CSCI 347 Final Report

Performed by: Simeon Shirshov, Jack Brown, Brady Ash, Brady Underwood, Andrew Cilker

Part 1: Proposal

Using the Heart Disease dataset, we want to solve the problem of predicting whether or not a person has heart disease. To do this, we will be using different combinations of the attributes in our dataset to attempt to find a correlation that gives us a good prediction of heart disease. While we know that correlation does not equal causation, if we do find a strong correlation that predicts if a person has heart disease, we can use that as a baseline for a more serious/in depth experiment. In the Heart Disease dataset, we have 13 total attributes: 6 numerical attributes including age, and 7 categorical attributes. This data set is not missing any values. We are going to use various clustering algorithms to attempt to solve this problem. Instead of picking and choosing variables, we may also attempt to plot the data as a whole, and then reduce dimensionality to preserve as much variance as possible, clustering on the result. If there is any part of our solution that we would need to leave to future entities, it would be the continuation of the experiments to prove causation. Otherwise, if we do not find a strong correlation or solution, it would be to gather more information or dimensions, and attempt our project again.<\p>

Part 2: Analysis

Naive Bayes

In [46]:
```
### Initial Imports
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
```

```
In [47]: data = open("./Data/heart.csv")
         arr = np.genfromtxt(data, delimiter=',')
         dataframe = pd.DataFrame(arr)
         Y = dataframe[13]
         X = dataframe.iloc[:, 0:13]
         # print(X)
         # print("================================")
         # print(Y)

         X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, rar

         gnb = GaussianNB()
         y_pred = gnb.fit(X_train, y_train).predict(X_test)
         totalPoints = X_test.shape[0]
         misPoints = (y_test != y_pred).sum()
         print("Number of mislabeled points out of a total %d points : %d" % (X_test.
         print("Accuracy: ", (totalPoints-misPoints)/ totalPoints)
```

```
Number of mislabeled points out of a total 54 points : 5
Accuracy:  0.9074074074074074
```

Logistic Regression

```
In [48]: from sklearn.metrics import accuracy_score
         from sklearn.linear_model import LogisticRegression

         data = open("./Data/heart.csv")
         arr = np.genfromtxt(data, delimiter=',')
         dataframe = pd.DataFrame(arr)
         Y = dataframe[13]
         X = dataframe.iloc[:, 0:13]
         X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, rar
         logreg = LogisticRegression(max_iter=1000)
         logreg.fit(X_train, y_train)
         y_pred = logreg.predict(X_test)
         accuracy = accuracy_score(y_test, y_pred)
         print('Accuracy:', accuracy)
```

```
Accuracy: 0.9259259259259259
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages
/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to c
onverge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion
  n_iter_i = _check_optimize_result(
```

Naive Bayes Just using columns 1,4,5,8,10,12 (non-categorical numeric values)

Predicting column 13:

```
In [49]: import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

def naive_bayes(dataset, column_indices):
    df = pd.read_csv(dataset)

    # Select the columns to be used for comparison
    X = df.iloc[:, column_indices]

    # Select the target variable
    y = df.iloc[:, -1]

    print(X)
    print(Y)

    # Split the data into training and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

    # Fit the Naive Bayes model on the training data
    model = GaussianNB()
    model.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = model.predict(X_test)

    # Calculate the accuracy of the model
    accuracy = np.mean(y_pred == y_test)

    return accuracy
columns = [1,4,5,8,10,12]
print(naive_bayes("./Data/heart.csv",columns))
```

```
     1.0  322.0  0.0  0.0.1  2.0.1  3.0.1
0    0.0  564.0  0.0    0.0    2.0    7.0
1    1.0  261.0  0.0    0.0    1.0    7.0
2    1.0  263.0  0.0    1.0    2.0    7.0
3    0.0  269.0  0.0    1.0    1.0    3.0
4    1.0  177.0  0.0    0.0    1.0    7.0
..   ...    ...  ...    ...    ...    ...
264  1.0  199.0  1.0    0.0    1.0    7.0
265  1.0  263.0  0.0    0.0    1.0    7.0
266  0.0  294.0  0.0    0.0    2.0    3.0
267  1.0  192.0  0.0    0.0    2.0    6.0
268  1.0  286.0  0.0    1.0    2.0    3.0

[269 rows x 6 columns]
0      2.0
1      1.0
2      2.0
3      1.0
4      1.0
      ...
265    1.0
266    1.0
267    1.0
268    1.0
269    2.0
Name: 13, Length: 270, dtype: float64
0.7222222222222222
```

Data Transformation

In [50]:
```
import pandas as pd
import numpy as np

#convert csv into dataframe
df = pd.read_csv('Data/heart.csv')

labels = ["age", "sex", "chest_pain_type", "resting_blood_pressure", "serum_
          "fasting_blood_sugar", "resting_electro_results", "max_heart_rate"
          "slope_peak_exercise_st", "major_vessels", "thal", "presence_of_di

df.columns = labels
np_array = np.array(df)
labeled_df = pd.DataFrame(np_array, columns=labels)

#print dataframe with all attributes
# print(labeled_df)

#delete categorical data from dataframe
labeled_df = labeled_df.drop(columns=["sex", "chest_pain_type","fasting_bloo

#print new dataframe with only numerical attributes
print(labeled_df)

column_names = list(labeled_df.columns)
print(column_names)
```

```
      age  resting_blood_pressure  serum_cholesterol  max_heart_rate  oldpeak  
0    67.0                   115.0              564.0           160.0      1.6   
\
1    57.0                   124.0              261.0           141.0      0.3   
2    64.0                   128.0              263.0           105.0      0.2   
3    74.0                   120.0              269.0           121.0      0.2   
4    65.0                   120.0              177.0           140.0      0.4   
..    ...                     ...                ...             ...      ...   
264  52.0                   172.0              199.0           162.0      0.5   
265  44.0                   120.0              263.0           173.0      0.0   
266  56.0                   140.0              294.0           153.0      1.3   
267  57.0                   140.0              192.0           148.0      0.4   
268  67.0                   160.0              286.0           108.0      1.5   

     presence_of_disease  
0                    1.0  
1                    2.0  
2                    1.0  
3                    1.0  
4                    1.0  
..                   ...  
264                  1.0  
265                  1.0  
266                  1.0  
267                  1.0  
268                  2.0  

[269 rows x 6 columns]
['age', 'resting_blood_pressure', 'serum_cholesterol', 'max_heart_rate', 'old
peak', 'presence_of_disease']
```

KMeans Clustering

In [51]:

```python
from sklearn.decomposition import PCA

pca = PCA(n_components=6)
pca.fit(labeled_df)
transformed = pca.transform(labeled_df)
labeled_array = labeled_df.to_numpy()
```

In [52]:

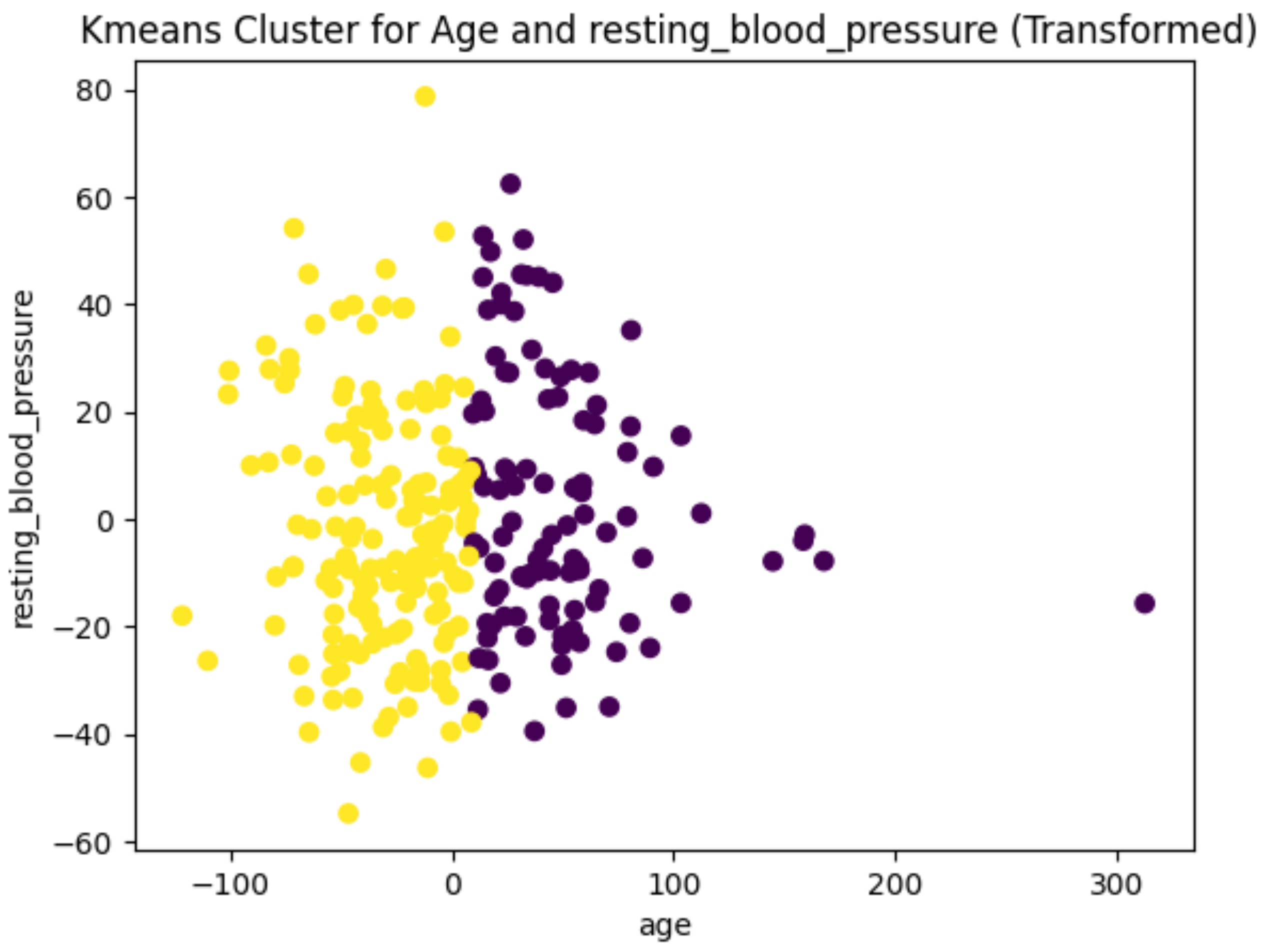
```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.colors as mcolors


#-------------------------------------------Resting Blood Pressure
kmeans = KMeans(n_clusters=2, random_state=0)
cluster_labels = kmeans.fit_predict(transformed)
plt.scatter(transformed[:, 0], transformed[:, 1], c=cluster_labels)
plt.title(' Kmeans Cluster for Age and resting_blood_pressure (Transformed)'
plt.xlabel('age')
plt.ylabel('resting_blood_pressure')
plt.show()


x = transformed[:, 0]
y = transformed[:, 1]
colors = labeled_df['presence_of_disease']
cmap = mcolors.ListedColormap(['yellow', 'purple'])
plt.scatter(x, y, c=colors, cmap=cmap)
yell = plt.plot([],[], marker="o", ms=10, ls="", mec=None, color='yellow', l
purp = plt.plot([],[], marker="o", ms=10, ls="", mec=None, color='purple', l
plt.legend(handles=[yell, purp], loc='upper right')
plt.xlabel('age')
plt.ylabel('resting_blood_pressure')
plt.title('Truth Cluster for Age and resting_blood_pressure (Transformed)')
plt.show()


kmeans = KMeans(n_clusters=2, random_state=0)
cluster_labels = kmeans.fit_predict(labeled_array)
plt.scatter(labeled_array[:, 0], labeled_array[:, 1], c=cluster_labels)
plt.title(' Kmeans Cluster for Age and resting_blood_pressure')
plt.xlabel('age')
plt.ylabel('resting_blood_pressure')
plt.show()


x = labeled_df['age']
y = labeled_df['resting_blood_pressure']
colors = labeled_df['presence_of_disease']
cmap = mcolors.ListedColormap(['yellow', 'purple'])
plt.scatter(x, y, c=colors, cmap=cmap)
plt.legend(handles=[yell, purp], loc='upper right')
plt.xlabel('age')
plt.ylabel('resting_blood_pressure')
plt.title('Truth Cluster for Age and resting_blood_pressure')
plt.show()

#--------------------------------------------serum_cholesterol
kmeans = KMeans(n_clusters=2, random_state=0)
cluster_labels = kmeans.fit_predict(transformed)
plt.scatter(transformed[:, 0], transformed[:, 2], c=cluster_labels)
plt.title(' Kmeans Cluster for Age and serum_cholesterol (Transformed)')
plt.xlabel('age')
plt.ylabel('serum_cholesterol')
plt.show()


x = transformed[:, 0]
y = transformed[:, 2]
colors = labeled_df['presence_of_disease']
cmap = mcolors.ListedColormap(['yellow', 'purple'])
```

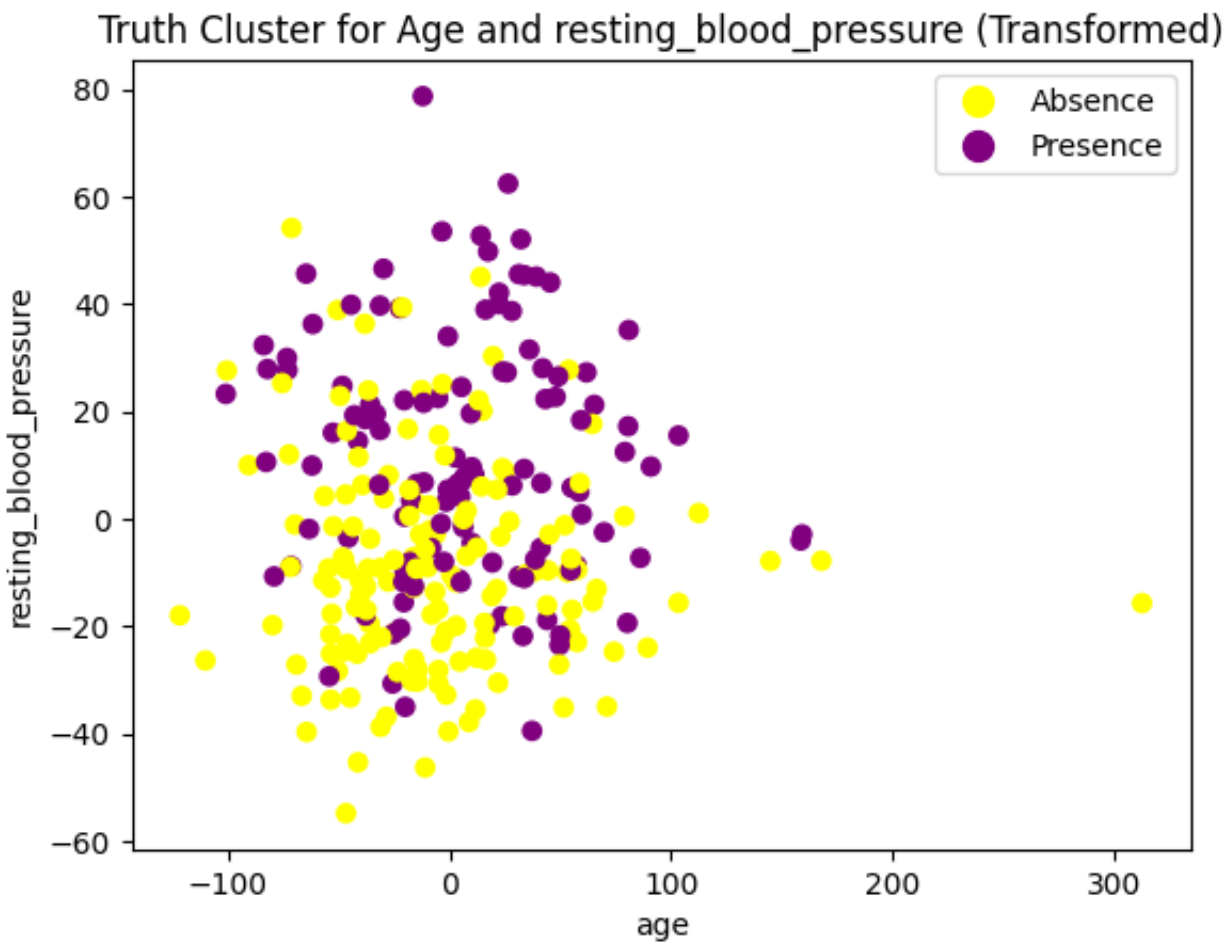
```
plt.scatter(x, y, c=colors, cmap=cmap)
yell = plt.plot([],[], marker="o", ms=10, ls="", mec=None, color='yellow', l
purp = plt.plot([],[], marker="o", ms=10, ls="", mec=None, color='purple', l
plt.legend(handles=[yell, purp], loc='upper right')
plt.xlabel('age')
plt.ylabel('serum_cholesterol')
plt.title('Truth Cluster for Age and serum_cholesterol (Transformed)')
plt.show()


kmeans = KMeans(n_clusters=2, random_state=0)
cluster_labels = kmeans.fit_predict(labeled_array)
plt.scatter(labeled_array[:, 0], labeled_array[:, 2], c=cluster_labels)
plt.title(' Kmeans Cluster for Age and serum_cholesterol')
plt.xlabel('age')
plt.ylabel('serum_cholesterol')
plt.show()


x = labeled_df['age']
y = labeled_df['serum_cholesterol']
colors = labeled_df['presence_of_disease']
cmap = mcolors.ListedColormap(['yellow', 'purple'])
plt.scatter(x, y, c=colors, cmap=cmap)
plt.legend(handles=[yell, purp], loc='upper right')
plt.xlabel('age')
plt.ylabel('serum_cholesterol')
plt.title('Truth Cluster for Age and serum_cholesterol')
plt.show()

#-------------------------------------------max_heart_rate
kmeans = KMeans(n_clusters=2, random_state=0)
cluster_labels = kmeans.fit_predict(transformed)
plt.scatter(transformed[:, 0], transformed[:, 3], c=cluster_labels)
plt.title(' Kmeans Cluster for Age and max_heart_rate (Transformed)')
plt.xlabel('age')
plt.ylabel('max_heart_rate')
plt.show()


x = transformed[:, 0]
y = transformed[:, 3]
colors = labeled_df['presence_of_disease']
cmap = mcolors.ListedColormap(['yellow', 'purple'])
plt.scatter(x, y, c=colors, cmap=cmap)
yell = plt.plot([],[], marker="o", ms=10, ls="", mec=None, color='yellow', l
purp = plt.plot([],[], marker="o", ms=10, ls="", mec=None, color='purple', l
plt.legend(handles=[yell, purp], loc='upper right')
plt.xlabel('age')
plt.ylabel('max_heart_rate')
plt.title('Truth Cluster for Age and max_heart_rate (Transformed)')
plt.show()
```

```
kmeans = KMeans(n_clusters=2, random_state=0)
cluster_labels = kmeans.fit_predict(labeled_array)
plt.scatter(labeled_array[:, 0], labeled_array[:, 3], c=cluster_labels)
plt.title(' Kmeans Cluster for Age and max_heart_rate')
plt.xlabel('age')
plt.ylabel('max_heart_rate')
plt.show()


x = labeled_df['age']
y = labeled_df['max_heart_rate']
colors = labeled_df['presence_of_disease']
cmap = mcolors.ListedColormap(['yellow', 'purple'])
plt.scatter(x, y, c=colors, cmap=cmap)
plt.legend(handles=[yell, purp], loc='upper right')
plt.xlabel('age')
plt.ylabel('max_heart_rate')
plt.title('Truth Cluster for Age and max_heart_rate')
plt.show()

#------------------------------------------oldpeak
kmeans = KMeans(n_clusters=2, random_state=0)
cluster_labels = kmeans.fit_predict(transformed)
plt.scatter(transformed[:, 0], transformed[:, 4], c=cluster_labels)
plt.title(' Kmeans Cluster for Age and oldpeak (Transformed)')
plt.xlabel('age')
plt.ylabel('oldpeak')
plt.show()


x = transformed[:, 0]
y = transformed[:, 4]
colors = labeled_df['presence_of_disease']
cmap = mcolors.ListedColormap(['yellow', 'purple'])
plt.scatter(x, y, c=colors, cmap=cmap)
yell = plt.plot([],[], marker="o", ms=10, ls="", mec=None, color='yellow', l
purp = plt.plot([],[], marker="o", ms=10, ls="", mec=None, color='purple', l
plt.legend(handles=[yell, purp], loc='upper right')
plt.xlabel('age')
plt.ylabel('oldpeak')
plt.title('Truth Cluster for Age and oldpeak (Transformed)')
plt.show()


kmeans = KMeans(n_clusters=2, random_state=0)
cluster_labels = kmeans.fit_predict(labeled_array)
plt.scatter(labeled_array[:, 0], labeled_array[:, 4], c=cluster_labels)
plt.title(' Kmeans Cluster for Age and oldpeak')
plt.xlabel('age')
plt.ylabel('oldpeak')
plt.show()
```

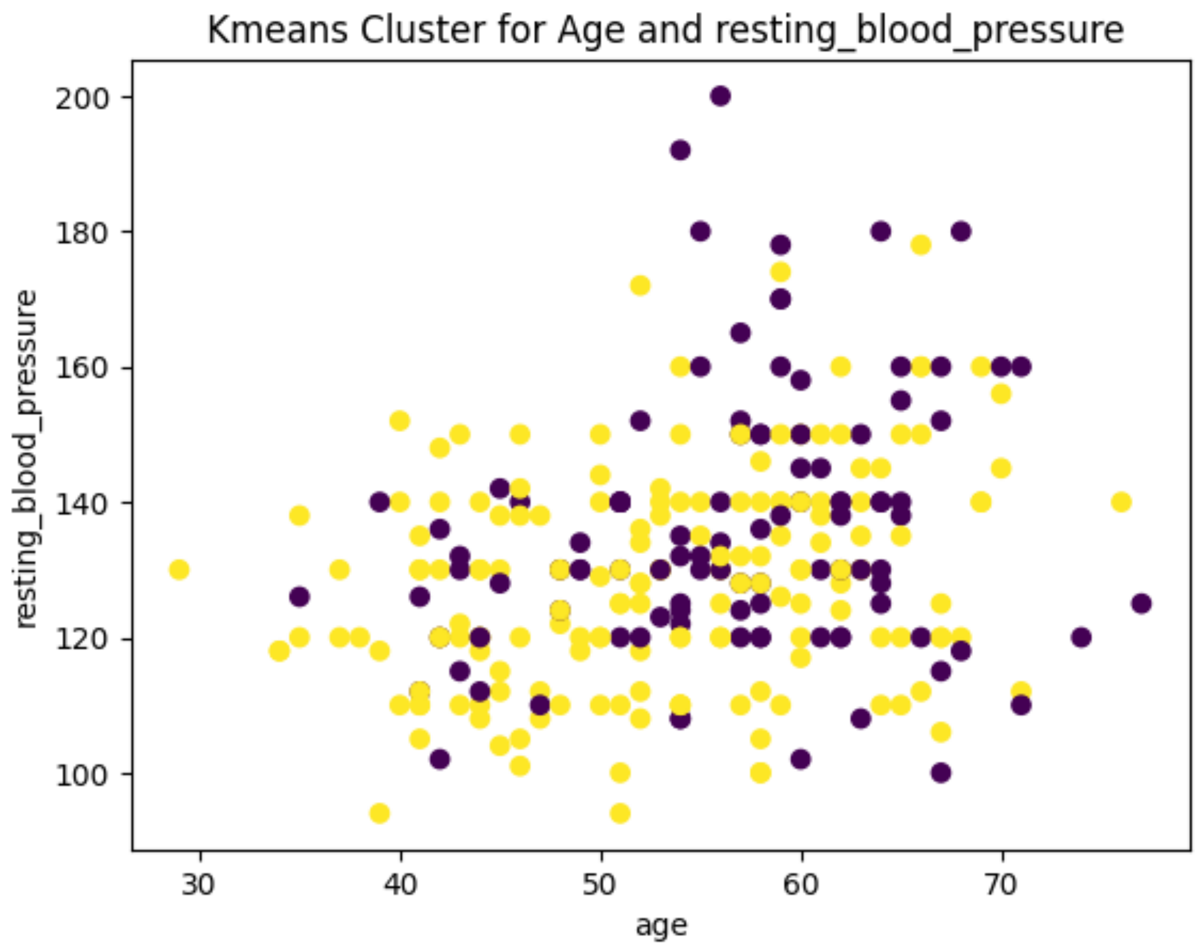
```
x = labeled_df['age']
y = labeled_df['oldpeak']
colors = labeled_df['presence_of_disease']
cmap = mcolors.ListedColormap(['yellow', 'purple'])
plt.scatter(x, y, c=colors, cmap=cmap)
plt.legend(handles=[yell, purp], loc='upper right')
plt.xlabel('age')
plt.ylabel('oldpeak')
plt.title('Truth Cluster for Age and oldpeak')
plt.show()
```

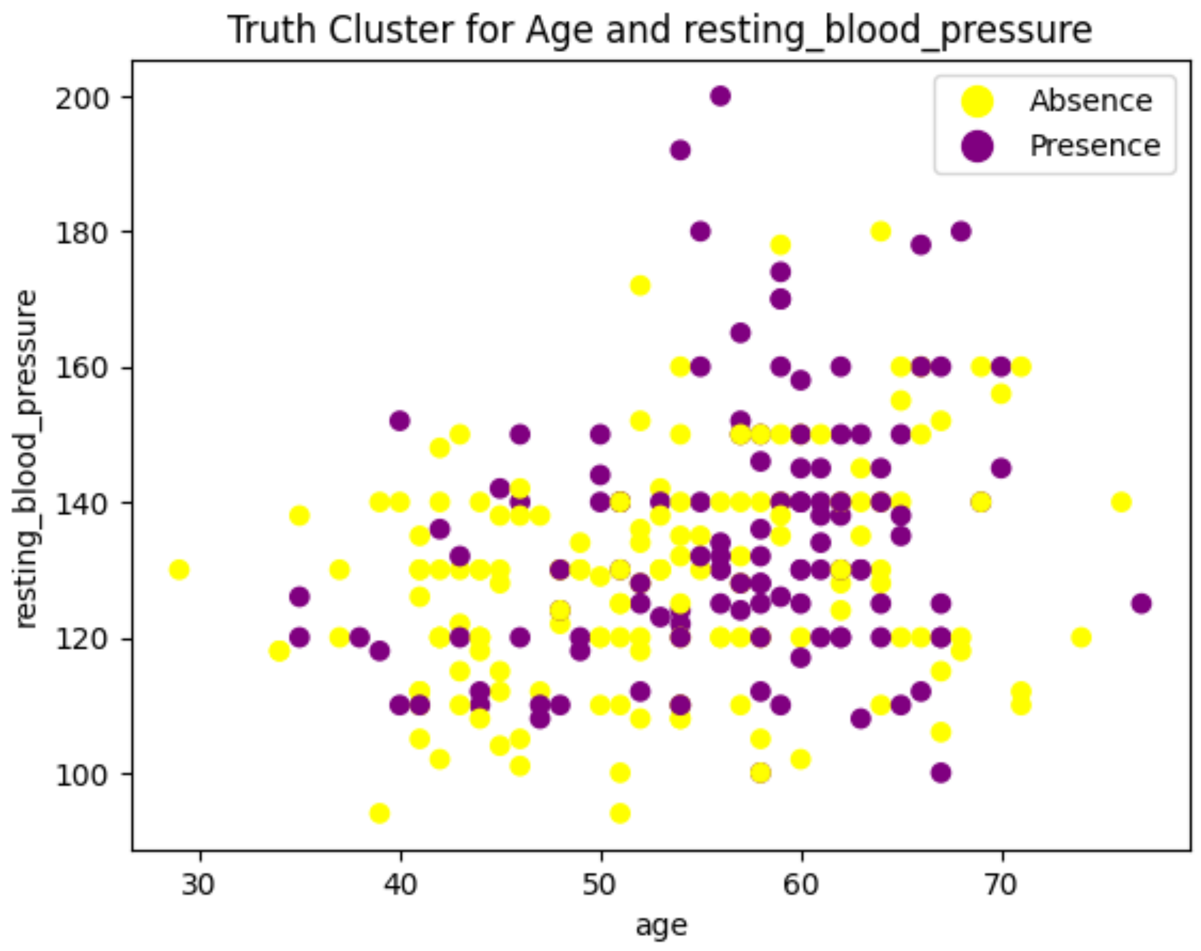
```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages
/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to
suppress the warning
  warnings.warn(
```



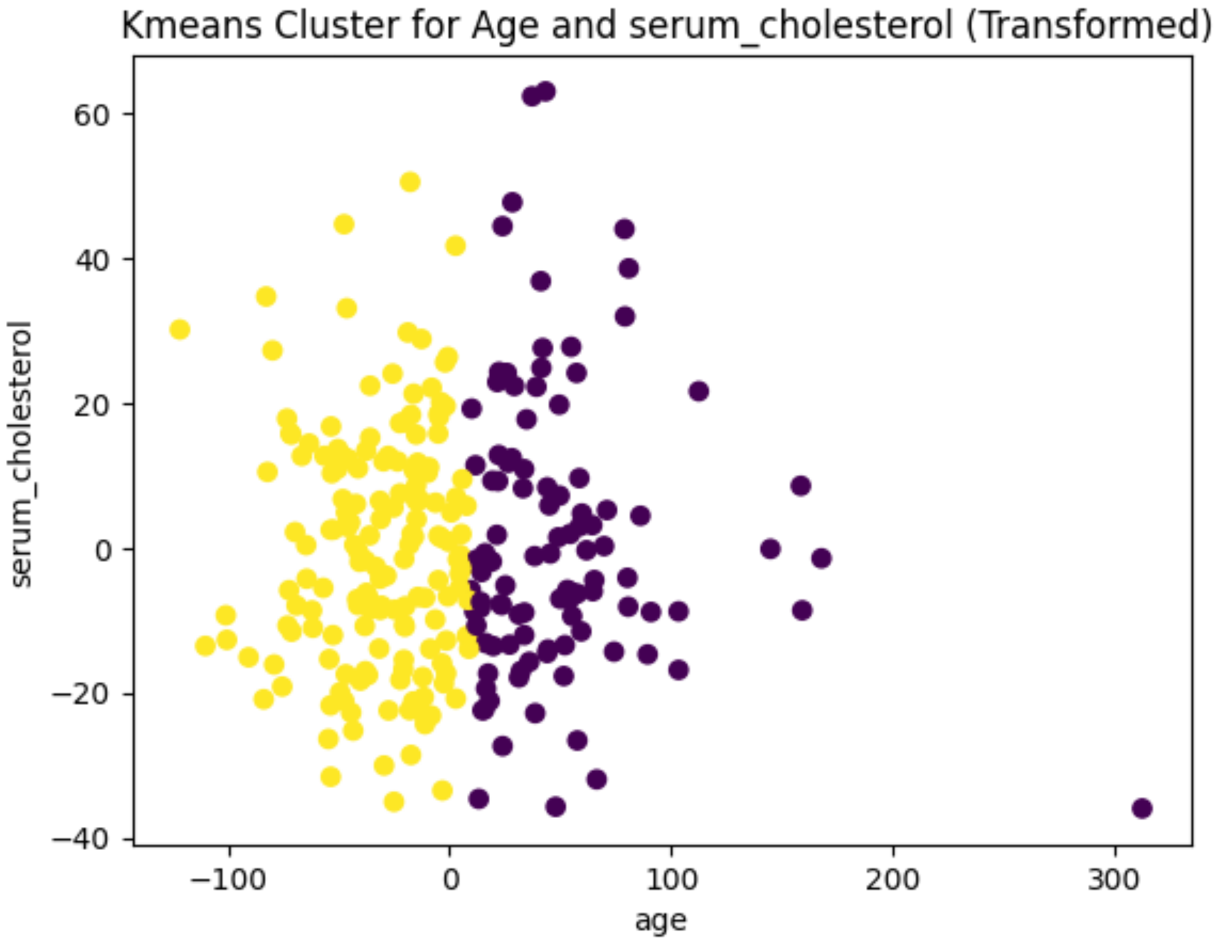


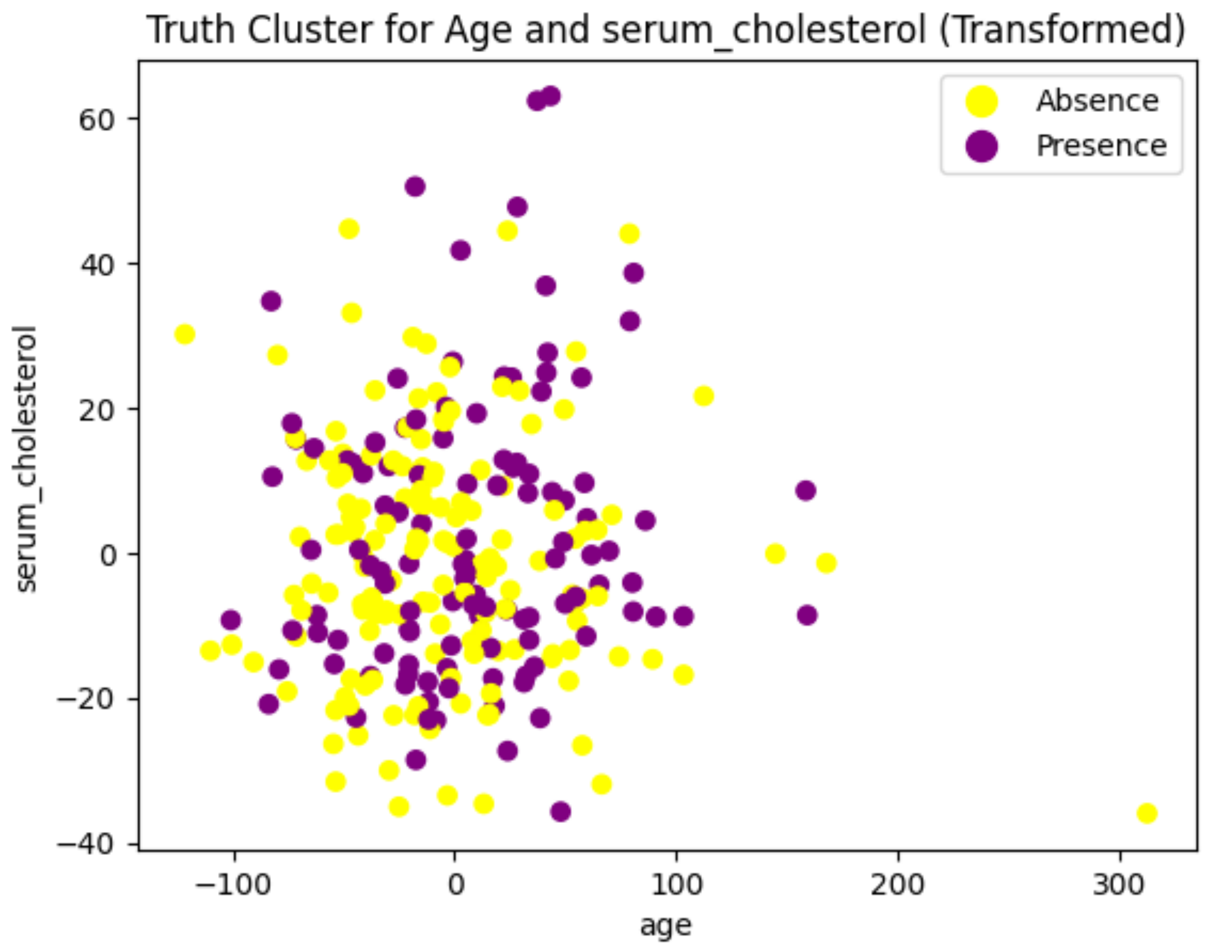
```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages
/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to
suppress the warning
  warnings.warn(
```


Kmeans Cluster for Age and resting_blood_pressure
resting_blood_pressure
age
200
180
160
140
120
100
30
40
50
60
70

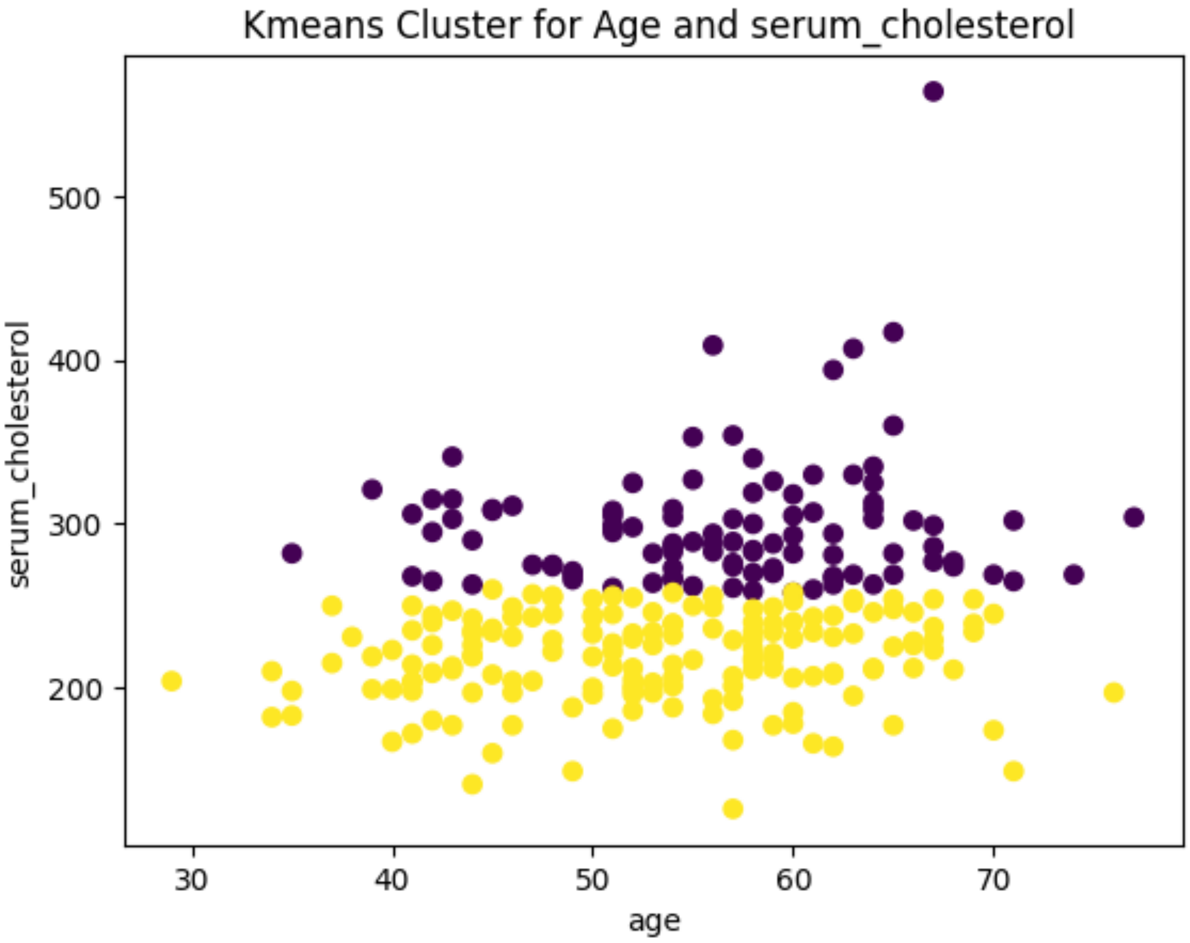


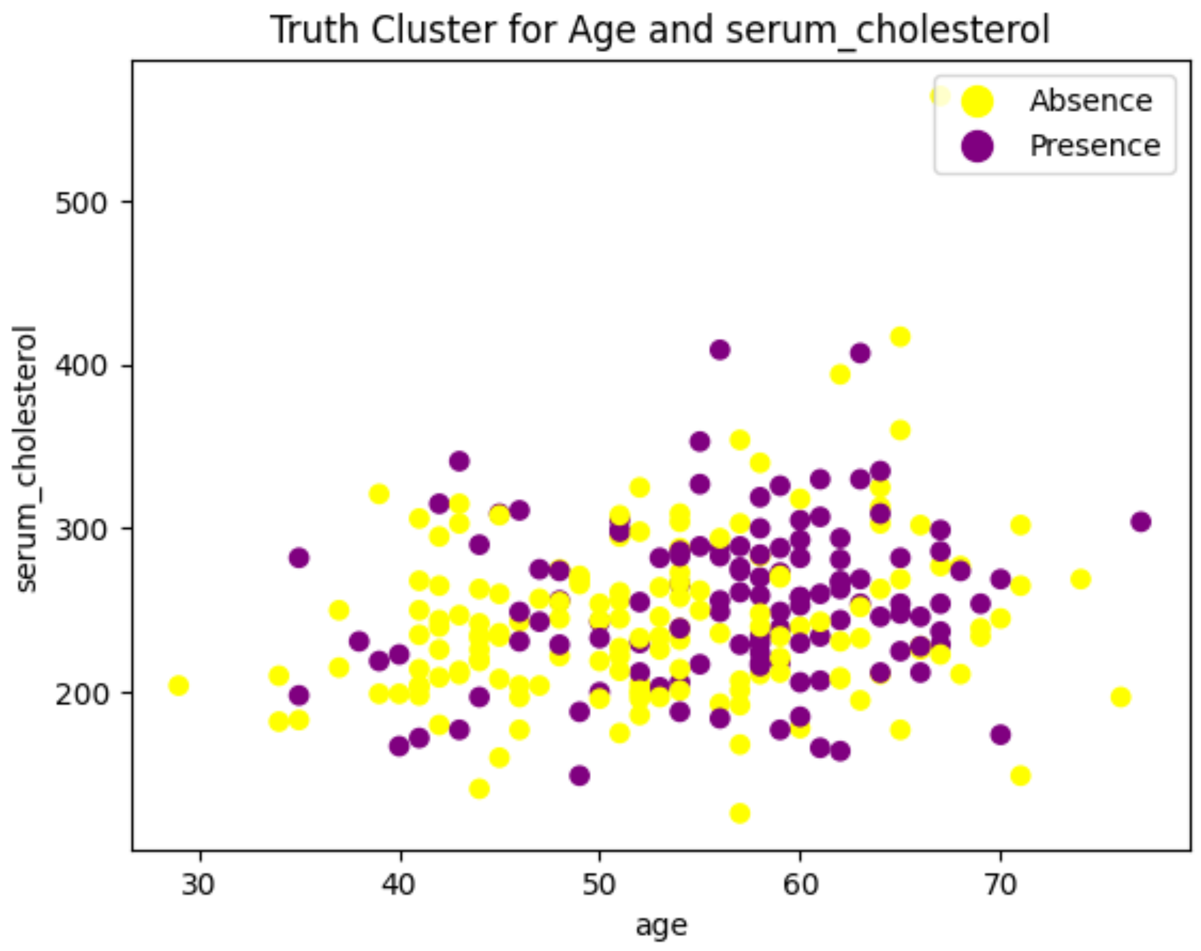
```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages
/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to
suppress the warning
  warnings.warn(
```


Kmeans Cluster for Age and serum_cholesterol (Transformed)
serum_cholesterol
age
60
40
20
0
−20
−40
−100
0
100
200
300

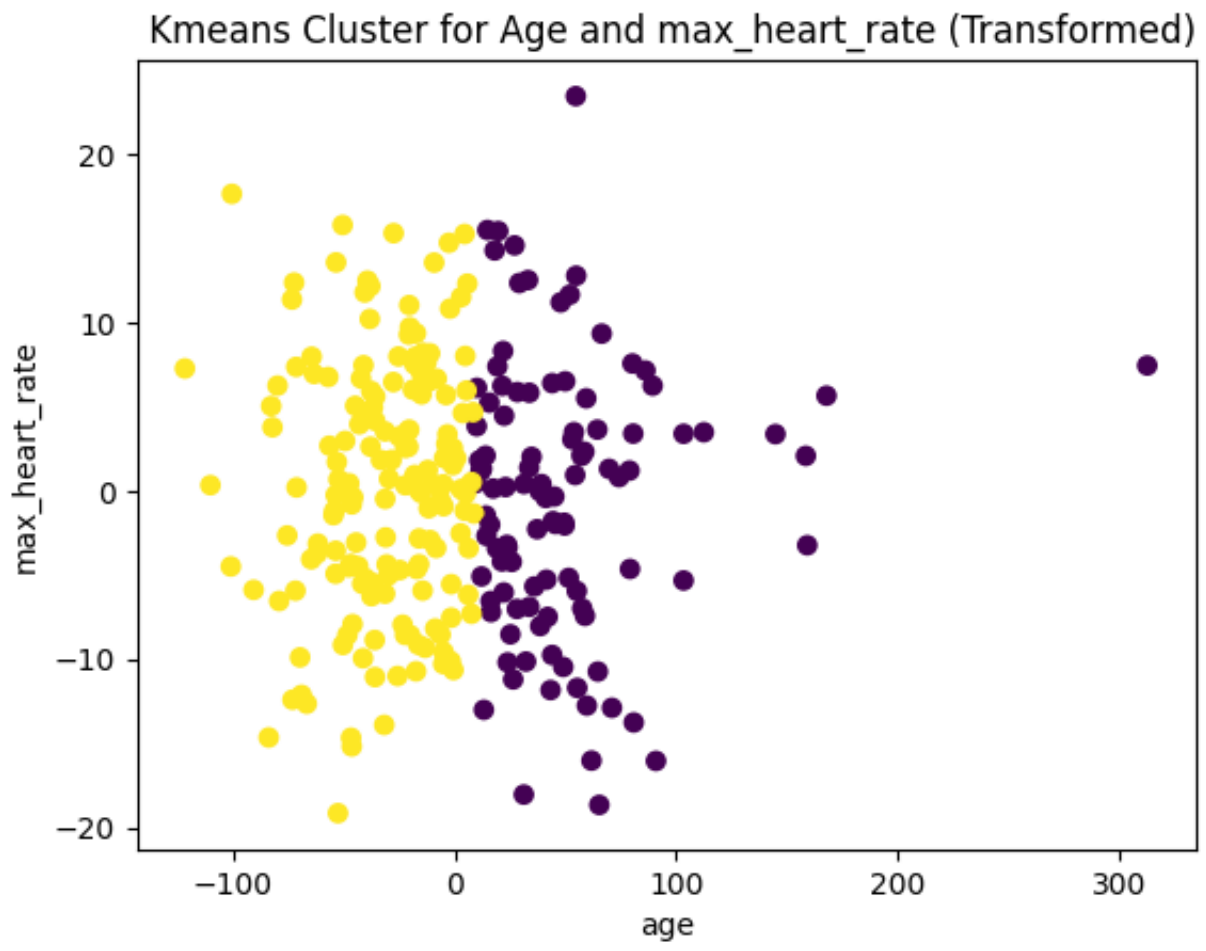


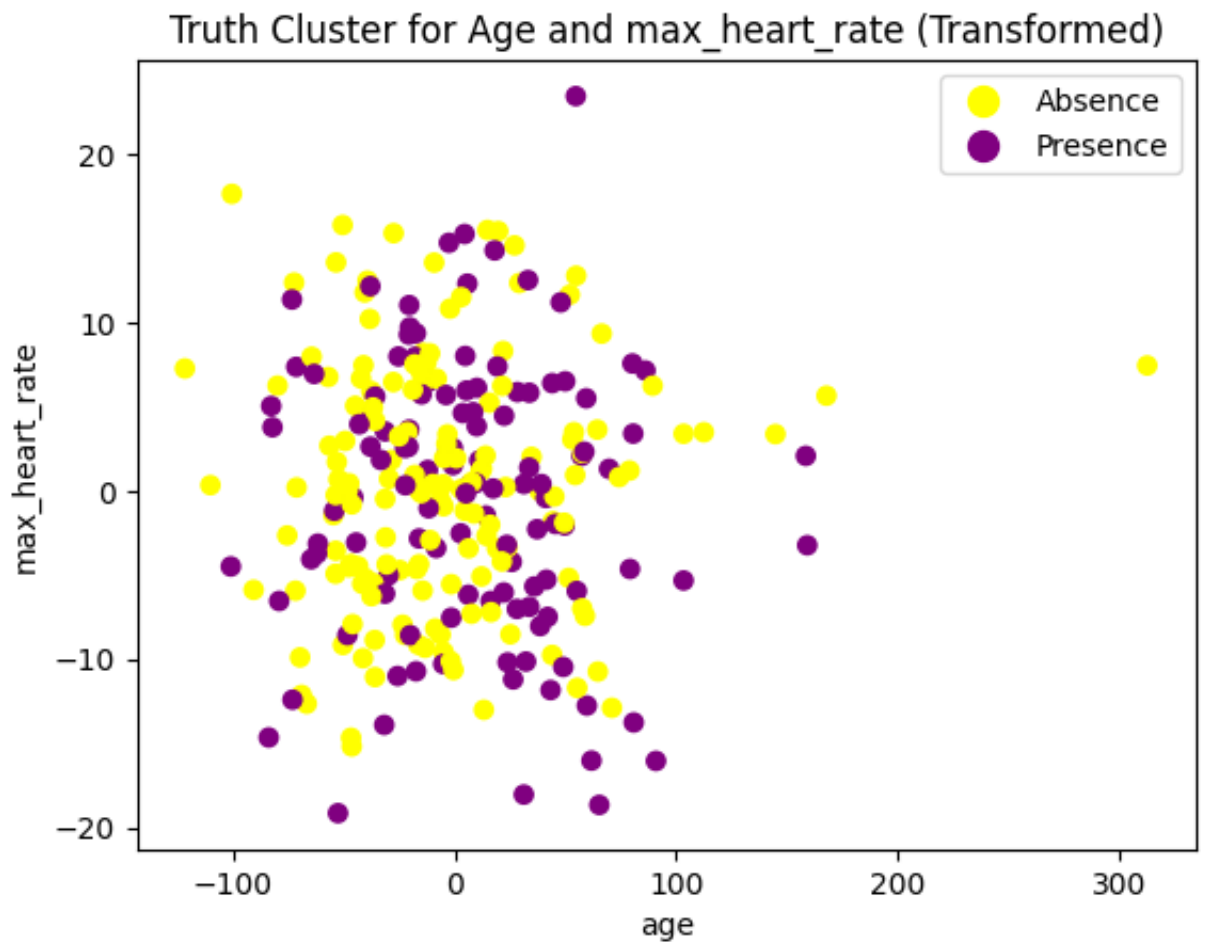
```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages
/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to
suppress the warning
  warnings.warn(
```


Kmeans Cluster for Age and serum_cholesterol
serum_cholesterol
age
500
400
300
200
30
40
50
60
70

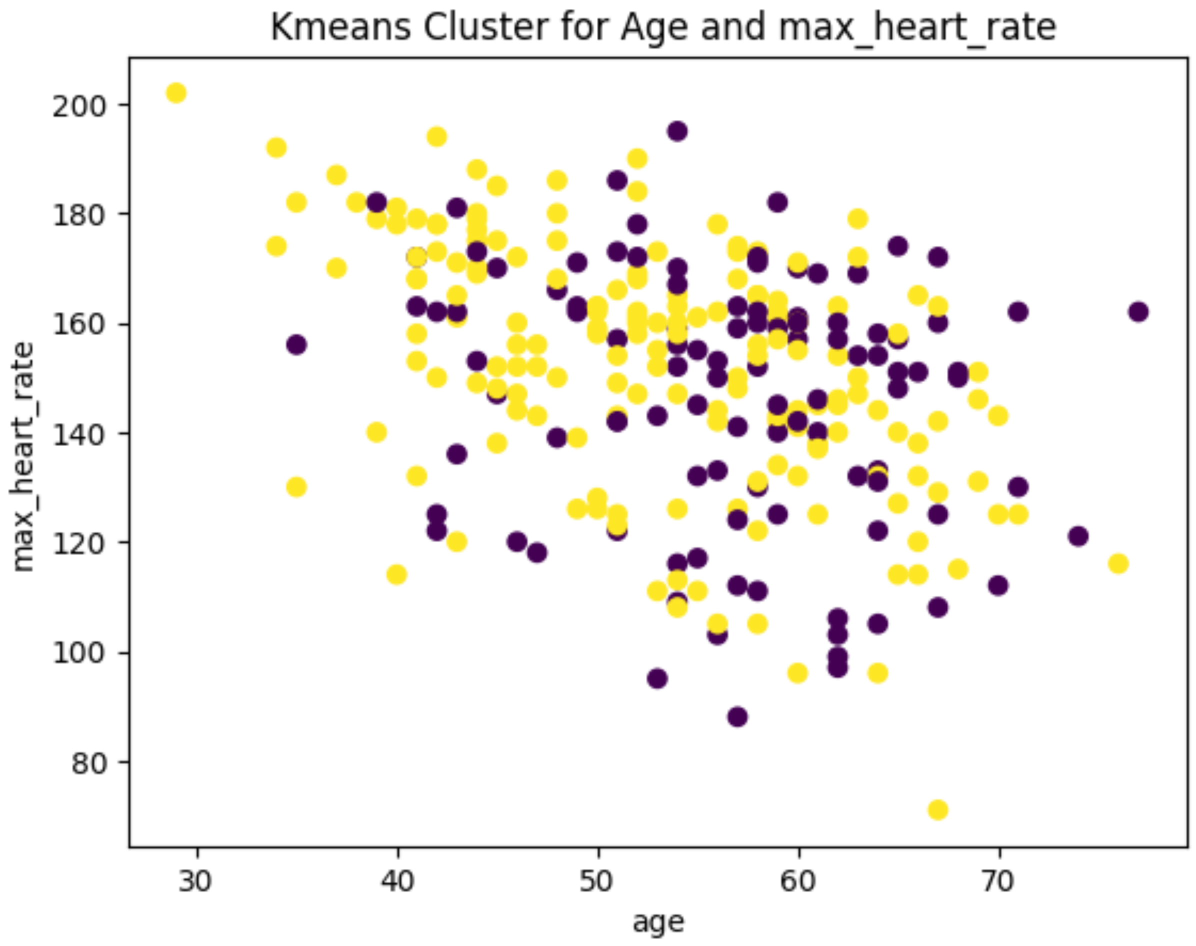


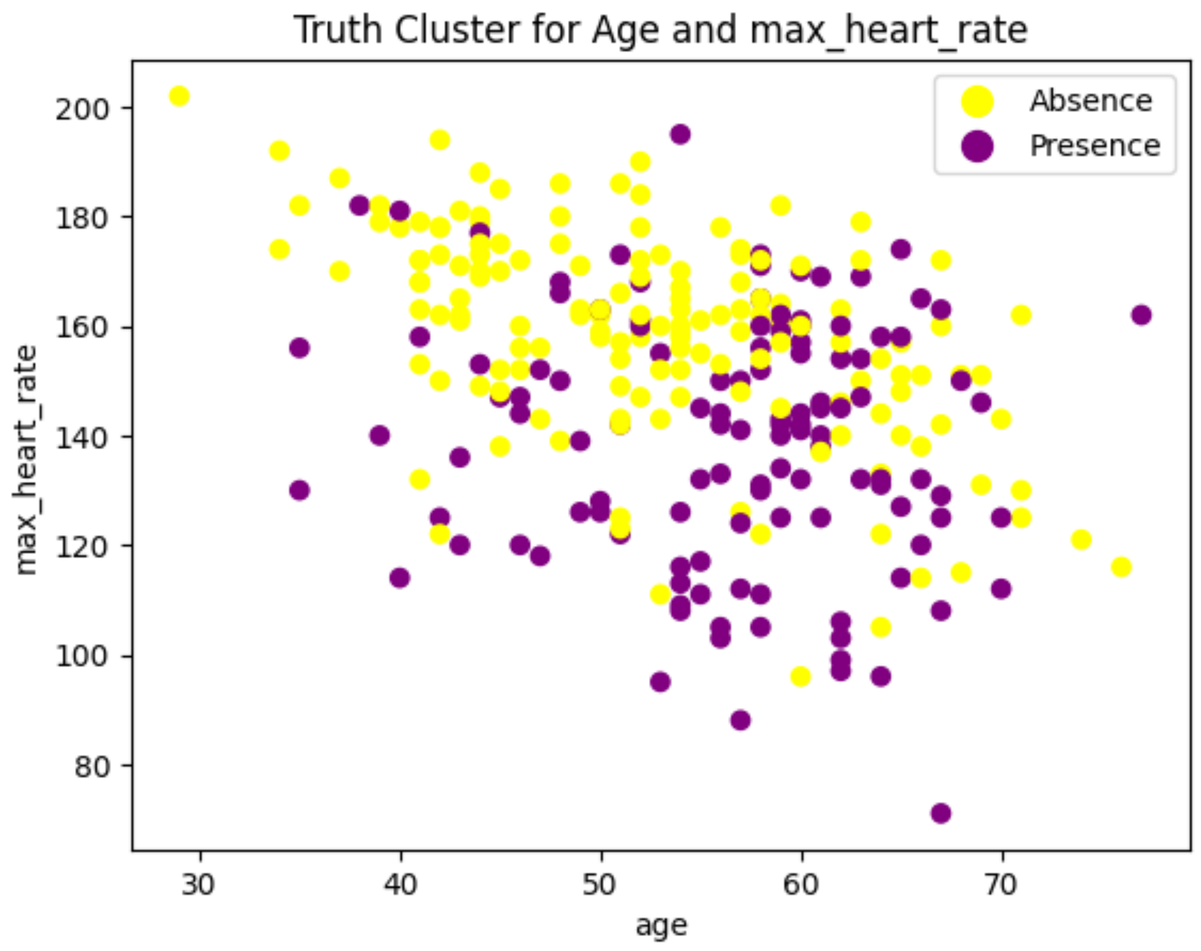
```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages
/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to
suppress the warning
  warnings.warn(
```


Kmeans Cluster for Age and max_heart_rate (Transformed)
max_heart_rate
20
10
0
−10
−20
−100
0
100
200
300
age

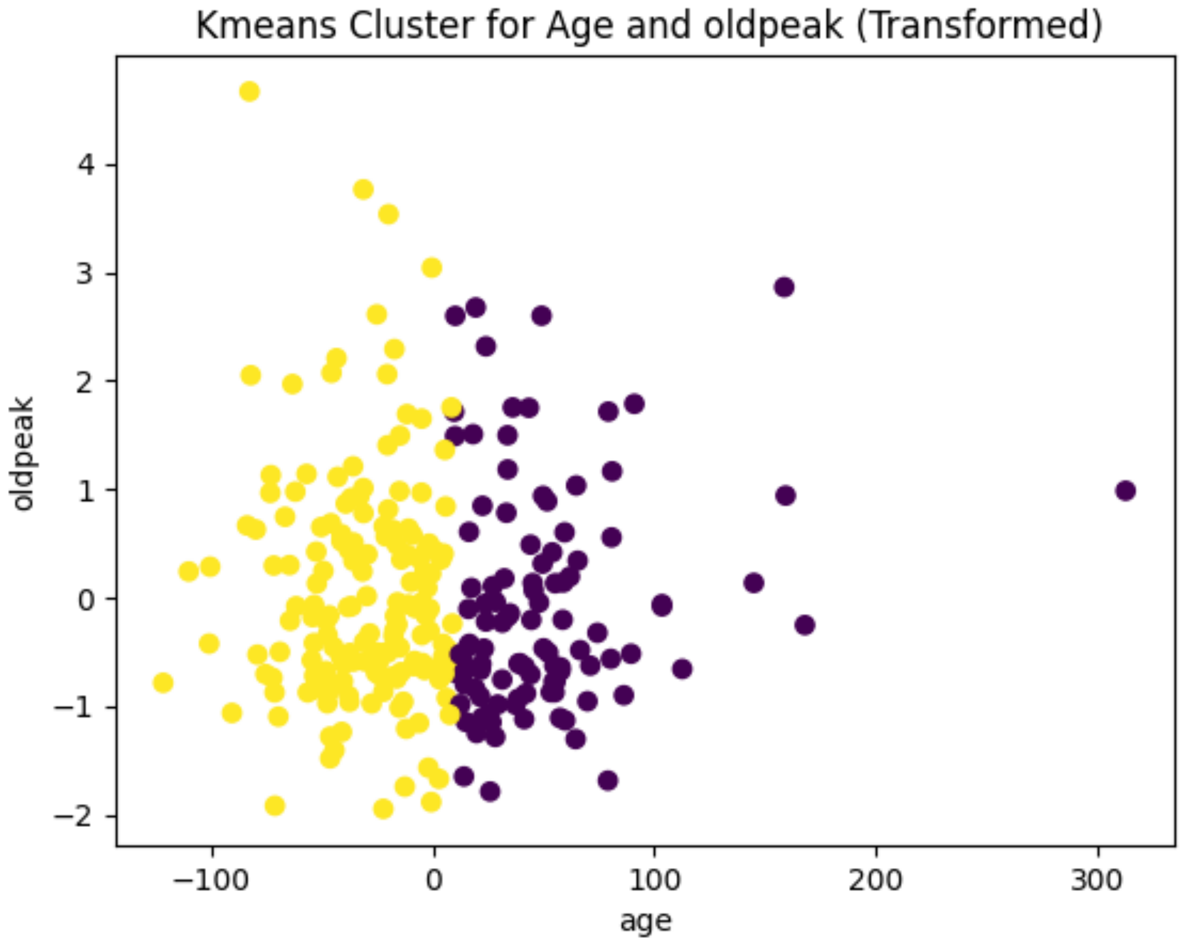


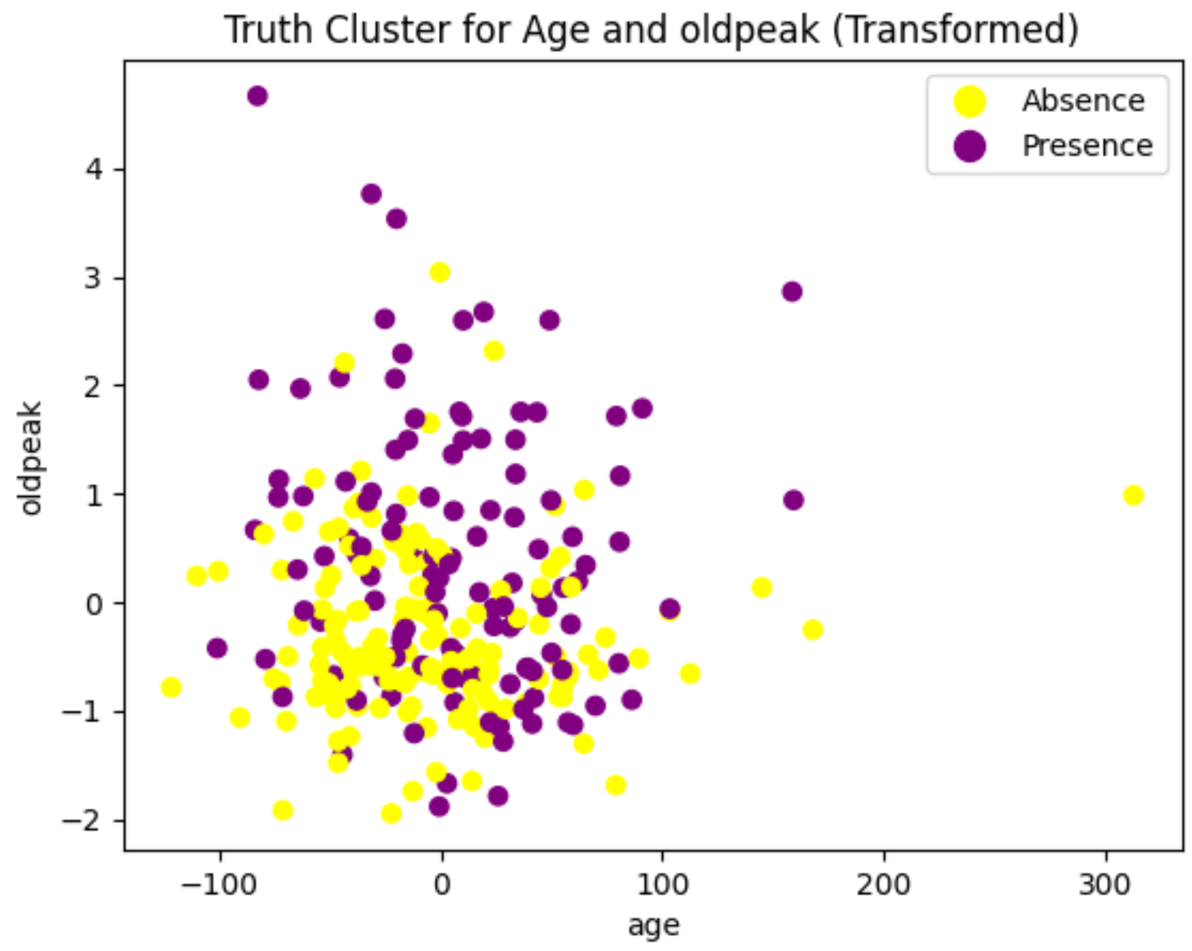
```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages
/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to
suppress the warning
  warnings.warn(
```


Kmeans Cluster for Age and max_heart_rate
max_heart_rate
age
200
180
160
140
120
100
80
30
40
50
60
70

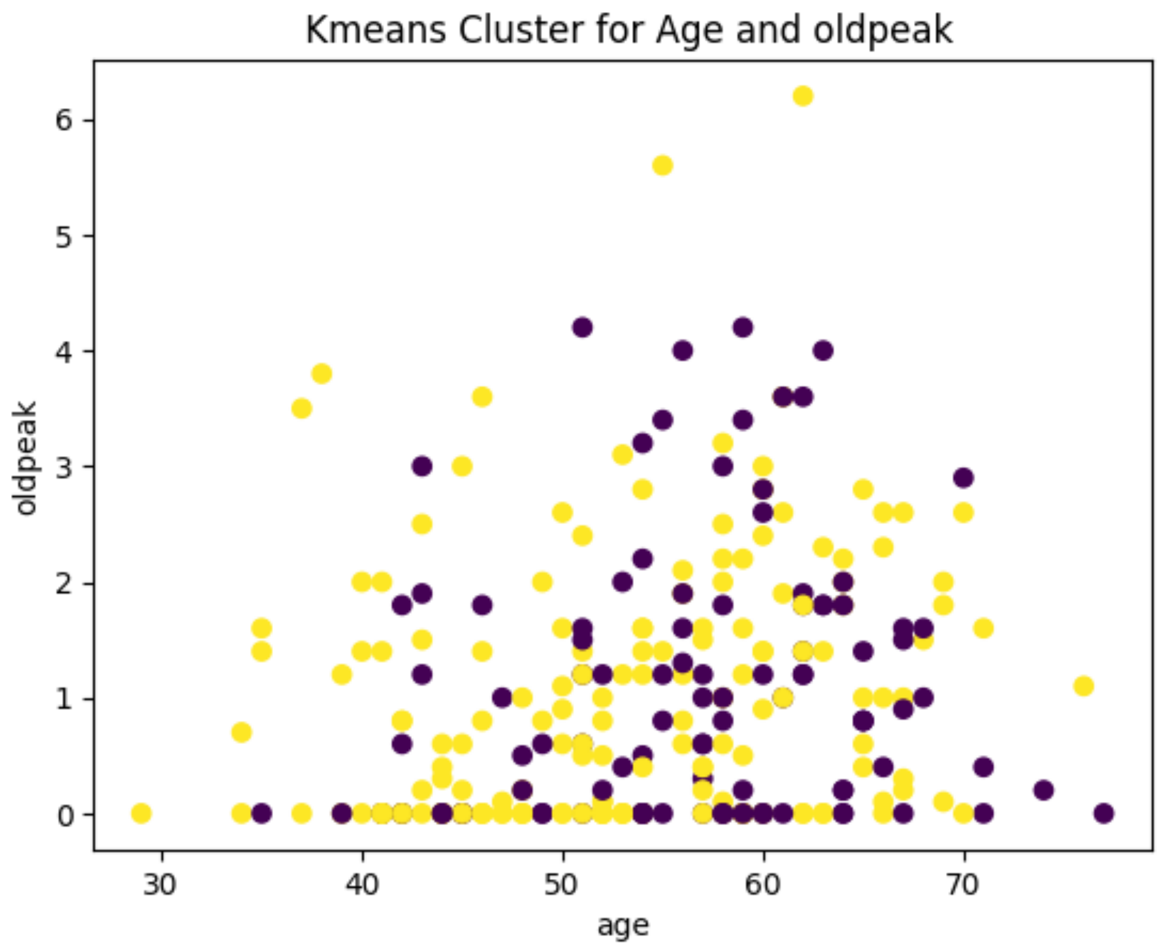


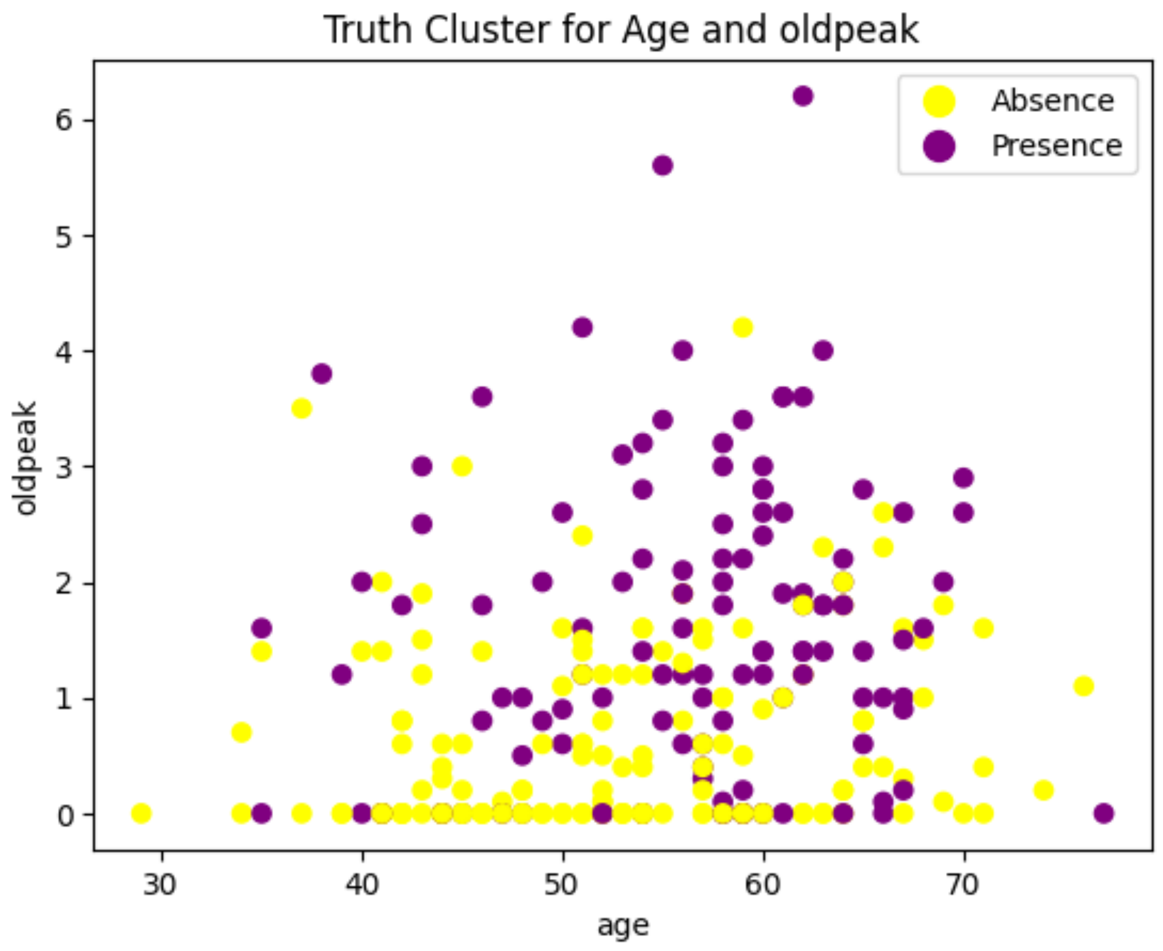
```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages
/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to
suppress the warning
  warnings.warn(
```


Kmeans Cluster for Age and oldpeak (Transformed)
oldpeak
age
4
3
2
1
0
−1
−2
−100
0
100
200
300



```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages
/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to
suppress the warning
  warnings.warn(
```


Kmeans Cluster for Age and oldpeak
oldpeak
age
0
1
2
3
4
5
6
30
40
50
60
70

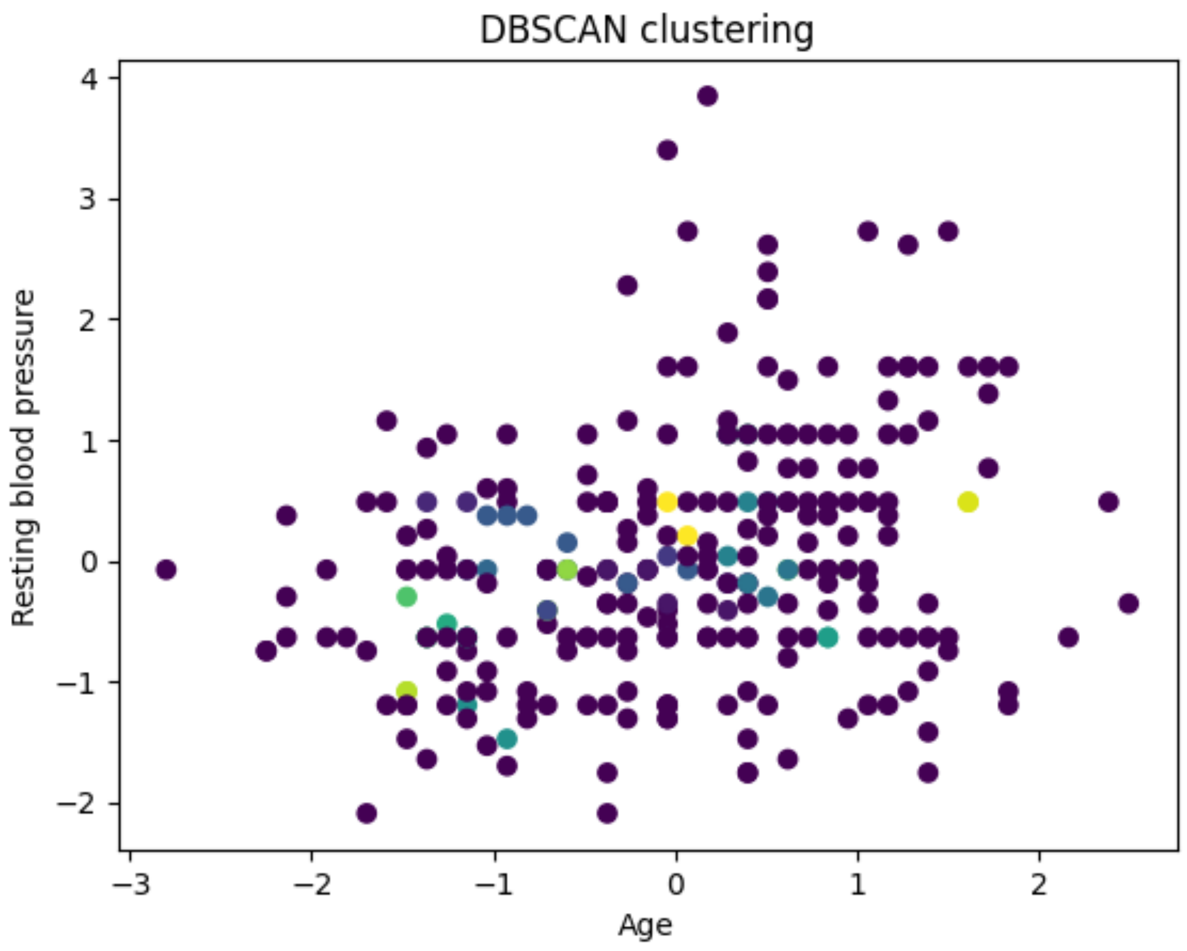


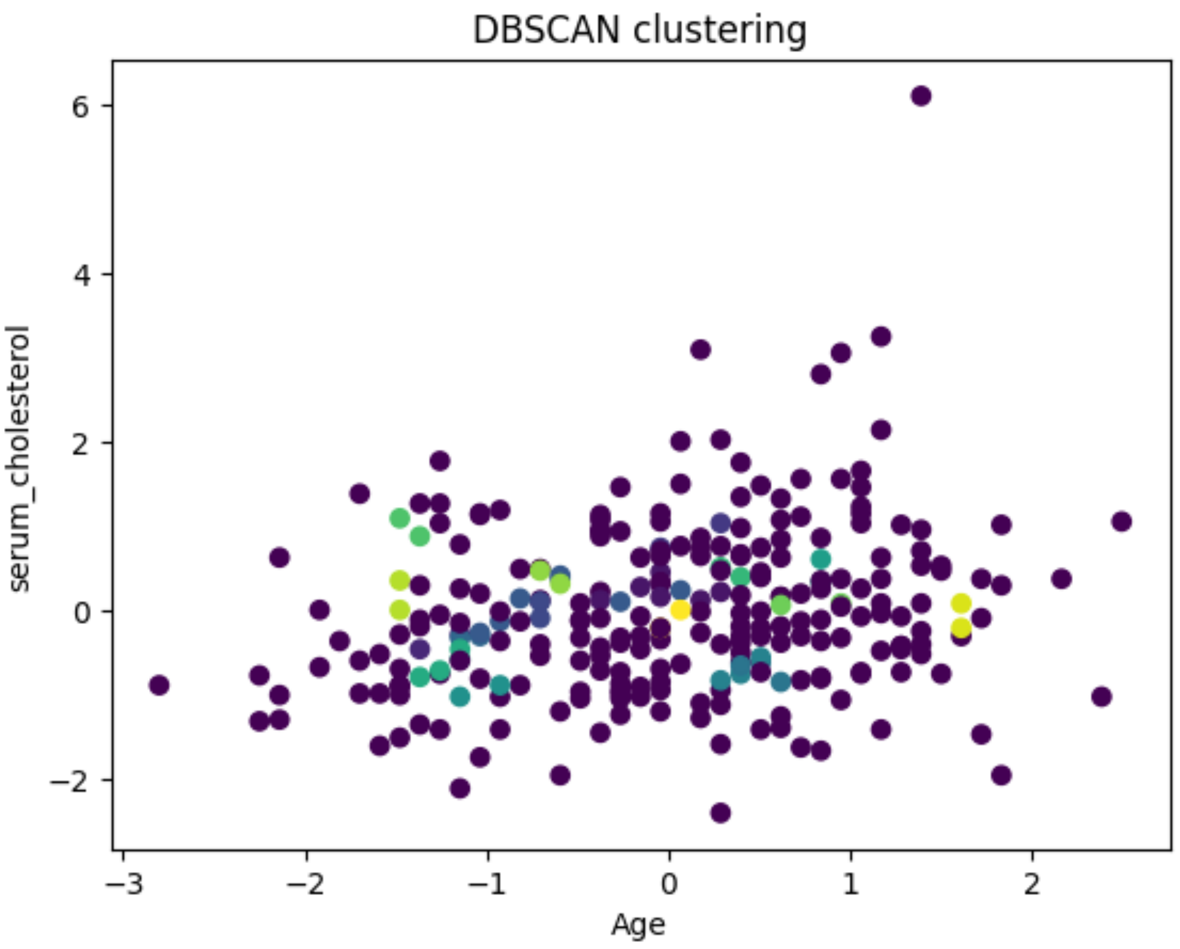
DBSCAN Clustering

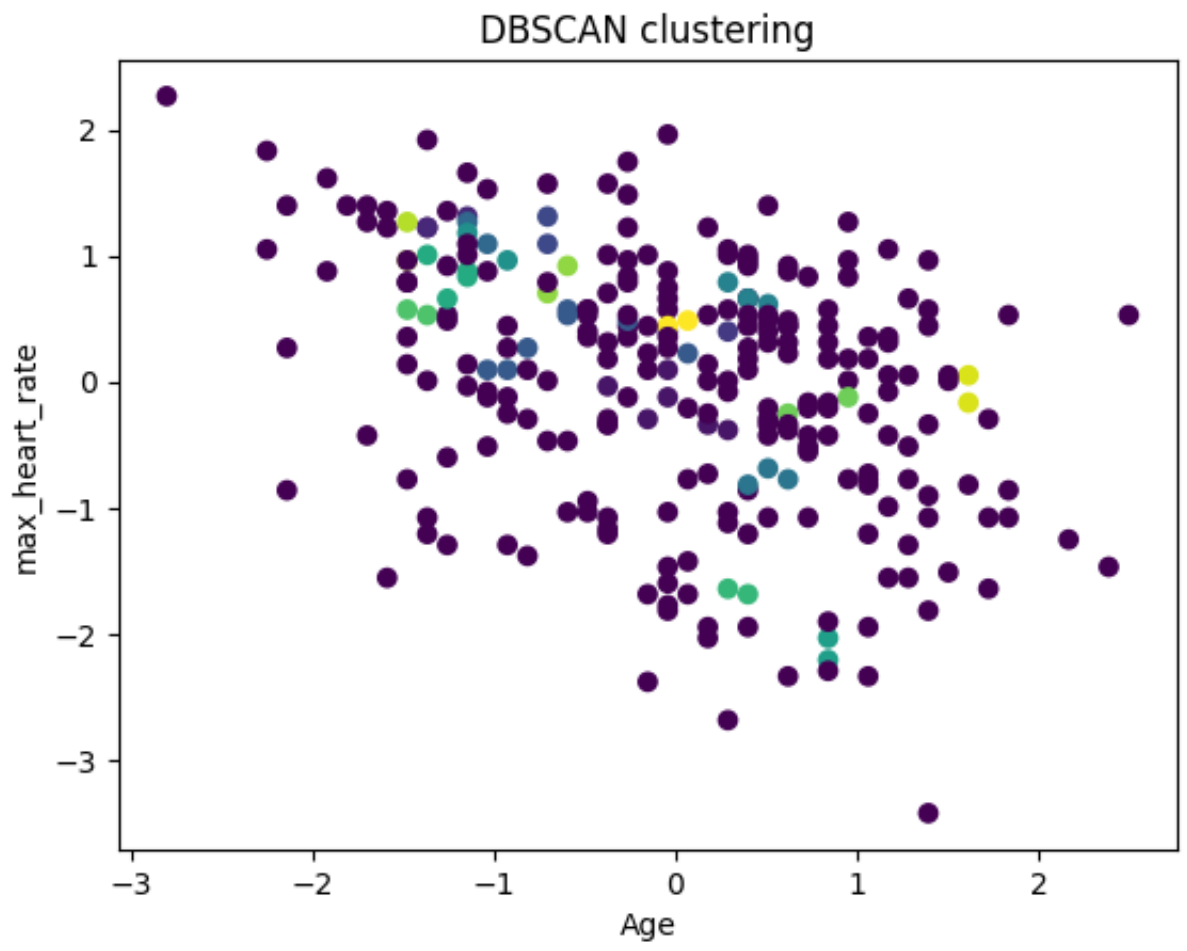
In [53]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=6)
pca.fit(labeled_df)
transformed = pca.transform(labeled_df)
labeled_array = labeled_df.to_numpy()
```

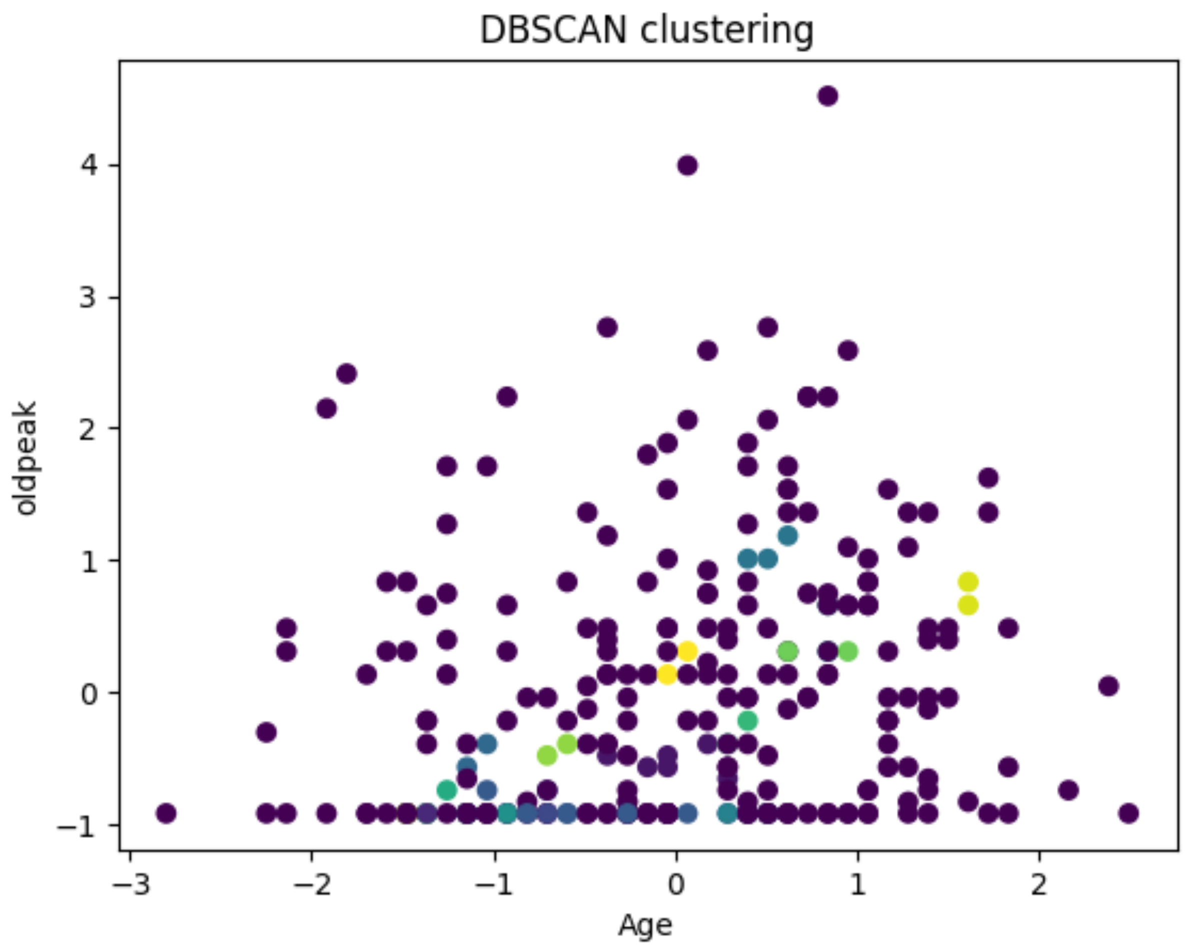
In [54]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN

#-------------------------------------------Resting Blood Pressure
scaler = StandardScaler()
X = scaler.fit_transform(labeled_df.drop('presence_of_disease', axis=1))

dbscan = DBSCAN(eps=.5, min_samples=2)
clusters = dbscan.fit_predict(X)

plt.scatter(X[:, 0], X[:, 1], c=clusters, cmap='viridis')
plt.xlabel('Age')
plt.ylabel('Resting blood pressure')
plt.title('DBSCAN clustering')
plt.show()
```

```
#-------------------------------------------serum_cholesterol
scaler = StandardScaler()
X = scaler.fit_transform(labeled_df.drop('presence_of_disease', axis=1))

dbscan = DBSCAN(eps=.5, min_samples=2)
clusters = dbscan.fit_predict(X)

plt.scatter(X[:, 0], X[:, 2], c=clusters, cmap='viridis')
plt.xlabel('Age')
plt.ylabel('serum_cholesterol')
plt.title('DBSCAN clustering')
plt.show()

#-------------------------------------------max_heart_rate
scaler = StandardScaler()
X = scaler.fit_transform(labeled_df.drop('presence_of_disease', axis=1))

dbscan = DBSCAN(eps=.5, min_samples=2)
clusters = dbscan.fit_predict(X)

plt.scatter(X[:, 0], X[:, 3], c=clusters, cmap='viridis')
plt.xlabel('Age')
plt.ylabel('max_heart_rate')
plt.title('DBSCAN clustering')
plt.show()

#-------------------------------------------oldpeak
scaler = StandardScaler()
X = scaler.fit_transform(labeled_df.drop('presence_of_disease', axis=1))

dbscan = DBSCAN(eps=.5, min_samples=2)
clusters = dbscan.fit_predict(X)

plt.scatter(X[:, 0], X[:, 4], c=clusters, cmap='viridis')
plt.xlabel('Age')
plt.ylabel('oldpeak')
plt.title('DBSCAN clustering')
plt.show()
```


DBSCAN clustering
Resting blood pressure
Age
4
3
2
1
0
−1
−2
−3
−2
−1
0
1
2


DBSCAN clustering
serum_cholesterol
Age
6
4
2
0
−2
−3
−2
−1
0
1
2


DBSCAN clustering
max_heart_rate
Age
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2



Part 3: Summary

What problem were you trying to solve or help solve?

The problem that we could be trying to solve or help solve is identifying the presence or absence of heart disease based on the given attributes. Going forward, this could be used for early detection and prevention of heart disease, which is a leading cause of death worldwide. By analyzing the data using clustering and classification techniques, we can gain insights into the factors that contribute to heart disease. We also wanted to help prediction methods by gaining a better understanding of the interconnectedness of data.

Describe the Data:

It is a Multivariate data set consisting of 270 instances, 13 attributes, and no missing values. There are 6 numerical attributes, 'resting_blood_pressure' ,'serum_cholesterol', 'max_heart_rate', 'oldpeak', 'slope_peak_exercise_st', and ‘age’ as well as 7 categorical attributes.

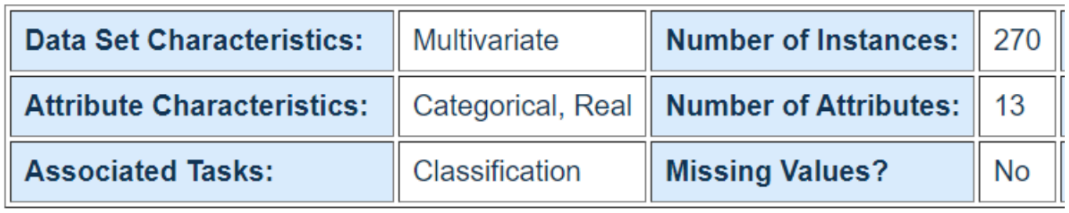
What pre-processing techniques did you apply and why (justify the use of each technique you used, for example label encoding vs. one-hot encoding)?

Because there is no missing data and categorical variables are already label encoded we didn't have to apply any pre-processing technique

What data mining techniques did you apply and why (justify the use of each technique you used, for example why did you use k-means instead of DBSCAN)?

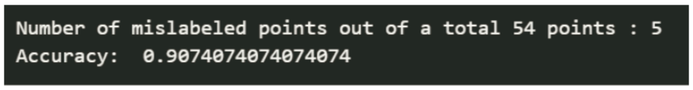
We used both DBScan and k-means for clustering to provide a more comprehensive analysis of the dataset. DBScan can identify clusters of arbitrary shapes and sizes, while Kmeans is efficient and easy to implement. By comparing and contrasting the results obtained from each algorithm, we can gain a deeper understanding of the structure of the data and identify potential issues, such as overlapping clusters or outliers. We also used both Naive Bayes and Logistic Regression classification methods to predict heart disease using 80% of our dataset for training and 20% of our dataset for testing. Naive Bayes is better for handling high-dimensional datasets and provides a reliable baseline model while Logistic Regression is capable of capturing complex relationships between features and can better account for correlated variables. By comparing the performance of both models, we can better understand the underlying data structure and make better predictions.

Include relevant visualizations and tables summarizing your data and your findings



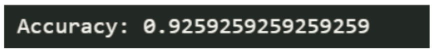
Data Set Characteristics:	Multivariate	**Number of Instances:**	270
Attribute Characteristics:	Categorical, Real	**Number of Attributes:**	13
Associated Tasks:	Classification	**Missing Values?**	No

Naive Bayes



```
Number of mislabeled points out of a total 54 points : 5
Accuracy:  0.9074074074074074
```

Logistic Regression



```
Accuracy: 0.9259259259259259
```

What did you learn through your analysis?

The clustering was accomplished using both DBSCAN and k-means. Each numerical attribute in the data set was compared to age in order to predict the presence of heart disease. The predictions were visualized using DBSCAN and k-means clusters. These were then compared to the true clusters. This process was conducted using both normalized and non-normalized data. The results showed that the DBSCAN and k-means clusters did a poor job of predicting the presence of heart disease. Both methods struggled with creating accurate results regardless of parameters. From this, we learned that the Naive Bayes and Logistic Regression algorithms result in better outcomes.

There is an interesting comparison to be made between the performance of the Naive Bayes algorithm and the performance of the Logistic Regression algorithm. For both algorithms, we trained them using 80% of the data, and tested on 20%. The Bayes algorithm assumes that each feature/column is independent, estimating the target value based on the probabilities of each feature. At first glance, I assumed this algorithm would be superior to the Logistic Regression algorithm, based on the assumption that each of these features were independent. However, I was pleasantly surprised to see that the Logistic Regression algorithm performed better than the Bayes algorithm by roughly 2%, to 92.5%. The Logistic Regression algorithm assumes that there is a linear relationship between the features and the target value. In this assumption, the model uses a loss function to better optimize its parameters and output. Further research showed that the algorithm did particularly well with classifying binary problems such as: if a user has heart disease(1) or not(2). I believe, with further tuning of the dataset and/or additional features, this algorithm could have an even better accuracy.

Was anything about your results surprising or unexpected?

One surprising element was that running the naive bayes algorithm on only the columns containing real numeric values as opposed to categorical column data in the form of ordered, binary, and nominal values was much less accurate than the naive bayes algorithm on all potential columns. This is not entirely surprising because using more data with naive bayes is typically better, but it seemed like an experiment worth trying because I wanted to find how predictive the hard data values such as heart rate compared to the self-reported values such as “chest pain type”.

How will your work help with understanding the problem you set out to solve?

Our work in prediction can help understand the problem of predicting heart disease. While we were able to achieve 92% accuracy on a 54 point testing set we can likely further improve this prediction with more data and a refined algorithm.

What else would you do if you had more time?

If we had more time it would be interesting to scale up our dataset to create a prediction algorithm with a higher accuracy. In addition, it would be interesting to see how adding/pruning features would factor into the accuracy of both the Bayes algorithm and Logistic Regression. Another interesting feature to implement would be a function that finds all the conditional probabilities of all column elements and then finds how predictive they are of the 13th column (final column of heart disease severity). This function could then rank each factor or even pairs of factors that are most predictive of heart disease severity.

```
In [54]:
```