'center': 'thal'}>,
        <Axes: title={'center': 'sick_int'}>]], dtype=object)
```

Questions assigned to the following page: 1.5 and 1.6

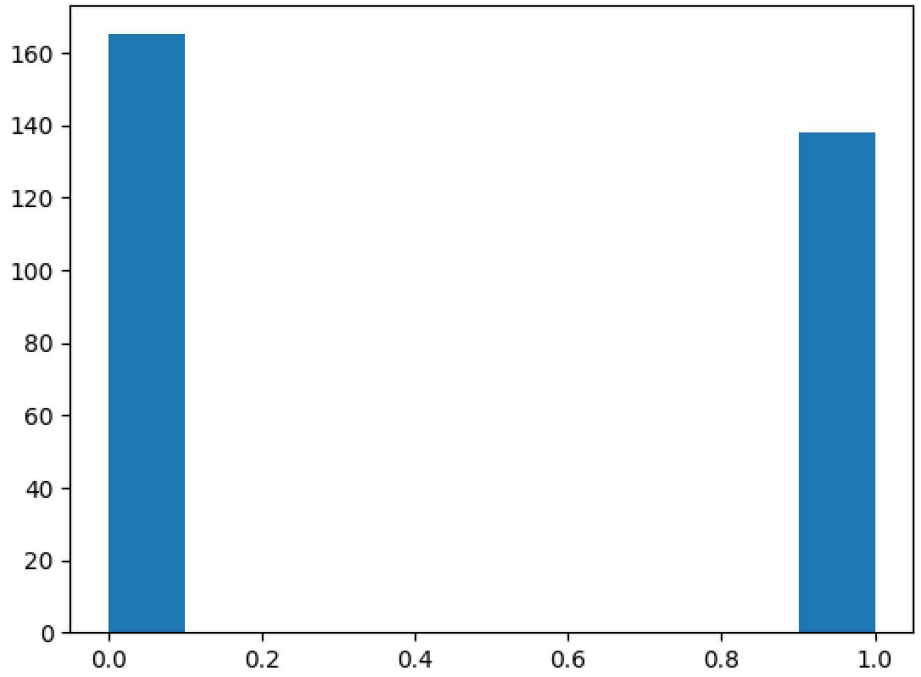


0.4.4 We also want to make sure we are dealing with a balanced dataset. In this case, we want to confirm whether or not we have an equitable number of sick and healthy individuals to ensure that our classifier will have a sufficiently balanced dataset to adequately classify the two. Plot a histogram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

Question assigned to the following page: 1.6

[33]:
```
data["sick_int"].hist(grid=False)
sick_count = data["sick_int"].value_counts()
print(sick_count)
print("There are 165 healthy patients and 138 sick patients in the dataset.␣
 ↪This provides a relatively balanced dataset.")
```

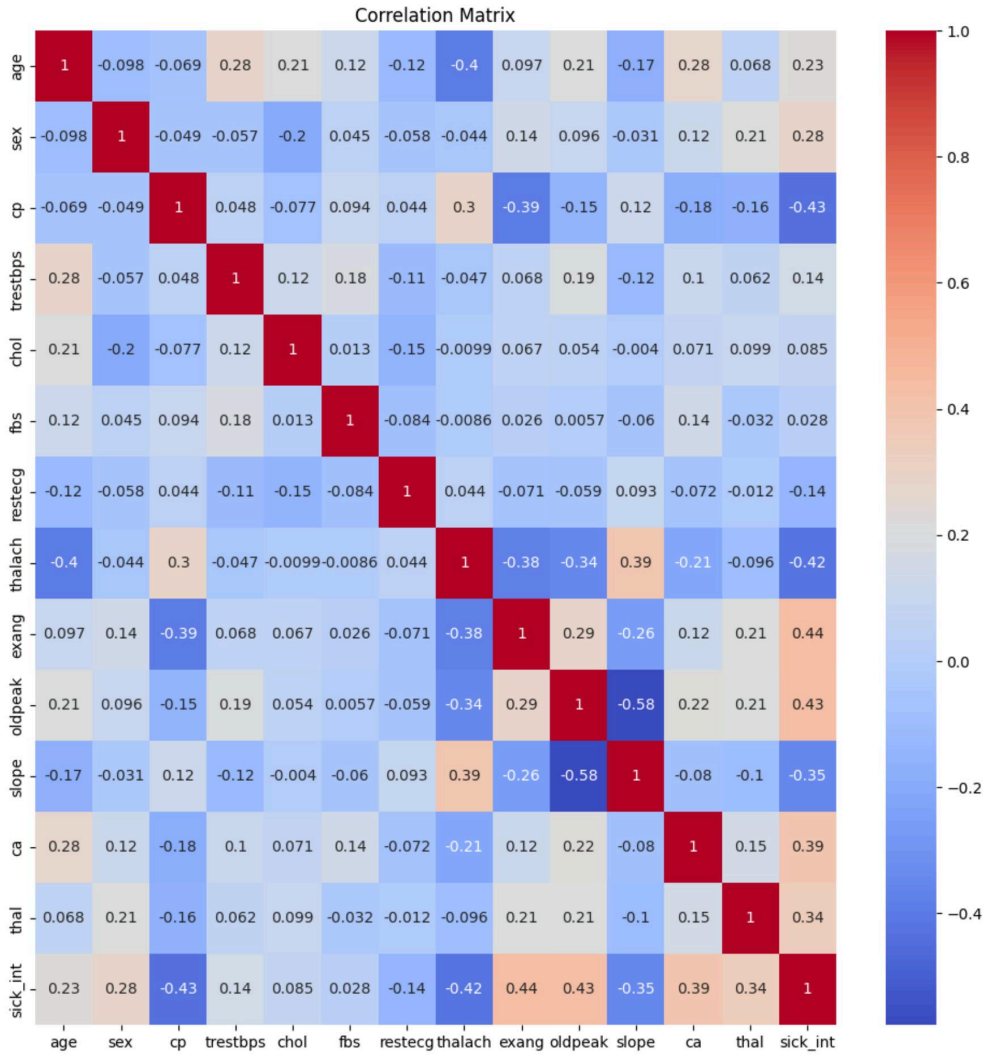
```
sick_int
0    165
1    138
Name: count, dtype: int64
There are 165 healthy patients and 138 sick patients in the dataset. This
provides a relatively balanced dataset.
```



Question assigned to the following page: 1.7

0.4.5 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)?

[34]:
```
correlation_matrix = data.corr()
plt.figure(figsize=(12, 12))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
plt.title("Correlation Matrix")
plt.show()
```



Questions assigned to the following page: 2.1 and 1.7

From the correlation matrix, it is clear that 3 variables have a moderate negative correlation with the target value. These variables are: cp, thalach, slope. The variable with the highest correlation is cp, which is the chest pain type. This makes sense as chest pain is a common symptom of heart disease. The other two variables, thalach and slope, are also related to heart rate and exercise, which are also related to heart disease.

Apart from this, there are a few variables that have a moderate positive correlation with the target value. These variables are: exang, oldpeak, ca, and thal. exang is exercise induced angina which is a symptom of coronary heart disease. It is the pain induced due to making your heart work harder (colloquially). This could lead to heart diseas and thus has a positive correlation. ca reports the number of major vessels colored by flourosopy. This is a measure of the severity of the heart disease and thus has a positive correlation. thal is a measure of the blood flow to the heart and thus has a positive correlation. oldpeak is the depression induced by exercise relative to rest. This is a measure of the heart's ability to handle stress and thus has a positive correlation.

Some variables tend to have higher correlations than others because: - Inherent relationships in the data: Variables that are directly related to the target variable may show higher correlations. - Common underlying factors: Variables that are influenced by common underlying factors may exhibit higher correlations. - Magnitude of effects: Variables that have a larger effect on the target variable may exhibit higher correlations.

0.5 Part 2. Prepare the 'Raw' Data and run a KNN Model

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!

0.5.1 Save the label column as a separate array and then drop it from the dataframe.

[35]:
```
sick_int_arr = data["sick_int"].values
data.drop('sick_int', axis=1, inplace=True)
```

0.5.2 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.

[36]:
```
x_train, x_test, y_train, y_test = train_test_split(data, sick_int_arr,␣
↪test_size=0.3)
```

Questions assigned to the following page: 2.1, 2.2, and 2.3

[37]:
```
print("Training set X shape: ", x_train.shape)
print("Training set Y shape: ", y_train.shape)
print("Testing set X shape: ", x_test.shape)
print("Testing set Y shape: ", y_test.shape)
```

```
Training set X shape:  (212, 13)
Training set Y shape:  (212,)
Testing set X shape:  (91, 13)
Testing set Y shape:  (91,)
```

0.5.3 We'll explore how not processing your data can impact model performance by using the K-Nearest Neighbor classifier. One thing to note was because KNN's rely on Euclidean distance, they are highly sensitive to the relative magnitude of different features. Let's see that in action! Implement a K-Nearest Neighbor algorithm on our raw data and report the results. For this initial implementation simply use the default settings. Refer to the KNN Documentation for details on implementation. Report on the accuracy of the resulting model.

[38]:
```
# k-Nearest Neighbors algorithm
knn = KNeighborsClassifier()
knn.fit(x_train, y_train)
```

[38]:
```
KNeighborsClassifier()
```

[39]:
```
knn.score(x_test, y_test)
```

[39]:
```
0.5934065934065934
```

0.5.4 Now implement a pipeline of your choice. You can opt to handle categoricals however you wish, however please scale your numeric features using standard scaler. Use the fit_transform() to fit this pipeline to your training data. and then transform() to apply that pipeline to your test data

Hint: 1. Create separate pipelines for numeric and categorical features with Pipeline() and then combining them with ColumnTransformer() 2. First, fit the full pipeline with the training data. Then, apply it to the test data as well.

0.5.5 Pipeline:

[40]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer, make_column_transformer
```

[41]:
```
numerical_features = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
categorical_features = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca',␣
↪'thal']

numeric_transformer = Pipeline(steps=[('scaler', StandardScaler())])
```

Questions assigned to the following page: 2.3 and 2.4

```
categorical_transformer = Pipeline(steps=[('onehot', OneHotEncoder())])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ]
)

preprocessor.fit(x_train)
x_train_transformed = preprocessor.transform(x_train)
```

[42]:
```
# Pipeline my test data
x_test_transformed = preprocessor.transform(x_test)
```

0.5.6 Now retrain your model and compare the accuracy metrics (Accuracy, Precision, Recall, F1 Score) with the raw and pipelined data.

[43]:
```
# k-Nearest Neighbors algorithm
knn_new = KNeighborsClassifier()
knn_new.fit(x_train_transformed, y_train)
```

[43]:
```
KNeighborsClassifier()
```

[44]:
```
test_pred = knn_new.predict(x_test_transformed)
print("Accuracy: ", metrics.accuracy_score(y_test, test_pred))
print("Precision: ", metrics.precision_score(y_test, test_pred))
print("Recall: ", metrics.recall_score(y_test, test_pred))
print("F1 Score: ", metrics.f1_score(y_test, test_pred))
```

```
Accuracy:  0.8461538461538461
Precision:  0.8444444444444444
Recall:  0.8444444444444444
F1 Score:  0.8444444444444444
```

Scaling the features provided a significant increase in the model's performance. The accuracy of the model increased from 0.5934 to 0.8462. Scaling the features allows the model to better understand the relationships between the features and the target variable. This is because scaling the features ensures that the features are on the same scale and so the euclidean distances are not skewed by the magnitude of the features.

Question assigned to the following page: 2.4

0.5.7 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, 7, 9, 10, 20, and 50. Run your model for each value and report the accuracy for each. (HINT leverage python's ability to loop to run through the array and generate results without needing to manually code each iteration).

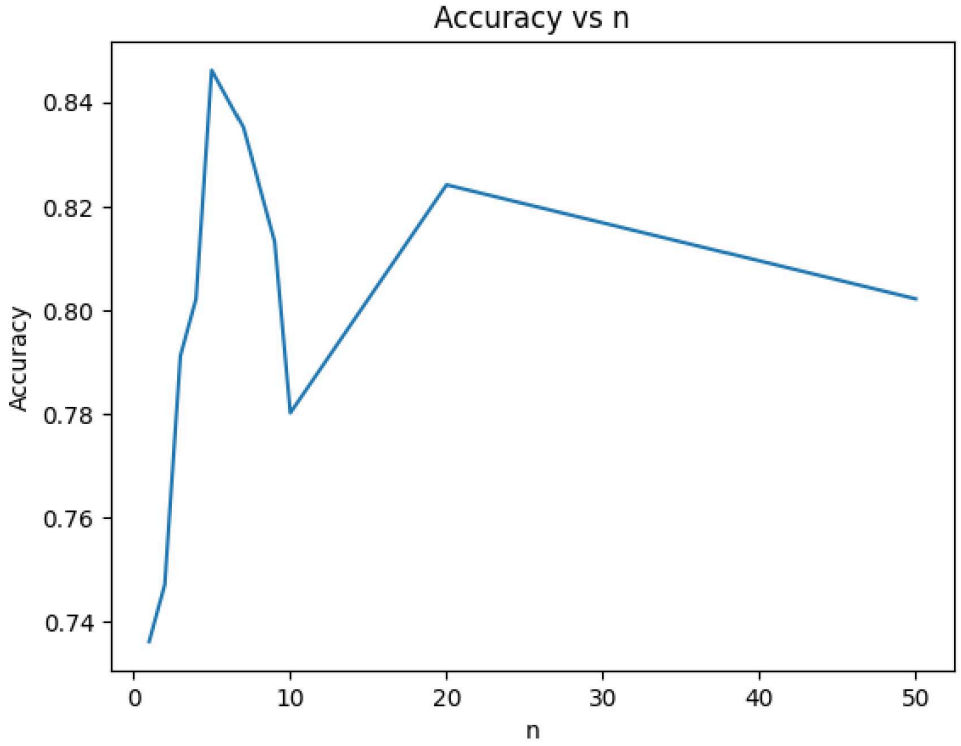
[46]:
```
n = [1, 2, 3, 4, 5, 7, 9, 10, 20, 50]
acc_n = []

for i in n:
    knn_i = KNeighborsClassifier(n_neighbors=i)
    knn_i.fit(x_train_transformed, y_train)
    test_pred_i = knn_i.predict(x_test_transformed)
    acc_n.append(metrics.accuracy_score(y_test, test_pred_i))
    print("Accuracy for k = ", i, " is ", acc_n[-1])

plt.plot(n, acc_n)
plt.xlabel('n')
plt.ylabel('Accuracy')
plt.title('Accuracy vs n')
plt.show()
```

```
Accuracy for k =  1  is  0.7362637362637363
Accuracy for k =  2  is  0.7472527472527473
Accuracy for k =  3  is  0.7912087912087912
Accuracy for k =  4  is  0.8021978021978022
Accuracy for k =  5  is  0.8461538461538461
Accuracy for k =  7  is  0.8351648351648352
Accuracy for k =  9  is  0.8131868131868132
Accuracy for k =  10  is  0.7802197802197802
Accuracy for k =  20  is  0.8241758241758241
Accuracy for k =  50  is  0.8021978021978022
```

Question assigned to the following page: 2.4



0.6 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 result.

0.6.1 Linear Decision Boundary Methods

0.6.2 Logistic Regression

Let's now try another classifier,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.

0.6.3 Implement a Logistical Regression Classifier. Review the Logistical Regression Documentation for how to implement the model.

0.6.4 Report metrics for:

1. Accuracy
2. Precision
3. Recall
4. F1 Score

Questions assigned to the following page: 3.1 and 3.2

[47]:
```
# Logistic Regression
log_reg = LogisticRegression()
log_reg.fit(x_train_transformed, y_train)
test_pred_log = log_reg.predict(x_test_transformed)
print("Accuracy: ", metrics.accuracy_score(y_test, test_pred_log))
print("Precision: ", metrics.precision_score(y_test, test_pred_log))
print("Recall: ", metrics.recall_score(y_test, test_pred_log))
print("F1 Score: ", metrics.f1_score(y_test, test_pred_log))
```

```
Accuracy:  0.8351648351648352
Precision:  0.875
Recall:  0.7777777777777778
F1 Score:  0.823529411764706
```

0.6.5 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 measures the ratio of correctly predicted instances to the total number of instances in the dataset. Accuracy is easy to understand and interpret, making it a common metric for evaluating classifier performance. However, accuracy may not be the best metric for imbalanced datasets where one class dominates the others. In such cases, high accuracy can be misleading if the model performs poorly on minority classes.

Precision measures the ratio of correctly predicted positive observations to the total predicted positive observations. It focuses on the relevance of positive predictions, indicating how many of the predicted positive cases are actually relevant. Precision is crucial in scenarios where false positives are costly or undesirable.

Recall measures the ratio of correctly predicted positive observations to the total actual positive observations. It focuses on capturing all positive instances, indicating how many of the actual positive cases are correctly identified. Recall is crucial in scenarios where false negatives are costly or undesirable.

F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics. F1-score is significant when there is an uneven class distribution or when false positives and false negatives have different costs.

In summary, each metric serves a specific purpose in evaluating classifier performance: - Accuracy provides an overall assessment but may not be suitable for imbalanced datasets. - Precision emphasizes the quality of positive predictions and is essential when false positives are costly. - Recall emphasizes the completeness of positive predictions and is essential when false negatives are costly. - F1-score balances precision and recall, making it suitable for imbalanced datasets or when false positives and false negatives have different impacts.

Examples: - In email spam detection, precision is crucial to avoid misclassifying legitimate emails as spam (false positives). A high precision ensures that important emails are not erroneously marked as spam. - In medical diagnosis, recall is crucial to ensure that all actual positive cases (e.g., patients with a disease) are correctly identified, minimizing the risk of missing a diagnosis. - In credit card

Questions assigned to the following page: 3.2, 3.3, and 3.4

fraud detection, a balance between precision and recall (as reflected in F1-score) is essential. We want to minimize both false positives (incorrectly flagging non-fraudulent transactions) and false negatives (missing fraudulent transactions).

0.6.6 Let's tweak a few settings. First let's set your solver to 'sag', your max_iter= 10, and set penalty = 'none' and rerun your model. Let's see how your results change!

[48]:
```
# Logistic Regression
log_reg_sag = LogisticRegression(solver='sag', max_iter=10, penalty='none')
log_reg_sag.fit(x_train_transformed, y_train)
test_pred_log_sag = log_reg_sag.predict(x_test_transformed)
print("Accuracy: ", metrics.accuracy_score(y_test, test_pred_log_sag))
print("Precision: ", metrics.precision_score(y_test, test_pred_log_sag))
print("Recall: ", metrics.recall_score(y_test, test_pred_log_sag))
print("F1 Score: ", metrics.f1_score(y_test, test_pred_log_sag))
```

```
Accuracy:  0.8131868131868132
Precision:  0.8333333333333334
Recall:  0.7777777777777778
F1 Score:  0.8045977011494253
```

```
/opt/homebrew/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:1182: FutureWarning:
`penalty='none'`has been deprecated in 1.2 and will be removed in 1.4. To keep
the past behaviour, set `penalty=None`.
  warnings.warn(
/opt/homebrew/lib/python3.11/site-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
```

0.6.7 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.

[51]:
```
# Logistic Regression
log_reg_sag = LogisticRegression(solver="sag", max_iter=10000, penalty="none")
log_reg_sag.fit(x_train_transformed, y_train)
test_pred_log_sag = log_reg_sag.predict(x_test_transformed)
print("Accuracy: ", metrics.accuracy_score(y_test, test_pred_log_sag))
print("Precision: ", metrics.precision_score(y_test, test_pred_log_sag))
print("Recall: ", metrics.recall_score(y_test, test_pred_log_sag))
print("F1 Score: ", metrics.f1_score(y_test, test_pred_log_sag))
```

```
Accuracy:  0.8021978021978022
Precision:  0.8
```

Questions assigned to the following page: 3.4 and 3.5

```
Recall:  0.8
F1 Score:  0.8000000000000002
```

```
/opt/homebrew/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:1182: FutureWarning:
`penalty='none'`has been deprecated in 1.2 and will be removed in 1.4. To keep
the past behaviour, set `penalty=None`.
  warnings.warn(
```

0.6.8 Explain what you changed, and why do you think that may have altered the outcome.

The warning "ConvergenceWarning: The max_iter was reached which means the coef_ did not converge" indicates that the logistic regression model did not converge within the maximum number of iterations specified. This can occur when the model is unable to find the optimal coefficients due to the complexity of the data or the chosen solver. To address this issue, we can increase the maximum number of iterations to allow the model to converge. Thus, we changed the max_iter parameter to 10000 to provide more iterations for the model to converge. This change allowed the model to find the optimal coefficients and converge successfully.

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

[52]:
```
log_reg_liblinear = LogisticRegression(solver="liblinear", penalty="l1")
log_reg_liblinear.fit(x_train_transformed, y_train)
test_pred_log_liblinear = log_reg_liblinear.predict(x_test_transformed)
print("Accuracy: ", metrics.accuracy_score(y_test, test_pred_log_liblinear))
print("Precision: ", metrics.precision_score(y_test, test_pred_log_liblinear))
print("Recall: ", metrics.recall_score(y_test, test_pred_log_liblinear))
print("F1 Score: ", metrics.f1_score(y_test, test_pred_log_liblinear))
```

```
Accuracy:  0.8351648351648352
Precision:  0.8571428571428571
Recall:  0.8
F1 Score:  0.8275862068965518
```

0.6.10 Explain what what the two solver approaches are, and why liblinear may have produced an improved outcome (but not always, and it's ok if your results show otherwise!).

'sag' stands for Stochastic Average Gradient, which is a variant of the stochastic gradient descent (SGD) optimization algorithm. It approximates the true gradient of the cost function using a random subset of samples (mini-batch) at each iteration, which makes it efficient for large datasets. 'sag' is particularly suitable when the dataset is large and the number of features is moderate.

'liblinear' is a method that uses coordinate descent as the optimization algorithm for Logistic Regression. Coordinate descent optimizes the cost function by iteratively updating each feature's coefficient while holding others fixed. It performs well when the dataset is small to medium-sized and the number of features is relatively high.

Questions assigned to the following page: 3.5, 3.6, and 3.7

'liblinear' may have produced an improved outcome in this case because: - 'liblinear' may produce improved outcomes when dealing with small to medium-sized datasets with a relatively high number of features. In such cases, coordinate descent, as used in 'liblinear', can converge faster and more effectively than stochastic optimization algorithms like 'sag'. The dataset we are using falls in the small to medium-sized category, and thus 'liblinear' may have produced an improved outcome. - 'liblinear' may also perform well when the dataset requires strong regularization. Coordinate descent tends to perform better when the regularization strength is high, as it can handle L1 (Lasso) and L2 (Ridge) regularization effectively. Thus, using the 'l1' penalty with 'liblinear' may have produced an improved outcome.

0.6.11 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.

0.6.12 Implement a Support Vector Machine classifier on your pipelined data. Review the SVM Documentation for how to implement a model. For this implementation you can simply use the default settings, but set probability = True.

[53]:
```
# SVM
svm = SVC(probability=True)
svm.fit(x_train_transformed, y_train)
test_pred_svm = svm.predict(x_test_transformed)
```

0.6.13 Report the accuracy, precision, recall, F1 Score, of your model, but in addition, plot a Confusion Matrix of your model's performance

recommend using `from sklearn.metrics import ConfusionMatrixDisplay` for this one!

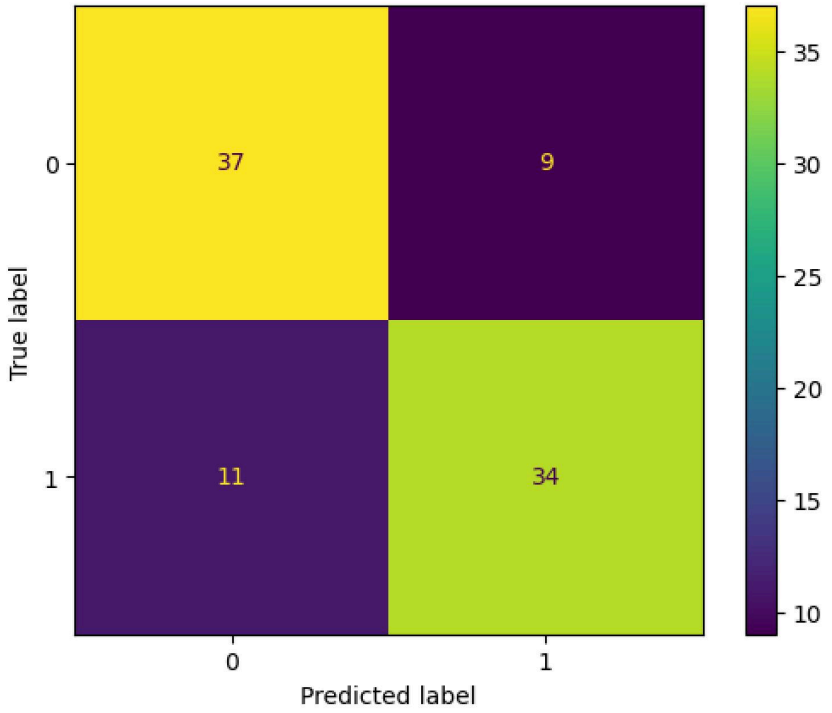
[54]:
```
print("Accuracy: ", metrics.accuracy_score(y_test, test_pred_svm))
print("Precision: ", metrics.precision_score(y_test, test_pred_svm))
print("Recall: ", metrics.recall_score(y_test, test_pred_svm))
print("F1 Score: ", metrics.f1_score(y_test, test_pred_svm))
```

```
Accuracy:  0.7802197802197802
Precision:  0.7906976744186046
Recall:  0.7555555555555555
F1 Score:  0.7727272727272727
```

[55]:
```
from sklearn.metrics import ConfusionMatrixDisplay
cm = confusion_matrix(y_test, test_pred_svm)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
```

[55]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x12a82cad0>
```

Questions assigned to the following page: 3.7 and 3.8



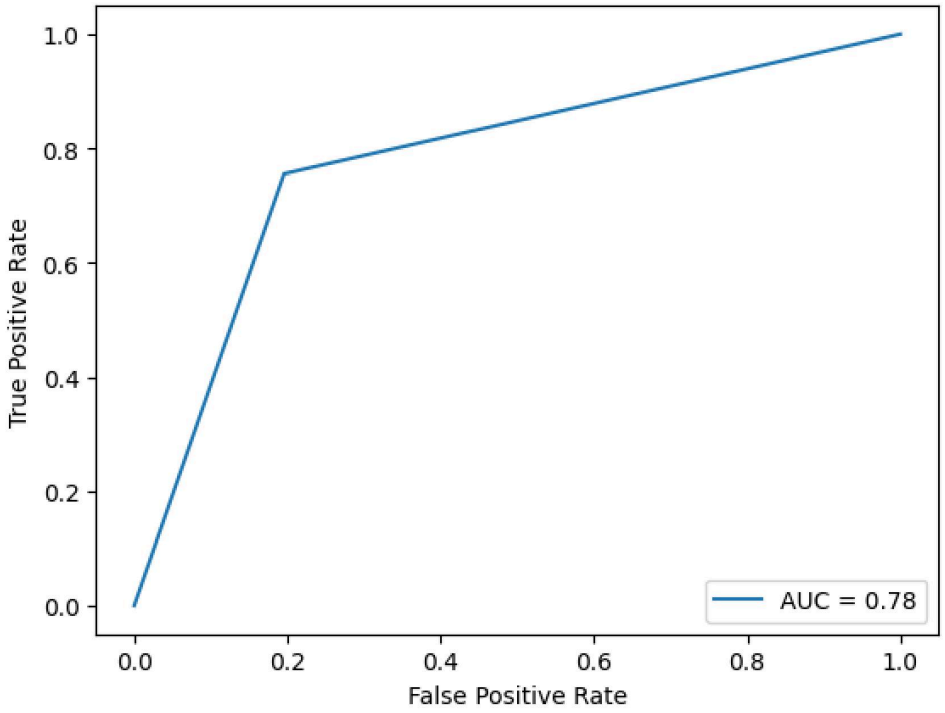
0.6.14 Plot a Receiver Operating Characteristic curve, or ROC curve, and describe what it is and what the results indicate

recommend using the `metrics.roc_curve` `metrics.auc` and `metrics.RocCurveDisplay` for this one!

[56]:
```
fpr, tpr, thresholds = metrics.roc_curve(y_test, test_pred_svm)
roc_auc = metrics.auc(fpr, tpr)
display = metrics.RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc)
display.plot()
```

[56]: `<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x12a8c2550>`

Questions assigned to the following page: 3.8 and 3.9



The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The area under an ROC curve is a measure of the usefulness of a test in general, where a greater area means a more useful test, so the areas under ROC curves are used to compare the usefulness of tests. Here we see a relatively low area under the curve indicating a poorly performing model.

0.6.15 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.

[57]:
```
# SVM
svm_lin = SVC(kernel='linear', probability=True)
svm_lin.fit(x_train_transformed, y_train)
test_pred_svm_lin = svm_lin.predict(x_test_transformed)
```

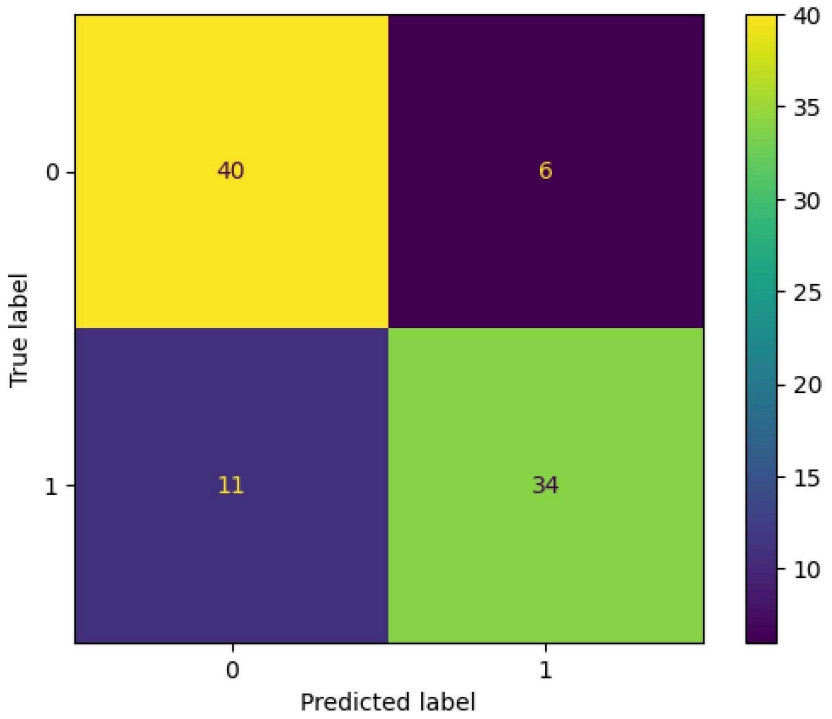
[61]:
```
print("Accuracy: ", metrics.accuracy_score(y_test, test_pred_svm_lin))
print("Precision: ", metrics.precision_score(y_test, test_pred_svm_lin))
print("Recall: ", metrics.recall_score(y_test, test_pred_svm_lin))
print("F1 Score: ", metrics.f1_score(y_test, test_pred_svm_lin))
cm = confusion_matrix(y_test, test_pred_svm_lin)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
```

Question assigned to the following page: 3.9

```
disp.plot()
```

```
Accuracy:  0.8131868131868132
Precision:  0.85
Recall:  0.7555555555555555
F1 Score:  0.7999999999999998
```

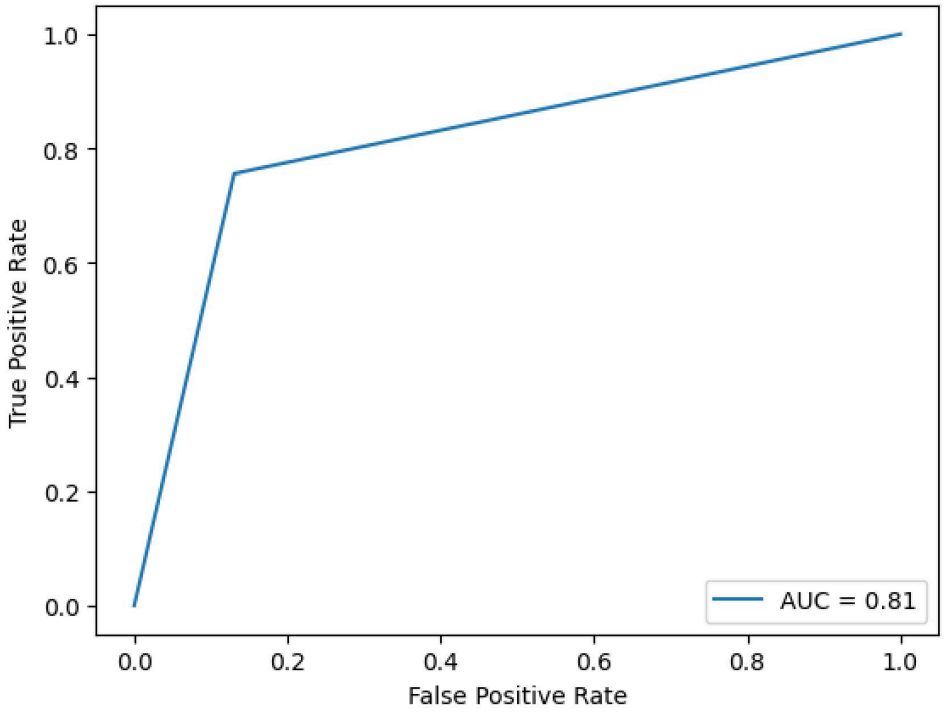
[61]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x12a836550>



[59]:
```
fpr, tpr, thresholds = metrics.roc_curve(y_test, test_pred_svm_lin)
roc_auc = metrics.auc(fpr, tpr)
display = metrics.RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc)
display.plot()
```

[59]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x12a299750>

Question assigned to the following page: 3.9



0.6.16 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 early model used SVC with the default kernel parameter, which typically defaults to 'rbf' (Radial Basis Function). The 'rbf' kernel is a popular choice and is suitable for non-linear classification tasks. It maps the input features into a higher-dimensional space to find a linear decision boundary. However, the default 'rbf' kernel may not provide optimal results for datasets with linearly separable classes, as it may introduce unnecessary complexity.

The recent model used SVC with the 'linear' kernel parameter explicitly specified. The 'linear' kernel is suitable for linearly separable datasets. It creates a decision boundary that is a hyperplane in the original feature space, making it a simpler and more interpretable model. By specifying the 'linear' kernel, the recent model explicitly instructs SVC to use a linear decision boundary, which may lead to improved performance on linearly separable datasets.

The change from the default 'rbf' kernel to the 'linear' kernel impacts the decision boundary of the SVC model. With the 'linear' kernel, the decision boundary becomes a hyperplane in the original feature space, making it more interpretable and easier to understand. This change may lead to improved performance on linearly separable datasets, as the model can find a simpler and more accurate decision boundary.

Questions assigned to the following page: 3.9 and 3.10

Thus, our recent model using the 'linear' kernel may have achieved improved results due to the linearly separable nature of the dataset, which allows the 'linear' kernel to create a more accurate decision boundary.

0.6.17 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?

Differences between the models: - Optimization: - Logistic Regression: In Logistic Regression, the optimization objective is to maximize the likelihood of the observed data given the model parameters (coefficients). It uses the logistic function (sigmoid function) to model the probability that a given data point belongs to a particular class. The decision boundary is determined by finding the coefficients that maximize the likelihood of the observed data. - Linear SVM: In Linear SVM, the optimization objective is to find the hyperplane that maximizes the margin between the classes. The margin is the distance between the hyperplane and the nearest data points from each class. The decision boundary is determined by finding the hyperplane that maximizes the margin.

- Loss function:
  - Logistic Regression: Logistic Regression uses the logistic loss function, which is a convex function that penalizes misclassifications. The logistic loss function is used to model the probability of a data point belonging to a particular class.
  - Linear SVM: Linear SVM uses the hinge loss function, which is a convex function that penalizes misclassifications. The hinge loss function is used to maximize the margin between the classes.
- Outliers:
  - Logistic Regression: Logistic Regression is sensitive to outliers, as it tries to model the probability of a data point belonging to a particular class. Outliers can affect the estimated probabilities and coefficients.
  - Linear SVM: Linear SVM is less sensitive to outliers, as it focuses on maximizing the margin between the classes. Outliers that are far from the decision boundary have little impact on the hyperplane.