```
import pandas as pd
import numpy as np
from plotnine import *

import warnings
warnings.filterwarnings('ignore')

from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler #Z-score variables

from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.metrics import silhouette_score, silhouette_samples

from sklearn.model_selection import KFold # k-fold cv
from sklearn.model_selection import LeaveOneOut #LOO cv
from sklearn.model_selection import train_test_split # simple TT split
cv

from sklearn.neighbors import NearestNeighbors

from sklearn.cluster import AgglomerativeClustering, KMeans, DBSCAN
from sklearn.mixture import GaussianMixture
from sklearn.tree import DecisionTreeClassifier

import matplotlib.pyplot as plt

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer

from sklearn import metrics
from sklearn.preprocessing import StandardScaler #Z-score variables

from sklearn.model_selection import train_test_split # simple TT split
cv
from sklearn.model_selection import KFold # k-fold cv
from sklearn.model_selection import LeaveOneOut #LOO cv
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import plot_confusion_matrix, plot_roc_curve

from sklearn.model_selection import GridSearchCV

import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

from plotnine import *
```

```
from sklearn.decomposition import PCA
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
import numpy as np


%matplotlib inline

from sklearn.metrics import accuracy_score, confusion_matrix,
f1_score, recall_score, precision_score, roc_auc_score,
plot_confusion_matrix, plot_roc_curve

import scipy.cluster.hierarchy as sch
```

Our group made made a number of changes to the Analysis Plan since #3. The first change was our dataset. We found that our original dataset did not contain a significant number of useful variables, and we wanted to utalize a heart disease data set that would allow us to throuroughly explore each of our questions. Our issue with the variables was that there were many binary and qualitative variables, but very few usable quantitative variables. This new dataset has a good mix of qualitative and quantitative variables, and allows us to conduct the depthful analysis we had hoped to achieve.

We also reworded/altered a few of our questions after recieving feedback from both our instructor and our peers. The opportunity to gain this outside persective gave us valuable insight into alterations we could make to our analysis plan in order to generate the best results.

Question 1) We decided to utalize question 3 from the analysis plan as our first question, as this was a great baseline to understand which coefficients were strongest. We also planned to use this logistic regression model throughout the remainder of the analysis.

Question 2) We added a KNN model to determine which model was most successful at predicting heart disease (Logistic Regression vs KNN).

Question 3) We utalized question 4 from the analysis plan with a few modifications. First off, we created three age groups rather than five due to the variation in number of data for some of the age groups. We also decided to utalize Accuracy and ROC/AUC to determine the accuracy for each age group.

Question 4) We used question 5 from the analysis plan. We decided to change the variables we were clustering with our new data set to Cholesterol and Max HR. We also decided to implement scatter plots to visualize how the variables are related to one another.

Question 5) For question 5, we wanted to add a decision tree to see how successful it would be at making classifications.

Question 6) For question 6, we made numerous altercations to improve our PCA model. We took out the categorical models from the PCA, selecting a subset of continuous variables and a few binary variables. We also added a 90% threshold to determine how many principal components we needed to achieve this threshold. Finally, we used the selected principal components to build a new logistic regression model. We then compared the accuracy of this model to the accuracy of the logistic regression model with the original variables.

We also added ggplot visualizations to each question in order to enhance the analysis.

Preliminary Loading and Cleaning of the Dataset

```
#read in data
DF = pd.read_csv("Heart_Disease_Prediction.csv")

#clean data
DF.dropna()

#create dummy variable for heart disease (we will be using heart
disease presence)
DF = pd.get_dummies(DF, columns = ["Heart Disease"])
DF
```

```
     Age  Sex  Chest pain type   BP  Cholesterol  FBS over 120  EKG
results  \
0     70    1                4  130          322             0
2
1     67    0                3  115          564             0
2
2     57    1                2  124          261             0
0
3     64    1                4  128          263             0
0
4     74    0                2  120          269             0
2
..   ...  ...              ...  ...          ...           ...
...
265   52    1                3  172          199             1
0
266   44    1                2  120          263             0
0
267   56    0                2  140          294             0
2
268   57    1                4  140          192             0
0
269   67    1                4  160          286             0
2

     Max HR  Exercise angina  ST depression  Slope of ST  \
0       109                0            2.4            2
```

```
1       160                   0            1.6            2
2       141                   0            0.3            1
3       105                   1            0.2            2
4       121                   1            0.2            1
..      ...                 ...            ...          ...
265     162                   0            0.5            1
266     173                   0            0.0            1
267     153                   0            1.3            2
268     148                   0            0.4            2
269     108                   1            1.5            2

     Number of vessels fluro  Thallium  Heart Disease_Absence  \
0                          3         3                      0
1                          0         7                      1
2                          0         7                      0
3                          1         7                      1
4                          1         3                      1
..                       ...       ...                    ...
265                        0         7                      1
266                        0         7                      1
267                        0         3                      1
268                        0         6                      1
269                        3         3                      0

     Heart Disease_Presence
0                         1
1                         0
2                         1
3                         0
4                         0
..                      ...
265                       0
266                       0
267                       0
268                       0
269                       1

[270 rows x 15 columns]
```

QUESTION 1: Which of the following variables is the strongest predictor of Heart Disease(Age, Sex, Chest pain type, BP, cholesterol, FBS over 120, EKG results, Max HR, Exercise angina, ST depression, slope of ST, thallium number of vesesels fluro)? What led to this conclusion?

First, lets gain an understanding of our dataset. Lets take a look at the distribution of Ages in our dataset.

```
(ggplot(DF, aes(x = "factor(Sex)", y = 'Age', fill = "factor(Sex)")) +
geom_boxplot() +
 theme_minimal() +
```

```
 xlab('Sex (0 = Female, 1 = Female)') +
 ylab('Age in Years') +
 ggtitle('Distribution of Ages by Sex') +
 guides(fill = guide_legend(title = 'Sex')))
```


```
<ggplot: (8776364310447)>
```

The above boxplot is a visual representation of the distribution of ages by sex.

From the visualization, it appears that the mean age is slightly higher for the Female group. How many males/females are there in our dataset?

```
print(np.unique(DF['Sex'], return_counts = True))
```

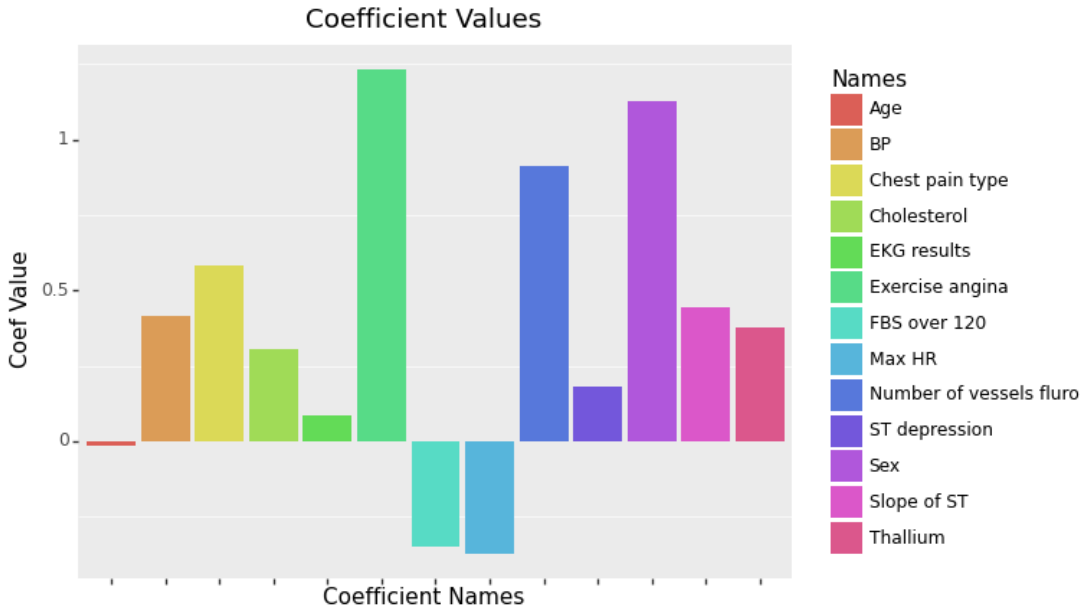
```
(array([0, 1]), array([ 87, 183]))
```

Thre are 183 Males in our dataset and 87 Females in our dataset. Undestanding your dataset is imporant before performing analysis; so now that we know a little more about our data, lets begin with the models.

```
# Create logistic regression model

preds = ["Age", "Sex", "Chest pain type", "BP", "Cholesterol", "FBS
over 120", "EKG results", "Max HR", "Exercise angina",
         "ST depression", "Slope of ST", "Thallium", "Number of
vessels fluro"]
```

```
cont = ["BP", "Cholesterol", "Max HR", "ST depression", "Number of
vessels fluro"]

X = DF[preds]
y = DF["Heart Disease_Presence"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.3, random_state = 5)

# zscore
z = StandardScaler()
X_train[cont] = z.fit_transform(X_train[cont])
X_test[cont] = z.transform(X_test[cont])

# create and fit model
lr = LogisticRegression()
lr.fit(X_train, y_train)

# create dataframes with coefficients
coef = pd.DataFrame({"Coefs": lr.coef_[0],
                     "Names": preds})

coef["Odds Coef"] = np.exp(coef["Coefs"])
coef

print(coef)

# create bar chart to represent the coefs
(ggplot(coef, aes(x = "Names", y = "Coefs", fill = "Names")) +
geom_bar(stat = "identity") +
 ggtitle("Coefficient Values") +
 labs(x = "Coefficient Names", y = "Coef Value") +
 theme(axis_text_x = element_blank(),
       panel_grid_major_y = element_blank(),
       panel_grid_major_x = element_blank()))
```

```
       Coefs                     Names  Odds Coef
0  -0.016537                       Age   0.983599
1   1.125363                       Sex   3.081334
2   0.584415          Chest pain type   1.793942
3   0.414544                        BP   1.513680
4   0.303510               Cholesterol   1.354606
5  -0.350877              FBS over 120   0.704070
6   0.087396               EKG results   1.091328
7  -0.373817                    Max HR   0.688103
8   1.231587          Exercise angina   3.426665
9   0.181508            ST depression   1.199025
10  0.442442               Slope of ST   1.556504
11  0.374381                  Thallium   1.454091
12  0.909533  Number of vessels fluro   2.483163
```




```
<ggplot: (8776361593652)>
```

The above bar graph visually represents the coefficient values for each variable. As we can see, Age is the lowest coefficient value while Exercise Angina and Sex are the highest. Further, FBS over 120 and Max HR have negative coefficients.

ANSWER TO QUESTION 1

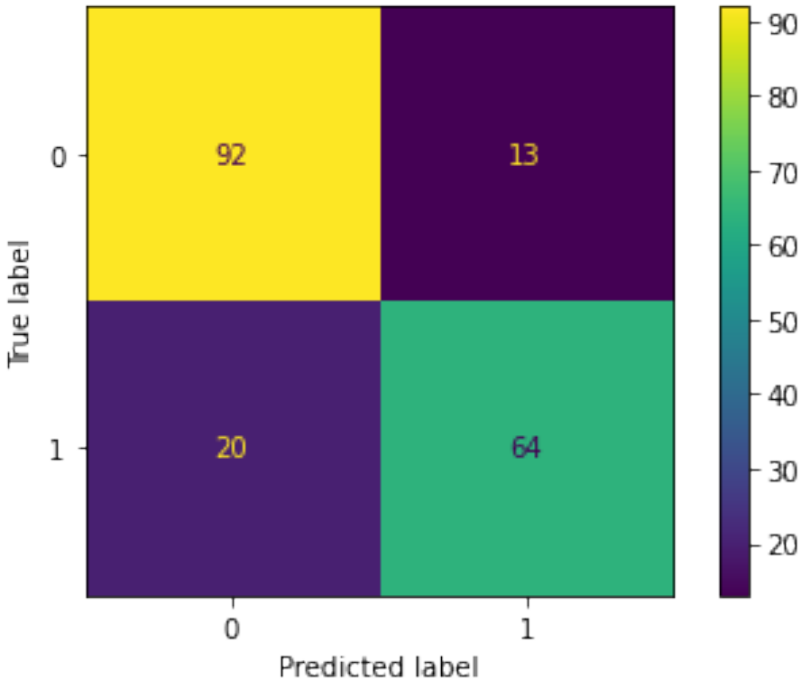
Out of all 13 of our predictors, Exercise Angina, Sex, Number of Vessels Fluro are all signifcant variables. However, Exercise Angina is the strongest predictor of heart disease. You can visaully see this from our bar graph of our coeficcients. Exercise Angina has the largest coefficent value. In order to better interpret our coefficients, we turned them into odds coefficients in order to view each of the variables as a multiplier. The interpretation of Exercise Angina's coefficicent is that the presence of exercise angina means a person is 3.43 times more likely to have a heart disease. This answer makes sense because exercise angina is a very good indicator as to whether someone has heart disease or not. Exercise angina is pain in the chest that comes with exercise that can feel like a pressure, heaviness, and/or tightness in the chest.

QUESTION 2: If a KNN is created with the same predictors as the model above, will the KNN or Logistic Regression model perform better? What lead to this conclusion?

```
# Creating KNN Model
knn = KNeighborsClassifier(n_neighbors = 9)
knn.fit(X_train, y_train)

# Creating Predictions
KNN_predtrain = knn.predict(X_train)
KNN_predtest = knn.predict(X_test)
```

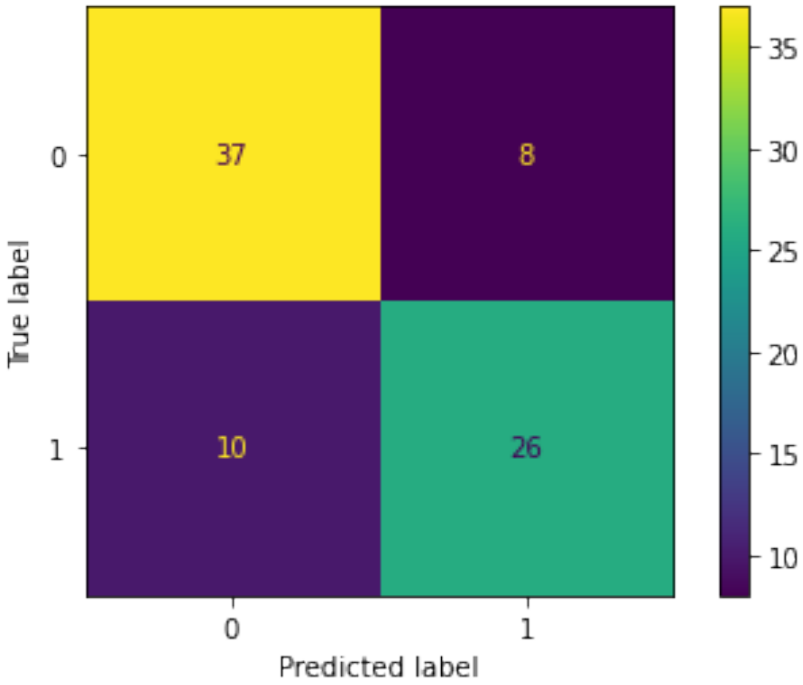
```
print("KNN Train Confusion Matrix: ", plot_confusion_matrix(knn,
X_train, y_train))
print("KNN Training Accuracy", accuracy_score(y_train, KNN_predtrain))
print("KNN Training ROC/AUC: ", roc_auc_score(y_train,
knn.predict_proba(X_train)[:,1]))
```

```
KNN Train Confusion Matrix:
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object
at 0x7fb681309550>
KNN Training Accuracy 0.8253968253968254
KNN Training ROC/AUC:  0.8997165532879818
```




```
print("KNN Test Confusion Matrix: ", plot_confusion_matrix(knn,
X_test, y_test))
print("KNN Testing Accuracy: ", accuracy_score(y_test, KNN_predtest))
print("KNN Testing ROC/AUC: ", roc_auc_score(y_test,
knn.predict_proba(X_test)[:,1]))
```

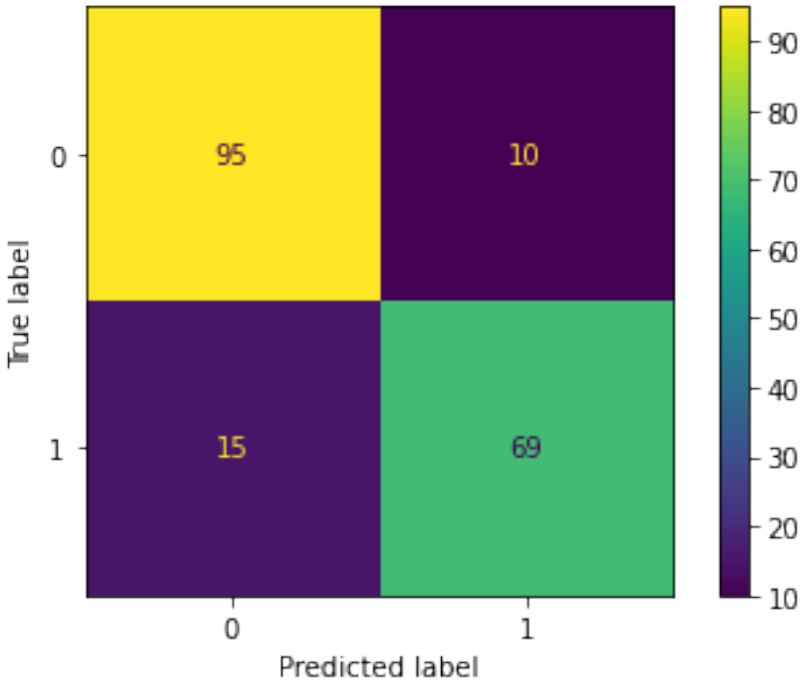
```
KNN Test Confusion Matrix:
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object
at 0x7fb67e8c2a60>
KNN Testing Accuracy:  0.7777777777777778
KNN Testing ROC/AUC:  0.8074074074074074
```

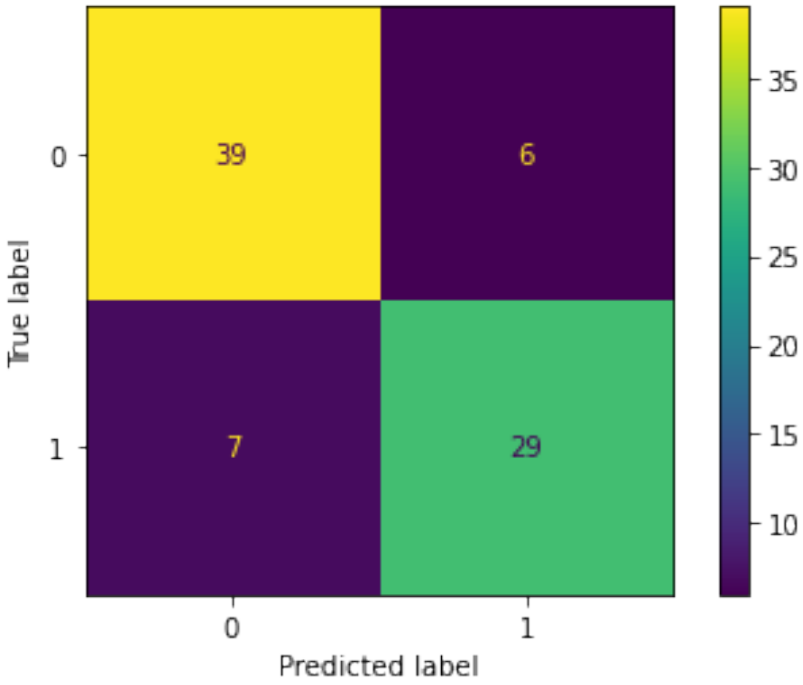



```
# Predictions and accuracy for Logistic Model

# Making Preds
trainpredvals = lr.predict(X_train)
print("Logistic Train Accuracy : ", accuracy_score(y_train,
trainpredvals))
print("Logistic Train ROC/AUC: ", roc_auc_score(y_train,
lr.predict_proba(X_train)[:,1]))

testpredvals = lr.predict(X_test)
print("Logistic Test Accuracy : ", accuracy_score(y_test,
testpredvals))
print("Logistic Test ROC/AUC: ", roc_auc_score(y_test,
lr.predict_proba(X_test)[:,1]))

#print confusion matrices
print("Train Confusion matrix: ", plot_confusion_matrix(lr, X_train,
y_train))
print("Test Confusion matrix: ", plot_confusion_matrix(lr, X_test,
y_test))
```

```
Logistic Train Accuracy :  0.8677248677248677
Logistic Train ROC/AUC:  0.9311791383219955
Logistic Test Accuracy :  0.8395061728395061
Logistic Test ROC/AUC:  0.9092592592592592
Train Confusion matrix:
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object
at 0x7fb67ea3d970>
Test Confusion matrix:
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object
at 0x7fb67e827d60>
```

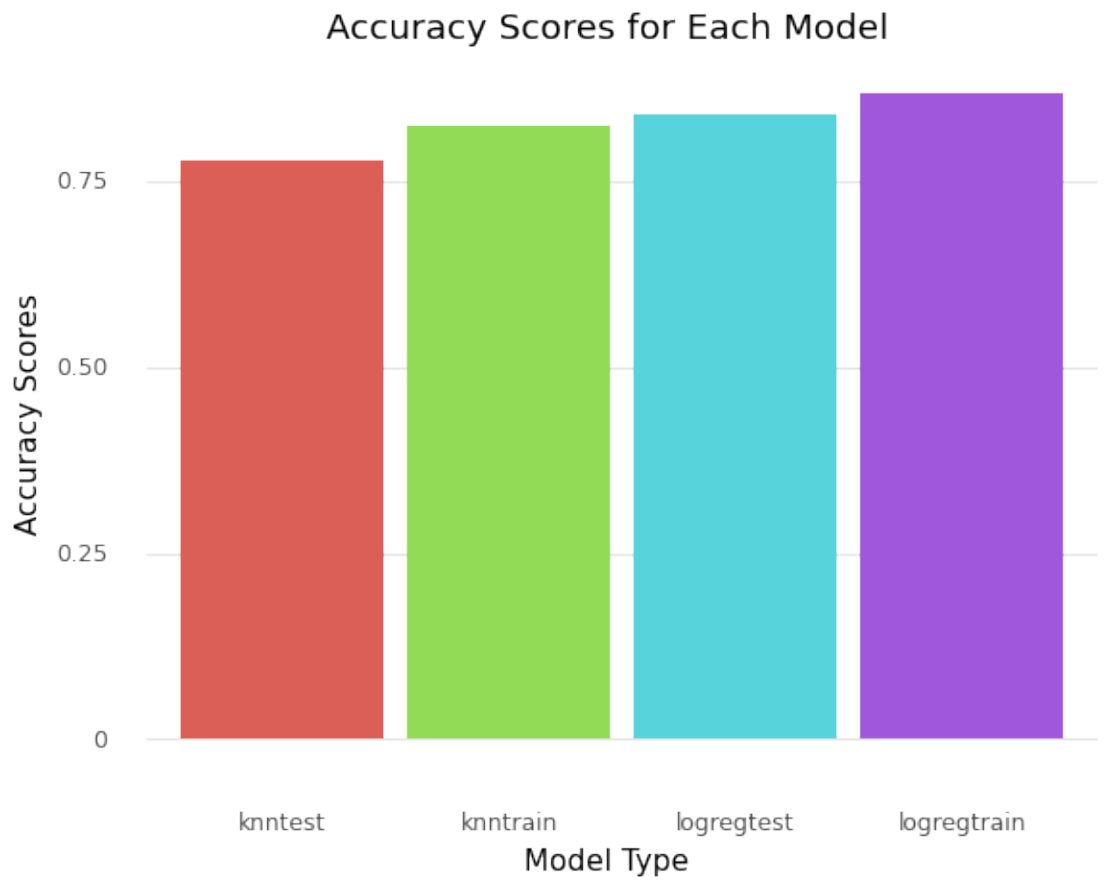





```
knntrain = accuracy_score(y_train, KNN_predtrain)
knntest = accuracy_score(y_test, KNN_predtest)
logregtrain = accuracy_score(y_train, trainpredvals)
logregtest = accuracy_score(y_test, testpredvals)
```

```
knntrain1 = roc_auc_score(y_train, knn.predict_proba(X_train)[:,1])
knntest1 = roc_auc_score(y_test, knn.predict_proba(X_test)[:,1])
logregtrain1 = roc_auc_score(y_train, lr.predict_proba(X_train)[:,1])
logregtest1 = roc_auc_score(y_test, lr.predict_proba(X_test)[:,1])

# Creating New Dataframes for Later Use
model = ["logregtrain", "logregtest", "knntrain", "knntest"]

accuracy = [logregtrain, logregtest, knntrain, knntest]
roc_auc = [logregtrain1, logregtest, knntrain, knntest]

newDF = pd.DataFrame({"ModelType": model, "Accuracy": accuracy})
newDF2 = pd.DataFrame({"ModelType": model, "ROC/AUC": roc_auc})


# Printing Visualizations
print("Graph of Accuracy Scores for KNN Model vs. Logisitic Model")
print((ggplot(newDF, aes(x = "ModelType", y = "Accuracy", fill =
"ModelType")) +
 geom_bar(stat = "identity") +
 labs(x = "Model Type", y = "Accuracy Scores") +
 ggtitle("Accuracy Scores for Each Model") +
 theme_minimal() +
 theme(panel_grid_major_x = element_blank(),
     panel_grid_minor_x = element_blank(),
      panel_grid_minor_y = element_blank(),
      legend_position = "none")))
```

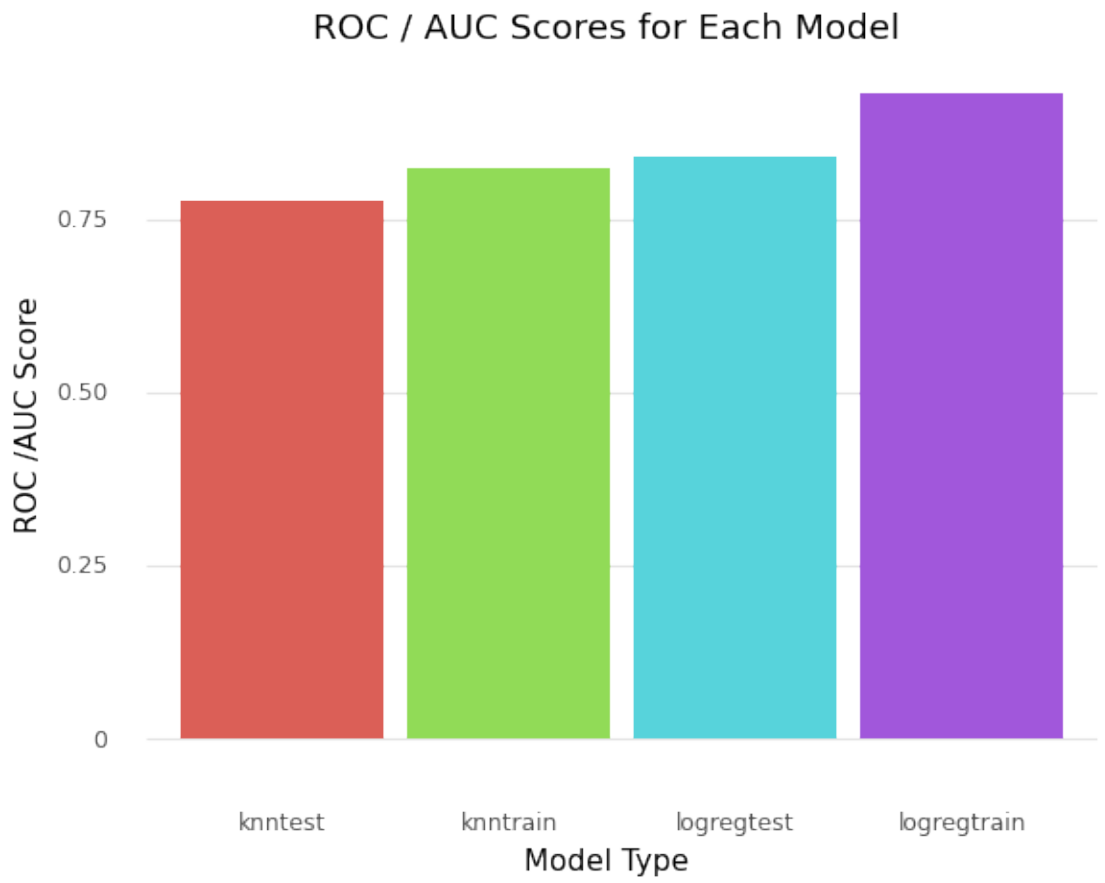
```
Graph of Accuracy Scores for KNN Model vs. Logisitic Model
```



The bar graph above displays the accuracy scores for each type of model (knn test set, knn train set, logisistic regression test set, and logistic regression train set). As we can see, the logistic regression train set had the highest accuracy score, while the knn test model had the lowest accuracy score. Further, we can see that the logistic regression train and test sets outperform both of the knn models.

```
print("Graph of ROC/AUC Scores for KNN Model vs. Logisitic Model")
print((ggplot(newDF2, aes(x = "ModelType", y = "ROC/AUC", fill =
"ModelType")) +
 geom_bar(stat = "identity") +
 labs(x = "Model Type", y = "ROC /AUC Score") +
 ggtitle("ROC / AUC Scores for Each Model") +
 theme_minimal() +
 theme(panel_grid_major_x = element_blank(),
     panel_grid_minor_x = element_blank(),
      panel_grid_minor_y = element_blank(),
      legend_position = "none")))
```

```
Graph of ROC/AUC Scores for KNN Model vs. Logisitic Model
```



The bar graph above displays the ROC/AUC scores for each type of model (knn test set, knn train set, logisistic regression test set, and logistic regression train set). As we can see, the logistic regression train set had the highest ROC/AUC score, while the knn test model had the lowest ROC/AUC score. Further, we can see that the logistic regression train and test sets outperform both of the knn models.

ANSWER TO QUESTION 2:

In order to answer this question, we needed to create a KNN model, and then compare how it performs to our Logistic Model we created in question 1. We decided that we were going to compare the accuracies of the two models (how good our model was at predicting heart disease). The training accuracy for our newly created KNN model was 0.8254 and our testing accuracy was 0.7778. This means that on data that our KNN model has never seen (like in a real life application), it predicts the presense of Heart Disease correctly roughly 78% of the time.

This is a great score, however, we did the same steps for the Logistic Model we created in question 1 and the accuracy scores were a bit higher. This is shown in the graph of accuracy scores. Our Logistic Model had a training accuracy of 0.8677 and a testing

accuracy of 0.8395; meaning our Logistic Model predicts Heart Disease correclty roughly 84% of the time on unseen data (like in a real life application).

Not only is our Logistic Model more accurate then our KNN model (as seen by the accuracy scores and confusion matrices), but it is also less overfit. A model is deemed overfit if it performs so good on training data that when it is exposed to unseen data, it suffers in accuracy. In data science, we want to avoid this at all costs because it can cause a model to get completely thrown out. Our KNN model had nearly a 5% deviation from its training and testing accuracy, while our Logistic Model had just under a 3% deviation in training and testing accuracy. Beacuse of this, we concluded that our KNN model is more overfit than our Logistic Regression Model.

Based on these accuracy scores, we determined the our Logisitc Model was the better model at predciting Heart Disease.

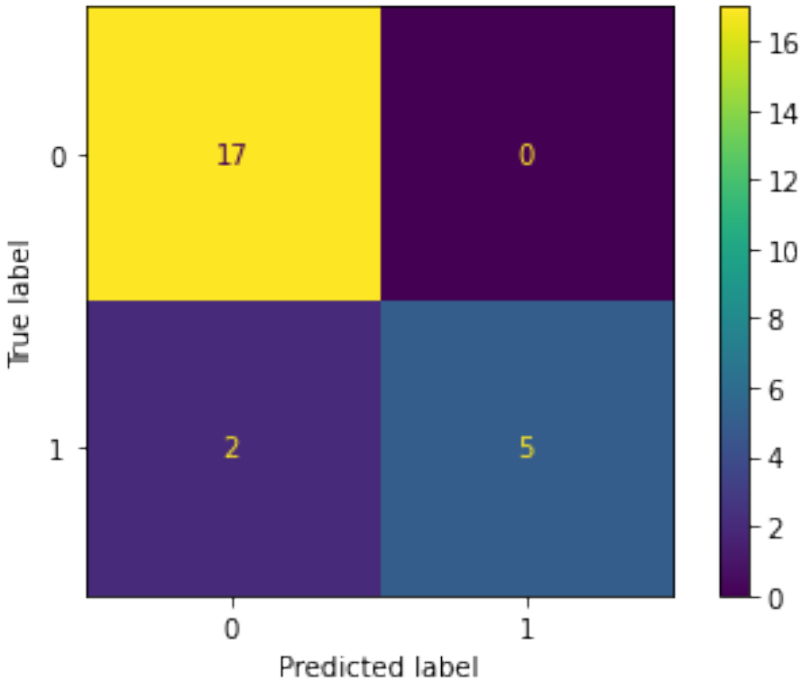
QUESTION 3:

Is there variance in the accuracy of our model for people of different age groups (i.e. for our logistic regression model, is there a significantly higher accuracy score for any age groups < 30, 30-39, 40-49, etc.) If so, how could this impact the reliability of our model and our dataset and how can this be overcome?

```
#create groups for the different age ranges
#group <= 29
group1 = (X_test["Age"] < 50)
group2 = (X_test["Age"] >= 50) & (X_test["Age"] < 59)
group3 = (X_test["Age"] >= 60)

#predict for each age group using the logistic regrssion model from
question 1
pred1 = lr.predict(X_test[preds].loc[group1])
pred2 = lr.predict(X_test[preds].loc[group2])
pred3 = lr.predict(X_test[preds].loc[group3])

#plot metrics to compare accuracy between age groups
print("Accuracy < 50 : ", accuracy_score(y_test.loc[group1], pred1))
print("Accuracy 50-59 : ", accuracy_score(y_test.loc[group2], pred2))
print("Accuracy 60+ : ", accuracy_score(y_test.loc[group3], pred3))

#print confusion matrices
print("Confusion matrices: ")
print("< 50 ")
plot_confusion_matrix(lr, X_test.loc[group1], y_test.loc[group1])
```

```
Accuracy < 50 :  0.9166666666666666
Accuracy 50-59 :  0.7878787878787878
Accuracy 60+ :  0.8636363636363636
Confusion matrices:
< 50
```

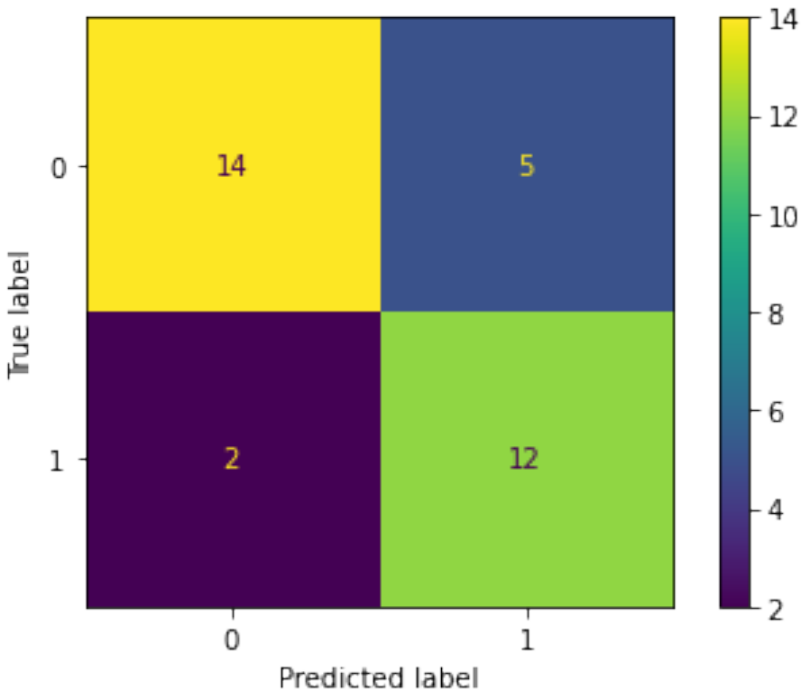
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fb6813a8130>
```




```
print("50-59")
plot_confusion_matrix(lr, X_test.loc[group2], y_test.loc[group2])
```

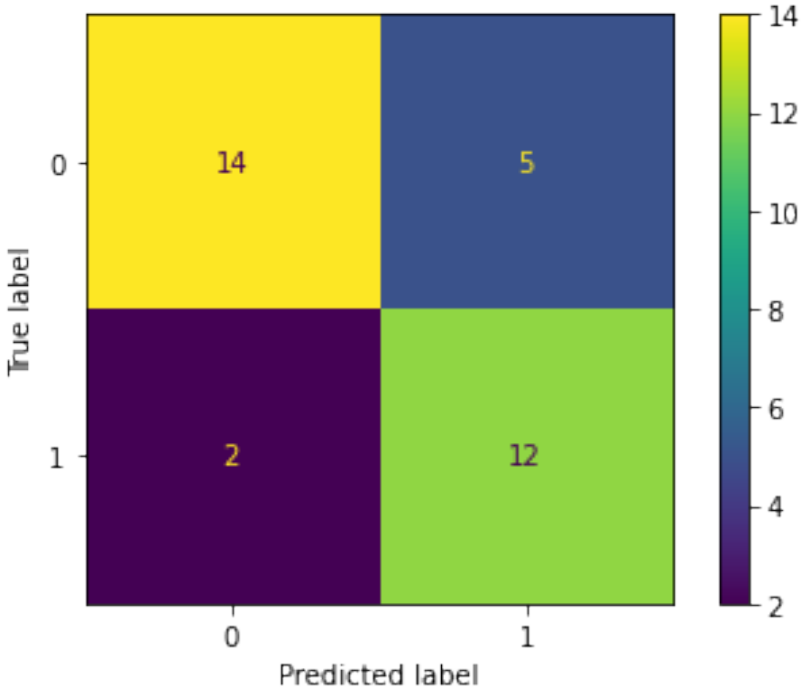
```
50-59
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fb67e994c10>
```

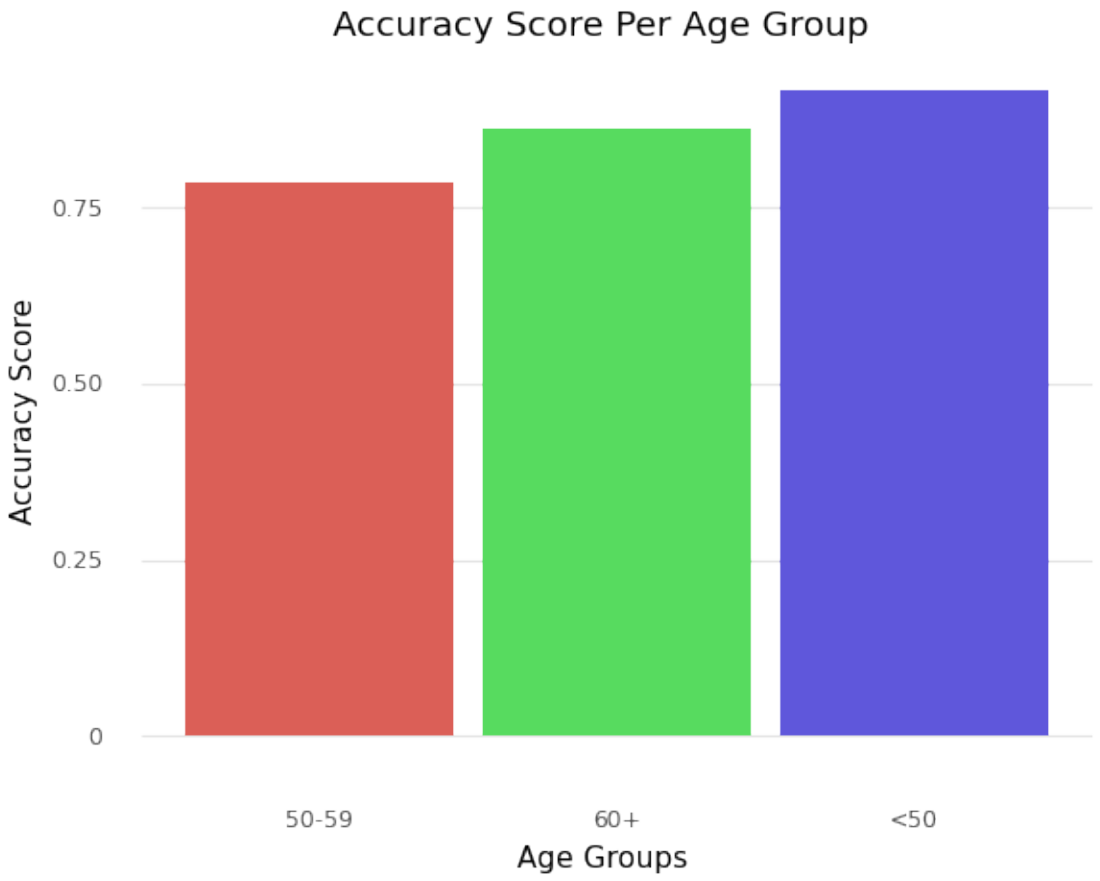



```
print("50-59")
plot_confusion_matrix(lr, X_test.loc[group2], y_test.loc[group2])
```

```
50-59

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fb67e6675e0>
```



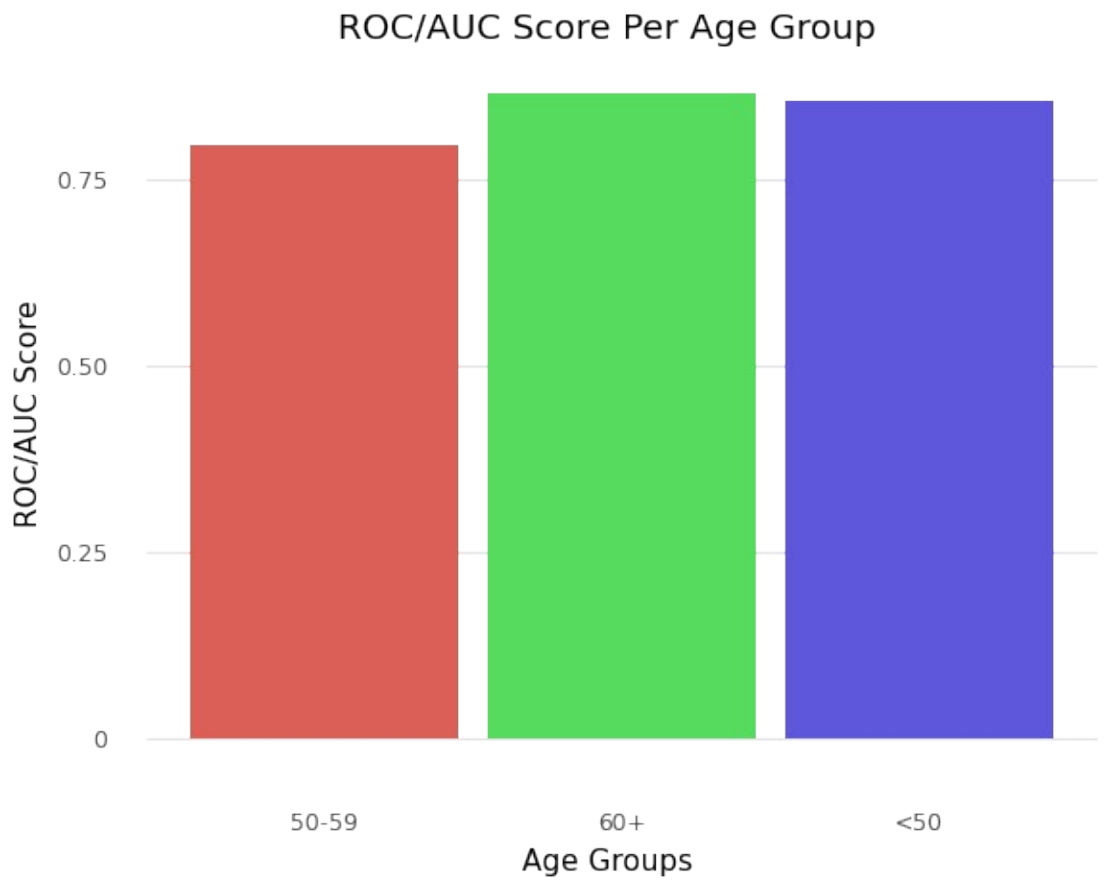
```
accscorebelow50 = accuracy_score(y_test.loc[group1], pred1)
accscore50to59 = accuracy_score(y_test.loc[group2], pred2)
accscore60plus = accuracy_score(y_test.loc[group3], pred3)

# Visualiations for the accuracy for each age group
agegroups = ["<50", "50-59", "60+"]
accuracyscores = [accscorebelow50, accscore50to59, accscore60plus]

newDF = pd.DataFrame({"agegroups": agegroups, "accuracyscores":
accuracyscores})

(ggplot(newDF, aes(x = "agegroups", y = "accuracyscores", fill =
"agegroups")) +
 geom_bar(stat = "identity") +
 labs(x = "Age Groups", y = "Accuracy Score") +
 ggtitle("Accuracy Score Per Age Group") +
 theme_minimal() +
 theme(panel_grid_major_x = element_blank(),
     panel_grid_minor_x = element_blank(),
      panel_grid_minor_y = element_blank(),
      legend_position = "none"))
```



```
<ggplot: (8776361286033)>
```

The bar graph above visually demonstrates the accuracy scores for each age group. From the bar graph, we can see that the model had the highest accuracy score when predicting ages < 50, and the lowest accuracy score when predicting models 50-59.

```
# Visualiations of the ROC/AUC for each age group
rocscorebelow50 = roc_auc_score(y_test.loc[group1], pred1)
rocscore50to59 = roc_auc_score(y_test.loc[group2], pred2)
rocscore60plus = roc_auc_score(y_test.loc[group3], pred3)

rocscores = [rocscorebelow50, rocscore50to59, rocscore60plus]

newDF = pd.DataFrame({"agegroups": agegroups, "accuracyscores":
rocscores})

(ggplot(newDF, aes(x = "agegroups", y = "rocscores", fill =
"agegroups")) +
 geom_bar(stat = "identity") +
 labs(x = "Age Groups", y = "ROC/AUC Score") +
 ggtitle("ROC/AUC Score Per Age Group") +
 theme_minimal() +
 theme(panel_grid_major_x = element_blank(),
     panel_grid_minor_x = element_blank(),
      panel_grid_minor_y = element_blank(),
      legend_position = "none"))
```




```
<ggplot: (8776361286765)>
```

The bar graph above visually demonstrates the ROC/AUC scores for each age group. From the bar graph, we can see that the model had the highest ROC/AUC score when predicting ages > 60, and the lowest ROC/AUC score when predicting models 50-59.

ANSWER TO QUESTION 3

There is variance in the accuracy of our model for individuals in different age groups. We were able to come to this conclusion by splitting our age varaible into three seperate groups - less than 50, ages 50 to 59, and 60 plus - and comparing all three accuracies. Age group less than 50 and greater than 60 were our two highest accuracies. Heart Disease is less common for younger subject and more common for older subjects, so this may be why out model struggles more with those in that middle age group between 50 and 59. Further, when looking at the ROC/AUC values for each age group, we can see that the age group 60+ has the highest ROC/AUC score, and the 50 to 59 age group is still the lowest.

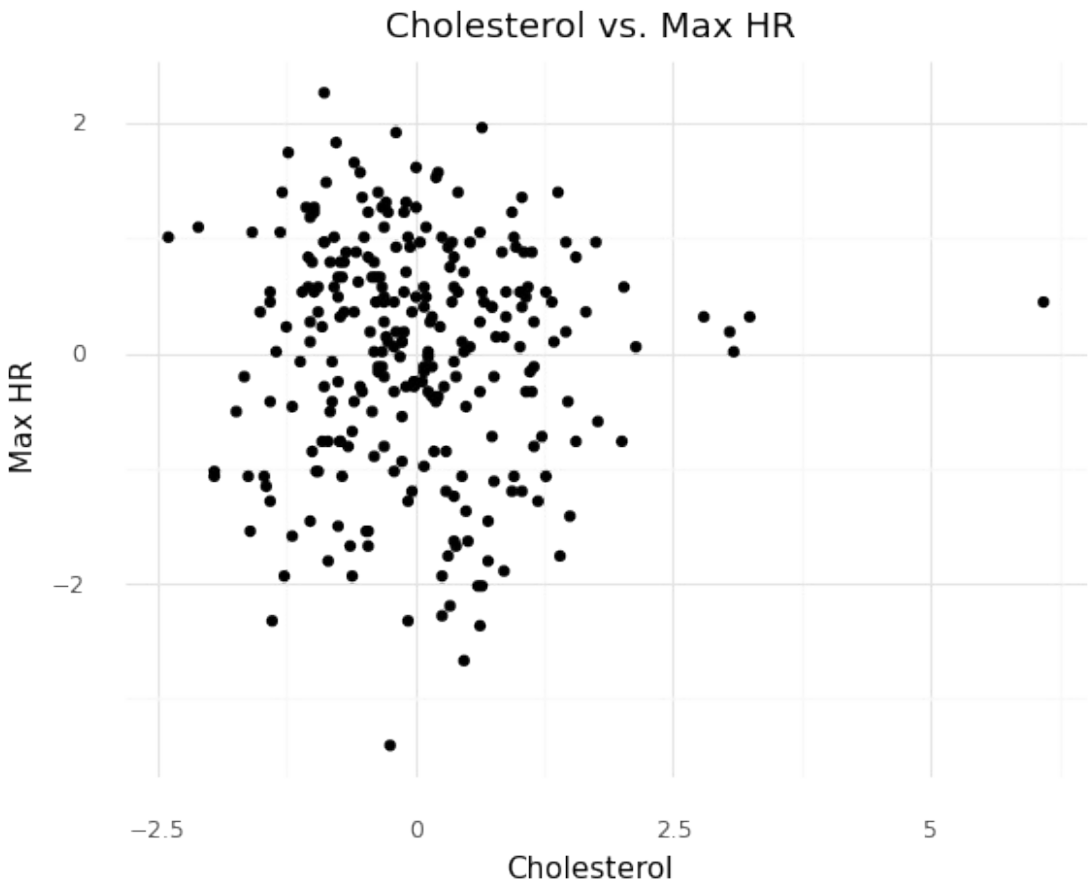
The difference in these three accuracies could potentially effect our models reliability. Due to the importance of accuracy classifying heart disease, with the result possibly being the difference between life and death, it would not be wise to utalize this model when there is varying accuracy for different age groups. To combat this, it would be helpful to re-train

our model on a larger dataset than has a roughly equal number of participants in each age group.

QUESTION 4:

When clustering Cholesterol and MAX HR, what sort of clusters emerge, and are there significant patterns that may characterize the clusters? What clustering method is most successful, and why is this likely the case? What is likely the driver behind the formation of these clusters, and what characteristics describe these clusters? Further, how do these clusters help explain the variables and their relation to Heart Disease?

```
#plot predictors to visually assess which clustering model will
potentially be the best fit
features = ["Cholesterol", "Max HR"]
X2 = DF[features]

#zscore
z = StandardScaler()
X2[features] = z.fit_transform(X2[features])

#create scatter plot
(ggplot(X2, aes(x = "Cholesterol", y = "Max HR")) + geom_point() +
 theme_minimal() + labs(title = "Cholesterol vs. Max HR"))
```



```
<ggplot: (8776361259692)>
```

The scatterplot above demonstrates the relationship between Cholesterol and Max HR in the data set.

```
#Agglomerative Clustering
predictors = ['Cholesterol', 'Max HR']

X1 = DF[predictors]

z = StandardScaler()
X1[predictors] = z.fit_transform(X1[predictors])

hac = AgglomerativeClustering(affinity = "euclidean",
                              linkage ='average',
                              n_clusters = 2)


hac.fit(X1)

# Dendrogram
dendro = sch.dendrogram(sch.linkage(X1, method = 'average'))
print(dendro)

# Silhouette Escore
membership = hac.labels_
print("Silhouette Score: ", silhouette_score(X1, membership))
```

```
{'icoord': [[35.0, 35.0, 45.0, 45.0], [25.0, 25.0, 40.0, 40.0], [15.0,
15.0, 32.5, 32.5], [55.0, 55.0, 65.0, 65.0], [23.75, 23.75, 60.0,
60.0], [75.0, 75.0, 85.0, 85.0], [95.0, 95.0, 105.0, 105.0], [165.0,
165.0, 175.0, 175.0], [155.0, 155.0, 170.0, 170.0], [145.0, 145.0,
162.5, 162.5], [135.0, 135.0, 153.75, 153.75], [125.0, 125.0, 144.375,
144.375], [115.0, 115.0, 134.6875, 134.6875], [100.0, 100.0,
124.84375, 124.84375], [215.0, 215.0, 225.0, 225.0], [205.0, 205.0,
220.0, 220.0], [195.0, 195.0, 212.5, 212.5], [185.0, 185.0, 203.75,
203.75], [112.421875, 112.421875, 194.375, 194.375], [235.0, 235.0,
245.0, 245.0], [255.0, 255.0, 265.0, 265.0], [240.0, 240.0, 260.0,
260.0], [275.0, 275.0, 285.0, 285.0], [250.0, 250.0, 280.0, 280.0],
[325.0, 325.0, 335.0, 335.0], [315.0, 315.0, 330.0, 330.0], [305.0,
305.0, 322.5, 322.5], [295.0, 295.0, 313.75, 313.75], [265.0, 265.0,
304.375, 304.375], [345.0, 345.0, 355.0, 355.0], [365.0, 365.0, 375.0,
375.0], [350.0, 350.0, 370.0, 370.0], [395.0, 395.0, 405.0, 405.0],
[385.0, 385.0, 400.0, 400.0], [435.0, 435.0, 445.0, 445.0], [425.0,
425.0, 440.0, 440.0], [415.0, 415.0, 432.5, 432.5], [392.5, 392.5,
423.75, 423.75], [360.0, 360.0, 408.125, 408.125], [455.0, 455.0,
465.0, 465.0], [475.0, 475.0, 485.0, 485.0], [495.0, 495.0, 505.0,
505.0], [480.0, 480.0, 500.0, 500.0], [460.0, 460.0, 490.0, 490.0],
[525.0, 525.0, 535.0, 535.0], [515.0, 515.0, 530.0, 530.0], [555.0,
555.0, 565.0, 565.0], [545.0, 545.0, 560.0, 560.0], [575.0, 575.0,
```

585.0, 585.0], [595.0, 595.0, 605.0, 605.0], [580.0, 580.0, 600.0,
600.0], [552.5, 552.5, 590.0, 590.0], [522.5, 522.5, 571.25, 571.25],
[475.0, 475.0, 546.875, 546.875], [384.0625, 384.0625, 510.9375,
510.9375], [284.6875, 284.6875, 447.5, 447.5], [615.0, 615.0, 625.0,
625.0], [645.0, 645.0, 655.0, 655.0], [635.0, 635.0, 650.0, 650.0],
[675.0, 675.0, 685.0, 685.0], [665.0, 665.0, 680.0, 680.0], [642.5,
642.5, 672.5, 672.5], [620.0, 620.0, 657.5, 657.5], [705.0, 705.0,
715.0, 715.0], [695.0, 695.0, 710.0, 710.0], [735.0, 735.0, 745.0,
745.0], [725.0, 725.0, 740.0, 740.0], [785.0, 785.0, 795.0, 795.0],
[775.0, 775.0, 790.0, 790.0], [765.0, 765.0, 782.5, 782.5], [755.0,
755.0, 773.75, 773.75], [732.5, 732.5, 764.375, 764.375], [702.5,
702.5, 748.4375, 748.4375], [638.75, 638.75, 725.46875, 725.46875],
[825.0, 825.0, 835.0, 835.0], [815.0, 815.0, 830.0, 830.0], [805.0,
805.0, 822.5, 822.5], [885.0, 885.0, 895.0, 895.0], [875.0, 875.0,
890.0, 890.0], [865.0, 865.0, 882.5, 882.5], [855.0, 855.0, 873.75,
873.75], [845.0, 845.0, 864.375, 864.375], [915.0, 915.0, 925.0,
925.0], [905.0, 905.0, 920.0, 920.0], [945.0, 945.0, 955.0, 955.0],
[985.0, 985.0, 995.0, 995.0], [975.0, 975.0, 990.0, 990.0], [965.0,
965.0, 982.5, 982.5], [950.0, 950.0, 973.75, 973.75], [935.0, 935.0,
961.875, 961.875], [912.5, 912.5, 948.4375, 948.4375], [854.6875,
854.6875, 930.46875, 930.46875], [813.75, 813.75, 892.578125,
892.578125], [682.109375, 682.109375, 853.1640625, 853.1640625],
[366.09375, 366.09375, 767.63671875, 767.63671875], [153.3984375,
153.3984375, 566.865234375, 566.865234375], [1005.0, 1005.0, 1015.0,
1015.0], [1025.0, 1025.0, 1035.0, 1035.0], [1010.0, 1010.0, 1030.0,
1030.0], [1055.0, 1055.0, 1065.0, 1065.0], [1045.0, 1045.0, 1060.0,
1060.0], [1075.0, 1075.0, 1085.0, 1085.0], [1095.0, 1095.0, 1105.0,
1105.0], [1080.0, 1080.0, 1100.0, 1100.0], [1052.5, 1052.5, 1090.0,
1090.0], [1020.0, 1020.0, 1071.25, 1071.25], [1125.0, 1125.0, 1135.0,
1135.0], [1115.0, 1115.0, 1130.0, 1130.0], [1145.0, 1145.0, 1155.0,
1155.0], [1122.5, 1122.5, 1150.0, 1150.0], [1175.0, 1175.0, 1185.0,
1185.0], [1165.0, 1165.0, 1180.0, 1180.0], [1136.25, 1136.25, 1172.5,
1172.5], [1195.0, 1195.0, 1205.0, 1205.0], [1235.0, 1235.0, 1245.0,
1245.0], [1225.0, 1225.0, 1240.0, 1240.0], [1215.0, 1215.0, 1232.5,
1232.5], [1200.0, 1200.0, 1223.75, 1223.75], [1255.0, 1255.0, 1265.0,
1265.0], [1285.0, 1285.0, 1295.0, 1295.0], [1305.0, 1305.0, 1315.0,
1315.0], [1335.0, 1335.0, 1345.0, 1345.0], [1325.0, 1325.0, 1340.0,
1340.0], [1310.0, 1310.0, 1332.5, 1332.5], [1290.0, 1290.0, 1321.25,
1321.25], [1275.0, 1275.0, 1305.625, 1305.625], [1260.0, 1260.0,
1290.3125, 1290.3125], [1211.875, 1211.875, 1275.15625, 1275.15625],
[1375.0, 1375.0, 1385.0, 1385.0], [1395.0, 1395.0, 1405.0, 1405.0],
[1380.0, 1380.0, 1400.0, 1400.0], [1365.0, 1365.0, 1390.0, 1390.0],
[1415.0, 1415.0, 1425.0, 1425.0], [1445.0, 1445.0, 1455.0, 1455.0],
[1435.0, 1435.0, 1450.0, 1450.0], [1420.0, 1420.0, 1442.5, 1442.5],
[1505.0, 1505.0, 1515.0, 1515.0], [1495.0, 1495.0, 1510.0, 1510.0],
[1485.0, 1485.0, 1502.5, 1502.5], [1475.0, 1475.0, 1493.75, 1493.75],
[1465.0, 1465.0, 1484.375, 1484.375], [1431.25, 1431.25, 1474.6875,
1474.6875], [1377.5, 1377.5, 1452.96875, 1452.96875], [1355.0, 1355.0,
1415.234375, 1415.234375], [1243.515625, 1243.515625, 1385.1171875,
1385.1171875], [1154.375, 1154.375, 1314.31640625, 1314.31640625],

[1535.0, 1535.0, 1545.0, 1545.0], [1525.0, 1525.0, 1540.0, 1540.0],
[1555.0, 1555.0, 1565.0, 1565.0], [1575.0, 1575.0, 1585.0, 1585.0],
[1595.0, 1595.0, 1605.0, 1605.0], [1580.0, 1580.0, 1600.0, 1600.0],
[1560.0, 1560.0, 1590.0, 1590.0], [1532.5, 1532.5, 1575.0, 1575.0],
[1645.0, 1645.0, 1655.0, 1655.0], [1635.0, 1635.0, 1650.0, 1650.0],
[1625.0, 1625.0, 1642.5, 1642.5], [1665.0, 1665.0, 1675.0, 1675.0],
[1695.0, 1695.0, 1705.0, 1705.0], [1685.0, 1685.0, 1700.0, 1700.0],
[1670.0, 1670.0, 1692.5, 1692.5], [1633.75, 1633.75, 1681.25,
1681.25], [1615.0, 1615.0, 1657.5, 1657.5], [1735.0, 1735.0, 1745.0,
1745.0], [1725.0, 1725.0, 1740.0, 1740.0], [1715.0, 1715.0, 1732.5,
1732.5], [1755.0, 1755.0, 1765.0, 1765.0], [1785.0, 1785.0, 1795.0,
1795.0], [1775.0, 1775.0, 1790.0, 1790.0], [1760.0, 1760.0, 1782.5,
1782.5], [1723.75, 1723.75, 1771.25, 1771.25], [1636.25, 1636.25,
1747.5, 1747.5], [1553.75, 1553.75, 1691.875, 1691.875],
[1234.345703125, 1234.345703125, 1622.8125, 1622.8125], [1045.625,
1045.625, 1428.5791015625, 1428.5791015625], [360.1318359375,
360.1318359375, 1237.10205078125, 1237.10205078125], [1815.0, 1815.0,
1825.0, 1825.0], [1845.0, 1845.0, 1855.0, 1855.0], [1865.0, 1865.0,
1875.0, 1875.0], [1850.0, 1850.0, 1870.0, 1870.0], [1835.0, 1835.0,
1860.0, 1860.0], [1820.0, 1820.0, 1847.5, 1847.5], [1905.0, 1905.0,
1915.0, 1915.0], [1895.0, 1895.0, 1910.0, 1910.0], [1885.0, 1885.0,
1902.5, 1902.5], [1833.75, 1833.75, 1893.75, 1893.75], [1965.0,
1965.0, 1975.0, 1975.0], [1955.0, 1955.0, 1970.0, 1970.0], [1945.0,
1945.0, 1962.5, 1962.5], [1935.0, 1935.0, 1953.75, 1953.75], [1925.0,
1925.0, 1944.375, 1944.375], [1995.0, 1995.0, 2005.0, 2005.0],
[1985.0, 1985.0, 2000.0, 2000.0], [1934.6875, 1934.6875, 1992.5,
1992.5], [2025.0, 2025.0, 2035.0, 2035.0], [2045.0, 2045.0, 2055.0,
2055.0], [2030.0, 2030.0, 2050.0, 2050.0], [2015.0, 2015.0, 2040.0,
2040.0], [2085.0, 2085.0, 2095.0, 2095.0], [2075.0, 2075.0, 2090.0,
2090.0], [2065.0, 2065.0, 2082.5, 2082.5], [2027.5, 2027.5, 2073.75,
2073.75], [1963.59375, 1963.59375, 2050.625, 2050.625], [2105.0,
2105.0, 2115.0, 2115.0], [2125.0, 2125.0, 2135.0, 2135.0], [2110.0,
2110.0, 2130.0, 2130.0], [2155.0, 2155.0, 2165.0, 2165.0], [2145.0,
2145.0, 2160.0, 2160.0], [2120.0, 2120.0, 2152.5, 2152.5],
[2007.109375, 2007.109375, 2136.25, 2136.25], [1863.75, 1863.75,
2071.6796875, 2071.6796875], [1805.0, 1805.0, 1967.71484375,
1967.71484375], [798.616943359375, 798.616943359375, 1886.357421875,
1886.357421875], [2185.0, 2185.0, 2195.0, 2195.0], [2235.0, 2235.0,
2245.0, 2245.0], [2225.0, 2225.0, 2240.0, 2240.0], [2215.0, 2215.0,
2232.5, 2232.5], [2205.0, 2205.0, 2223.75, 2223.75], [2190.0, 2190.0,
2214.375, 2214.375], [2255.0, 2255.0, 2265.0, 2265.0], [2275.0,
2275.0, 2285.0, 2285.0], [2295.0, 2295.0, 2305.0, 2305.0], [2345.0,
2345.0, 2355.0, 2355.0], [2335.0, 2335.0, 2350.0, 2350.0], [2325.0,
2325.0, 2342.5, 2342.5], [2315.0, 2315.0, 2333.75, 2333.75], [2300.0,
2300.0, 2324.375, 2324.375], [2280.0, 2280.0, 2312.1875, 2312.1875],
[2260.0, 2260.0, 2296.09375, 2296.09375], [2202.1875, 2202.1875,
2278.046875, 2278.046875], [2365.0, 2365.0, 2375.0, 2375.0], [2395.0,
2395.0, 2405.0, 2405.0], [2385.0, 2385.0, 2400.0, 2400.0], [2370.0,
2370.0, 2392.5, 2392.5], [2425.0, 2425.0, 2435.0, 2435.0], [2415.0,
2415.0, 2430.0, 2430.0], [2445.0, 2445.0, 2455.0, 2455.0], [2422.5,

2422.5, 2450.0, 2450.0], [2465.0, 2465.0, 2475.0, 2475.0], [2485.0,
2485.0, 2495.0, 2495.0], [2470.0, 2470.0, 2490.0, 2490.0], [2535.0,
2535.0, 2545.0, 2545.0], [2525.0, 2525.0, 2540.0, 2540.0], [2515.0,
2515.0, 2532.5, 2532.5], [2505.0, 2505.0, 2523.75, 2523.75], [2480.0,
2480.0, 2514.375, 2514.375], [2436.25, 2436.25, 2497.1875, 2497.1875],
[2381.25, 2381.25, 2466.71875, 2466.71875], [2575.0, 2575.0, 2585.0,
2585.0], [2565.0, 2565.0, 2580.0, 2580.0], [2555.0, 2555.0, 2572.5,
2572.5], [2595.0, 2595.0, 2605.0, 2605.0], [2635.0, 2635.0, 2645.0,
2645.0], [2625.0, 2625.0, 2640.0, 2640.0], [2615.0, 2615.0, 2632.5,
2632.5], [2655.0, 2655.0, 2665.0, 2665.0], [2623.75, 2623.75, 2660.0,
2660.0], [2685.0, 2685.0, 2695.0, 2695.0], [2675.0, 2675.0, 2690.0,
2690.0], [2641.875, 2641.875, 2682.5, 2682.5], [2600.0, 2600.0,
2662.1875, 2662.1875], [2563.75, 2563.75, 2631.09375, 2631.09375],
[2423.984375, 2423.984375, 2597.421875, 2597.421875], [2240.1171875,
2240.1171875, 2510.703125, 2510.703125], [2175.0, 2175.0,
2375.41015625, 2375.41015625], [1342.4871826171875,
1342.4871826171875, 2275.205078125, 2275.205078125], [80.0, 80.0,
1808.8461303710938, 1808.8461303710938], [41.875, 41.875,
944.4230651855469, 944.4230651855469], [5.0, 5.0, 493.14903259277344,
493.14903259277344]], 'dcoord': [[0.0, 0.1772802076708238,
0.1772802076708238, 0.0], [0.0, 0.28669217330125407,
0.28669217330125407, 0.1772802076708238], [0.0, 0.3829949994171254,
0.3829949994171254, 0.28669217330125407], [0.0, 0.5318406230124716,
0.5318406230124716, 0.0], [0.3829949994171254, 1.010506268437949,
1.010506268437949, 0.5318406230124716], [0.0, 0.3033444121538131,
0.3033444121538131, 0.0], [0.0, 0.27136814262154285,
0.27136814262154285, 0.0], [0.0, 0.04324739966125413,
0.04324739966125413, 0.0], [0.0, 0.07643222973971849,
0.07643222973971849, 0.04324739966125413], [0.0, 0.08703283508347204,
0.08703283508347204, 0.07643222973971849], [0.0, 0.2901041411445724,
0.2901041411445724, 0.08703283508347204], [0.0, 0.34794838893694313,
0.34794838893694313, 0.2901041411445724], [0.0, 0.5040975564888103,
0.5040975564888103, 0.34794838893694313], [0.27136814262154285,
0.5394349555733607, 0.5394349555733607, 0.5040975564888103], [0.0,
0.10422480315570333, 0.10422480315570333, 0.0], [0.0,
0.2970778548907218, 0.2970778548907218, 0.10422480315570333], [0.0,
0.5206618198810536, 0.5206618198810536, 0.2970778548907218], [0.0,
0.6683264023026952, 0.6683264023026952, 0.5206618198810536],
[0.5394349555733607, 0.8492691489489239, 0.8492691489489239,
0.6683264023026952], [0.0, 0.0, 0.0, 0.0], [0.0, 0.04324739966125435,
0.04324739966125435, 0.0], [0.0, 0.08315668701021096,
0.08315668701021096, 0.04324739966125435], [0.0, 0.12990108838123754,
0.12990108838123754, 0.0], [0.08315668701021096, 0.1915554769618948,
0.1915554769618948, 0.12990108838123754], [0.0, 0.03876687751736324,
0.03876687751736324, 0.0], [0.0, 0.05815031627604492,
0.05815031627604492, 0.03876687751736324], [0.0, 0.10758866512411748,
0.10758866512411748, 0.05815031627604492], [0.0, 0.19407212171744634,
0.19407212171744634, 0.10758866512411748], [0.1915554769618948,
0.2380505405674676, 0.2380505405674676, 0.19407212171744634], [0.0,
0.07246928218538054, 0.07246928218538054, 0.0], [0.0,

0.15114404178440957, 0.15114404178440957, 0.0], [0.07246928218538054,
0.18326019444686018, 0.18326019444686018, 0.15114404178440957], [0.0,
0.0, 0.0, 0.0], [0.0, 0.10612860137705242, 0.10612860137705242, 0.0],
[0.0, 0.038766877517363185, 0.038766877517363185, 0.0], [0.0,
0.09643245349555761, 0.09643245349555761, 0.038766877517363185], [0.0,
0.13672016769185683, 0.13672016769185683, 0.09643245349555761],
[0.10612860137705242, 0.22201957181882537, 0.22201957181882537,
0.13672016769185683], [0.18326019444686018, 0.3227742053008481,
0.3227742053008481, 0.22201957181882537], [0.0, 0.05807932824944268,
0.05807932824944268, 0.0], [0.0, 0.05807932824944268,
0.05807932824944268, 0.0], [0.0, 0.0724692821853805,
0.0724692821853805, 0.0], [0.05807932824944268, 0.11166810480416704,
0.11166810480416704, 0.0724692821853805], [0.05807932824944268,
0.2667267341686971, 0.2667267341686971, 0.11166810480416704], [0.0,
0.11615865649888524, 0.11615865649888524, 0.0], [0.0,
0.12974573697612019, 0.12974573697612019, 0.11615865649888524], [0.0,
0.05807932824944256, 0.05807932824944256, 0.0], [0.0,
0.11966341552740979, 0.11966341552740979, 0.05807932824944256], [0.0,
0.038766877517363296, 0.038766877517363296, 0.0], [0.0,
0.0947851312299946, 0.0947851312299946, 0.0], [0.038766877517363296,
0.14030339851118023, 0.14030339851118023, 0.0947851312299946],
[0.11966341552740979, 0.27340127765587086, 0.27340127765587086,
0.14030339851118023], [0.12974573697612019, 0.3433731030164257,
0.3433731030164257, 0.27340127765587086], [0.2667267341686971,
0.37171710774161276, 0.37171710774161276, 0.3433731030164257],
[0.3227742053008481, 0.487944391107957, 0.487944391107957,
0.37171710774161276], [0.2380505405674676, 0.5452661783947853,
0.5452661783947853, 0.487944391107957], [0.0, 0.1299010883812375,
0.1299010883812375, 0.0], [0.0, 0.05807932824944265,
0.05807932824944265, 0.0], [0.0, 0.1051767022663779,
0.1051767022663779, 0.05807932824944265], [0.0, 0.0947851312299946,
0.0947851312299946, 0.0], [0.0, 0.15217700621683217,
0.15217700621683217, 0.0947851312299946], [0.1051767022663779,
0.22517564925632128, 0.22517564925632128, 0.15217700621683217],
[0.1299010883812375, 0.29567074168428403, 0.29567074168428403,
0.22517564925632128], [0.0, 0.05807932824944265, 0.05807932824944265,
0.0], [0.0, 0.11770347546609869, 0.11770347546609869,
0.05807932824944265], [0.0, 0.0473925656149973, 0.0473925656149973,
0.0], [0.0, 0.11114362893796889, 0.11114362893796889,
0.0473925656149973], [0.0, 0.08864010383541195, 0.08864010383541195,
0.0], [0.0, 0.10653792514614674, 0.10653792514614674,
0.08864010383541195], [0.0, 0.12592945299310487, 0.12592945299310487,
0.10653792514614674], [0.0, 0.1731879443780272, 0.1731879443780272,
0.12592945299310487], [0.11114362893796889, 0.24493764837632237,
0.24493764837632237, 0.1731879443780272], [0.11770347546609869,
0.3383258710017307, 0.3383258710017307, 0.24493764837632237],
[0.29567074168428403, 0.5560577284369612, 0.5560577284369612,
0.3383258710017307], [0.0, 0.0473925656149973, 0.0473925656149973,
0.0], [0.0, 0.21490840079881918, 0.21490840079881918,
0.0473925656149973], [0.0, 0.3234381751976381, 0.3234381751976381,

0.21490840079881918], [0.0, 0.08864010383541195, 0.08864010383541195,
0.0], [0.0, 0.11510496217250074, 0.11510496217250074,
0.08864010383541195], [0.0, 0.17240340740767282, 0.17240340740767282,
0.11510496217250074], [0.0, 0.22688339699538407, 0.22688339699538407,
0.17240340740767282], [0.0, 0.2975275570491906, 0.2975275570491906,
0.22688339699538407], [0.0, 0.08864010383541174, 0.08864010383541174,
0.0], [0.0, 0.11510496217250071, 0.11510496217250071,
0.08864010383541174], [0.0, 0.1421776968449917, 0.1421776968449917,
0.0], [0.0, 0.05807932824944268, 0.05807932824944268, 0.0], [0.0,
0.0805546930103961, 0.0805546930103961, 0.05807932824944268], [0.0,
0.149952447981651, 0.149952447981651, 0.0805546930103961],
[0.1421776968449917, 0.20281292151967723, 0.20281292151967723,
0.149952447981651], [0.0, 0.2288143530219625, 0.2288143530219625,
0.20281292151967723], [0.11510496217250071, 0.31164038078487116,
0.31164038078487116, 0.2288143530219625], [0.2975275570491906,
0.44391028945567046, 0.44391028945567046, 0.31164038078487116],
[0.3234381751976381, 0.610578938773632, 0.610578938773632,
0.44391028945567046], [0.5560577284369612, 0.7979165492125964,
0.7979165492125964, 0.610578938773632], [0.5452661783947853,
0.9372973710321736, 0.9372973710321736, 0.7979165492125964],
[0.8492691489489239, 1.1606672014800572, 1.1606672014800572,
0.9372973710321736], [0.0, 0.21755959884353346, 0.21755959884353346,
0.0], [0.0, 0.31250283913164345, 0.31250283913164345, 0.0],
[0.21755959884353346, 0.4374654844795404, 0.4374654844795404,
0.31250283913164345], [0.0, 0.0864947993225087, 0.0864947993225087,
0.0], [0.0, 0.16409481242150697, 0.16409481242150697,
0.0864947993225087], [0.0, 0.19828853646177635, 0.19828853646177635,
0.0], [0.0, 0.23696282807498642, 0.23696282807498642, 0.0],
[0.19828853646177635, 0.29673269674824365, 0.29673269674824365,
0.23696282807498642], [0.16409481242150697, 0.4770645170284533,
0.4770645170284533, 0.29673269674824365], [0.4374654844795404,
0.7257601680189937, 0.7257601680189937, 0.4770645170284533], [0.0,
0.07246928218538053, 0.07246928218538053, 0.0], [0.0,
0.08870986141055368, 0.08870986141055368, 0.07246928218538053], [0.0,
0.13118214777649403, 0.13118214777649403, 0.0], [0.08870986141055368,
0.2254096502543967, 0.2254096502543967, 0.13118214777649403], [0.0,
0.18773173787244626, 0.18773173787244626, 0.0], [0.0,
0.2815976068086694, 0.2815976068086694, 0.18773173787244626],
[0.2254096502543967, 0.40261577404437365, 0.40261577404437365,
0.2815976068086694], [0.0, 0.07753375503472654, 0.07753375503472654,
0.0], [0.0, 0.03876687751736324, 0.03876687751736324, 0.0], [0.0,
0.050663363955348484, 0.050663363955348484, 0.03876687751736324],
[0.0, 0.08844472972549859, 0.08844472972549859, 0.050663363955348484],
[0.07753375503472654, 0.1700417262397021, 0.1700417262397021,
0.08844472972549859], [0.0, 0.16090841344475296, 0.16090841344475296,
0.0], [0.0, 0.04739256561499729, 0.04739256561499729, 0.0], [0.0,
0.07753375503472652, 0.07753375503472652, 0.0], [0.0, 0.0, 0.0, 0.0],
[0.0, 0.08877961898569539, 0.08877961898569539, 0.0],
[0.07753375503472652, 0.11314272282182954, 0.11314272282182954,
0.08877961898569539], [0.04739256561499729, 0.1361544862748071,

0.1361544862748071, 0.11314272282182954], [0.0, 0.17143567700515133,
0.17143567700515133, 0.1361544862748071], [0.16090841344475296,
0.2464967373948427, 0.2464967373948427, 0.17143567700515133],
[0.1700417262397021, 0.28825981016425795, 0.28825981016425795,
0.2464967373948427], [0.0, 0.0473925656149973, 0.0473925656149973,
0.0], [0.0, 0.05807932824944266, 0.05807932824944266, 0.0],
[0.0473925656149973, 0.08462922560035634, 0.08462922560035634,
0.05807932824944266], [0.0, 0.14032317999916588, 0.14032317999916588,
0.08462922560035634], [0.0, 0.043247399661254335,
0.043247399661254335, 0.0], [0.0, 0.043247399661254335,
0.043247399661254335, 0.0], [0.0, 0.08315668701021096,
0.08315668701021096, 0.043247399661254335], [0.043247399661254335,
0.12884682051394789, 0.12884682051394789, 0.08315668701021096], [0.0,
0.0, 0.0, 0.0], [0.0, 0.04324739966125432, 0.04324739966125432, 0.0],
[0.0, 0.08128237635171615, 0.08128237635171615, 0.04324739966125432],
[0.0, 0.10401957592297673, 0.10401957592297673, 0.08128237635171615],
[0.0, 0.2064030748135318, 0.2064030748135318, 0.10401957592297673],
[0.12884682051394789, 0.25815534033234694, 0.25815534033234694,
0.2064030748135318], [0.14032317999916588, 0.2802746743613287,
0.2802746743613287, 0.25815534033234694], [0.0, 0.39508093589281706,
0.39508093589281706, 0.2802746743613287], [0.28825981016425795,
0.4988272767095121, 0.4988272767095121, 0.39508093589281706],
[0.40261577404437365, 0.6701100214612886, 0.6701100214612886,
0.4988272767095121], [0.0, 0.1163006325520898, 0.1163006325520898,
0.0], [0.0, 0.2057668986806288, 0.2057668986806288,
0.1163006325520898], [0.0, 0.0947851312299946, 0.0947851312299946,
0.0], [0.0, 0.11615865649888532, 0.11615865649888532, 0.0], [0.0,
0.12990108838123757, 0.12990108838123757, 0.0], [0.11615865649888532,
0.2406029295098849, 0.2406029295098849, 0.12990108838123757],
[0.0947851312299946, 0.39463386077840623, 0.39463386077840623,
0.2406029295098849], [0.2057668986806288, 0.44735207565821966,
0.44735207565821966, 0.39463386077840623], [0.0, 0.0, 0.0, 0.0], [0.0,
0.08877961898569535, 0.08877961898569535, 0.0], [0.0,
0.14176242657527122, 0.14176242657527122, 0.08877961898569535], [0.0,
0.019383438758681648, 0.019383438758681648, 0.0], [0.0,
0.05815031627604483, 0.05815031627604483, 0.0], [0.0,
0.1537301631763142, 0.1537301631763142, 0.05815031627604483],
[0.019383438758681648, 0.24156913878098624, 0.24156913878098624,
0.1537301631763142], [0.14176242657527122, 0.3005613204843993,
0.3005613204843993, 0.24156913878098624], [0.0, 0.31328394079323124,
0.31328394079323124, 0.3005613204843993], [0.0, 0.04739256561499726,
0.04739256561499726, 0.0], [0.0, 0.12916817667193858,
0.12916817667193858, 0.04739256561499726], [0.0, 0.21284450707429337,
0.21284450707429337, 0.12916817667193858], [0.0, 0.08864010383541195,
0.08864010383541195, 0.0], [0.0, 0.14240963726141137,
0.14240963726141137, 0.0], [0.0, 0.20610936083364534,
0.20610936083364534, 0.14240963726141137], [0.08864010383541195,
0.2678787960838747, 0.2678787960838747, 0.20610936083364534],
[0.21284450707429337, 0.34898420883290376, 0.34898420883290376,
0.2678787960838747], [0.31328394079323124, 0.5652986785897635,

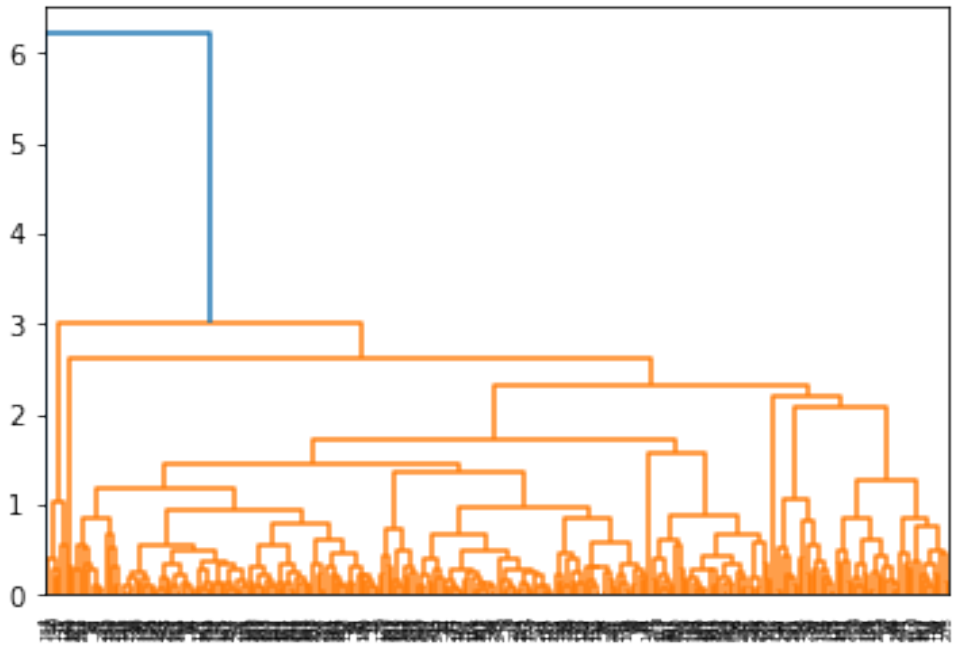
0.5652986785897635, 0.34898420883290376], [0.44735207565821966,
0.8475430417634661, 0.8475430417634661, 0.5652986785897635],
[0.6701100214612886, 0.973735158662314, 0.973735158662314,
0.8475430417634661], [0.7257601680189937, 1.3499305595491995,
1.3499305595491995, 0.973735158662314], [1.1606672014800572,
1.4393840429202949, 1.4393840429202949, 1.3499305595491995], [0.0,
0.16194437517225269, 0.16194437517225269, 0.0], [0.0,
0.07753375503472659, 0.07753375503472659, 0.0], [0.0,
0.08864010383541195, 0.08864010383541195, 0.0], [0.07753375503472659,
0.15727928784996995, 0.15727928784996995, 0.08864010383541195], [0.0,
0.2051542574338655, 0.2051542574338655, 0.15727928784996995],
[0.16194437517225269, 0.3909416494528098, 0.3909416494528098,
0.2051542574338655], [0.0, 0.16194437517225288, 0.16194437517225288,
0.0], [0.0, 0.2619999820145311, 0.2619999820145311,
0.16194437517225288], [0.0, 0.5320224621320837, 0.5320224621320837,
0.2619999820145311], [0.3909416494528098, 0.619129723571025,
0.619129723571025, 0.5320224621320837], [0.0, 0.038766877517363296,
0.038766877517363296, 0.0], [0.0, 0.05993092390018888,
0.05993092390018888, 0.038766877517363296], [0.0, 0.07849968369018337,
0.07849968369018337, 0.05993092390018888], [0.0, 0.13491277485983505,
0.13491277485983505, 0.07849968369018337], [0.0, 0.18569609527314884,
0.18569609527314884, 0.13491277485983505], [0.0, 0.10422480315570327,
0.10422480315570327, 0.0], [0.0, 0.26501756698738566,
0.26501756698738566, 0.10422480315570327], [0.18569609527314884,
0.27955099622023943, 0.27955099622023943, 0.26501756698738566], [0.0,
0.0, 0.0, 0.0], [0.0, 0.08864010383541195, 0.08864010383541195, 0.0],
[0.0, 0.1253619970789742, 0.1253619970789742, 0.08864010383541195],
[0.0, 0.1964908844304668, 0.1964908844304668, 0.1253619970789742],
[0.0, 0.07753375503472659, 0.07753375503472659, 0.0], [0.0,
0.18618035117565854, 0.18618035117565854, 0.07753375503472659], [0.0,
0.23951325984283414, 0.23951325984283414, 0.18618035117565854],
[0.1964908844304668, 0.33470213683199407, 0.33470213683199407,
0.23951325984283414], [0.27955099622023943, 0.427608227392133,
0.427608227392133, 0.33470213683199407], [0.0, 0.05807932824944267,
0.05807932824944267, 0.0], [0.0, 0.058150316276045055,
0.058150316276045055, 0.0], [0.05807932824944267, 0.19954336158388125,
0.19954336158388125, 0.058150316276045055], [0.0, 0.14493856437076114,
0.14493856437076114, 0.0], [0.0, 0.3320865746958648,
0.3320865746958648, 0.14493856437076114], [0.19954336158388125,
0.5861907955792752, 0.5861907955792752, 0.3320865746958648],
[0.427608227392133, 0.6594698810384934, 0.6594698810384934,
0.5861907955792752], [0.619129723571025, 0.86920891492827,
0.86920891492827, 0.6594698810384934], [0.0, 1.5543463399506283,
1.5543463399506283, 0.86920891492827], [1.4393840429202949,
1.701435618838162, 1.701435618838162, 1.5543463399506283], [0.0,
0.04324739966125435, 0.04324739966125435, 0.0], [0.0,
0.08864010383541193, 0.08864010383541193, 0.0], [0.0,
0.1748919645715673, 0.1748919645715673, 0.08864010383541193], [0.0,
0.22100249208438139, 0.22100249208438139, 0.1748919645715673], [0.0,
0.42841914332600123, 0.42841914332600123, 0.22100249208438139],

[0.04324739966125435, 0.5065805378342119, 0.5065805378342119,
0.42841914332600123], [0.0, 0.4062304529528744, 0.4062304529528744,
0.0], [0.0, 0.21740784655614168, 0.21740784655614168, 0.0], [0.0,
0.26633885695708626, 0.26633885695708626, 0.0], [0.0,
0.019383438758681648, 0.019383438758681648, 0.0], [0.0,
0.13046217338012855, 0.13046217338012855, 0.019383438758681648], [0.0,
0.19801476056630118, 0.19801476056630118, 0.13046217338012855], [0.0,
0.278878698876434, 0.278878698876434, 0.19801476056630118],
[0.26633885695708626, 0.3740206202431062, 0.3740206202431062,
0.278878698876434], [0.21740784655614168, 0.5543436073444352,
0.5543436073444352, 0.3740206202431062], [0.4062304529528744,
0.828367981386839, 0.828367981386839, 0.5543436073444352],
[0.5065805378342119, 1.0481143526945198, 1.0481143526945198,
0.828367981386839], [0.0, 0.34013596395367035, 0.34013596395367035,
0.0], [0.0, 0.11615865649888532, 0.11615865649888532, 0.0], [0.0,
0.37978661987551526, 0.37978661987551526, 0.11615865649888532],
[0.34013596395367035, 0.4996496974054922, 0.4996496974054922,
0.37978661987551526], [0.0, 0.08877961898569539, 0.08877961898569539,
0.0], [0.0, 0.19587787438730803, 0.19587787438730803,
0.08877961898569539], [0.0, 0.2300938760187676, 0.2300938760187676,
0.0], [0.19587787438730803, 0.3466293983461286, 0.3466293983461286,
0.2300938760187676], [0.0, 0.038766877517363185, 0.038766877517363185,
0.0], [0.0, 0.17755923797139078, 0.17755923797139078, 0.0],
[0.038766877517363185, 0.24847398608524593, 0.24847398608524593,
0.17755923797139078], [0.0, 0.0473925656149973, 0.0473925656149973,
0.0], [0.0, 0.1291681766719386, 0.1291681766719386,
0.0473925656149973], [0.0, 0.16433792564251928, 0.16433792564251928,
0.1291681766719386], [0.0, 0.2983760572279156, 0.2983760572279156,
0.16433792564251928], [0.24847398608524593, 0.43109740121433515,
0.43109740121433515, 0.2983760572279156], [0.3466293983461286,
0.6011790168712188, 0.6011790168712188, 0.43109740121433515],
[0.4996496974054922, 0.8319225172846985, 0.8319225172846985,
0.6011790168712188], [0.0, 0.27456737371884365, 0.27456737371884365,
0.0], [0.0, 0.34644128720336986, 0.34644128720336986,
0.27456737371884365], [0.0, 0.4543571839783711, 0.4543571839783711,
0.34644128720336986], [0.0, 0.3592972966920929, 0.3592972966920929,
0.0], [0.0, 0.09691719379340813, 0.09691719379340813, 0.0], [0.0,
0.14116309478045158, 0.14116309478045158, 0.09691719379340813], [0.0,
0.22604535488839375, 0.22604535488839375, 0.14116309478045158], [0.0,
0.22971704901093315, 0.22971704901093315, 0.0], [0.22604535488839375,
0.33227333280050786, 0.33227333280050786, 0.22971704901093315], [0.0,
0.11615865649888536, 0.11615865649888536, 0.0], [0.0,
0.4503631980996454, 0.4503631980996454, 0.11615865649888536],
[0.33227333280050786, 0.481604228795339, 0.481604228795339,
0.4503631980996454], [0.3592972966920929, 0.7446627867996847,
0.7446627867996847, 0.481604228795339], [0.4543571839783711,
0.8585980933661276, 0.8585980933661276, 0.7446627867996847],
[0.8319225172846985, 1.2506282413282697, 1.2506282413282697,
0.8585980933661276], [1.0481143526945198, 2.0652863077768178,
2.0652863077768178, 1.2506282413282697], [0.0, 2.204175010841276,

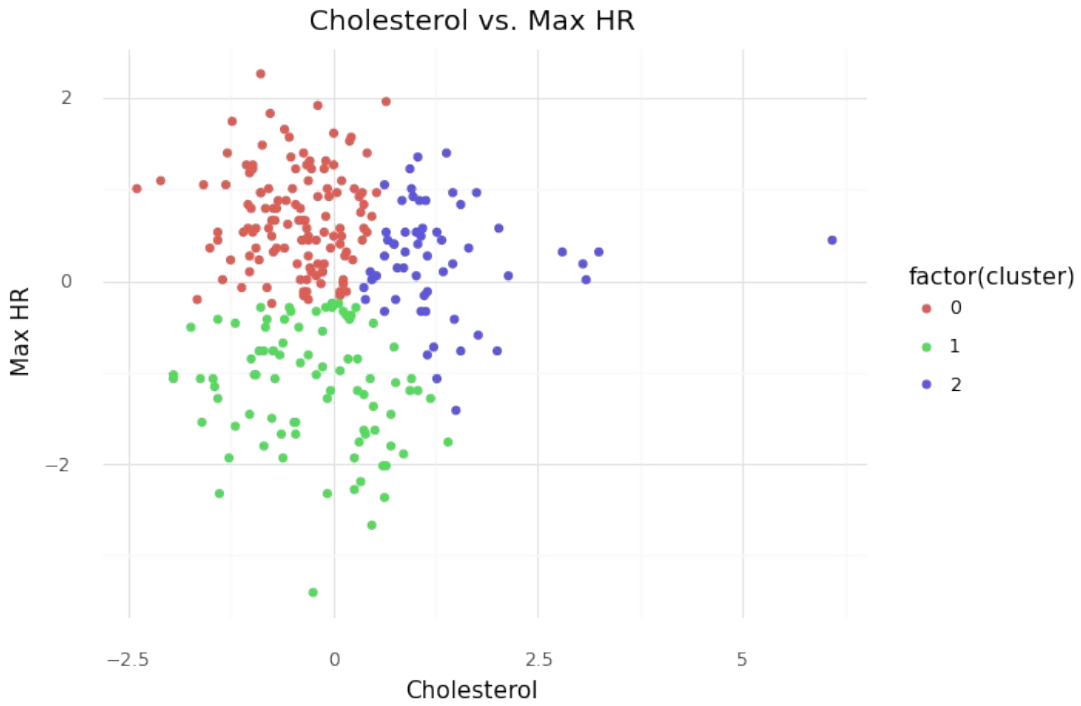
2.204175010841276, 2.0652863077768178], [1.701435618838162,
2.3226411203410953, 2.3226411203410953, 2.204175010841276],
[0.3033444121538131, 2.618586614269613, 2.618586614269613,
2.3226411203410953], [1.010506268437949, 3.010631685851865,
3.010631685851865, 2.618586614269613], [0.0, 6.2123859266300405,
6.2123859266300405, 3.010631685851865]], 'ivl': ['1', '188', '52',
'9', '181', '71', '123', '60', '100', '174', '222', '229', '224',
'51', '53', '47', '19', '77', '214', '180', '138', '39', '251', '129',
'146', '10', '236', '99', '190', '90', '142', '134', '28', '159',
'62', '264', '148', '219', '78', '155', '215', '139', '32', '83',
'94', '55', '127', '125', '265', '158', '179', '17', '43', '128',
'151', '197', '254', '45', '212', '41', '164', '136', '169', '266',
'149', '263', '105', '130', '152', '243', '102', '115', '162', '107',
'238', '246', '178', '195', '166', '233', '141', '209', '66', '232',
'211', '194', '63', '216', '184', '259', '150', '21', '85', '54',
'40', '132', '160', '79', '12', '42', '5', '120', '109', '235', '182',
'108', '135', '167', '268', '38', '248', '124', '200', '240', '87',
'247', '204', '25', '249', '30', '64', '13', '234', '177', '187',
'97', '217', '68', '111', '168', '161', '207', '198', '26', '69',
'256', '242', '2', '70', '6', '202', '35', '154', '218', '104', '171',
'7', '72', '245', '58', '116', '137', '221', '29', '163', '48', '196',
'89', '220', '20', '170', '143', '44', '213', '131', '140', '34',
'84', '81', '225', '261', '31', '231', '74', '106', '36', '86', '76',
'18', '23', '144', '22', '205', '8', '113', '157', '201', '203',
'241', '260', '91', '239', '114', '14', '88', '112', '199', '255',
'192', '185', '206', '193', '24', '210', '228', '262', '96', '27',
'252', '267', '75', '230', '61', '244', '258', '67', '165', '101',
'15', '46', '147', '237', '56', '191', '253', '57', '186', '73',
'250', '80', '93', '173', '156', '11', '49', '118', '122', '145',
'133', '126', '226', '37', '4', '183', '95', '121', '59', '119',
'175', '269', '3', '65', '92', '82', '189', '257', '33', '223', '227',
'0', '110', '103', '172', '153', '176', '16', '50', '117', '98',
'208'], 'leaves': [1, 188, 52, 9, 181, 71, 123, 60, 100, 174, 222,
229, 224, 51, 53, 47, 19, 77, 214, 180, 138, 39, 251, 129, 146, 10,
236, 99, 190, 90, 142, 134, 28, 159, 62, 264, 148, 219, 78, 155, 215,
139, 32, 83, 94, 55, 127, 125, 265, 158, 179, 17, 43, 128, 151, 197,
254, 45, 212, 41, 164, 136, 169, 266, 149, 263, 105, 130, 152, 243,
102, 115, 162, 107, 238, 246, 178, 195, 166, 233, 141, 209, 66, 232,
211, 194, 63, 216, 184, 259, 150, 21, 85, 54, 40, 132, 160, 79, 12,
42, 5, 120, 109, 235, 182, 108, 135, 167, 268, 38, 248, 124, 200, 240,
87, 247, 204, 25, 249, 30, 64, 13, 234, 177, 187, 97, 217, 68, 111,
168, 161, 207, 198, 26, 69, 256, 242, 2, 70, 6, 202, 35, 154, 218,
104, 171, 7, 72, 245, 58, 116, 137, 221, 29, 163, 48, 196, 89, 220,
20, 170, 143, 44, 213, 131, 140, 34, 84, 81, 225, 261, 31, 231, 74,
106, 36, 86, 76, 18, 23, 144, 22, 205, 8, 113, 157, 201, 203, 241,
260, 91, 239, 114, 14, 88, 112, 199, 255, 192, 185, 206, 193, 24, 210,
228, 262, 96, 27, 252, 267, 75, 230, 61, 244, 258, 67, 165, 101, 15,
46, 147, 237, 56, 191, 253, 57, 186, 73, 250, 80, 93, 173, 156, 11,
49, 118, 122, 145, 133, 126, 226, 37, 4, 183, 95, 121, 59, 119, 175,
269, 3, 65, 92, 82, 189, 257, 33, 223, 227, 0, 110, 103, 172, 153,

176, 16, 50, 117, 98, 208], 'color_list': ['C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C0'], 'leaves_color_list': ['C0', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1',

```
'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1', 'C1']}
Silhouette Score:  0.710688682801176
```




```
#KMEANS
features = ["Cholesterol", "Max HR"]
X2 = DF[features]

#zscore
z = StandardScaler()
X2[features] = z.fit_transform(X2[features])

#km
km = KMeans(n_clusters = 3)
km.fit(X2[features])

membership = km.predict(X2[features])
X2["cluster"] = membership

#evaluate model
print("Silhouette Score: " , silhouette_score(X2[features],
membership))

#plot
(ggplot(X2, aes(x = "Cholesterol", y = "Max HR", color =
"factor(cluster)")) +
 geom_point() + theme_minimal() + labs(title = "Cholesterol vs. Max
HR"))

Silhouette Score:  0.372010963946033
```

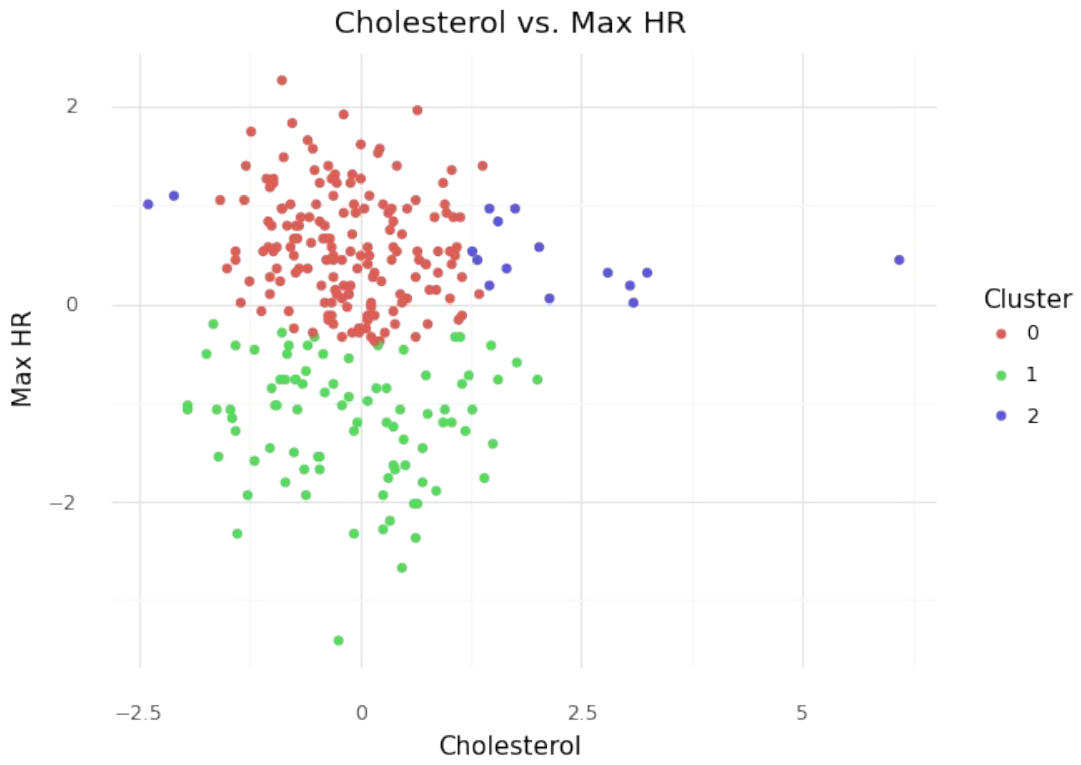



```
<ggplot: (8776361062340)>
```

The scatterplot above demonstrates the clusters obtained from the KMeans model.

```
#Gaussian Mixture Model
X3 = DF[["Cholesterol", "Max HR"]]

z = StandardScaler()
X3[["Cholesterol", "Max HR"]] = z.fit_transform(X3)

gmm = GaussianMixture(n_components = 3)
gmm.fit(X3)

X3["cluster"] = gmm.predict(X3)

print("Silhouette Score: ", silhouette_score(X3[features],
X3["cluster"]))

(ggplot(X3, aes(x = "Cholesterol", y = "Max HR", color =
"factor(cluster)")) + geom_point() +
 theme_minimal() + labs(title = "Cholesterol vs. Max HR") +
 scale_color_discrete(name = "Cluster"))
```

```
Silhouette Score:  0.3605276137121836
```

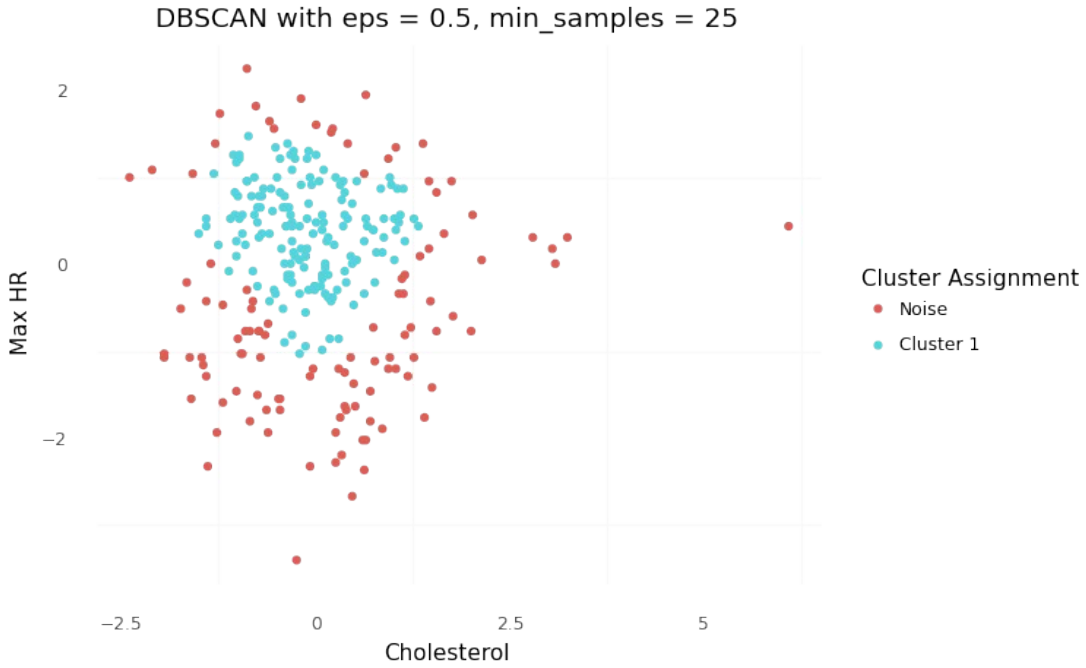



```
<ggplot: (8776361056859)>
```

The scatterplot above demonstrates the clusters found between Max HR and Cholesterol from the GMM model.

```
#DBSCAN
features = ["Cholesterol", "Max HR"]
X4 = DF[features]

#zscore
z = StandardScaler()
X4[features] = z.fit_transform(X2[features])


db = DBSCAN(eps = 0.5, min_samples = 25)
db.fit(X4[features])

labsList = ["Noise"]
labsList = labsList  + ["Cluster " + str(i) for i in
range(1,len(set(db.labels_)))]

X4["assignments"] = db.labels_

print((ggplot(X4, aes(x = "Cholesterol", y = "Max HR", color =
"factor(assignments)")) +
 geom_point() +
```

```
 theme_minimal() +
 scale_color_discrete(name = "Cluster Assignment",
                      labels = labsList) +
theme(panel_grid_major = element_blank()) +
labs(title = "DBSCAN with eps = 0.5, min_samples = 25")))

print("DBCAN Silhouette Score: ",
silhouette_score(X4[["Cholesterol","Max HR"]], X4["assignments"]))
```




```
DBCAN Silhouette Score:  0.2491147835771584
```

The scatterplot above demonstrates the clusters found between Max HR and Cholesterol from the DBSCAN model with EPS = 0.5 and min_samples = 25.

QUESTION 5:

When running a decision tree on our data, how many leaves is our tree made up of? How deep is our tree? How accurate is the model running? Is our tree overfit? How can we tell?

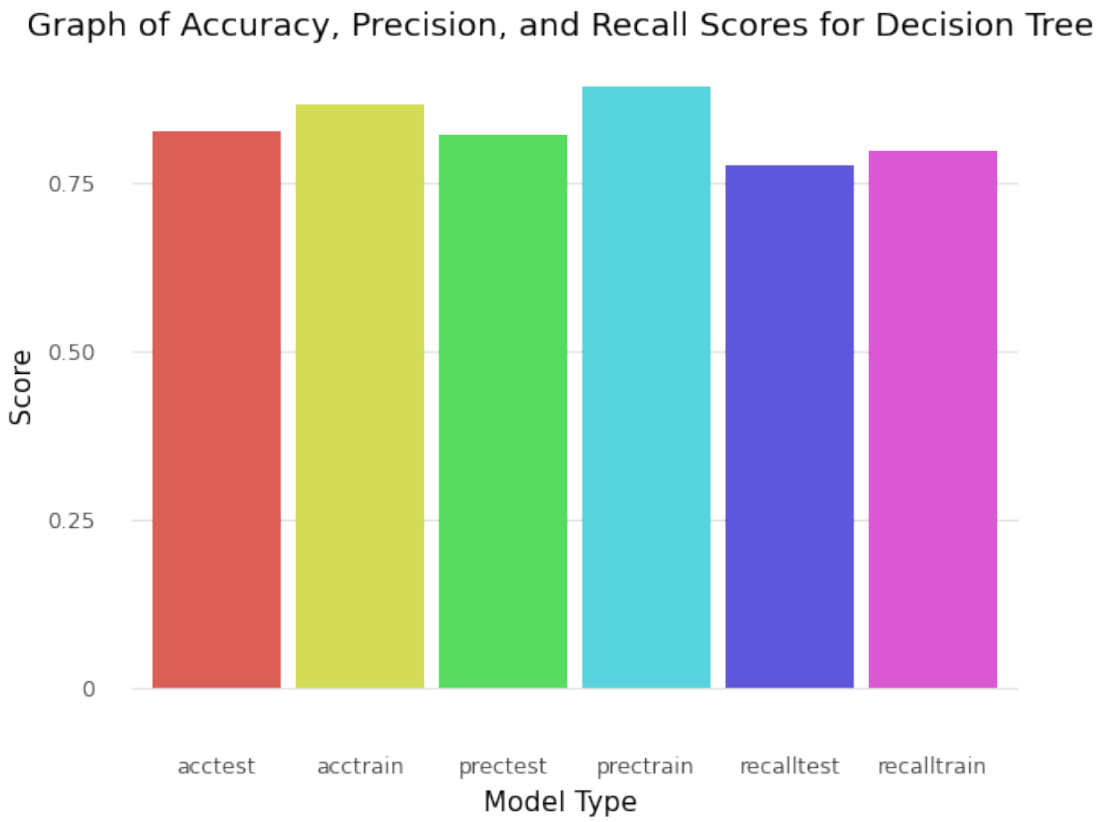
```
# Decision Tree
feats = ["Age", "Sex", "Chest pain type", "BP", "Cholesterol", "FBS
over 120", "EKG results", "Max HR", "Exercise angina",
         "ST depression", "Slope of ST", "Thallium", "Number of
vessels fluro"]

X7 = DF[feats]

# Splitting, Z-scoring
X_train, X_test, y_train, y_test = train_test_split(X7, y, test_size =
```

```
0.3,
                                                    random_state = 5)
z = StandardScaler()
z.fit(X_train[cont])

X_train[cont] = z.transform(X_train[cont])
X_test[cont] = z.transform(X_test[cont])

# Creating Model (Set min samples leaf to 10)
tree = DecisionTreeClassifier(min_samples_leaf = 10)
tree.fit(X_train, y_train)

DecisionTreeClassifier(min_samples_leaf=10)

from traitlets.config.application import
TRAITLETS_APPLICATION_RAISE_CONFIG_FILE_ERROR
# Printing tree metrics
print("Train Acc: ", accuracy_score(y_train, tree.predict(X_train)))
print("Test Acc: ", accuracy_score(y_test, tree.predict(X_test)))

trainacc = accuracy_score(y_train, tree.predict(X_train))
testacc = accuracy_score(y_test, tree.predict(X_test))

print("TEST Precision : ", precision_score(y_train,
tree.predict(X_train)))
print("TRAIN Precision: ", precision_score(y_test,
tree.predict(X_test)))

trainprec = precision_score(y_train, tree.predict(X_train))
testprec = precision_score(y_test, tree.predict(X_test))

print("TEST Recall : ", recall_score(y_test, tree.predict(X_test)))
print("TRAIN Recall: ", recall_score(y_train, tree.predict(X_train)))

trainrec = recall_score(y_train, tree.predict(X_train))
testrec = recall_score(y_test, tree.predict(X_test))

print("TEST ROC/AUC : ", roc_auc_score(y_test,
tree.predict_proba(X_test)[:,1]))
print("TRAIN ROC/AUC: ", roc_auc_score(y_train,
tree.predict_proba(X_train)[:,1]))

trainroc = roc_auc_score(y_train, tree.predict_proba(X_train)[:,1])
testroc = roc_auc_score(y_test, tree.predict_proba(X_test)[:,1])

# Creating New Dataframes for use in visualizations
score = ["acctrain", "acctest", "prectrain", "prectest",
"recalltrain", "recalltest"]
roc = ["testroc", "trainroc"]
```

```
accuracy = [trainacc, testacc, trainprec, testprec, trainrec, testrec]
roc_auc = [testroc, trainroc]

newDF = pd.DataFrame({"modeltype": score, "Accuracy": accuracy})
newDF2 = pd.DataFrame({"modeltype": roc, "ROC/AUC": roc_auc})
```

```
Train Acc:  0.8677248677248677
Test Acc:  0.8271604938271605
TEST Precision :  0.8933333333333333
TRAIN Precision:  0.8235294117647058
TEST Recall :  0.7777777777777778
TRAIN Recall:  0.7976190476190477
TEST ROC/AUC :  0.8283950617283949
TRAIN ROC/AUC:  0.9407596371882087
```

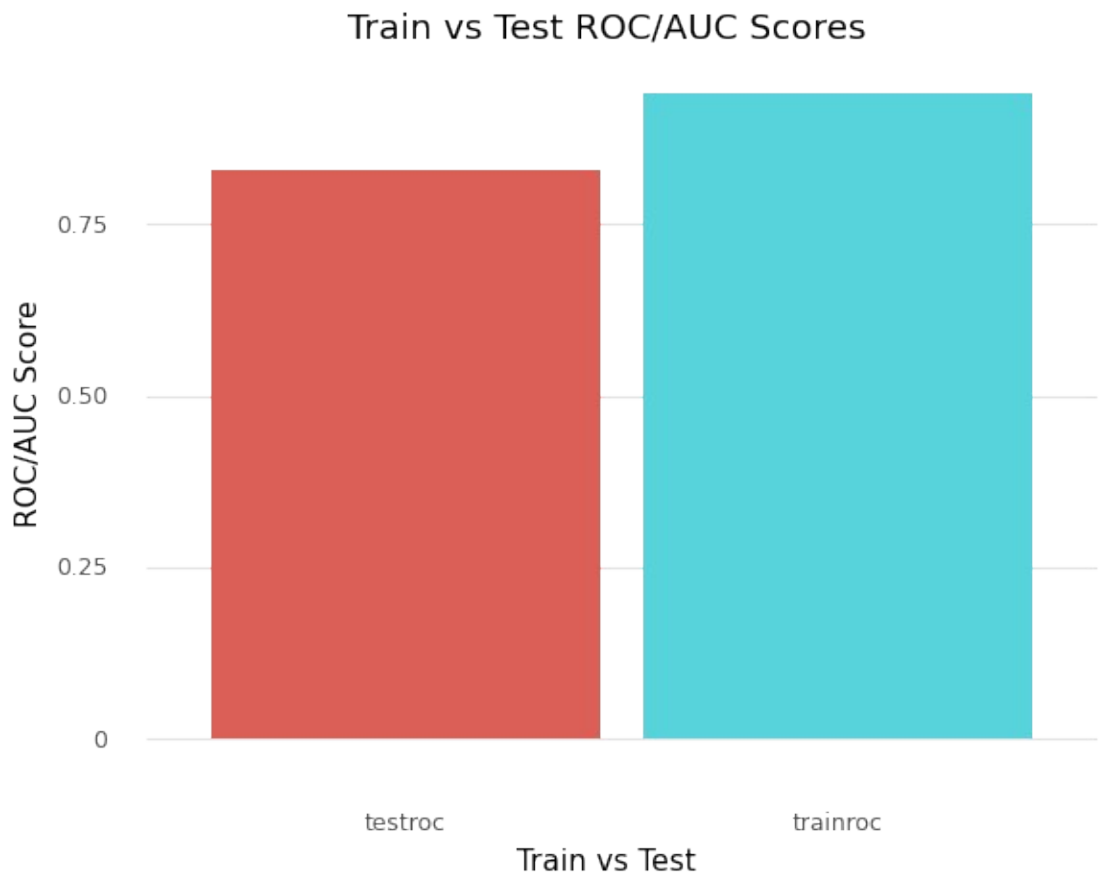
```
(ggplot(newDF, aes(x = "score", y = "accuracy", fill = "score")) +
 geom_bar(stat = "identity") +
 labs(x = "Model Type", y = "Score") +
 ggtitle("Graph of Accuracy, Precision, and Recall Scores for Decision
Tree") +
 theme_minimal() +
 theme(panel_grid_major_x = element_blank(),
     panel_grid_minor_x = element_blank(),
      panel_grid_minor_y = element_blank(),
      legend_position = "none"))
```




```
<ggplot: (8776361475750)>
```

The above bar graph demonstrates the accuracy scores, precision scores, and recall scores for both the train and test sets of the decision tree.

```
(ggplot(newDF2, aes(x = "roc", y = "roc_auc", fill = "roc")) +
 geom_bar(stat = "identity") +
 labs(x = "Train vs Test", y = "ROC/AUC Score") +
 ggtitle("Train vs Test ROC/AUC Scores") +
 theme_minimal() +
 theme(panel_grid_major_x = element_blank(),
     panel_grid_minor_x = element_blank(),
      panel_grid_minor_y = element_blank(),
      legend_position = "none"))
```

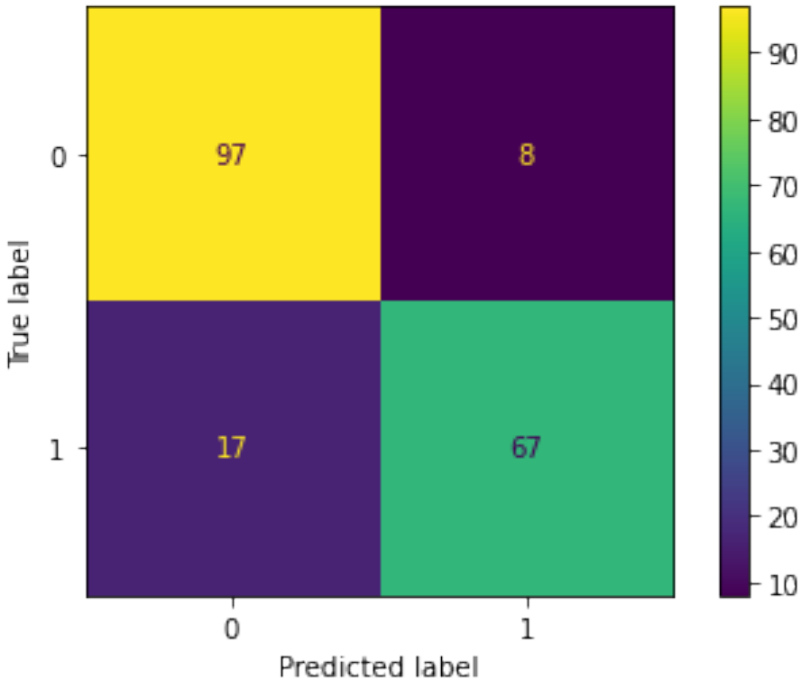



```
<ggplot: (8776361242164)>
```

The bar graph above visually demonstrates the difference in the ROC/AUC scores between the train and test sets.

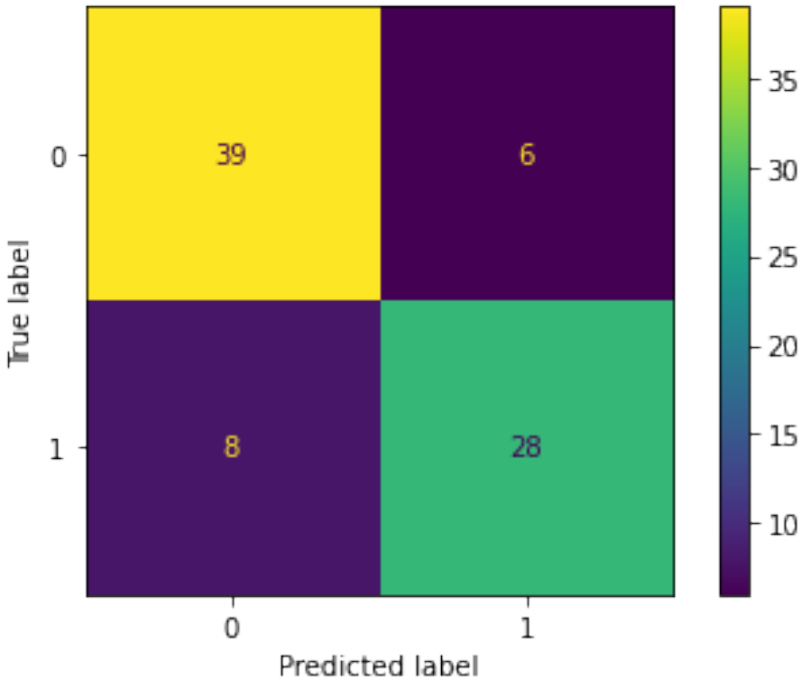
```
# Training Confusion Matrix for Tree
plot_confusion_matrix(tree, X_train, y_train)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fb67e571d00>
```




```
# Testing Confusion Matric for Tree
plot_confusion_matrix(tree, X_test, y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fb67e6fa5e0>
```



```
# Analyzing Depth of the Tree
print("Tree Depth: ", tree.get_depth())
print("Number of Leaves: ", tree.get_n_leaves())
```

```
Tree Depth:  4
Number of Leaves:  12
```

ANSWER TO QUESTION 5

In order to answer this question, we first have to make sure all imports are in order, and then we can begin creating our tree. We created this tree to predict the presense of Heart Disease based on all of our predictors using min leafs of 10. We then analyzed our accuracy, precision, recall, and ROC/AUC for training and testing data. In addtion to this, we printed our confusion matrices for both training and testing. The testing accuracy score for our tree was 0.8272, which means that it predicts Heart Disease correctly roughly 83 percent of the time on unseen data.

We know that tree is not overfit since our training and testing accuracy does not differ drastically; there is only a 4% deviation between the two accuracies.

We determined there were 12 leaves in our tree by using tree.get_n_leaves(), and found out that our tree had a depth of 4 by using tree.getdepth(). This means that our tree has four levels (depth) and a final count of 12 leaves at the bottom of our tree.

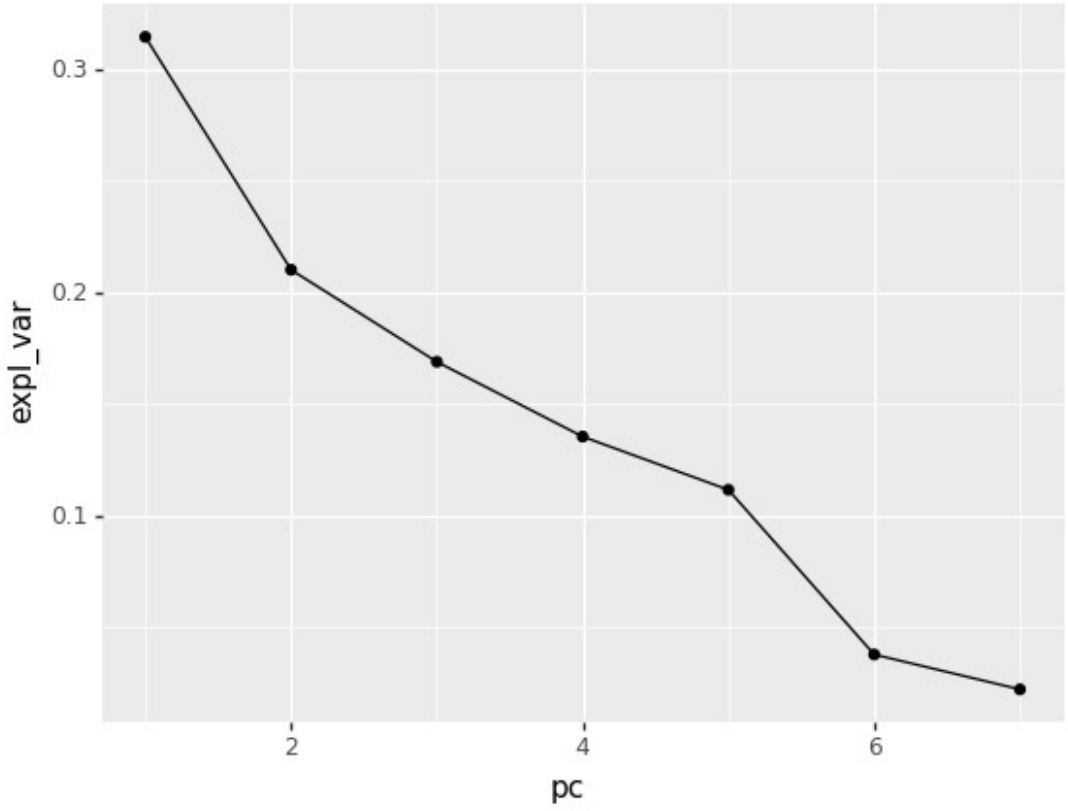
QUESTION 6: When comparing a logistic regression model using PCA on all of the predictors retaining enough PCs to keep 90% of the variance to the logistic regression model with original variables in question 1, how much of a difference is there in the accuracy score when predicting heart disease? Should the PCs be utilized over the original predictors in this model?

```
#PCA

df1vals = ["BP", "Cholesterol", "Max HR", "ST depression", "Number of
vessels fluro", "Sex", "FBS over 120", "Heart Disease_Presence"]
features1 = ["BP", "Cholesterol", "Max HR", "ST depression", "Number
of vessels fluro", "Sex", "FBS over 120"]
cont = ["BP", "Cholesterol", "Max HR", "ST depression", "Number of
vessels fluro"]

df1 = pd.DataFrame(DF[df1vals])

#zscore
z = StandardScaler()
df1[cont] = z.fit_transform(df1[cont])

#PCA
pca = PCA()
pca.fit(df1[features1])

#PCA data frame
```

```
pcaDF = pd.DataFrame({"expl_var" :
                      pca.explained_variance_ratio_,
                      "pc": range(1,8),
                      "cum_var":
                      pca.explained_variance_ratio_.cumsum()})

pcaDF.head()
```

```
   expl_var  pc   cum_var
0  0.314298   1  0.314298
1  0.210028   2  0.524326
2  0.168823   3  0.693150
3  0.135346   4  0.828496
4  0.111524   5  0.940020
```

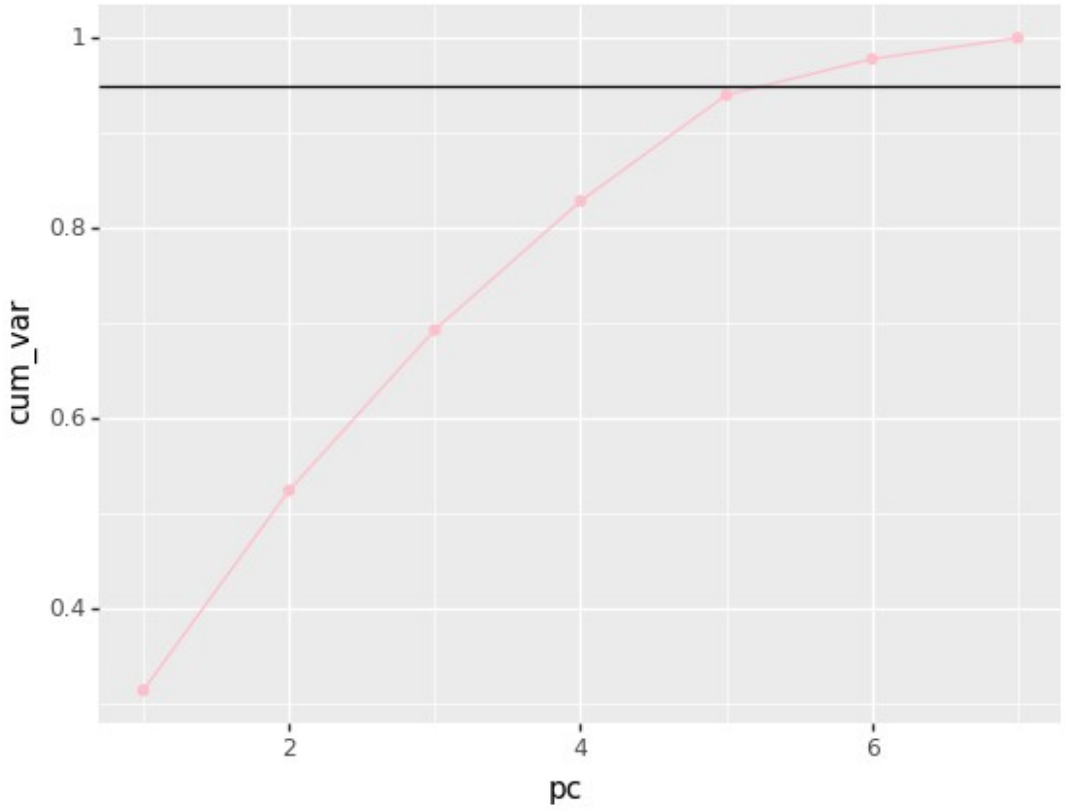
```
#scree plot

(ggplot(pcaDF, aes(x = "pc", y = "expl_var")) + geom_line() +
geom_point())
```




```
<ggplot: (8776361213980)>
```

The model above is a scree plot that demonstrates the amount of variance explained by each principal component in descending order.

```
#cum plot

(ggplot(pcaDF, aes(x = "pc", y = "cum_var")) + geom_line(color =
"pink") +
 geom_point(color = "pink") + geom_hline(yintercept = 0.95))
```




```
<ggplot: (8776361415641)>
```

The model above is a cumulative variance plot that demonstrates the total variance achieved by each principal components and the principal components before it.

```
#need to keep 5 components to account for at least 90% of the variance
pcomps5 = pca.transform(df1[features1])
pcomps5 = pd.DataFrame(pcomps5[:,0:5])

# Model
lr2 = LogisticRegression()
lr2.fit(df1[features1], y)

print("5 PCs :" , lr2.score(df1[features1], y))

rawpredvals = lr.predict(X_test)
print("Raw Variable Accuracy : ", accuracy_score(y_test, rawpredvals))
```

```
pcpredvals = lr2.predict(X_test[features1])
print("PC Accuracy: ", accuracy_score(y_test, pcpredvals))
```

```
5 PCs : 0.8
Raw Variable Accuracy :  0.8395061728395061
PC Accuracy:  0.8148148148148148
```

ANSWER TO QUESTION

By comparing our logistic regression model with principal components to the logistic regression model with the original variables, we were able to determine that there was only a slight drop in accuracy for the model with principal components. We utalized PCA to create 5 principal components which retained slightly over 90% of the original variance. With our PC logistic regression model being so successful, we believe it would be wise to utilize the principal components over the original predictors in this model. With the use of the principal components, our model will be quicker and more efficient.

REFERENCES: https://www.geeksforgeeks.org/how-to-change-legend-title-in-ggplot2-in-r/

https://stackoverflow.com/questions/14622421/how-to-change-legend-title-in-ggplot