Project 2

Graded

Student

TIYA CHOKHANI

Total Points

115 / 115 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

– 1 pt missing/incorrect plot

– 3 pts not readable / missing

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 **30** / 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 pts missing dimensions

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

✔ **– 0 pts** Correct

– 5 pts Not attempted

2.3 **Implement pipeline** **15** / 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

– 3 pts count all float/int as numeric

– 1 pt some wrong features

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

✔ **– 0 pts** Correct

– 1 pt Lack of discussion

– 1 pt Incomplete comparison

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

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

– 5 pts Not attempted

Question 3

Additional Learning Methods **60** / 60 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

– 1 pt missing .predict()

– 1 pt missing statistics

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

– 1 pt wrong solver

– 4 pts Answer not visible/wrong

– 1 pt Supposed to keep max iteration - 10

– 2 pts no statistics

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

– 1 pt Incorrect explanation

– 2 pts No explanation

– 1 pt Missing some values

– 2 pts No values reported

– 5 pts Answer not visible / missing

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

– 0.5 pts Missing true label vs predicted label axes labels

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

✔ **– 0 pts** Correct. As long as they plot the correct metric and the explanation makes sense.

– 0.5 pts Missing labels

– 1 pt Plot on prediction labels instead of prediction probabilities

– 2 pts Missing/Incorrect ROC plot

– 4 pts Unattempted/Unreadable

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

✔ **– 0 pts** Correct

– 0 pts Rbf kernel not mentioned

– 5 pts Unattempted/Assignment unreadable

– 2 pts No explanation

– 1 pt Missing ROC curve

– 1 pt Missing/wrong confusion matrix

– 1 pt not enough explanation

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

✔ **– 0 pts** Correct

– 1 pt Partially correct: Missing one of the two of log reg or svm

– 3 pts Incorrect explanation: doesn't compare log reg and svm

– 5 pts No attempt/Answer not readable

3.11 Implement Basic Decision Tree — **4** / 4 pts

- ✔ **– 0 pts** Correct
- **– 2 pts** KNN wrong/not attempted
- **– 2 pts** Decision tree wrong/not attempted
- **– 4 pts** Not attempted or wrong implementation

3.12 Categorical Preprocessing Only — **5** / 5 pts

- ✔ **– 0 pts** Correct
- **– 1 pt** Incorrect accuracies
- **– 1 pt** No passthrough
- **– 2 pts** Accuracies not reported
- **– 5 pts** Missing / wrong implementation

3.13 Explain the difference in accuracy — **6** / 6 pts

- ✔ **– 0 pts** Correct
- **– 2 pts** Partially correct explanation
- **– 6 pts** Missing / incorrect

No questions assigned to the following page.

25W-COM SCI-M148 Project 2 - Binary Classification Comparative Methods

Name: Tiya chokhani

UID: 305933966

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.

No questions assigned to the following page.

Background: The Dataset

For this exercise, we will be using a subset of the UCI Heart Disease dataset. This dataset was created by collecting clinical data from patients undergoing diagnostic tests for heart disease. All identifying information about the patients has been removed to protect their privacy. The dataset represents data from patients who were suspected of having heart disease and underwent several diagnostic tests, including blood tests, electrocardiograms (ECG), exercise stress tests, and fluoroscopic imaging.

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

age: Patient age in years

sex: Patient sex (1 = male; 0 = female)

c_pain: Chest pain type (0 = asymptomatic; 1 = atypical angina (unusual discomfort due to reduced blood flow to the heart); 2 = non-anginal pain (chest pain unrelated to the heart); 3 = typical angina (classic chest discomfort due to reduced blood flow to the heart))

rbp: Resting blood pressure in mm Hg (measured at hospital admission)

chol: Serum cholesterol level in mg/dL

high_fbs: Fasting blood sugar > 120 mg/dL (1 = true; 0 = false)

r_ecg: Resting electrocardiographic results (0 = probable thickened left ventricular wall; 1 = normal; 2 = ST-T wave abnormality)

hr_max: Maximum heart rate achieved during the stress test

has_ex_ang: Exercise-induced angina (1 = yes; 0 = no)

ecg_depress: Depression of the ST segment on ECG during exercise compared to rest (measured in mm)

stress_slope: Slope of the peak exercise ST segment (0 = downsloping (concerning); 1 = flat (abnormal); 2 = upsloping (normal))

num_vessels: Number of major vessels (0–3) showing good blood flow during fluoroscopy

thal_test_res: Thallium Stress Test result (assesses blood flow using trace amounts of radioactive thallium-201) (1 = normal; 2 = fixed defect; 7 = reversible defect)

heart_disease: Indicates whether heart disease is present (True = Disease; False = No disease)

Loading Essentials and Helper Functions

```
#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
```

Questions assigned to the following page: 1.2 and 1.1

```
import seaborn as sns # used for plot interactive graph.
from sklearn.model_selection import train_test_split, cross_val_score,
GridSearchCV
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix
import sklearn.metrics.cluster as smc
from sklearn.model_selection import KFold


from matplotlib import pyplot
import itertools

%matplotlib inline

import random

random.seed(42)
```

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.)

```
data = pd.read_csv('heartdisease.csv')
```

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.

```
data.head()
```

	age	sex	chest_pain	rpb	chol	high_fbs	r_ecg	hr_max	ex_ang	ecg_depress	stress_slope	nu
0	63	1	3	145	233	1	0	150	0	2.3	0	0
1	37	1	2	130	250	0	1	187	0	3.5	0	0
2	41	0	1	130	204	0	0	172	0	1.4	2	0
3	56	1	1	120	236	0	1	178	0	0.8	2	0
4	57	0	0	120	354	0	1	163	1	0.6	2	0

Questions assigned to the following page: 1.3 and 1.4

```
data.describe()
```

	age	sex	chest_pain	rpb	chol	high_fbs	r_ecg	hr_max
count	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
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000

```
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   chest_pain     303 non-null    int64
 3   rpb            303 non-null    int64
 4   chol           303 non-null    int64
 5   high_fbs       303 non-null    int64
 6   r_ecg          303 non-null    int64
 7   hr_max         303 non-null    int64
 8   ex_ang         303 non-null    int64
 9   ecg_depress    303 non-null    float64
 10  stress_slope   303 non-null    int64
 11  num_vessels    303 non-null    int64
 12  thal_test_res  303 non-null    int64
 13  heart_disease  303 non-null    bool
dtypes: bool(1), float64(1), int64(12)
memory usage: 31.2 KB
```

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

```
data["heart_disease_int"] = data["heart_disease"].astype(int)
data.drop('heart_disease', axis=1, inplace=True)
```

Question assigned to the following page: 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 ascertain how each variable performs (is it a binary, or limited selection, or does it follow a gradient?

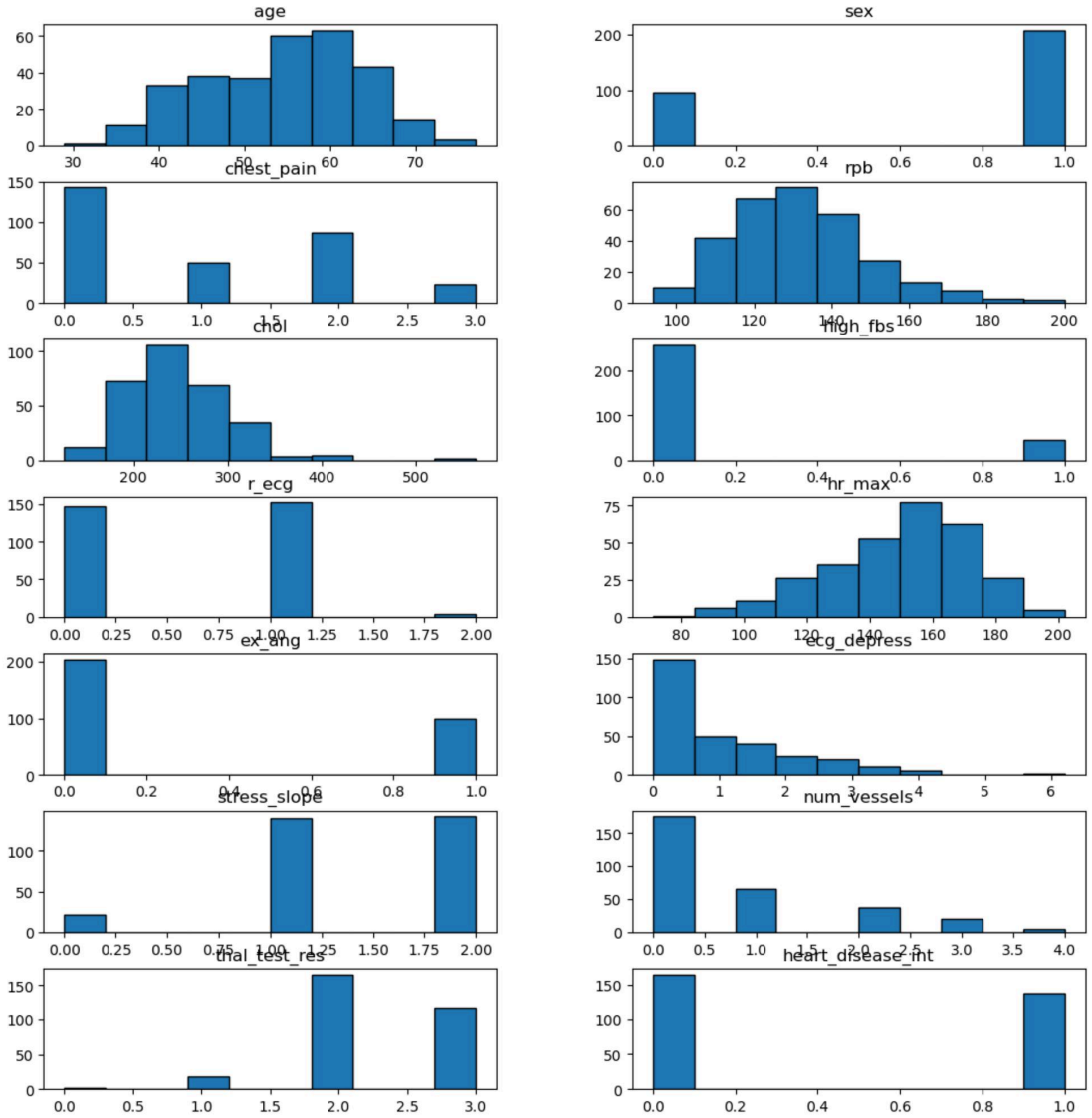
```
fig, axis = plt.subplots(7, 2, figsize=(12, 12))  # Corrected 'fogsize' to
'figsize'
fig.tight_layout(pad=5.0)

# Flatten the axis array
axis = axis.flatten()

# Plot histograms
data.hist(ax=axis, grid=False, edgecolor='black')

plt.show()
```

Question assigned to the following page: 1.5



Age – The distribution is slightly right-skewed, with most individuals aged between 40 and 65.

Resting Blood Pressure (rpb) – Appears normally distributed, with most values around 120-140 mmHg, but some individuals have higher outliers.

Cholesterol (chol) – Right-skewed, meaning most individuals have cholesterol levels below 300 mg/dL, but a few have very high values.

Max Heart Rate (hr_max) – Left-skewed, showing that most patients have a high heart rate during exercise.

ECG Depression (ecg_depress) – Highly skewed towards zero, indicating that many individuals have little or no ST depression.

Question assigned to the following page: 1.6

Sex – More males (1) than females (0) in the dataset. Chest Pain Type (chest_pain) – Imbalanced, with Type 3 being the most common.

Fasting Blood Sugar (high_fbs) – Mostly 0, indicating that most individuals do not have high fasting blood sugar.

Exercise-Induced Angina (ex_ang) – Mostly 0, meaning most patients do not experience angina during exercise.

Stress Test Slope (stress_slope) – Shows a clear categorical distribution, with one category being dominant.

Chest Pain Type (chest_pain) – Type 3 is the most common, followed by Type 2 and Type 1, while Type 0 is the least frequent, suggesting that many individuals experience some form of chest pain.

Resting ECG Results (r_ecg) – The distribution is dominated by category 1, while categories 0 and 2 are less common.

Number of Major Vessels Colored by Fluoroscopy (num_vessels) – Most individuals have 0 or 1 major vessel detected, while having 3 or more is relatively rare.

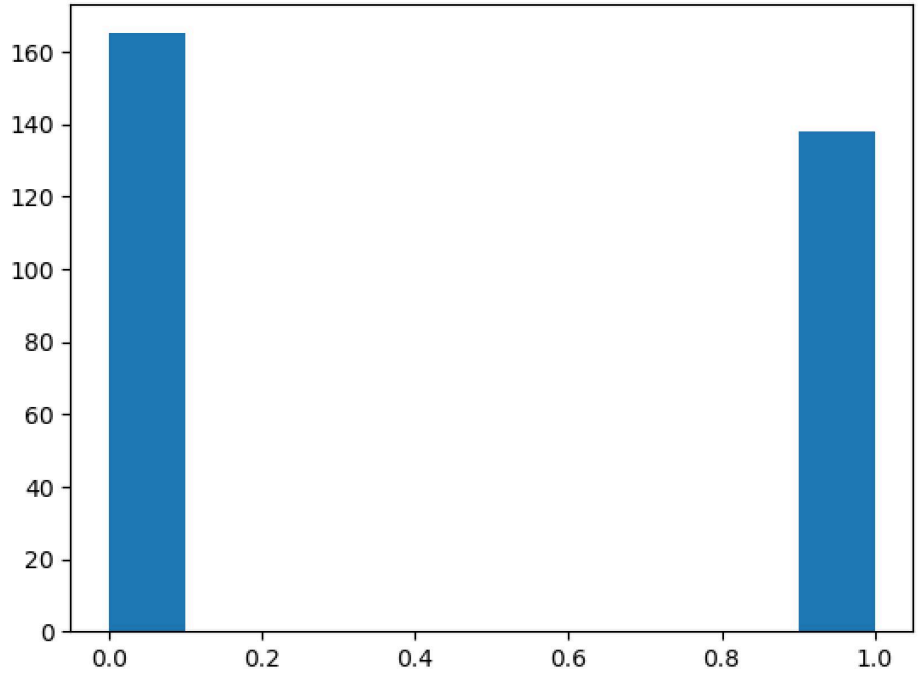
Thalassemia Test Results (thal_test_res) – The dataset includes multiple categories, with category 2 being the most common, while categories 1 and 3 appear less frequently.

Heart Disease Diagnosis (heart_disease_int) – Appears to be relatively balanced, though there may be a slight class imbalance favoring individuals without heart disease.

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 heart_disease target, and conduct a count of the number of diseased and healthy individuals and report on the results:

```
data["heart_disease_int"].hist(grid=False)
plt.show()
hd_count = data["heart_disease_int"].value_counts()
print(hd_count)
print("There are 165 healthy patients and 138 partients with heart disease this
is a relatively balanced data set")
```

Questions assigned to the following page: 1.6 and 1.7

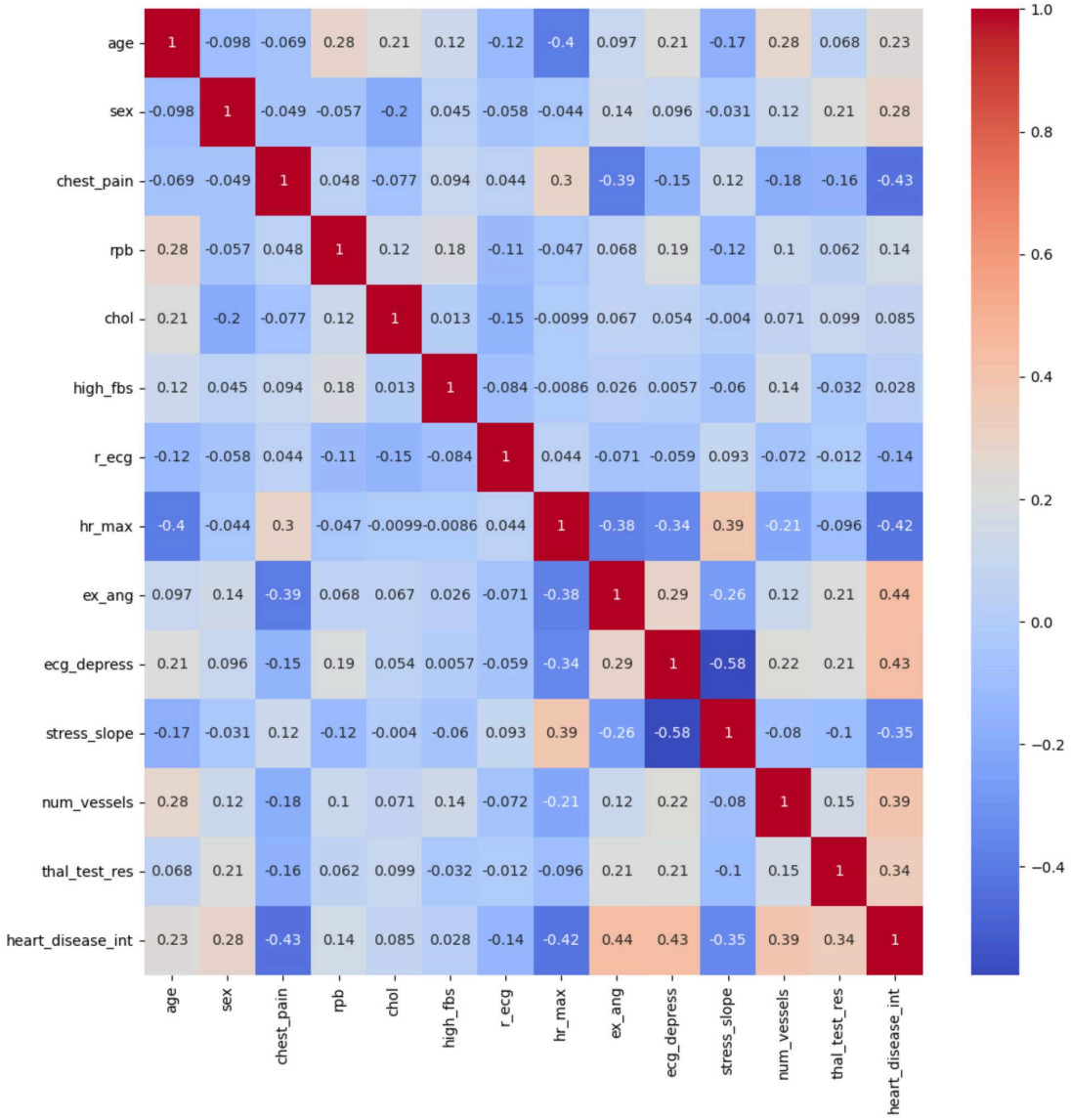



```
heart_disease_int
0    165
1    138
Name: count, dtype: int64
There are 165 healthy patients and 138 partients with heart disease this is a
relatively balanced data set
```

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)?

```
correlation_matrix = data.corr()
plt.figure(figsize=(12,12))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
plt.show()
```

Question assigned to the following page: 1.7



From the correlation matrix, it is clear that 3 variables have a moderate negative correlation with the target value. These variables are: chest_pain, hr_max, stress_slope. The variable with the highest correlation is chest_pain. This makes sense as chest pain is a common symptom of heart disease. The other two variables, hr_max, stress_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: ex_ang, ecg_depress, num_vessels, and thal_test_res. ex_ang 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. num_vessels 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_test_res is a measure of the blood flow to the heart and thus has

Questions assigned to the following page: 1.7, 2.1, and 2.2

a positive correlation. ecg_depress 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.

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!

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

```
hd_int_arr = data["heart_disease_int"].values
data.drop('heart_disease_int', axis=1, inplace=True)
```

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.

```
x_train, x_test, y_train, y_test = train_test_split(data, hd_int_arr,
test_size=0.3)
```

```
print("Trining set X shape: ", x_train.shape)
print("Trining set Y shape: ", y_train.shape)
print("Testing set X shape: ", x_test.shape)
print("Testing set Y shape: ", y_test.shape)
```

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

Questions assigned to the following page: 2.2 and 2.3

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.

```
knn = KNeighborsClassifier()
knn.fit(x_train, y_train)
```

```
KNeighborsClassifier()
```

```
knn.score(x_test, y_test)
```

```
0.6263736263736264
```

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.

Pipeline:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer, make_column_transformer
```

```
# Create pipelines
numerical_features = ['age', 'rpb', 'chol', 'hr_max', 'ecg_depress']
categorical_features = ['sex', 'chest_pain', 'high_fbs', 'r_ecg', 'ex_ang',
'stress_slope', 'num_vessels', 'thal_test_res']

numerical_transformer = Pipeline(steps=[('scaler', StandardScaler())])
categorical_transformer = Pipeline(steps=[('onehot', OneHotEncoder())])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
```

Questions assigned to the following page: 2.4, 3.1, and 3.2

```
    ]
)

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

```
# Pipeline the training and test data
x_test_transformed = preprocessor.transform(x_test)
```

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

```
# KNN
knn_new = KNeighborsClassifier()
knn_new.fit(x_train_transformed, y_train)
```

```
KNeighborsClassifier()
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
# Report Metrics
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:  0.8241758241758241
Precision:  0.8260869565217391
Recall:  0.8260869565217391
F1 Score:  0.8260869565217391
```

Scaling the features improved the models performance significantly. The accuracy of the model increased to 0.846 from 0.637. Scaling the features allows the model to better undersatnd 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 euclideans distanes are not skewed by the magnitude of the features.

Questions assigned to the following page: 3.1 and 3.2

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).

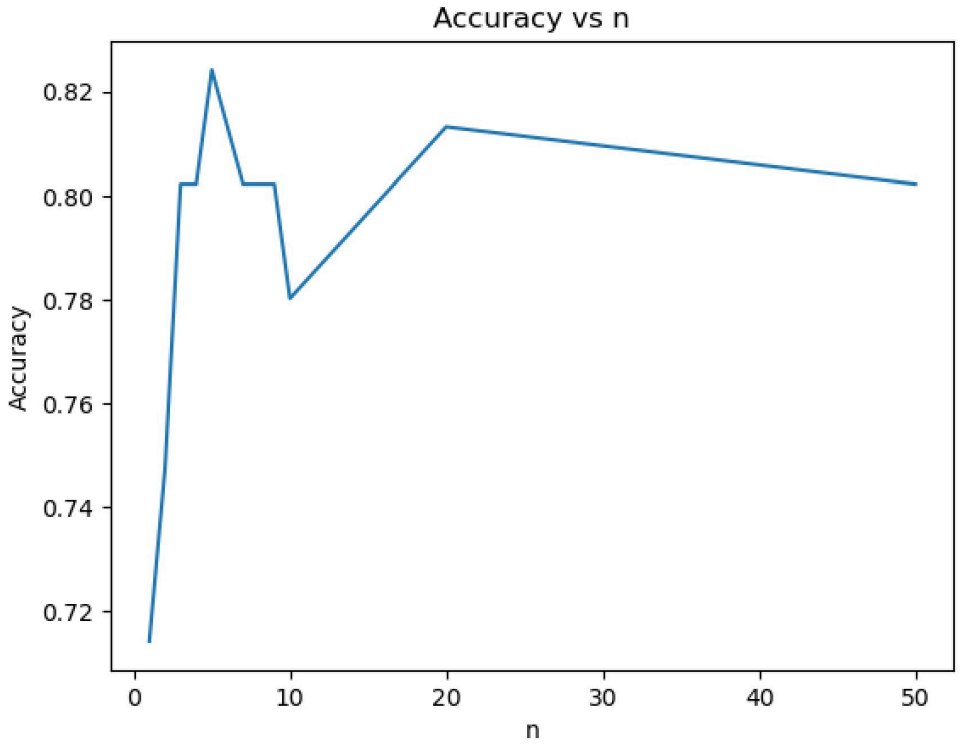
```
n = [1,2,3,4,5,7,9,10,20,50]
acc_n = []

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  0.7142857142857143
Accuracy for k =  2  is  0.7472527472527473
Accuracy for k =  3  is  0.8021978021978022
Accuracy for k =  4  is  0.8021978021978022
Accuracy for k =  5  is  0.8241758241758241
Accuracy for k =  7  is  0.8021978021978022
Accuracy for k =  9  is  0.8021978021978022
Accuracy for k =  10  is  0.7802197802197802
Accuracy for k =  20  is  0.8131868131868132
Accuracy for k =  50  is  0.8021978021978022
```

Questions assigned to the following page: 3.1 and 3.2



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.

Linear Decision Boundary Methods

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.

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

Report metrics for:

1. Accuracy
2. Precision

Question assigned to the following page: 3.3

3. Recall
4. F1 Score

```
# 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:  0.8021978021978022
Precision:  0.7916666666666666
Recall:  0.8260869565217391
F1 Score: 0.8085106382978723
```

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. Fl-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.

Questions assigned to the following page: 3.3, 3.4, and 3.5

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

Let's tweak a few settings. First let's set your solver to 'sag' (Stochastic Average Gradient), your max_iter= 10, and set penalty = None and rerun your model. Let's see how your results change!

```
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:  0.7472527472527473
Precision:  0.7254901960784313
Recall:  0.8043478260869565
F1 Score:  0.7628865979381443
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.py:349:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
```

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.

```
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))
```

Questions assigned to the following page: 3.4 and 3.5

```
Accuracy:  0.7362637362637363
Precision:  0.72
Recall:  0.782608695652174
F1 Score:  0.75
```

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.

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

```
log_reg_sag = LogisticRegression(solver='liblinear', max_iter=10000,
penalty='l1')
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:  0.7916666666666666
Recall:  0.8260869565217391
F1 Score:  0.8085106382978723
```

Explain 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: 3.7, 3.6, and 3.5

'liblinear' may not 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'. sag is designed for larger datasets but can still work well on smaller datasets when features aren't too many. liblinear works well when the number of features is relatively high, but our dataset has only 13 features. We used L1 (Lasso) regularization with liblinear, which tends to shrink some coefficients to zero. This could have affected model performance, especially if some features are more important than others. sag was run with penalty=None, meaning it used no regularization, potentially capturing more relationships in the data.

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.

More explanation here: https://en.wikipedia.org/wiki/Support_vector_machine.

For the sake of this project, you can regard it as a type of classifier.

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.

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

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

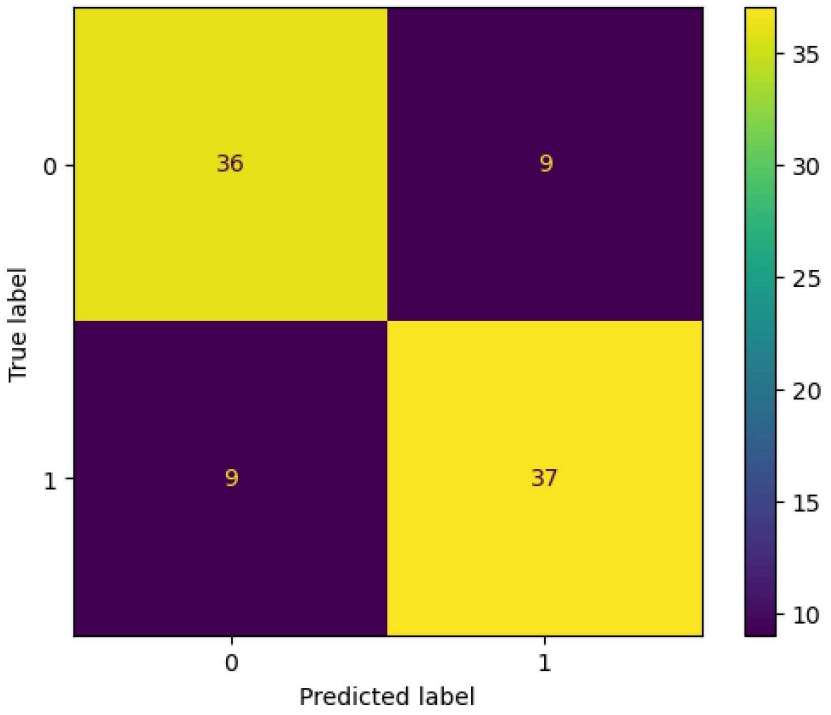
recommend using `from sklearn.metrics import ConfusionMatrixDisplay` for this one!

```
# Report Metrics
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:  0.8021978021978022
Precision:  0.8043478260869565
Recall:  0.8043478260869565
F1 Score:  0.8043478260869565
```

Questions assigned to the following page: 3.7 and 3.8

```
# Confusion Matrix
from sklearn.metrics import ConfusionMatrixDisplay
cm = confusion_matrix(y_test,test_pred_svm)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()
```

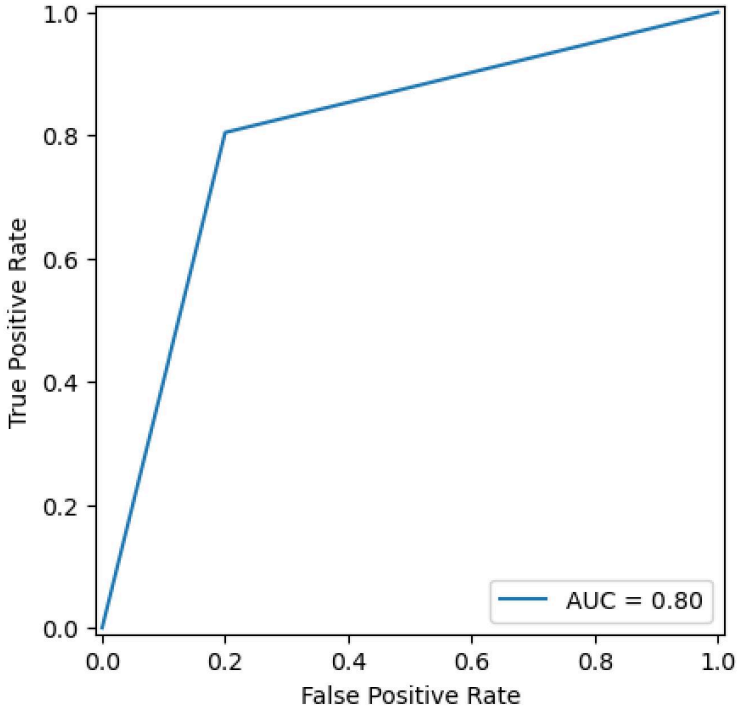


Plot a Receiver Operating Characteristic curve, or ROC curve, and describe what it is and what the results indicate

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

```
# ROC
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()
plt.show()
```

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.

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.

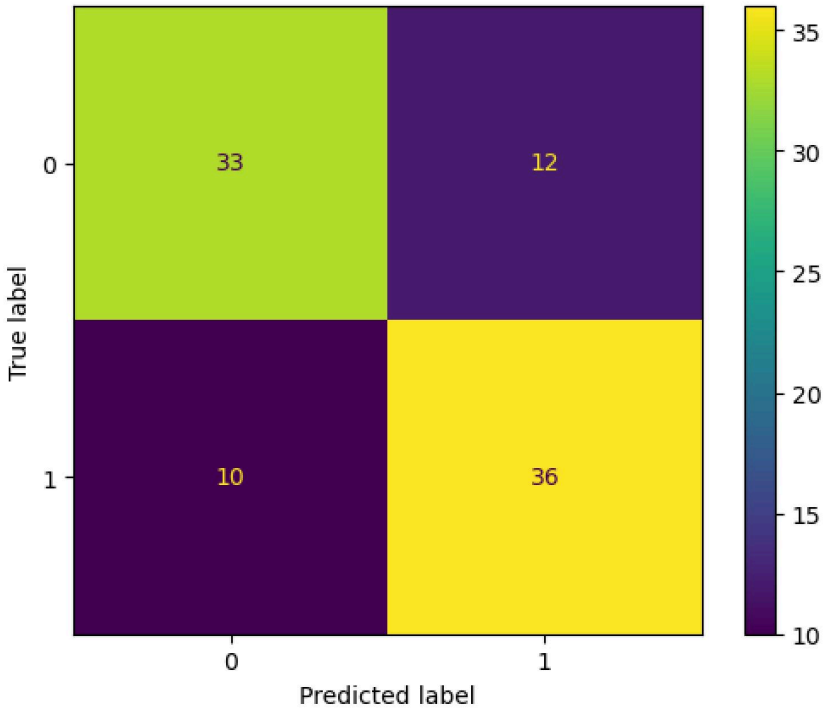
```
# Metrics
svm_lin = SVC(kernel='linear', probability=True)
svm_lin.fit (x_train_transformed, y_train)
test_pred_svm_lin = svm_lin.predict(x_test_transformed)
```

```
# Confusion Matrix
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)
```

Questions assigned to the following page: 3.8 and 3.9

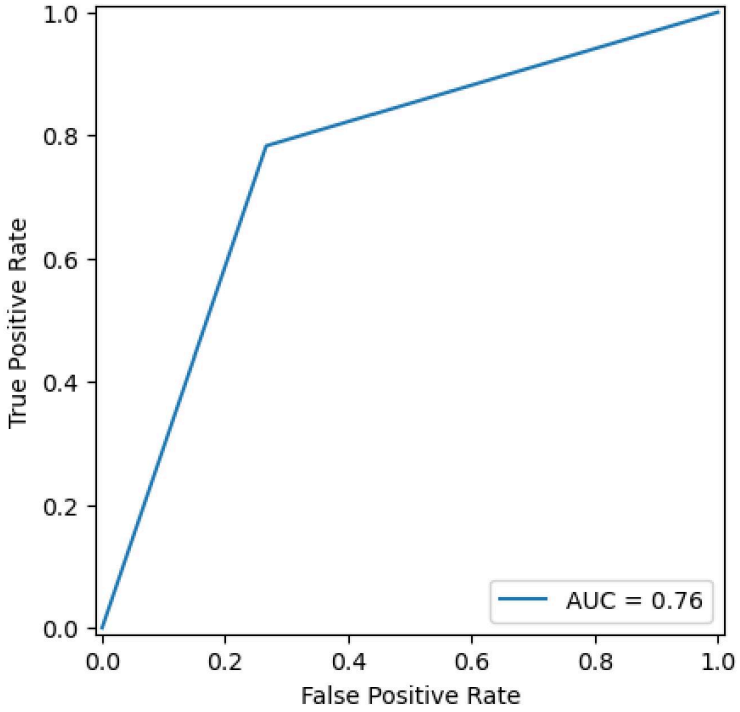
```
disp = ConfusionMatrixDisplay (confusion_matrix=cm)
disp.plot()
plt.show()
```

```
Accuracy:  0.7582417582417582
Precision:  0.75
Recall: 0.782608695652174
F1 Score:  0.7659574468085106
```




```
# ROC
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()
plt.show()
```

Questions assigned to the following page: 3.9 and 3.10



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.

In the initial model, SVC was used with the default kernel, which is the Radial Basis Function (RBF) kernel. The RBF kernel is a popular choice for non-linear classification tasks because it maps the input features into a higher-dimensional space, where a linear decision boundary can be found. This makes it effective for complex decision boundaries but can introduce unnecessary complexity if the dataset is actually linearly separable.

In the updated model, the kernel was explicitly set to 'linear', instructing SVC to find a linear decision boundary in the original feature space. The linear kernel is most effective when the data is linearly separable or nearly so, as it avoids the additional complexity of transforming the data into a higher-dimensional space.

The AUC (Area Under the ROC Curve) score changed from 0.80 to 0.76, a slight decrease. This shift suggests that: - The dataset may not be perfectly linearly separable – The RBF kernel, by mapping data into a higher-dimensional space, might have captured subtle non-linear relationships better than the linear kernel. - Regularization and Model Complexity Tradeoff – The linear kernel results in a simpler model, but simplicity does not always translate to better performance. The RBF kernel, despite being more complex, might have led to slightly better classification for this dataset.

Questions assigned to the following page: 3.10 and 3.11

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 sensitivito 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.

Decision Trees

Create both a Decision Tree and a KNN and fit them onto your fully preprocessed data, then calculate an accuracy score for both (https://scikit-learn.org/stable/api/sklearn.tree.html).

What are Decision Trees?

Decision Trees are a non-parametric supervised learning methods used for classification and regression. The goal is to split data into branches based on feature conditions, forming a tree-like structure where each internal node represents a decision, and each branch represents an outcome.

Compared to KNN, decision trees is less influenced by the high dimensionality of the data, and can make the model output more predicable.

For more explanation, see here: https://en.wikipedia.org/wiki/Decision_tree. For the sake of this project, you can regard it as a type of classifier.

```
from sklearn.tree import DecisionTreeClassifier
# Decision Tree
dt_classifier = DecisionTreeClassifier()
dt_classifier.fit(x_train_transformed, y_train)
y_pred_dt = dt_classifier.predict(x_test_transformed)

# KNN
knn_classifier = KNeighborsClassifier()  # Using 5 neighbors as default
knn_classifier.fit(x_train_transformed, y_train)
```

Questions assigned to the following page: 3.11 and 3.12

```
y_pred_knn = knn_classifier.predict(x_test_transformed)
```

```
# Decision Tree Accuracy
print("Decision Tree Accuracy:", metrics.accuracy_score(y_test, y_pred_dt))

# KNN Accuracy
print("KNN Accuracy:", metrics.accuracy_score(y_test, y_pred_knn))
```

```
Decision Tree Accuracy: 0.7802197802197802
KNN Accuracy: 0.8241758241758241
```

Categorical Preprocessing Only

Create a new preprocessing pipeline which ONLY preprocesses categorical values (leaving scalar variables in the data as they were originally, ie. no StandardScaler).
Process your data with this new pipeline, fit a decision tree and a KNN once more and report a new accuracy score for each.

Hint: Ensure that remainder = 'passthrough' in your ColumnTransformer to ensure scalar values are not dropped!

```
# Categorical Preprocessing Only
preprocessor_categorical = ColumnTransformer(
    transformers=[('cat', categorical_transformer, categorical_features)],
    remainder='passthrough'  # Keep numerical features as they are
)

x_train_transformed_cat = preprocessor_categorical.fit_transform(x_train)
x_test_transformed_cat = preprocessor_categorical.transform(x_test)
```

```
# Fit Decision Tree
dt_classifier_cat = DecisionTreeClassifier()
dt_classifier_cat.fit(x_train_transformed_cat, y_train)
y_pred_dt_cat = dt_classifier_cat.predict(x_test_transformed_cat)

# Fit KNN
knn_classifier_cat = KNeighborsClassifier()
knn_classifier_cat.fit(x_train_transformed_cat, y_train)
y_pred_knn_cat = knn_classifier_cat.predict(x_test_transformed_cat)
```

```
# Decision Tree Accuracy
print("Decision Tree Accuracy:", metrics.accuracy_score(y_test, y_pred_dt_cat))

# KNN Accuracy
print("KNN Accuracy:", metrics.accuracy_score(y_test, y_pred_knn_cat))
```

Questions assigned to the following page: 3.12 and 3.13

```
Decision Tree Accuracy: 0.8021978021978022
KNN Accuracy: 0.6263736263736264
```

Explain the difference in accuracy loss in Decision Trees vs KNNs when Standardization was removed.

When standardization was removed (i.e., only categorical features were preprocessed), the impact on Decision Trees and KNN was different:

1. Decision Trees:

   - Before (with Standardization): 78.02% Accuracy
   - After (Categorical Only): 80.22% Accuracy (Slight increase)

   Decision Trees are not affected by feature scaling because they split data based on feature thresholds rather than distance calculations. When standardization was removed, numerical features retained their original values, potentially making the splits more meaningful for classification. Categorical encoding (OneHotEncoder) may have allowed the model to capture relationships better, leading to a slight increase in accuracy.

2. K-Nearest Neighbors (KNN):

   - Before (with Standardization): 82.42% Accuracy
   - After (Categorical Only): 62.64% Accuracy (Significant drop)

   Without standardization, numerical features dominate distance calculations: Example: If age ranges from 30 to 80 and sex is 0 or 1, unscaled age has a much larger numerical range, overshadowing categorical variables. This distorts distance measurements, making KNN's similarity-based classification less reliable. Standardization equalized feature scales, preventing any one feature from disproportionately affecting distances. Without scaling, KNN's nearest neighbors become biased towards large-scale numerical features, leading to worse classification performance.

Printing Jupyter notebook to PDF (Google Colab Only, Optional)

It may take a few minutes to run