Heart Disease Classification Analysis

McMaster University - STATS 3DA3

Anna Rasheed - 400441966

2025-03-21

Heart Disease Classification Analysis

Overview

Heart disease remains one of the leading causes of mortality worldwide. Timely detection and accurate classification are critical for effective treatment and improved patient outcomes. This project aims to develop a model that can predict the presence of heart disease using a range of patient health metrics, such as blood pressure, cholesterol levels, and other relevant features.

About the Dataset

The dataset used in this project is sourced from the UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/Heart+Disease

This dataset contains 303 observations with 14 variables. There are 13 features and 1 target variable. The feature variables are `age`, `sex`, `cp`, `trestbps`, `chol`, `fbs`, `restecg`, `thalach`, `exang`, `oldpeak`, `slope`, `ca`, and `thal`.

The target variable is `num`.

The dataset contains both categorical and numerical variables. The categorical variables are `sex`, `cp`, `fbs`, `restecg`, `exang`, and `slope`. The numerical variables are `age`, `trestbps`, `chol`, `thalach`,`oldpeak` and `ca`.

The target variable `num` is a categorical variable with 5 classes: 0, 1, 2, 3, and 4. We will transform this variable into a binary variable with two classes: 0 and 1 later. The class 0 represents the absence of heart disease, while the class 1 represents the presence of heart disease.

Objectives

I will be analyzing this dataset using two different classification algorithms:

1. **Logistic Regression**:

    - I chose Logistic Regression as one of my classifiers for this assignment because it provides clear insight into the influence of each feature on the predicted probability of heart disease. Since our classification is binary (whether the patient has a heart disease or not), logistic regression is a good fit for this problem. The coefficients of the logistic

regression model can be interpreted as odds ratios, which allows us to understand the impact of each predictor variable on the likelihood of heart disease. This interpretability is crucial for medical applications where understanding the influence of different factors is important for diagnosis and treatment decisions.

2. **Support Vector Machines (SVM)**:

    - I chose Support Vector Machines (SVM) as my second classifier because it is effectine on relatively small datasets since they focus on the most informative data points and don't overfit easily. SVM also performs well with non-linear relationships between features and the target variable, which may be important for this dataset. It is tunable and can be adapted to different types of data distributions. While SVM is less interpretable than logistic regression, it can still provide valuable insights into the classification problem.

The metrics I will use to compare the performance of the classifiers are:

1. **Accuracy**:

    - Accuracy is the ratio of correctly predicted instances to the total instances in the dataset. It gives a straightforward measure of how well the model is performing overall.

2. **ROC Curve (Specificity & Sensitivity)**

    - The ROC curve is a graphical representation of the true positive rate (sensitivity) against the false positive rate (1 - specificity) at various threshold settings. The area under the ROC curve (AUC) provides a single measure of overall model performance, with values ranging from 0 to 1. A value of 0.5 indicates random guessing, while a value of 1 indicates perfect discrimination. We use specificity and sensitivity to understand the trade-off between true positive and false positive rates. This metric was chosen for this dataset since the dataset is not imbalanced.

I will be using Lasso as a feature selection technique to identify the most relevant features for predicting heart disease. The dataset will be split into training and testing sets to evaluate the performance of both models.

The goal is to classify whether a patient has heart disease or not.

```
#import
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn import svm
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score, accuracy_score, confusion_mat
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
```

```
#get dataset from UCI Machine Learning Repository
from ucimlrepo import fetch_ucirepo

# fetch dataset
heart_disease = fetch_ucirepo(id=45)
data = heart_disease.data.original

# variable information
print(heart_disease.variables)
```

```
       name     role         type demographic  \
0       age  Feature      Integer         Age
1       sex  Feature  Categorical         Sex
2        cp  Feature  Categorical        None
3  trestbps  Feature      Integer        None
4      chol  Feature      Integer        None
5       fbs  Feature  Categorical        None
```

6	restecg	Feature	Categorical	None
7	thalach	Feature	Integer	None
8	exang	Feature	Categorical	None
9	oldpeak	Feature	Integer	None
10	slope	Feature	Categorical	None
11	ca	Feature	Integer	None
12	thal	Feature	Categorical	None
13	num	Target	Integer	None

	description	units	missing_values
0	None	years	no
1	None	None	no
2	None	None	no
3	resting blood pressure (on admission to the ho...	mm Hg	no
4	serum cholestoral	mg/dl	no
5	fasting blood sugar > 120 mg/dl	None	no
6	None	None	no
7	maximum heart rate achieved	None	no
8	exercise induced angina	None	no
9	ST depression induced by exercise relative to ...	None	no
10	None	None	no
11	number of major vessels (0-3) colored by flour...	None	yes
12	None	None	yes
13	diagnosis of heart disease	None	no

```
data.shape
```

```
(303, 14)
```

```
data.head(5)
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	num
0	63	1	1	145	233	1	2	150	0	2.3	3	0.0	6.0	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3.0	3.0	2
2	67	1	4	120	229	0	2	129	1	2.6	2	2.0	7.0	1
3	37	1	3	130	250	0	0	187	0	3.5	3	0.0	3.0	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0.0	3.0	0

```
data.dtypes
```

```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca          float64
thal        float64
num           int64
dtype: object
```

We can see that some of the variables that are supposed to be categorical are listed as integers or floats. Let's change those variables to categorical.

```
#change the variables that are supposed to be categorical to categorical
data['sex'] = pd.Categorical(data['sex'])
data['cp'] = pd.Categorical(data['cp'])
data['fbs'] = pd.Categorical(data['fbs'])
```

```
data['restecg'] = pd.Categorical(data['restecg'])
data['exang'] = pd.Categorical(data['exang'])
data['slope'] = pd.Categorical(data['slope'])
data['thal'] = pd.Categorical(data['thal'])

#we will deal with the target variable later

#check the data types
data.dtypes
```

```
age            int64
sex         category
cp          category
trestbps       int64
chol           int64
fbs         category
restecg     category
thalach        int64
exang       category
oldpeak      float64
slope       category
ca           float64
thal        category
num            int64
dtype: object
```

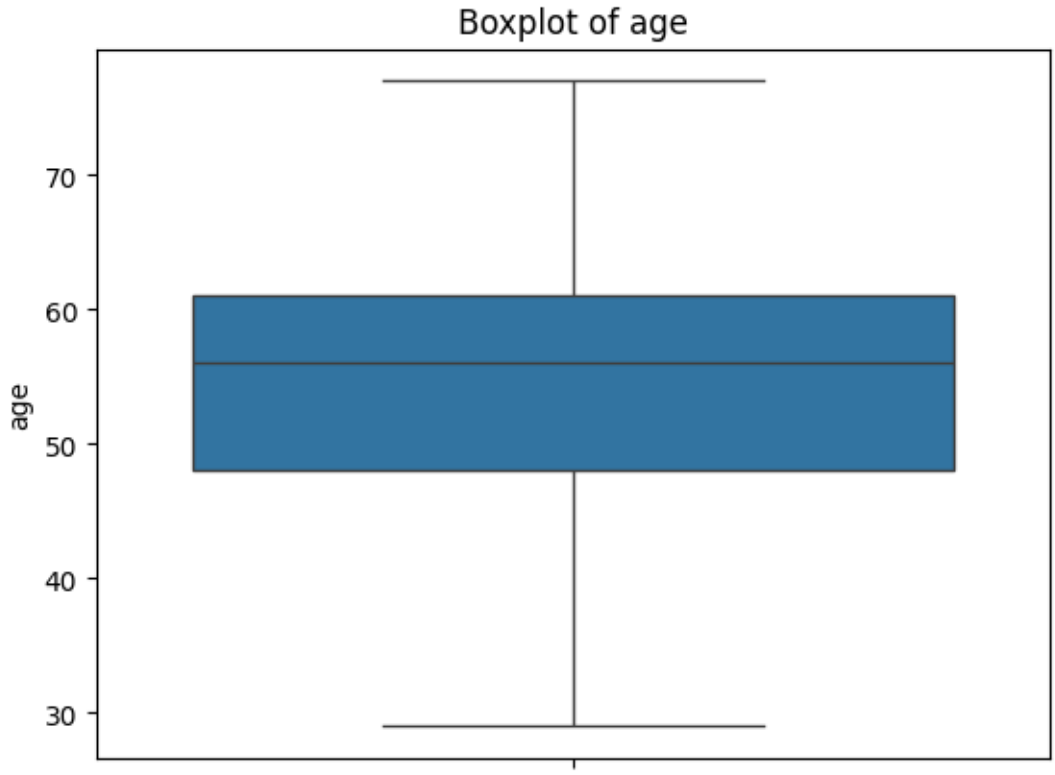
The variables are now in the correct form.

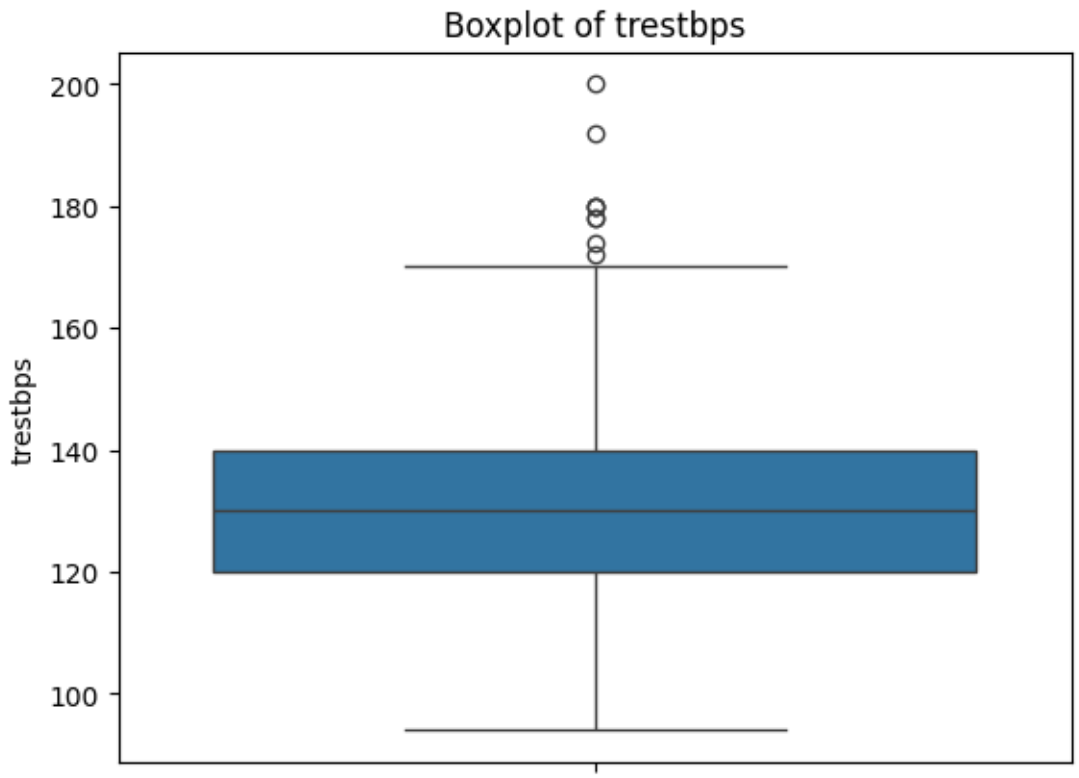
The variable `ca` is defined to be the number of major vessels (0-3) colored by fluoroscopy, which can be considered a categorical variable since it is discrete. However, we will leave it as an integer variable since that is how it is defined in the data dictionary.

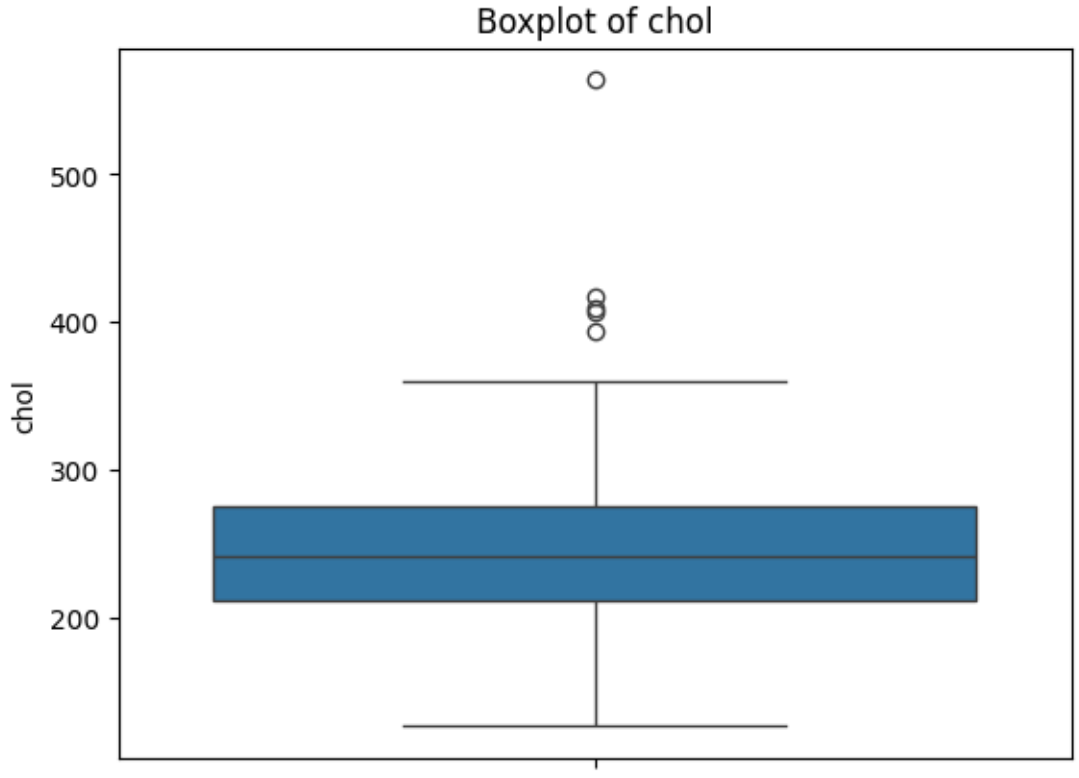
We will also need to handle missing values in the data set, which is also being done later.

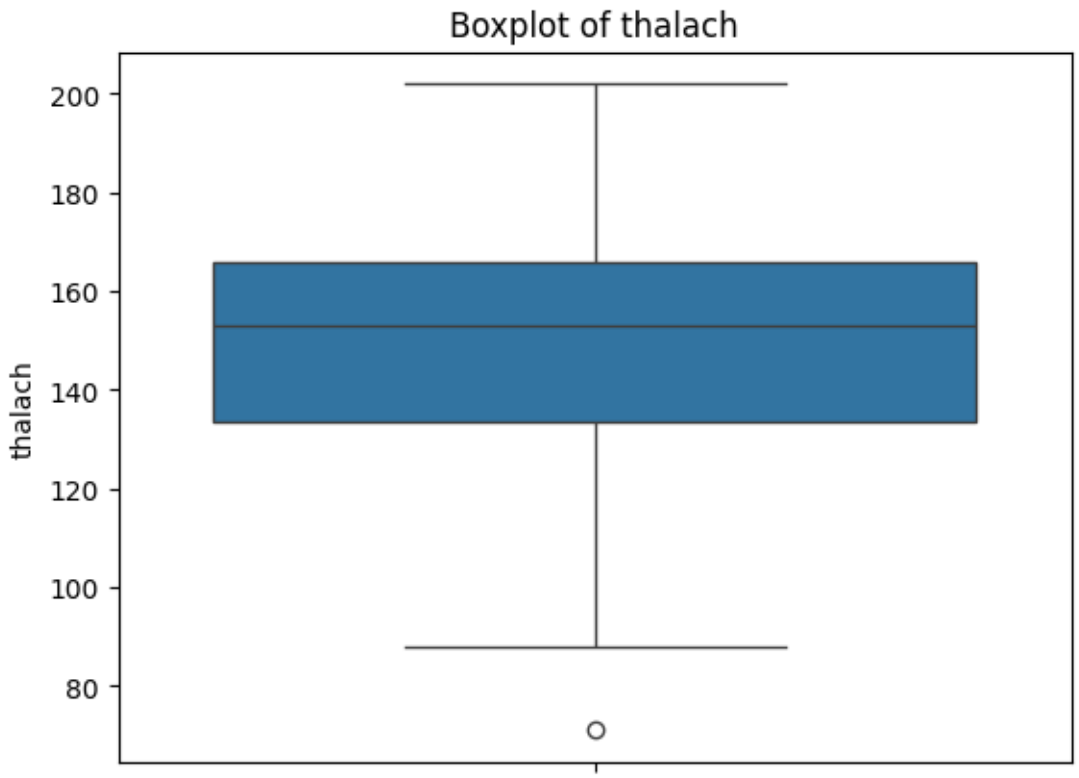
Since I would later like to use Logistic Regression and SVM, I will need to convert the categorical variables that have more than two levels into dummy variables later.

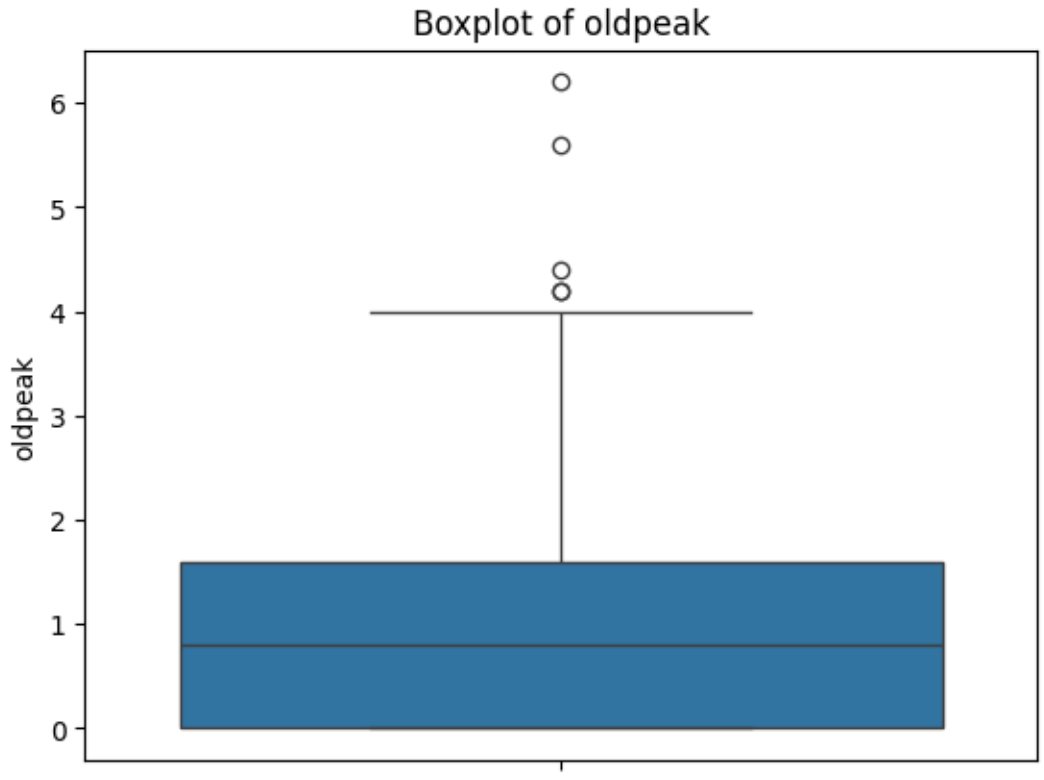
```
#create boxplots for identifying outliers in numerical features

features = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak', 'ca']
#for feature in features:

for f in features:
    sns.boxplot(data=data[f])
    plt.title(f'Boxplot of {f}')
    plt.show()
```

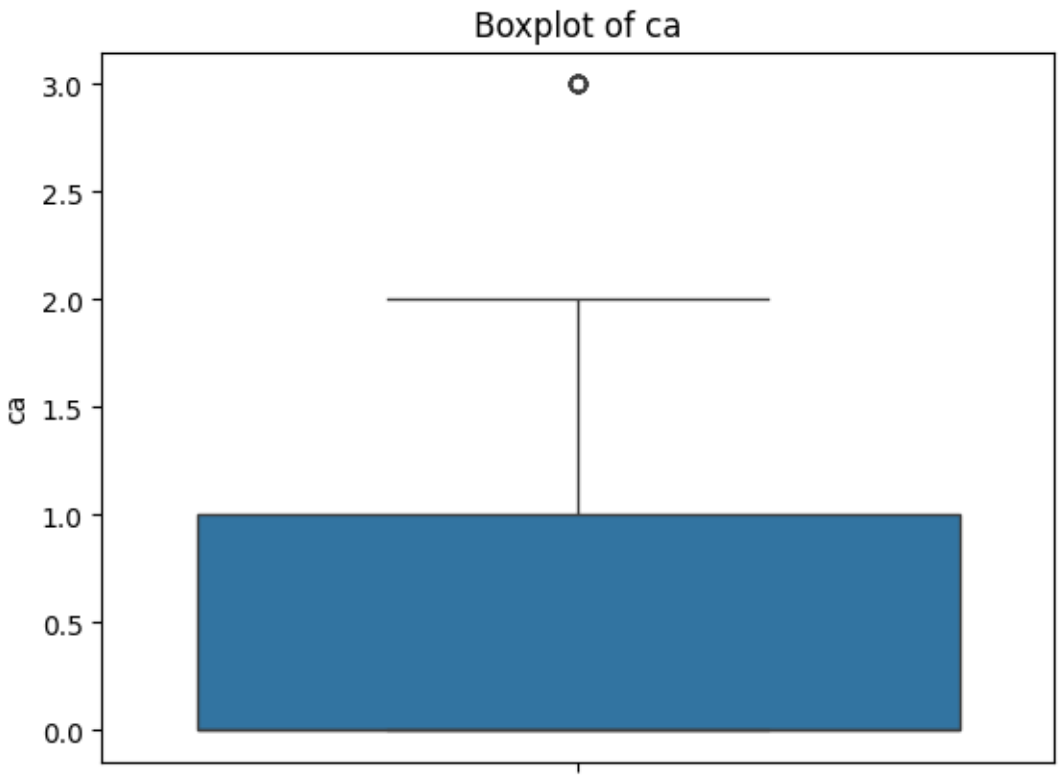



Boxplot of trestbps
trestbps
200
180
160
140
120
100


Boxplot of chol
chol
500
400
300
200

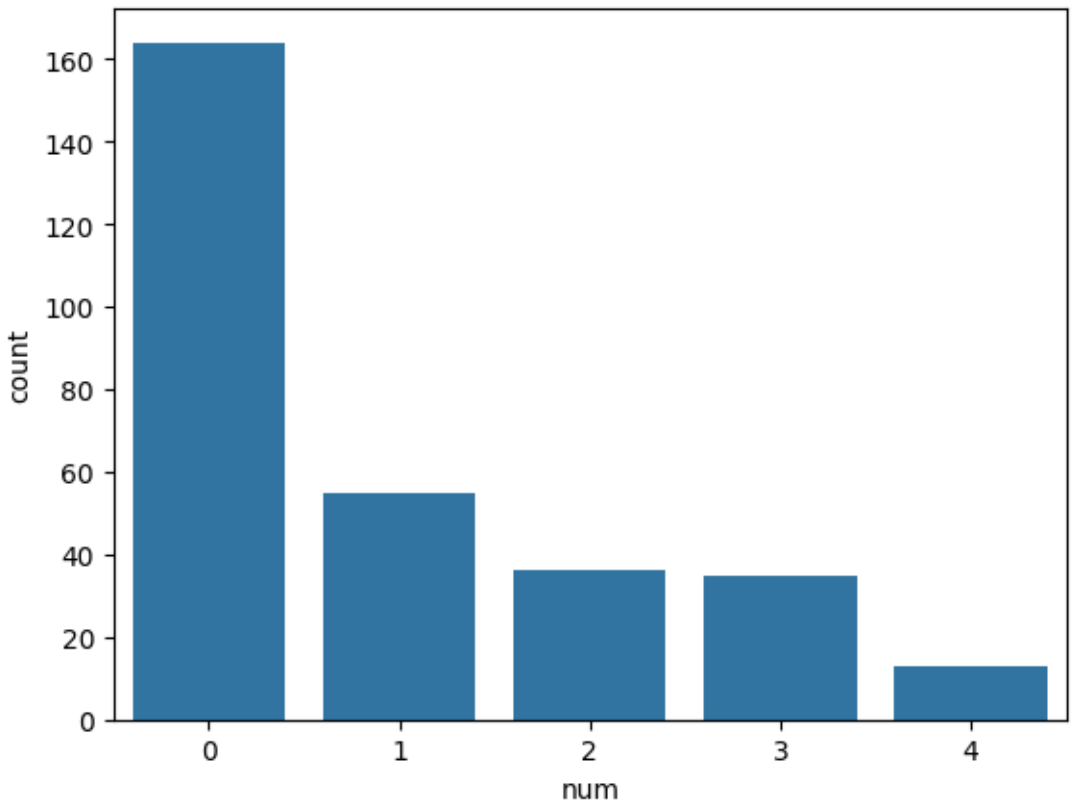

Boxplot of thalach
thalach
200
180
160
140
120
100
80


Boxplot of oldpeak
oldpeak
6
5
4
3
2
1
0



From these boxplots, we can see that there are a few outliers in the dataset. For example, the `chol` variable has a few outliers with very high values. For this assignment, we are told to skip the outlier analysis so we will not be handling or removing them.

```
#check for balancedness in the target variable
sns.countplot(x='num', data=data)
```




```
data['num'].value_counts()
```

```
num
0    164
1     55
2     36
3     35
4     13
Name: count, dtype: int64
```

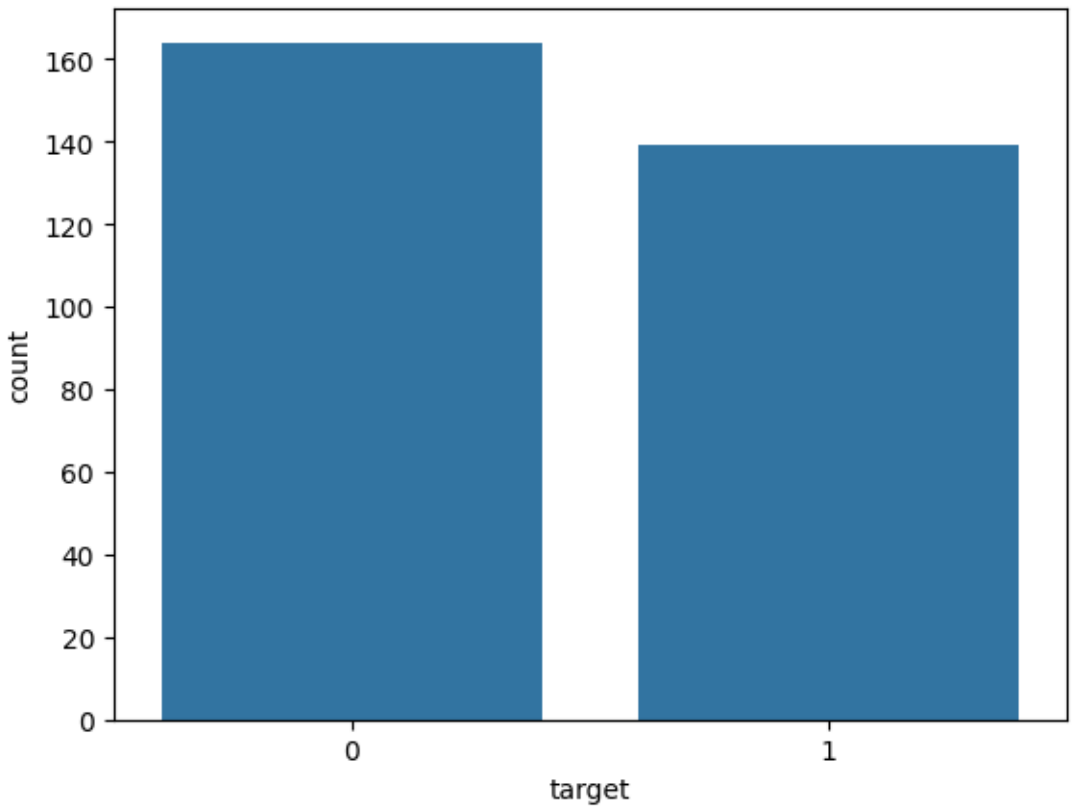
We can see that our data is not balanced. Class 0 has significantly more observations than the other classes. The data is right-skewed however since later we are combining the `num` values 1-4 into one category, this balancedness will look better and we can check again for balancedness after the transformation.

```
#make our target variable binary
# 0 = no heart disease, 1 = heart disease
data['target'] = 0
data.loc[data['num'] > 0, 'target'] = 1
data.drop('num', axis=1, inplace=True)
print(data['target'].value_counts())

#convert the target variable to categorical
data['target'] = pd.Categorical(data['target'])
data.head()
```

```
target
0    164
1    139
Name: count, dtype: int64
```

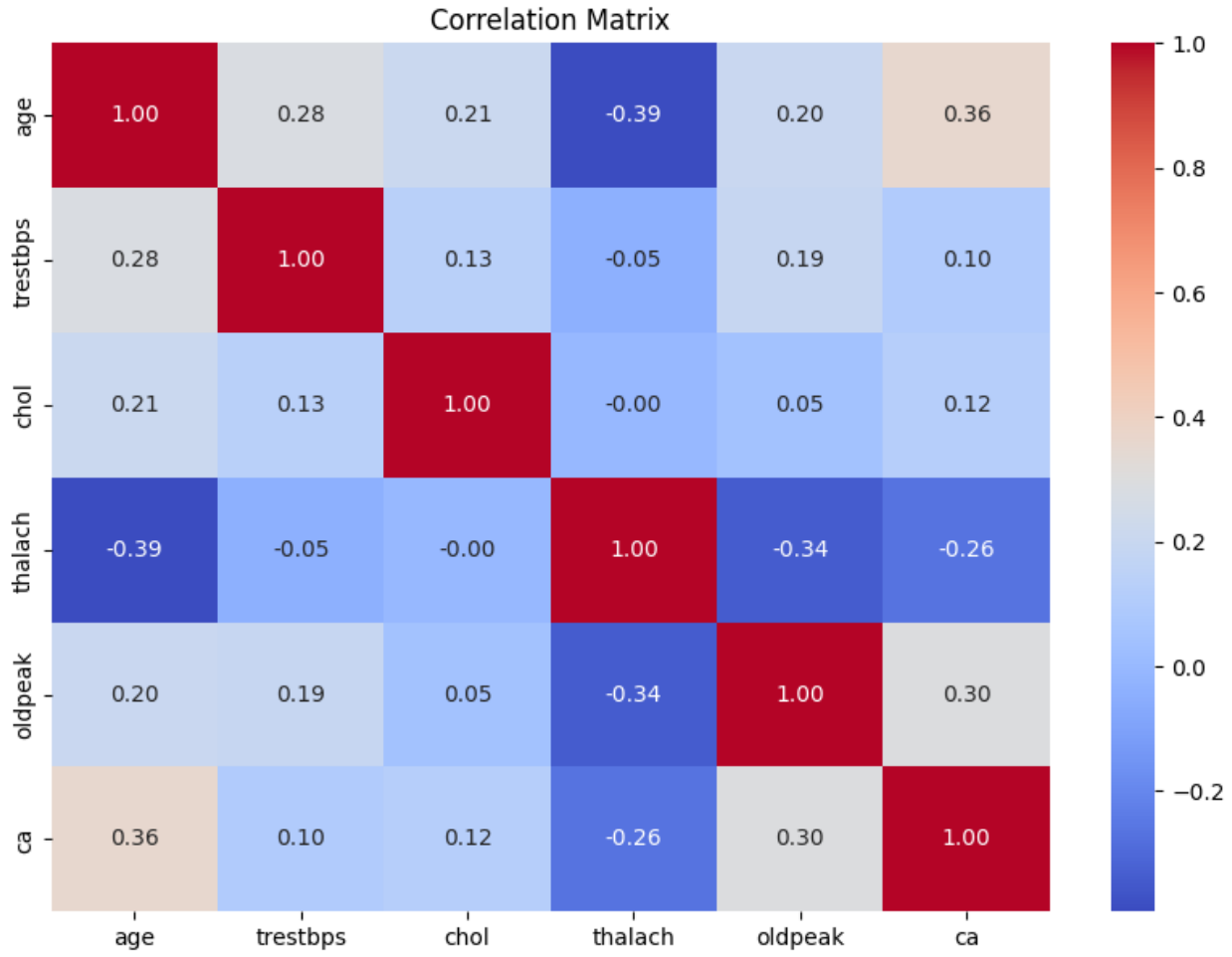
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	1	145	233	1	2	150	0	2.3	3	0.0	6.0	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3.0	3.0	1
2	67	1	4	120	229	0	2	129	1	2.6	2	2.0	7.0	1
3	37	1	3	130	250	0	0	187	0	3.5	3	0.0	3.0	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0.0	3.0	0

```
#check for balancedness in the target variable
sns.countplot(x='target', data=data)
```



After transforming our target variable, the data is much more balanced so we do not need to apply any transformations for balancing.

```
#create a correlation matrix to visualize relationships between numerical features
corr_matrix = data.select_dtypes(include=[np.number]).corr()


# Plot heatmap
plt.figure(figsize=(10, 7))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```



From this correlation matrix, we see that `thalach` and `age` seem to be negatively correlated. This means that as age increases, the maximum heart rate decreases. This is a common finding in heart disease studies, as older individuals tend to have lower maximum heart rates.

`Age` and `ca` have a moderate positive correlation, which means that as age increases, the number of major vessels colored by fluoroscopy also increases. This is also a common finding in heart disease studies, as older individuals tend to have more severe heart disease.

`Thalach` and `oldpeak` are negatively correlated which means that higher maximum heart rate is associated with lower ST depression after exercise.

`Chol` or cholestrol on the other hand seems to have low correlation with all others

```
#summary statistics
data.describe()
```

	age	trestbps	chol	thalach	oldpeak	ca
count	303.000000	303.000000	303.000000	303.000000	303.000000	299.000000
mean	54.438944	131.689769	246.693069	149.607261	1.039604	0.672241
std	9.038662	17.599748	51.776918	22.875003	1.161075	0.937438
min	29.000000	94.000000	126.000000	71.000000	0.000000	0.000000
25%	48.000000	120.000000	211.000000	133.500000	0.000000	0.000000
50%	56.000000	130.000000	241.000000	153.000000	0.800000	0.000000
75%	61.000000	140.000000	275.000000	166.000000	1.600000	1.000000
max	77.000000	200.000000	564.000000	202.000000	6.200000	3.000000

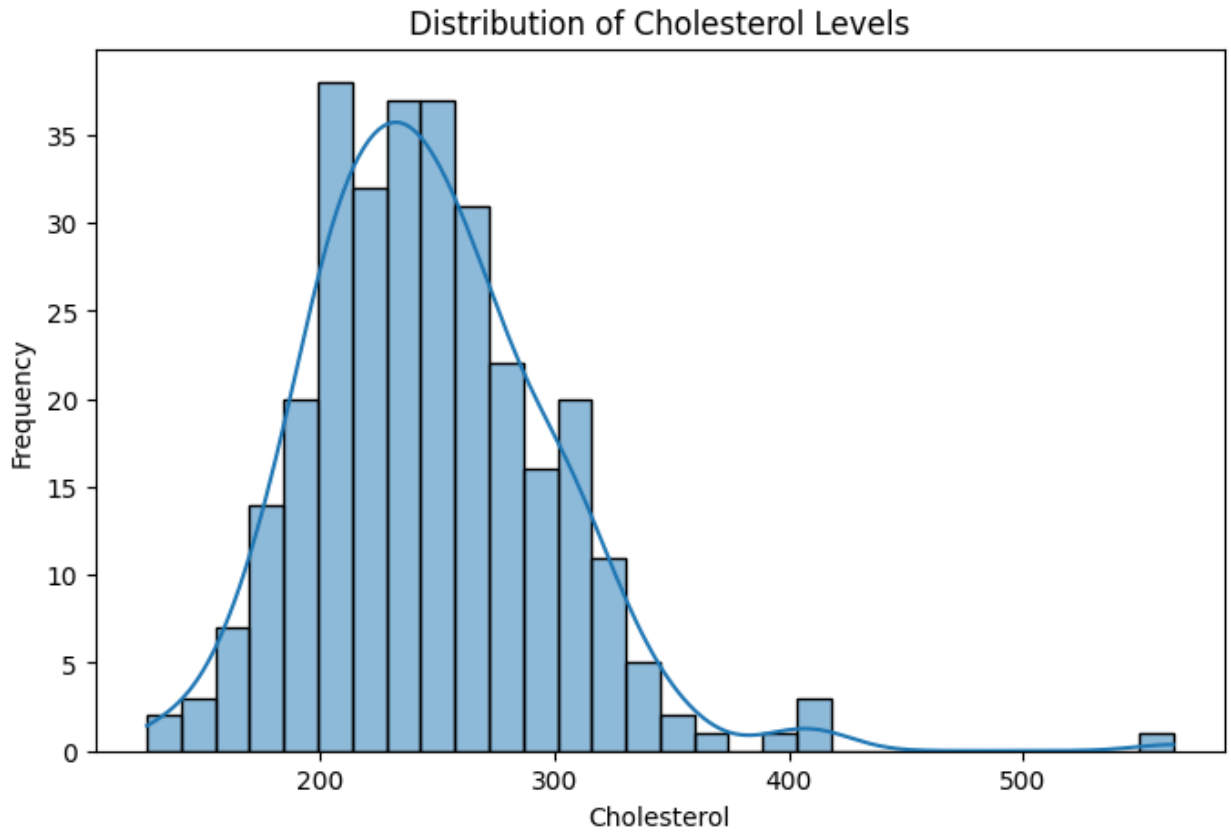
From the summary statistics, we can observe that the average age of patients in this data set is around 54 years, with a standard deviation of 9.08 years. The minimum age is 29 years, and the maximum age is 77 years.

We also see that the minimum of 0 in `oldpeak` and `ca` but these are both valid since a 0 makes sense in both of those cases.

The max value of `chol` is 564 which is quite high and could be an outlier. The standard deviation of `chol` is also quite high at 51.77 which indicates that there is a lot of variability in the cholesterol levels of patients in this data set.

There are also 299 observations with recorded `ca` instead of 303 which means that there are 4 missing values in this variable. We will need to handle these missing values later.

```
#plot the cholestrol values
plt.figure(figsize=(8, 5))
sns.histplot(data['chol'], bins=30, kde=True)
plt.title('Distribution of Cholesterol Levels')
plt.xlabel('Cholesterol')
plt.ylabel('Frequency')
plt.show()
```



We see that the distribution of the cholestrol levels of patients is right-skewed, with a few outliers on the right side of the distribution. This is consistent with the summary statistics, which show that the mean cholestrol level is higher than the median cholestrol level.

A log-transformation could be applied to the `chol` variable to make it more normally distributed.

```
#explore the categorical predictors.
cat_columns = data.select_dtypes(
    include=['category', 'object']
    ).columns

# print the summary of categorical columns
for col in cat_columns:
    print(data[col].value_counts(normalize=True))
```

```
sex
```

```
1    0.679868
0    0.320132
Name: proportion, dtype: float64
cp
4    0.475248
3    0.283828
2    0.165017
1    0.075908
Name: proportion, dtype: float64
fbs
0    0.851485
1    0.148515
Name: proportion, dtype: float64
restecg
0    0.498350
2    0.488449
1    0.013201
Name: proportion, dtype: float64
exang
0    0.673267
1    0.326733
Name: proportion, dtype: float64
slope
1    0.468647
2    0.462046
3    0.069307
Name: proportion, dtype: float64
thal
3.0    0.551495
7.0    0.388704
6.0    0.059801
Name: proportion, dtype: float64
```

```
target
0    0.541254
1    0.458746
Name: proportion, dtype: float64
```

```
#check for missing values
data.isnull().sum()
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          4
thal        2
target      0
dtype: int64
```

There are a total of 6 missing values. We will remove the rows that have missing values:

```
#drop rows with missing values
data.dropna(inplace=True)
data.isnull().sum().sum()
```

```
np.int64(0)
```

```
#check the shape of the dataset after dropping missing values
data.shape
```

```
(297, 14)
```

We can see that after dropping, the sum of the missing values is 0 and we now have 297 observationg so we have successfully deleted the rows with missing values.

```
#prepare dataset for clustering
clust_df = data.select_dtypes(include=[np.number])
clust_df.head()
```

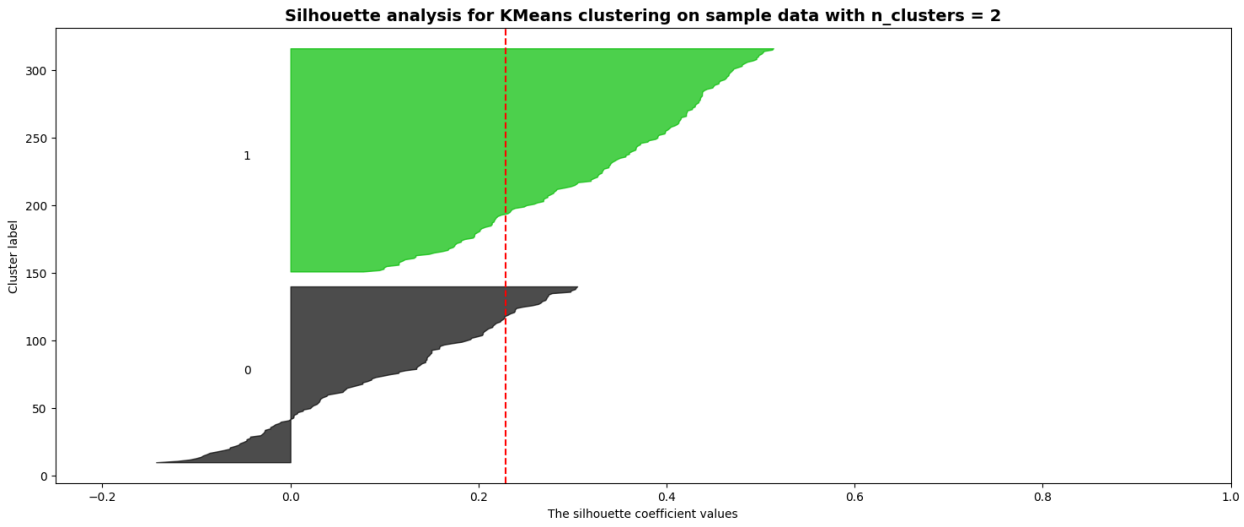
	age	trestbps	chol	thalach	oldpeak	ca
0	63	145	233	150	2.3	0.0
1	67	160	286	108	1.5	3.0
2	67	120	229	129	2.6	2.0
3	37	130	250	187	3.5	0.0
4	41	130	204	172	1.4	0.0

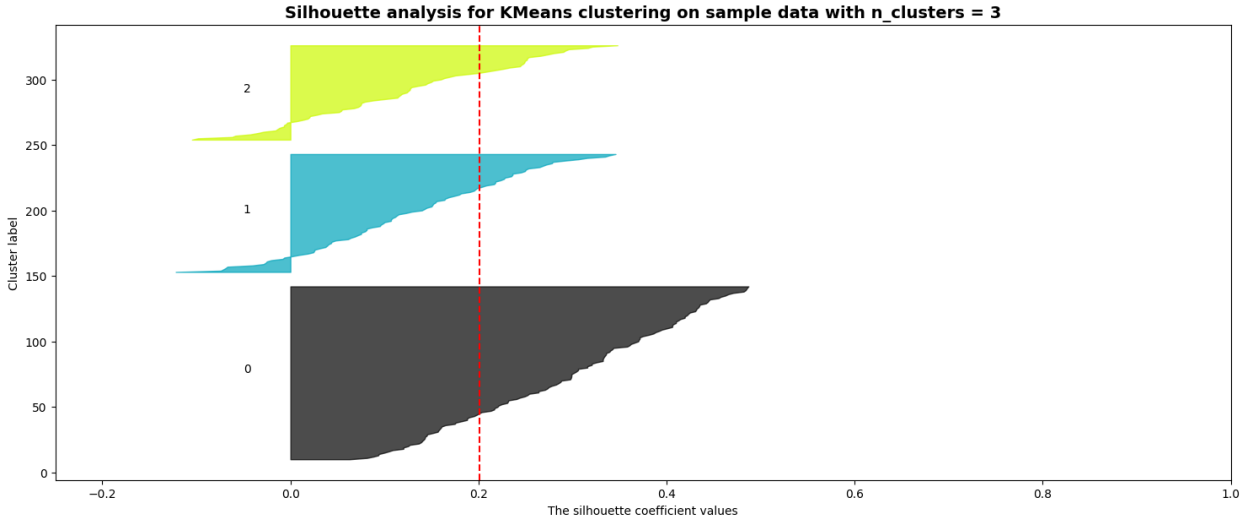
```
#standardize
scaler = StandardScaler()
clust_df_scaled = scaler.fit_transform(clust_df)
clust_df_scaled = pd.DataFrame(clust_df_scaled, columns=clust_df.columns)
print(clust_df_scaled.head())
```

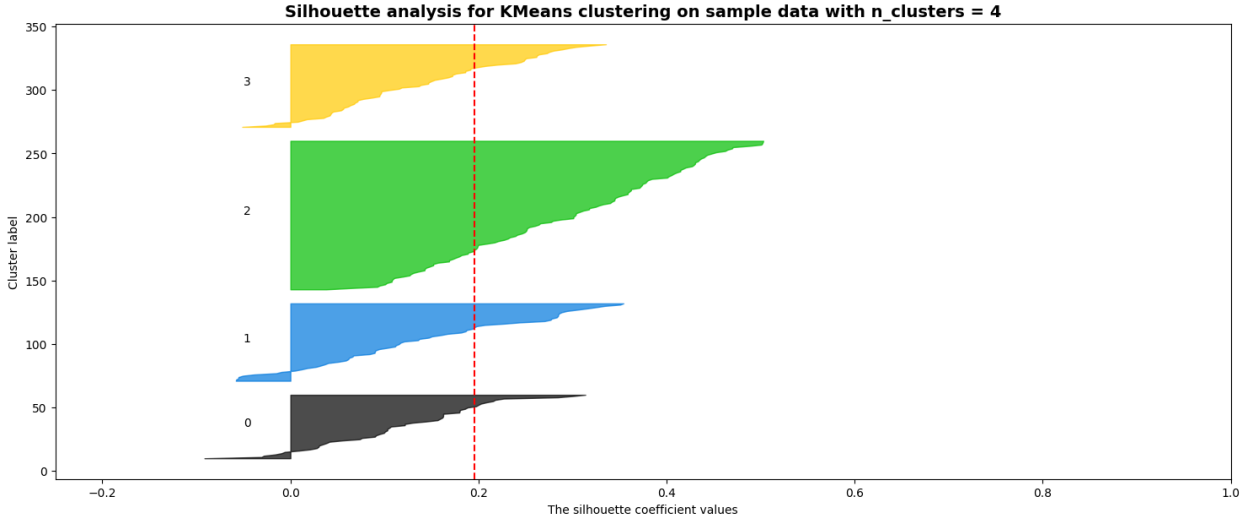
```
        age  trestbps      chol   thalach   oldpeak        ca
0  0.936181  0.750380 -0.276443  0.017494  1.068965 -0.721976
1  1.378929  1.596266  0.744555 -1.816334  0.381773  2.478425
2  1.378929 -0.659431 -0.353500 -0.899420  1.326662  1.411625
3 -1.941680 -0.095506  0.051047  1.633010  2.099753 -0.721976
4 -1.498933 -0.095506 -0.835103  0.978071  0.295874 -0.721976
```

```
#apply k-means clustering
range_n_clusters = range(2,5)
for n_clusters in range_n_clusters:
    km = KMeans(n_clusters = n_clusters, n_init = 20, random_state=0)
    cluster_labels_km = km.fit_predict(clust_df_scaled)
    silhouette_avg_km = silhouette_score(clust_df_scaled, cluster_labels_km)
    # compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(clust_df_scaled, cluster_labels_km)
    fig, ax1 = plt.subplots(1, 1)
    fig.set_size_inches(18, 7)
    ax1.set_xlim([-0.25, 1])# change this based on the silhouette range
    y_lower = 10

    for i in range(n_clusters):
        # aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels_km == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(
            np.arange(y_lower, y_upper),
            0,
            ith_cluster_silhouette_values,
            facecolor=color,
            edgecolor=color,
            alpha=0.7,
        )
```

```
        # label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # compute the new y_lower for next plot
        y_lower = y_upper + 10

    ax1.set_title("The silhouette plot for various cluster")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg_km, color="red", linestyle="--")
    plt.title(
        "Silhouette analysis for KMeans clustering on sample data with n_clusters = %d"
        % n_clusters,
        fontsize=14,
        fontweight="bold",
    )
```

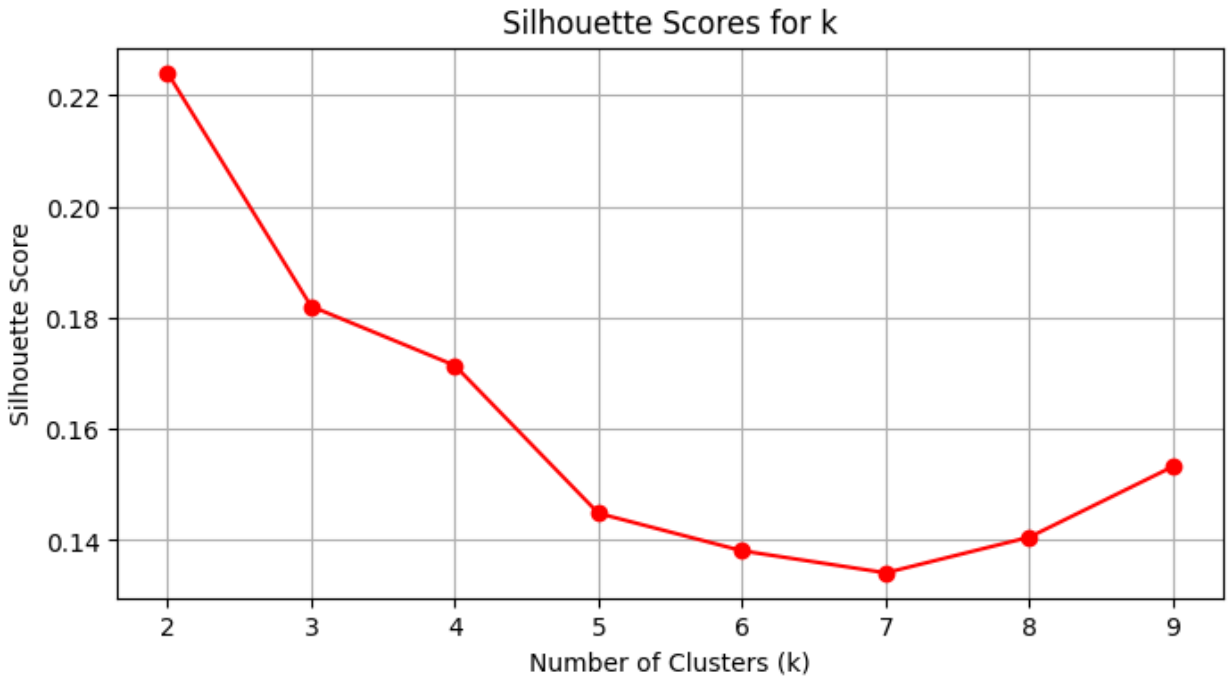







```
from sklearn.metrics import silhouette_score

silhouette_scores = []

for k in range(2, 10):
    kmeans = KMeans(n_clusters=k, random_state=42)
    preds = kmeans.fit_predict(clust_df_scaled)
    score = silhouette_score(clust_df_scaled, preds)
    silhouette_scores.append(score)

plt.figure(figsize=(8, 4))
```

```
plt.plot(range(2, 10), silhouette_scores, 'ro-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Scores for k')
plt.grid(True)
plt.show()
```



Using the silhouette score, we can see that the best number of clusters is 2. However, even the silhouette score for 2 clusters is less than 0.25. This means that the quality of our clusters is not great and it is not worth clustering. So instead, we can try to use PC Components to reduce the dimensionality.

```
#apply PCA
pca_X = PCA()
pca_X.fit(clust_df_scaled)
```

```
PCA()
```

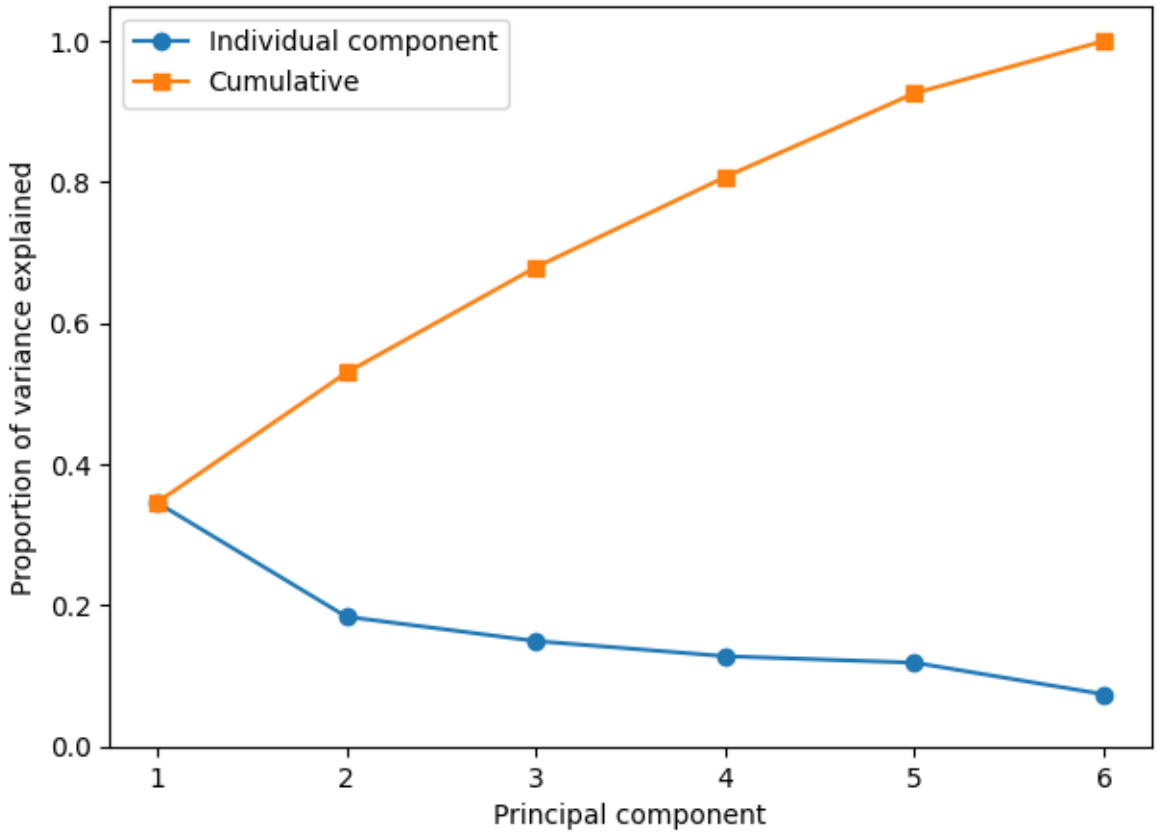
```
pca_X.components_.shape[0]
```

6

```
# Let's put the PC loading into a data frame.
pca_loadings = pd.DataFrame(
    pca_X.fit(clust_df_scaled).components_.T,
    index=clust_df_scaled.columns,
    columns=[f'PC{i+1}' for i in range(pca_X.n_components_)]
)
```

```
# PC scores in a data frame
pc_scores = pd.DataFrame(
    pca_X.fit_transform(clust_df_scaled),
    index=clust_df_scaled.index,
    columns=[f'PC{i+1}' for i in range(pca_X.n_components_)]
    )
```

```
num_components = pca_X.n_components_
pc_indices = np.arange(1, num_components + 1)

plt.figure(figsize=(7,5))

plt.plot(pc_indices, pca_X.explained_variance_ratio_, '-o', label='Individual component')
plt.plot(pc_indices, np.cumsum(pca_X.explained_variance_ratio_), '-s', label='Cumulative')

plt.ylabel('Proportion of variance explained')
plt.xlabel('Principal component')
plt.xlim(0.75, num_components + 0.25)
plt.ylim(0, 1.05)
plt.xticks(pc_indices)
plt.legend(loc=2)
plt.show()
```



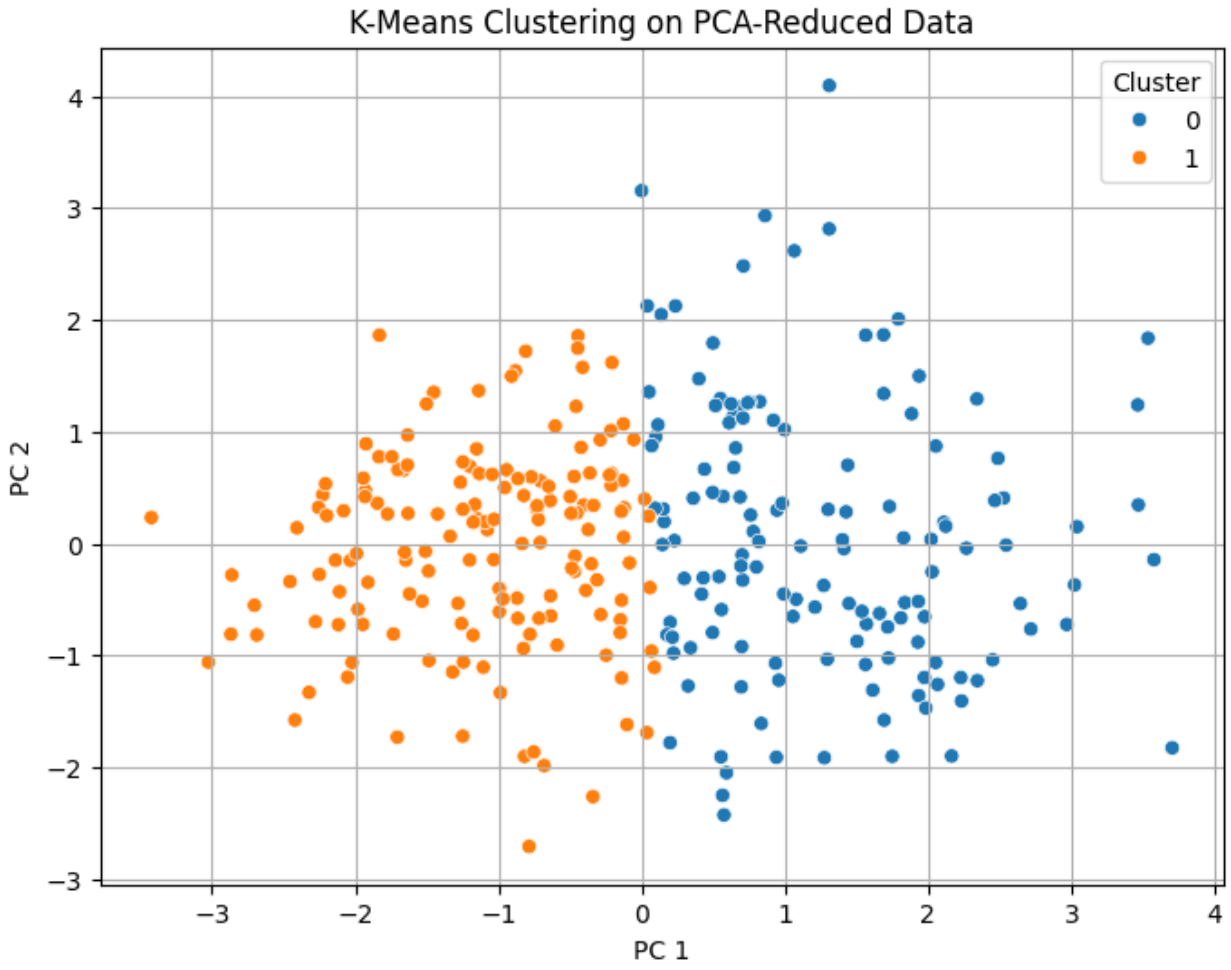
Using the elbow rule, we select 3 principal components since the variance explained by the remaining PCs increases slowly after that.

```
#subset the data using the top 3 PCs
pc_subset = pc_scores[['PC1', 'PC2', 'PC3']].copy()
```

Since k=2 gave us the highest silhouette score, we will now try applying k-means clustering using the PCA-reduced features.

```
kmeans = KMeans(n_clusters=2, random_state=1)
clusters = kmeans.fit_predict(pc_subset)


# add the cluster labels to the PC scores
pc_subset['Cluster'] = clusters
```

```
#visualize
plt.figure(figsize=(8,6))
sns.scatterplot(
    data=pc_subset, x='PC1', y='PC2', hue='Cluster'
)
plt.title('K-Means Clustering on PCA-Reduced Data')
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(title='Cluster')
plt.grid(True)
plt.show()
```



ALthough we performed clustering using all 3 PCs, we visualize the clusters in 2D using the first

two principal components for simplification prposes so we can view the clusters. We can see that the clusters are not very well separated which was indicated to us by the low silhouette score earlier.

Since both my chosen classifiers require standardization, I will standardize the data before splitting it into train and test sets.

Before doing that, I will create dummy variables for the features `cp`, `restecg`, `slope`, and `thal` since they are categorical variables with more than two levels for which there is no order.

```
# create dummy variables for cp, restecg, slope, and thal
encoded_data = pd.get_dummies(data, columns=['cp', 'restecg', 'slope', 'thal'], drop_first=True
encoded_data
```

	age	sex	trestbps	chol	fbs	thalach	exang	oldpeak	ca	target	cp_2	cp_3	cp_4	restecg_
0	63	1	145	233	1	150	0	2.3	0.0	0	False	False	False	False
1	67	1	160	286	0	108	1	1.5	3.0	1	False	False	True	False
2	67	1	120	229	0	129	1	2.6	2.0	1	False	False	True	False
3	37	1	130	250	0	187	0	3.5	0.0	0	False	True	False	False
4	41	0	130	204	0	172	0	1.4	0.0	0	True	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
297	57	0	140	241	0	123	1	0.2	0.0	1	False	False	True	False
298	45	1	110	264	0	132	0	1.2	0.0	1	False	False	False	False
299	68	1	144	193	1	141	0	3.4	2.0	1	False	False	True	False
300	57	1	130	131	0	115	1	1.2	1.0	1	False	False	True	False
301	57	0	130	236	0	174	0	0.0	1.0	1	True	False	False	False

```
X = encoded_data.drop(columns=['target'])
y = encoded_data['target']


#standardize the data


#since there are categorical variables,
# we will only standardize the numerical variables
```

```
numerical = X.select_dtypes(include=[np.number]).columns
X_std = X.copy()
scale = StandardScaler()
X_std[numerical] = scale.fit_transform(X[numerical])



#split the data
X_train, X_test, y_train, y_test = train_test_split(
    X_std, y, test_size=0.30, random_state=1, stratify=y
)
```

Logistic Regression:

```
# use cv to find the best C for logistic regression
log = LogisticRegressionCV(
    Cs = [0.001, 0.01, 0.1, 1, 10, 100],
    cv = 5,
    random_state = 1,
    max_iter = 1000,
    solver = 'liblinear',
    scoring= 'accuracy'
)
log.fit(X_train, y_train)
# print the best C
```

```
LogisticRegressionCV(Cs=[0.001, 0.01, 0.1, 1, 10, 100], cv=5, max_iter=1000,
                     random_state=1, scoring='accuracy', solver='liblinear')
```

```
print(f'Best C: {log.C_}')
```

```
Best C: [1.]
```

```
pred_prob1 = log.predict_proba(X_test)
pred_prob1[0:2]
```

```
array([[0.13598176, 0.86401824],
       [0.05213506, 0.94786494]])
```

Here, the second column is the predicted probability of a patient having heart disease.

```
y_pred = log.predict(X_test)
y_pred_proba = log.predict_proba(X_test)[:, 1]
accuracy_score(y_test, y_pred)
```

```
0.8555555555555555
```

```
# create a data frame with predicted probabilities and actual labels
df_log = pd.DataFrame(
    data = {'prob1': pred_prob1[:,1], 'y_test': y_test}
    )
df_log.head(5)
```

	prob1	y_test
95	0.864018	1
299	0.947865	1
221	0.016439	0
210	0.016938	0
43	0.066449	0

This dataframe will be used to calculate metrics later.

SVM:

```
# Use CV on the training set to tune gamma and nu
clf_w = svm.SVC(kernel="rbf", gamma="auto", C=10)

param_grid = {
    'C': [0.1, 1, 10, 100, 1000],
    'gamma': ['scale', 'auto',0.001, 0.01, 0.1],
    'kernel': ['rbf', 'poly', 'sigmoid'],
}

grid_search_w = GridSearchCV(
    estimator=clf_w,
    param_grid=param_grid,
    cv=5,
    scoring='accuracy'
)

grid_search_w.fit(X_train, y_train)

best_params = grid_search_w.best_params_
```

```
best_params
```

```
{'C': 1, 'gamma': 'auto', 'kernel': 'sigmoid'}
```

```
clf_w = svm.SVC(kernel=best_params['kernel'],gamma=best_params['gamma'], C=best_params['C'],pr
clf_w.fit(X_train, y_train)
pred = clf_w.predict(X_test)
accuracy_score(y_test, pred)
```

```
0.8333333333333334
```

```
#create dataframe with prediction probabilities
pred_prob_svm = clf_w.predict_proba(X_test)



df_svm = pd.DataFrame(
    data = {'prob1': pred_prob_svm[:,1], 'y_test': y_test}
    )
df_svm.head(5)
```

	prob1	y_test
95	0.838500	1
299	0.953318	1
221	0.023085	0
210	0.029920	0
43	0.091915	0

Feature Selection:

I will now apply feature selection to the Logistic Regression classifier by using L1 regularization (Lasso). L1 regularization adds a penalty equal to the absolute value of the magnitude of coefficients to the loss function. This encourages sparsity in the model, effectively performing feature selection by shrinking some coefficients to zero.

```
m_lasso = LogisticRegressionCV(
    penalty='l1',
    solver='saga',
    cv=5,
    max_iter=10000,
    scoring='accuracy'
)
m_lasso.fit(X_train, y_train)
print(1/m_lasso.C_[0])
```

```
2.782559402207126
```

```
m_lasso_pre = m_lasso.predict(X_test)
accuracy_score(y_test, m_lasso_pre)
```

```
0.8555555555555555
```

```
coef_df = pd.DataFrame({'Feature': X_train.columns, 'Coefficient': m_lasso.coef_.reshape(len(X
coef_df
```

	Feature	Coefficient
0	age	0.000000
1	sex	0.974805
2	trestbps	0.096042
3	chol	0.161006
4	fbs	0.000000
5	thalach	-0.395775
6	exang	0.476332
7	oldpeak	0.277075
8	ca	0.572020
9	cp_2	0.000000
10	cp_3	0.000000
11	cp_4	1.217249
12	restecg_1	0.000000
13	restecg_2	0.000000
14	slope_2	0.289797
15	slope_3	0.000000
16	thal_6.0	0.000000
17	thal_7.0	1.275906

We see that after applying Lasso, some of the coefficients are shrunk to zero, which means they are not important for the classification task.

We will select the non-zero coefficients as the important features for our model:

```
selected_features = X_train.columns[(coef_df['Coefficient'] != 0)]
print("Selected Features:", selected_features)
```

```
Selected Features: Index(['sex', 'trestbps', 'chol', 'thalach', 'exang', 'oldpeak', 'ca', 'cp_4
       'slope_2', 'thal_7.0'],
      dtype='object')
```

```
#create a dataframe for probabilities
m_lasso_pre_prob = m_lasso.predict_proba(X_test)
df_lasso = pd.DataFrame(
    data = {'prob1': m_lasso_pre_prob[:, 1], 'y_test': y_test}
    )
df_lasso.head(5)
```

	prob1	y_test
95	0.812898	1
299	0.949876	1
221	0.036957	0
210	0.032031	0
43	0.157752	0

```
#train new classifier with selected features
X_train_sel = X_train[selected_features]
X_test_sel = X_test[selected_features]

selected_log = LogisticRegression()
selected_log.fit(X_train_sel, y_train)
pred3 = selected_log.predict(X_test_sel)
accuracy_score(y_test, pred3)
```

```
0.8444444444444444
```

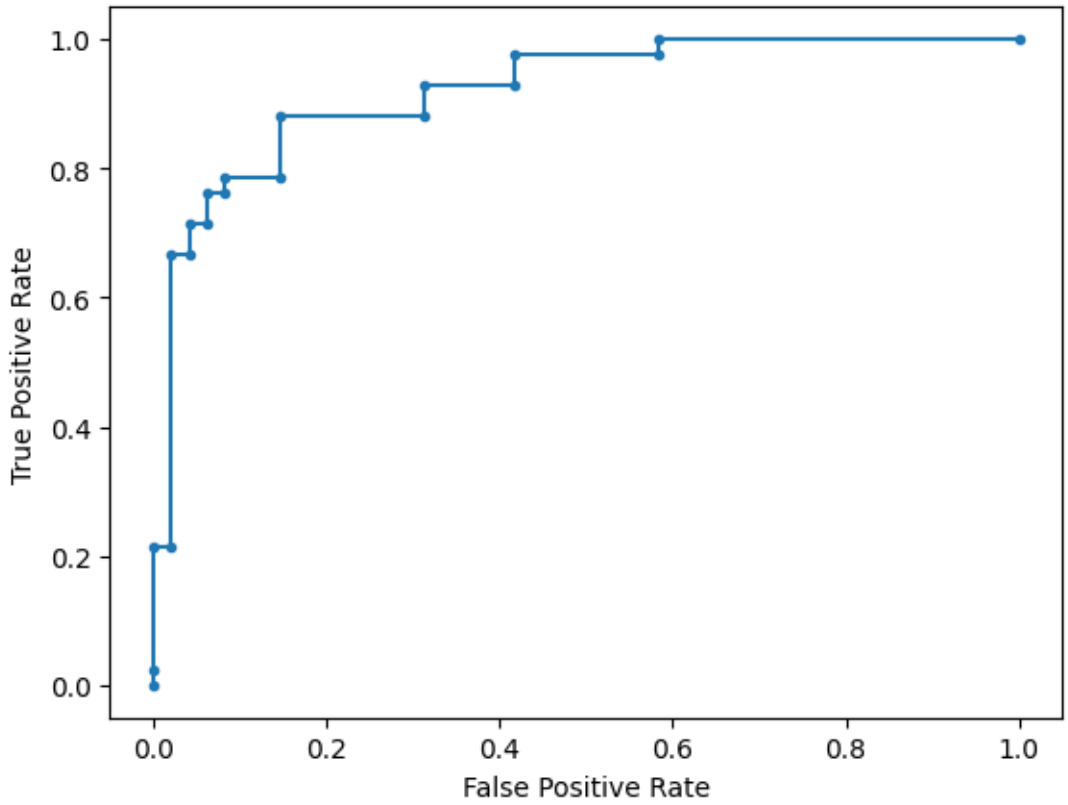
```
#create a dataframe with prediction probabilities
reduced_df = pd.DataFrame(
    data = {'prob1': selected_log.predict_proba(X_test_sel)[:,1], 'y_test': y_test}
    )
reduced_df.head(5)
```

	prob1	y_test
95	0.855555	1
299	0.969028	1
221	0.015076	0
210	0.012120	0
43	0.122104	0

Evaluate Classifier 1: Logistic Regression

```
# calculate false positive rate and true positive rate
log_fpr, log_tpr, thresholds = roc_curve(df_log.y_test, df_log.prob1)
```

```
# plot the roc curve for the model
plt.plot(log_fpr, log_tpr, marker='.', label='Logistic')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```




```
# calculate the AUC for the logistic regression with all predictors
log_auc = roc_auc_score(df_log.y_test, df_log.prob1)
log_auc
```

```
np.float64(0.9211309523809523)
```

The AUC of the ROC curve is 0.921 which indicates that the model has a good performance in distinguishing between patients with and without heart disease.

```
#Youden's J statistic
j_statistic = log_tpr - log_fpr
optimal_index = np.argmax(j_statistic)
optimal_threshold = thresholds[optimal_index]
optimal_threshold
```

```
np.float64(0.4210101398082611)
```

```
# K-S statistic
ks_threshold = thresholds[np.argmax(log_tpr - log_fpr)]
ks_threshold
```

```
np.float64(0.4210101398082611)
```

Both the Youden's J statistic and K-S statistic suggest a threshold of 0.42, which means that if the predicted probability of heart disease is greater than 0.42, we classify the patient as having heart disease.

```
# let's predict the labels using the optimal threshold
df_log['y_pred'] = df_log.prob1.apply(lambda x: 1 if x > optimal_threshold else 0)
```

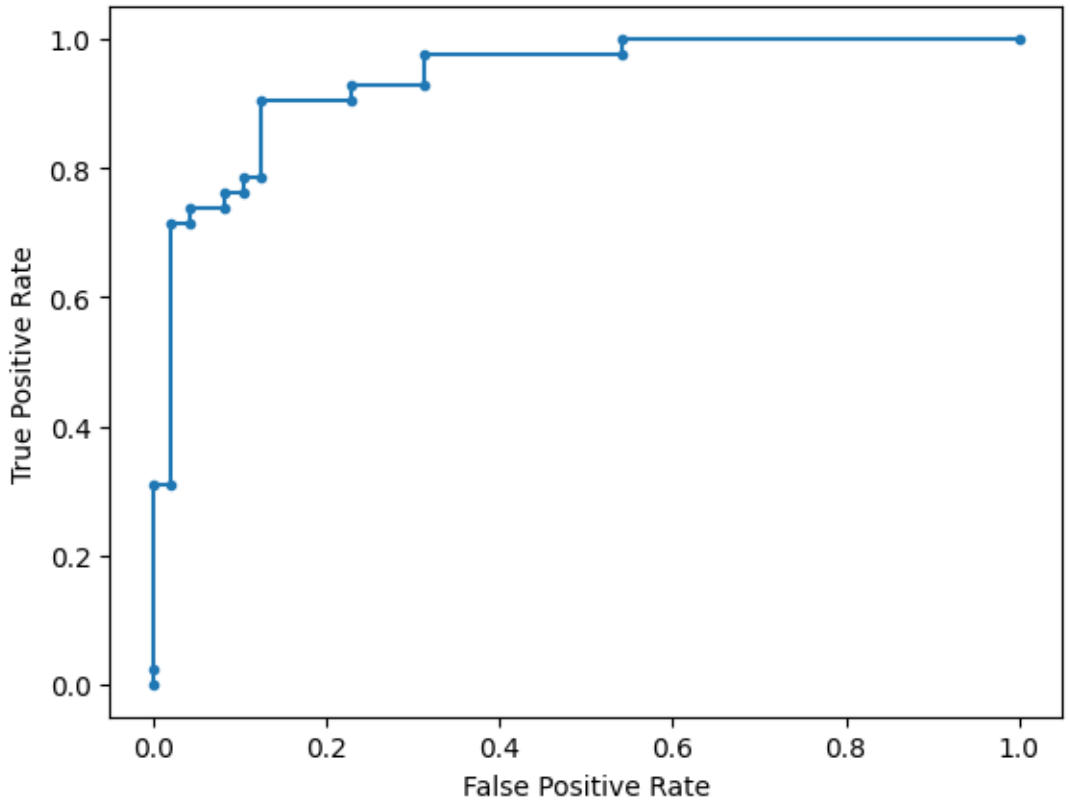
```
cm1 = confusion_matrix(df_log.y_test, df_log.y_pred)

total1 = sum(sum(cm1))

accuracy1 = (cm1[0,0]+cm1[1,1])/total1
print ('Accuracy : ', accuracy1)

sensitivity1 = cm1[1,1]/(cm1[1,0]+cm1[1,1])
print('Sensitivity : ', sensitivity1 )

specificity1 = cm1[0,0]/(cm1[0,0]+cm1[0,1])
print('Specificity : ', specificity1)
```

```
Accuracy :  0.8555555555555555
Sensitivity :  0.8571428571428571
Specificity :  0.8541666666666666
```

Evaluate Classifier 2: SVM

```
#calculate false positive rate and true positive rate
svm_fpr, svm_tpr, thresholds = roc_curve(df_svm.y_test, df_svm.prob1)
```

```
#plot the roc curve for the model
plt.plot(svm_fpr, svm_tpr, marker='.', label='SVM')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```




```
#calculate the AUC for the SVM
svm_auc = roc_auc_score(df_svm.y_test, df_svm.prob1)
svm_auc
```

```
np.float64(0.9379960317460317)
```

The AUC of the ROC curve for the SVM model is 0.937 which is means that the model has a good performance in distinguishing between patients with and without heart disease.

```
#Youden's J statistic
j_statistic = svm_tpr - svm_fpr
optimal_index = np.argmax(j_statistic)
optimal_threshold = thresholds[optimal_index]
optimal_threshold
```

```
np.float64(0.3677384459628099)
```

```
# K-S statistic
ks_threshold = thresholds[np.argmax(svm_tpr - svm_fpr)]
ks_threshold
```

```
np.float64(0.3677384459628099)
```

Both the Youden's J statistic and K-S statistic give around the same threshold of 0.36. This means that if the predicted probability of heart disease is greater than 0.36, we classify the patient as having heart disease.

```
# let's predict the labels using the optimal threshold
df_svm['y_pred'] = df_svm.prob1.apply(lambda x: 1 if x > optimal_threshold else 0)
```

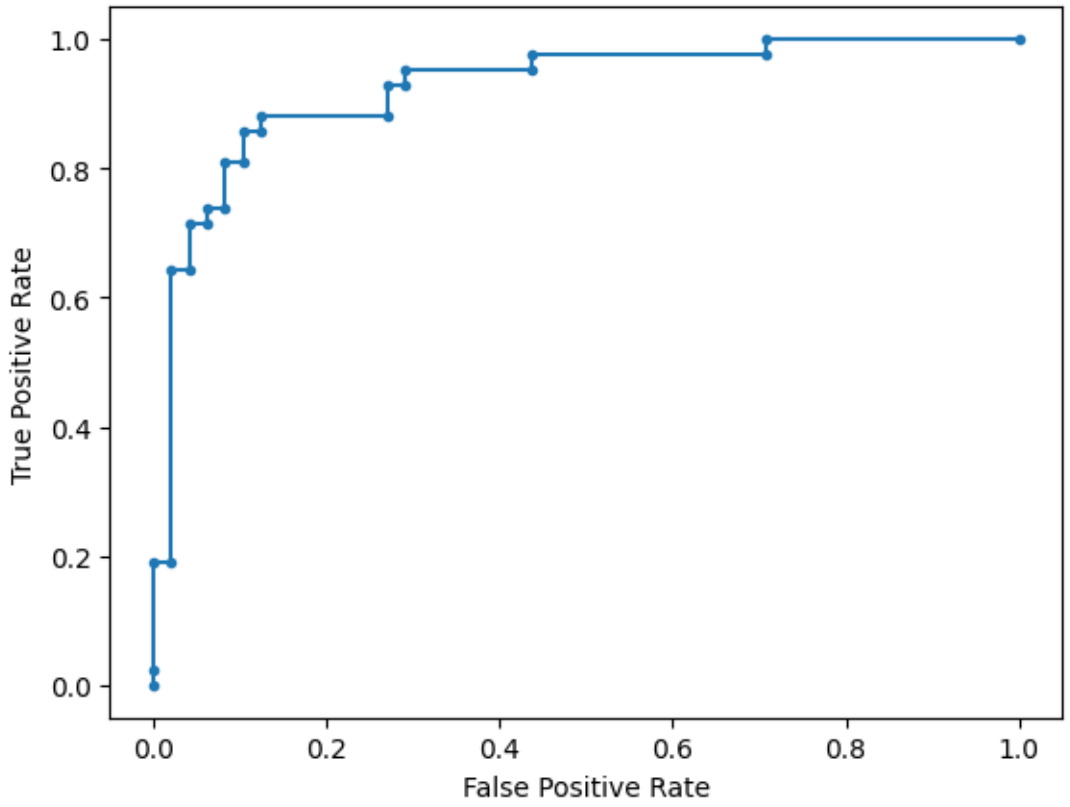
```
cm2 = confusion_matrix(df_svm.y_test, df_svm.y_pred)

total2 = sum(sum(cm2))

accuracy2 = (cm2[0,0]+cm2[1,1])/total2
print ('Accuracy : ', accuracy2)

sensitivity2 = cm2[1,1]/(cm2[1,0]+cm2[1,1])
print('Sensitivity : ', sensitivity2 )

specificity2 = cm2[0,0]/(cm2[0,0]+cm2[0,1])
print('Specificity : ', specificity2)
```

```
Accuracy :  0.8777777777777778
Sensitivity :  0.8809523809523809
Specificity :  0.875
```

Evaluate Classifier 3: Logistic Regression with Lasso

```
#calculate false positive rate and true positive rate
lasso_fpr, lasso_tpr, thresholds = roc_curve(reduced_df.y_test, reduced_df.prob1)
```

```
#plot the roc curve for the model
plt.plot(lasso_fpr, lasso_tpr, marker='.', label='Lasso')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```



```
#calculate the AUC for the lasso
lasso_auc = roc_auc_score(reduced_df.y_test, reduced_df.prob1)
lasso_auc
```

```
np.float64(0.9250992063492064)
```

The AUC of the ROC curve for the logistic regression model after Lasso was applied is 0.925. This is very similar to the AUC of the original logistic regression model, which indicates that the performance of the model has not changed significantly after applying Lasso.

```
#Youden's J statistic
j_statistic = lasso_tpr - lasso_fpr
optimal_index = np.argmax(j_statistic)
optimal_threshold = thresholds[optimal_index]
optimal_threshold
```

```
np.float64(0.38016196605023883)
```

```
# K-S statistic
ks_threshold = thresholds[np.argmax(lasso_tpr - lasso_fpr)]
ks_threshold
```

```
np.float64(0.38016196605023883)
```

Both the Youden's J Statistic and K-S statistic give a threshold of 0.38.

```
# let's predict the labels using the optimal threshold
reduced_df['y_pred'] = reduced_df.prob1.apply(lambda x: 1 if x > optimal_threshold else 0)
```

```
#confusion matrix
cm3 = confusion_matrix(reduced_df.y_test, reduced_df.y_pred)


total3 = sum(sum(cm3))
```

```
accuracy3 = (cm3[0,0]+cm3[1,1])/total3
print ('Accuracy : ', accuracy3)

sensitivity3 = cm3[1,1]/(cm3[1,0]+cm3[1,1])
print('Sensitivity : ', sensitivity3 )

specificity3 = cm3[0,0]/(cm3[0,0]+cm3[0,1])
print('Specificity : ', specificity3)
```

```
Accuracy :  0.8666666666666667
Sensitivity :  0.8571428571428571
Specificity :  0.875
```

Discussing my findings:

To clearly summarize my findings, I will create a dataframe that lists the accuracy, sensitivity, specificity, and AUC for each of the classifiers:

```
results = pd.DataFrame({
    'Classifier': ['Logistic Regression', 'SVM', 'LR with Lasso'],
    'Accuracy': [accuracy1, accuracy2, accuracy3],
    'Sensitivity': [sensitivity1, sensitivity2, sensitivity3],
    'Specificity': [specificity1, specificity2, specificity3],
    'AUC': [log_auc, svm_auc, lasso_auc]
})
results
```

	Classifier	Accuracy	Sensitivity	Specificity	AUC
0	Logistic Regression	0.855556	0.857143	0.854167	0.921131
1	SVM	0.877778	0.880952	0.875000	0.937996
2	LR with Lasso	0.866667	0.857143	0.875000	0.925099

We see that all of the classifiers have almost the same accuracy, with SVM having the highest accuracy.

In a medical setting, misclassifying a patient with heart disease as not having heart disease is more serious than the other way around. Therefore, sensitivity hold a little more importance in this analysis. We see that the Lasso classifier has the highest sensitivity, followed by SVM, and then logistic regression.

The lasso model also has the highest AUC but they are all very close. The SVM model has the lowest AUC, which is still good but not as good as the other two.

Impact of Feature Selection:

The impact feature selection was slightly significat in our model using Lasso since it led to higher accuracy and higher sensitivity. This means that the model is better at identifying patients with heart disease. The Lasso model also has a higher AUC, which indicates that it has a better overall performance in distinguishing between patients with and without heart disease.

The best and most interpretable model was the Logistic Regression Model with **Lasso**

```
features = pd.DataFrame({
        'Feature': X_train.columns,
        'Coefficient': m_lasso.coef_[0]
    }).sort_values(by='Coefficient', ascending=False)


#sort features based on absoute value
features['Absolute Coefficient'] = features['Coefficient'].abs()
features.sort_values(by='Absolute Coefficient', ascending=False, inplace=True)
features
```

	Feature	Coefficient	Absolute Coefficient
17	thal_7.0	1.275906	1.275906
11	cp_4	1.217249	1.217249
1	sex	0.974805	0.974805
8	ca	0.572020	0.572020

	Feature	Coefficient	Absolute Coefficient
6	exang	0.476332	0.476332
5	thalach	-0.395775	0.395775
14	slope_2	0.289797	0.289797
7	oldpeak	0.277075	0.277075
3	chol	0.161006	0.161006
2	trestbps	0.096042	0.096042
0	age	0.000000	0.000000
13	restecg_2	0.000000	0.000000
4	fbs	0.000000	0.000000
9	cp_2	0.000000	0.000000
10	cp_3	0.000000	0.000000
15	slope_3	0.000000	0.000000
12	restecg_1	0.000000	0.000000
16	thal_6.0	0.000000	0.000000

We see that the `thal7.0` and `cp_4` have the highest coefficients which means that they are the most important features in the model. The `thal7.0` variable is a dummy variable that indicates whether the patient has a reversible defect or not. The `cp_4` variable is a dummy variable that indicates whether the patient has asymptomatic chest pain or not. This makes sense since patients with reversible defects and asymptomatic chest pain are more likely to have heart disease.

The sex of the patients is anotherinfluential variable in predicting heart disease since it has a high absolute coefficient. Other influential variables are `exang`, `thalach`, and `ca`.

`Exang` which is exercise induced angina, is highly positively correlated with heart disease, which makes sense since if a patient has angina, they are more likely to have heart disease.

`Thalach` which is the maximum heart rate achieved, is negatively correlated with heart disease, which makes sense since if a patient has a higher maximum heart rate, they are less likely to have heart disease.

`ca` which is the number of major vessels colored by fluoroscopy, is also positively correlated with heart disease, which makes sense since if a patient has more major vessels colored, they are more

likely to have heart disease.

Extension

As an extension, I wanted to explore subgroups in this dataset. The subgroups we saw in question 7 were not very well separated and the silhouette score was low. This means that the clusters are not very meaningful. However, we can still try to interpret the clusters by splitting the dataset by the cluster labels and training another logistic regression model.

```
#to do so, we can use the cluster labels as a new feature


X_clustered = X.copy()
X_clustered['Cluster'] = clusters



#standardize the data
X_clustered_std = X_clustered.copy()
numerical = X_clustered_std.select_dtypes(include=[np.number]).columns
X_clustered_std[numerical] = scale.fit_transform(X_clustered_std[numerical])

#split the data

X_trainC, X_testC, y_trainC, y_testC = train_test_split(
    X_clustered_std, y, test_size=0.30, random_state=1, stratify=y
)
```

```
#train logistic regression
log_clust = LogisticRegression(max_iter=10000)
log_clust.fit(X_trainC, y_trainC)
pred_prob_clust = log_clust.predict_proba(X_testC)
pred_prob_clust[0:2]
```

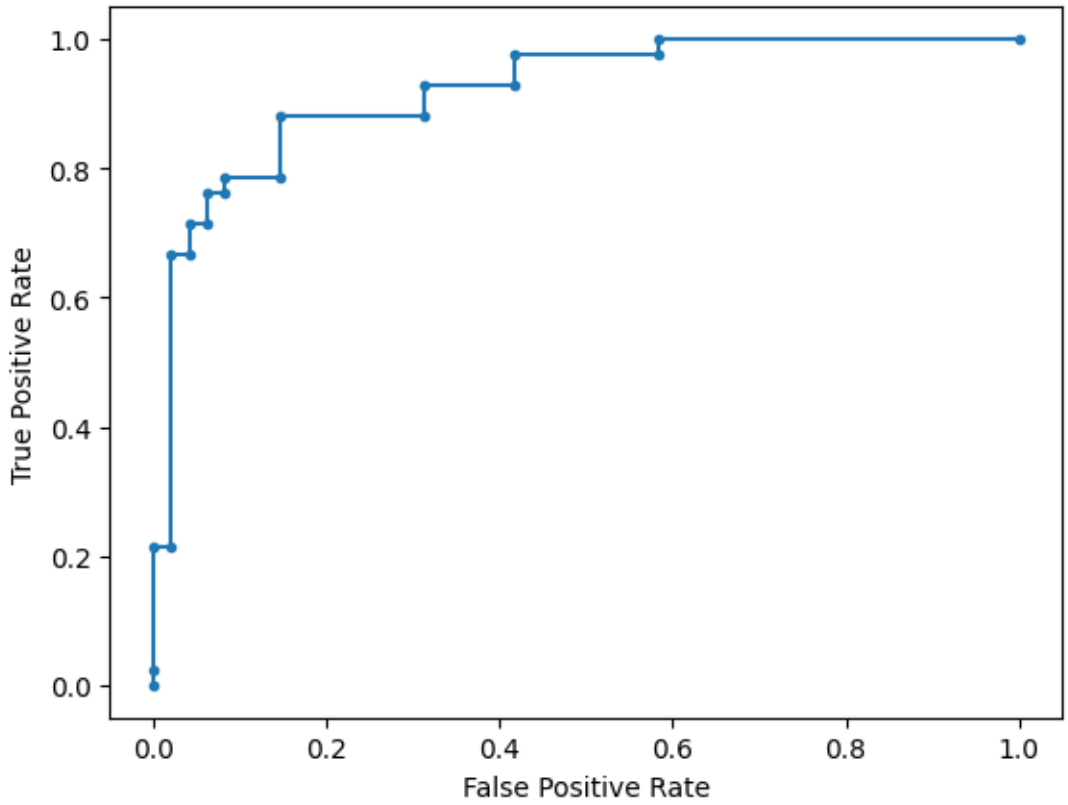
```
array([[0.15827814, 0.84172186],
```

```
       [0.05988172, 0.94011828]])
```

```
df_clust_data = pd.DataFrame(
    data = {'prob1':pred_prob1[:,1], 'y_test': y_test}
)
df_clust_data.head(5)
```

	prob1	y_test
95	0.864018	1
299	0.947865	1
221	0.016439	0
210	0.016938	0
43	0.066449	0

```
clust_fpr, clust_tpr, thresholds = roc_curve(df_clust_data.y_test, df_clust_data.prob1)
#plot the roc curve for the model
plt.plot(clust_fpr, clust_tpr, marker='.', label='Clustered')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```




```
# calculate the AUC for the logistic regression with all predictors
clust_auc = roc_auc_score(df_clust_data.y_test, df_clust_data.prob1)
clust_auc
```

```
np.float64(0.9211309523809523)
```

```
#Youden's J statistic
j_statistic = clust_tpr - clust_fpr
optimal_index = np.argmax(j_statistic)
optimal_threshold = thresholds[optimal_index]
optimal_threshold
```

```
np.float64(0.4210101398082611)
```

```
#predict
df_clust_data['y_pred'] = df_clust_data.prob1.apply(lambda x: 1 if x > optimal_threshold else 0
cm_clust = confusion_matrix(df_clust_data.y_test, df_clust_data.y_pred)
total_clust = sum(sum(cm_clust))
accuracy_clust = (cm_clust[0,0]+cm_clust[1,1])/total_clust
print ('Accuracy : ', accuracy_clust)
sensitivity_clust = cm_clust[1,1]/(cm_clust[1,0]+cm_clust[1,1])
print('Sensitivity : ', sensitivity_clust )
specificity_clust = cm_clust[0,0]/(cm_clust[0,0]+cm_clust[0,1])
print('Specificity : ', specificity_clust)
```

```
Accuracy :  0.8555555555555555
Sensitivity :  0.8571428571428571
Specificity :  0.8541666666666666
```

```
#dd these to the results dataframe
new = pd.DataFrame({
    'Classifier': 'Clustered',
    'Accuracy': accuracy_clust,
    'Sensitivity': sensitivity_clust,
    'Specificity': specificity_clust,
    'AUC': clust_auc
},index = [4])

updated_results = pd.concat([results,new])
updated_results
```

	Classifier	Accuracy	Sensitivity	Specificity	AUC
0	Logistic Regression	0.855556	0.857143	0.854167	0.921131
1	SVM	0.877778	0.880952	0.875000	0.937996
2	LR with Lasso	0.866667	0.857143	0.875000	0.925099

	Classifier	Accuracy	Sensitivity	Specificity	AUC
4	Clustered	0.855556	0.857143	0.854167	0.921131

This clustered model performs almost the same as Logistic Regression with Lasso.