Classification algorithms in machine learning use input training data to predict the likelihood that subsequent data will fall into one of the predetermined categories. One of the most common uses of classification is filtering emails into “spam” or “non-spam.”

In short, classification is a form of “pattern recognition,” with classification algorithms applied to the training data to find the same pattern (similar words or sentiments, number sequences, etc.) in future sets of data.

Popular Classification Algorithms: Naïve Bias,Logistic regression,KMeans,KNN

Heart disease prediction using classification algorithms :

A heart attack which is analogous to acute myocardial infarction (AMI) is one of the most serious diseases in the segment of cardiovascular disease. It occurs due to the interruption of blood circulation to muscle of the heart which damages the heart the muscle. Diagnosing heart disease is also a crucial task. The symptoms, physical examination, and understanding of the different signs of this disease are required to diagnose heart disease. Different factors including cholester,, genetic heart disease, high blood pressure, low physical activity, obesity, and smoking can be reasons for the occurrence of heart disease.

Logistic Regression: Logistic regression is a Machine Learning method used for classification tasks. It is a predictive analytic technique based on the probability idea. The dependent variable in logistic regression is binary (coded as 1 or 0). The goal is to discover a link between characteristics and the likelihood of a specific outcome.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```
heart_data = pd.read_csv('/content/heart.csv')
```

```
heart_data.head()
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

```
heart_data.tail()
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

```
heart_data.shape
```

```
(303, 14)
```

```
heart_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
```

```
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trtbps    303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalachh  303 non-null    int64
 8   exng      303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slp       303 non-null    int64
 11  caa       303 non-null    int64
 12  thall     303 non-null    int64
 13  output    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

```
heart_data.isnull().sum()
```

```
age         0
sex         0
cp          0
trtbps      0
chol        0
fbs         0
restecg     0
thalachh    0
exng        0
oldpeak     0
slp         0
caa         0
thall       0
output      0
dtype: int64
```

```
heart_data.describe()
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.

```
heart_data['output'].value_counts()
```

```
1    165
0    138
Name: output, dtype: int64
```

1 --> Defective heart 0 --> Healthy heart

```
x = heart_data.drop(columns='output',axis = 1)
y = heart_data['output']
```

```
print(x)
```

```
     age  sex  cp  trtbps  chol  fbs  restecg  thalachh  exng  oldpeak  slp  \
0     63    1   3     145   233    1        0       150     0      2.3    0
1     37    1   2     130   250    0        1       187     0      3.5    0
2     41    0   1     130   204    0        0       172     0      1.4    2
3     56    1   1     120   236    0        1       178     0      0.8    2
4     57    0   0     120   354    0        1       163     1      0.6    2
..   ...  ...  ..     ...   ...  ...      ...       ...   ...      ...  ...
298   57    0   0     140   241    0        1       123     1      0.2    1
299   45    1   3     110   264    0        1       132     0      1.2    1
300   68    1   0     144   193    1        1       141     0      3.4    1
301   57    1   0     130   131    0        1       115     1      1.2    1
302   57    0   1     130   236    0        0       174     0      0.0    1

     caa  thall
0      0      1
1      0      2
2      0      2
```

```
3      0      2
4      0      2
..    ...    ...
298    0      3
299    0      3
300    2      3
301    1      3
302    1      2

[303 rows x 13 columns]
```

```
print(y)
```

```
0      1
1      1
2      1
3      1
4      1
      ..
298    0
299    0
300    0
301    0
302    0
Name: output, Length: 303, dtype: int64
```

```
from IPython.testing import test
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,stratify=y,random_state=2)
```

```
print(x.shape,x_train.shape,x_test.shape)
```

```
(303, 13) (242, 13) (61, 13)
```

Logistic Regression

```
model = LogisticRegression()
```

```
model.fit(x_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Conver
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-regression
  n_iter_i = _check_optimize_result(
▾ LogisticRegression
LogisticRegression()
```

```
x_train_prediction = model.predict(x_train)
training_data_accuracy = accuracy_score(x_train_prediction,y_train)
```

```
print('Accuracy on Training data:',training_data_accuracy)
```

```
Accuracy on Training data: 0.8512396694214877
```

```
x_test_prediction = model.predict(x_test)
test_data_accuracy = accuracy_score(x_test_prediction,y_test)
```

```
print('Accuracy on Test data:',test_data_accuracy)
```

```
Accuracy on Test data: 0.819672131147541
```

```
input_data=(41,0,1,130,204,0,172,0,1,4,2,0,2)

input_data_as_numpy_array = np.asarray(input_data)
input_data_reshaped=input_data_as_numpy_array.reshape(1,-1)
prediction = model.predict(input_data_reshaped)
print(prediction)

if (prediction[0]==0):
  print('Person doesnot have heart disease')
else:
  print('Person has a heart disease')
```

```
[1]
Person has a heart disease
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LogisticRegressi
  warnings.warn(
```

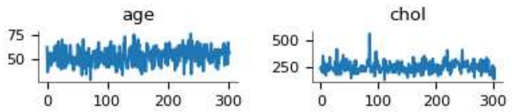
Kmeans clustering

```
X=heart_data[["age","chol"]]
X
```

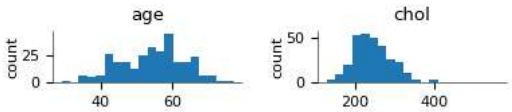
	age	chol
0	63	233
1	37	250
2	41	204
3	56	236
4	57	354
...	...	...
298	57	241
299	45	264
300	68	193
301	57	131
302	57	236

303 rows × 2 columns

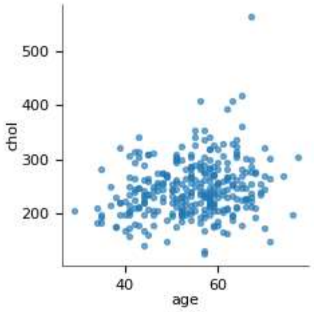
Values



Distributions

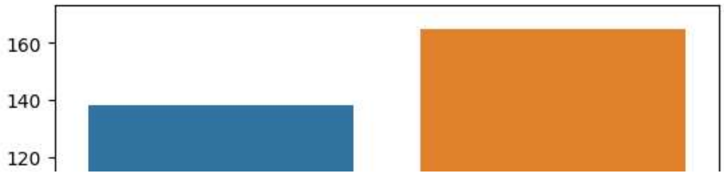


2-d distributions



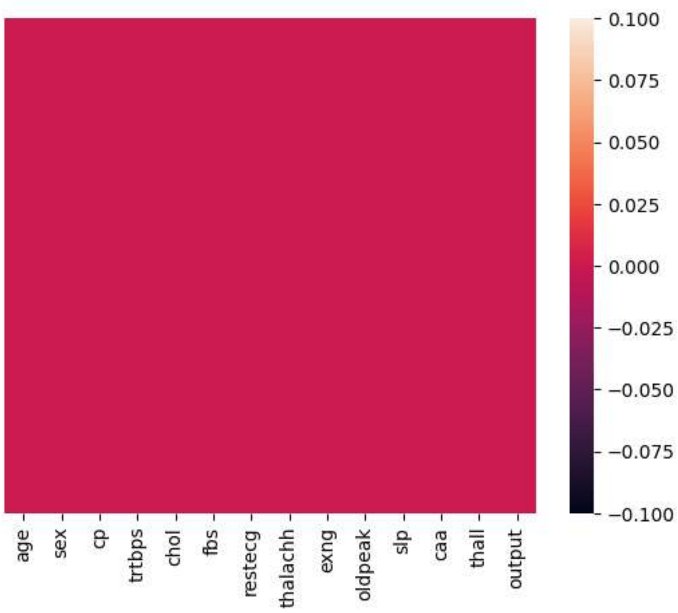

```
sns.countplot(x='output',data=heart_data)
```

```
<Axes: xlabel='output', ylabel='count'>
```




```
sns.heatmap(heart_data.isnull(),yticklabels=False)
```

```
<Axes: >
```



Conclusion:

This study utilized the Heart Disease UCI dataset which consisted of fourteen variables including age, sex, cp, fbs, restecg, thalac, exang, oldpeak, slope, ca, thal, and output to determine how well the logistic regression algorithm performs in predicting cardiovascular disease. Based on the results of data validation, the accuracy of the prediction results is 85% and the error rate tends to be small at 0.1406565. These results demonstrate that this algorithm can be utilized as a prediction algorithm in the current study. According to cardiovascular disease predictions, gender, trestbps - blood pressure level, thalach - heart rate, and the number of vessels affected by fluoroscopy have significant influence on possibility of heart disease. Increase in these variables value will have an impact on overall cardiovascular performance.

KNN Algorithm

1.KNN is used for both classifications as well as regression tasks in Machine learning.

2.KNN tries to find similarities between predictors and values that are within the dataset.

3.KNN uses a non-parametric method as there is not a particular finding of parameters to a particular functional form

Working of KNN Algorithm:

Initially, we select a value for K in our KNN algorithm.

Now we go for a distance measure. Let's consider Eucleadean distance here. Find the euclidean distance of k neighbours.

Now we check all the neighbours to the new point we have given and see which is nearest to our point. We only check for k-nearest here.

Now we see to which class there is the highest number obtained. The max number is chosen and we assign our new point to that class.

In this way, we use the KNN algorithm

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

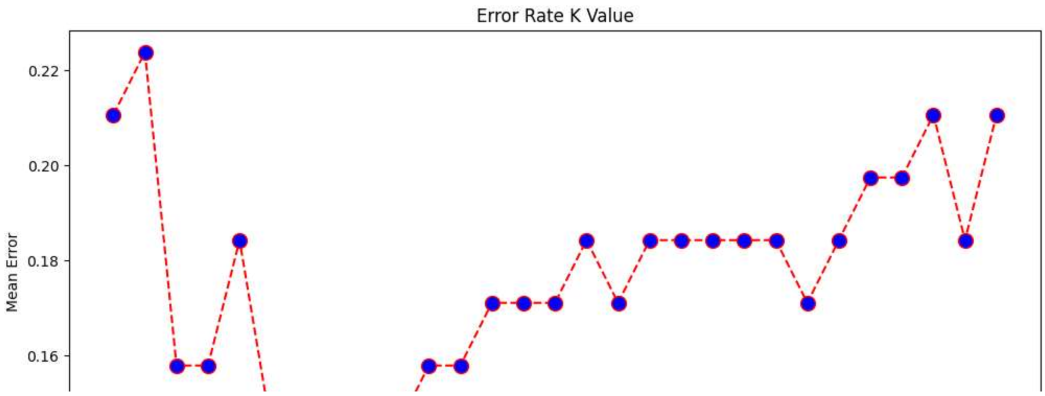
```
df=pd.read_csv('/content/heart.csv')
df.head()
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

```
x=df.iloc[:,0:13].values
y= df['output'].values
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

```
error=[]
for i in range(1, 30):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train, y_train)
    pred_i = knn.predict(x_test)
    error.append(np.mean(pred_i != y_test))
plt.figure(figsize=(12, 6))
plt.plot(range(1, 30), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
print("Minimum error:-",min(error),"at K =",error.index(min(error))+1)
```

```
Minimum error:- 0.13157894736842105 at K = 7
```




```
classifier= KNeighborsClassifier(n_neighbors=7)
classifier.fit(x_train, y_train)
y_pred= classifier.predict(x_test)
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
```

```
accuracy_score(y_test, y_pred)
```

```
0.868421052631579
```

Conclusion:

Manually determining the odds of cardiovascular disease based on risk factors can be hard. Using Machine learning techniques we can predict the outcome with the help of existing data. But still, we can’t trust the machine always. As you can see from this prediction, we got some percentage of “False positives and False negatives”. The only way to prevent cardiovascular disease is to stay healthy.

KMeans

This research investigates anomaly detection in the healthcare domain to effectively predict heart disease using unsupervised K-means clustering algorithm. Our proposed model first determines an optimal value of K using the Silhouette method to form the clusters for finding the anomalies

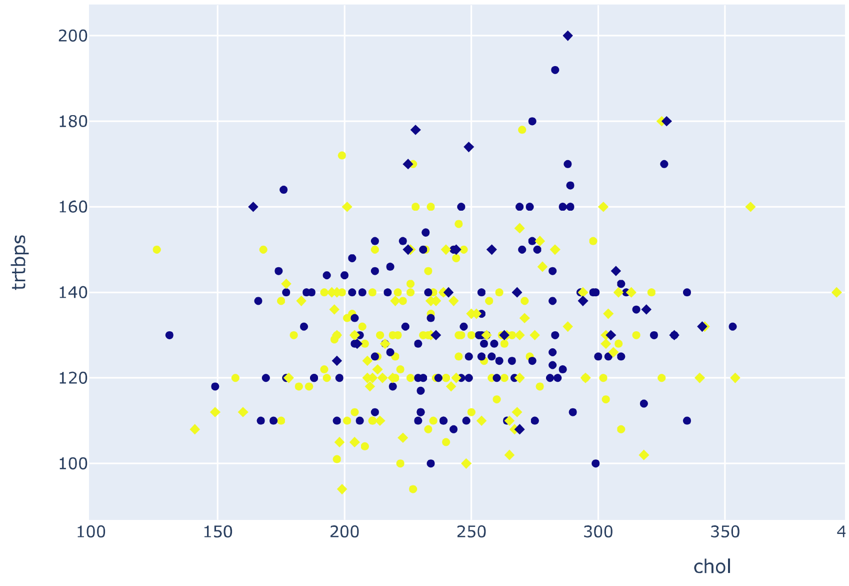
K-means clusters are a nice way to visualize data when we are not sure what we are looking for. Finding clusters then labeling the data with the cluster labels to create your own “target” feature is a great way to handle unlabeled data for unsupervised machine learning. As you can see, one application can be finding hidden features that may indicate a disease state in patients. Finding the features that most accurately indicate a given disease can save both money and lives.

```
import pandas as pd
df = pd.read_csv('/content/heart.csv')
```

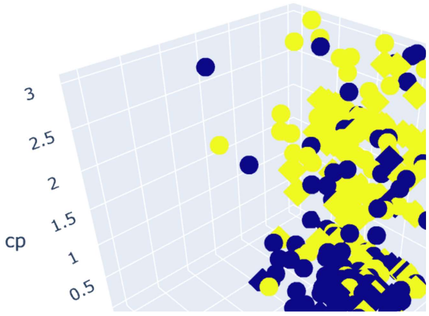
```
df.head()
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2

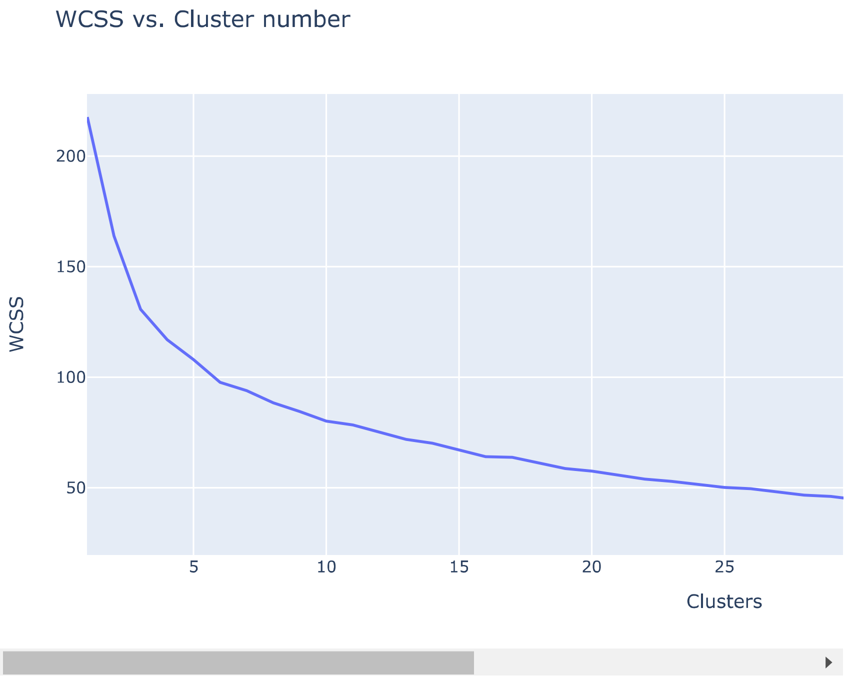
```
import plotly.express as px
fig = px.scatter(df, x='chol', y="trtbps", color='output', symbol="sex")
fig.show()
```




```
import plotly.express as px
fig = px.scatter_3d(df, x='oldpeak', y="age",z='cp', color='output', symbol="sex")
fig.show()
```




```
from sklearn.model_selection import train_test_split
def normalize(df, features):
    result = df.copy()
    for feature_name in features:
        max_value = df[feature_name].max()
        min_value = df[feature_name].min()
        result[feature_name] = (df[feature_name] - min_value) / (max_value - min_value)
    return result

normalised_df = normalize(df,df.columns)

X_train, X_test, y_train, y_test = train_test_split( normalised_df.drop(["output"], axis=1), normalised_df["output"], test_size=0.33, ra


import plotly.graph_objects as go
from sklearn.cluster import KMeans, AgglomerativeClustering, AffinityPropagation, DBSCAN


wcss = []
clusters = 50
for i in range(1, clusters):
    kmeans = KMeans(n_clusters = i, init = "k-means++", max_iter = 500, n_init = 10, random_state = 123)
    kmeans.fit(X_train.values)
    wcss.append(kmeans.inertia_)

fig = go.Figure(data = go.Scatter(x = [i for i in range(1, clusters)], y = wcss))


fig.update_layout(title='WCSS vs. Cluster number',
                  xaxis_title='Clusters',
                  yaxis_title='WCSS')
fig.show()
```

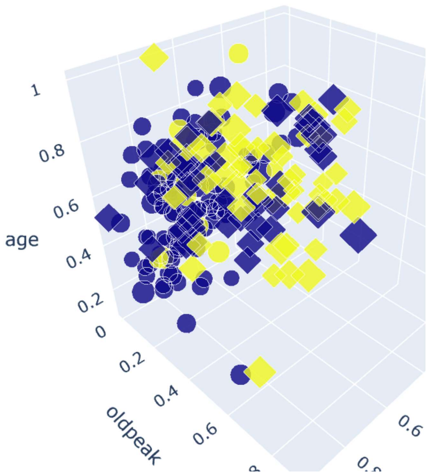


```
kmeans = KMeans(n_clusters = 2, init="k-means++", max_iter = 500, n_init = 10, random_state = 42)
identified_clusters = kmeans.fit_predict(X_train.values)
```

```
copy_X_train = X_train.copy()
copy_X_train['Cluster'] = identified_clusters
```

```
from scipy.spatial import distance
distance_from_centroid = []
for values in  copy_X_train.values:
    distance_from_centroid.append(distance.euclidean(values[:-1],kmeans.cluster_centers_[int(values[-1])]))

copy_X_train['distance_from_centriods'] = distance_from_centroid
```

```
fig = px.scatter_3d(copy_X_train,  x='thalachh', y="oldpeak",z='age',
              color='Cluster', opacity = 0.8, size='distance_from_centriods',symbol=y_train, size_max=30)
fig.show()
```




```
from sklearn.metrics import classification_report, confusion_matrix

for i in range(1,3):
    knn1 = KMeans(i)
    knn1.fit(X_train,y_train)
    pred1 = knn1.predict(X_test)
    print(f"For Knn-{i}: \n")
    print(classification_report(y_test,pred1))
    print('===================================')
```

```
For Knn-1:

              precision    recall  f1-score   support

         0.0       0.42      1.00      0.59        42
         1.0       0.00      0.00      0.00        58

    accuracy                           0.42       100
   macro avg       0.21      0.50      0.30       100
weighted avg       0.18      0.42      0.25       100

===================================
For Knn-2:

              precision    recall  f1-score   support

         0.0       0.72      0.55      0.62        42
         1.0       0.72      0.84      0.78        58

    accuracy                           0.72       100
   macro avg       0.72      0.70      0.70       100
weighted avg       0.72      0.72      0.71       100

===================================
```

```
/usr/local/lib/python3.10/dist-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

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cc

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cc

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cc

/usr/local/lib/python3.10/dist-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
```

Conclusion: From the analysis of the data using the K-Means Algorithm we get to know that age is the mainfactor in the prediction of the disease as the centroids of the plots where we consider age as an attribute has the biggest concentration and similarities. The model predicts the disease using

0s completed at 11:30 PM