C.K.Pithawala College of Engineering and Technology, Surat
Div. : A
Subject : Machine Learning
Subject Code : 3170724
1.Name:KUNAL MAURYA Enrollment: 200090107084
2.Name:KALASH THAKKAR Enrollment: 200090107043
Data Set : Heart Failure Prediction Dataset
Data Set Link :https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction/data
Academic Year : 2022-2023
Subject Faculty : Dr. Ami Tusharkant Choksi

https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction/data

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
pd.options.display.float_format = '{:.2f}'.format
import warnings
warnings.filterwarnings('ignore')
```

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

	Age	Sex	ChestPainType	RestingBP	Cholesterol	Fasting
0	40	M	ATA	140	289	
1	49	F	NAP	160	180	
2	37	M	ATA	130	283	
3	48	F	ASY	138	214	
4	54	M	NAP	150	195	

```
categorical_feature = df.dtypes==object
final_categorical_feature = df.columns[categorical_feature].tolist()
# ---------------------------------------------------------
final_numeric_feature = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']
```

CO-1. ASSIGNMENT

1.Implement the techniques to deal with outliers.

```
def outlier_detect(df, col):
    q1_col = Q1[col]
    iqr_col = IQR[col]
    q3_col = Q3[col]
    return df[((df[col] < (q1_col - 1.5 * iqr_col)) |(df[col] > (q3_col + 1.5 * iqr_col)))]
```

```
# ------------------------------------------------------------
def outlier_detect_normal(df, col):
    m = df[col].mean()
    s = df[col].std()
    return df[((df[col]-m)/s).abs()>3]

# ------------------------------------------------------------
def lower_outlier(df, col):
    q1_col = Q1[col]
    iqr_col = IQR[col]
    q3_col = Q3[col]
    lower = df[(df[col] < (q1_col - 1.5 * iqr_col))]
    return lower

# ------------------------------------------------------------
def upper_outlier(df, col):
    q1_col = Q1[col]
    iqr_col = IQR[col]
    q3_col = Q3[col]
    upper = df[(df[col] > (q3_col + 1.5 * iqr_col))]
    return upper

# ------------------------------------------------------------
def replace_upper(df, col):
    q1_col = Q1[col]
    iqr_col = IQR[col]
    q3_col = Q3[col]
    tmp = 9999999
    upper = q3_col + 1.5 * iqr_col
    df[col] = df[col].where(lambda x: (x < (upper)), tmp)
    df[col] = df[col].replace(tmp, upper)
    print('outlier replace with upper bound - {}' .format(col))

# ------------------------------------------------------------
def replace_lower(df, col):
    q1_col = Q1[col]
    iqr_col = IQR[col]
    q3_col = Q3[col]
    tmp = 1111111
    lower = q1_col - 1.5 * iqr_col
    df[col] = df[col].where(lambda x: (x > (lower)), tmp)
    df[col] = df[col].replace(tmp, lower)
    print('outlier replace with lower bound - {}' .format(col))

# ------------------------------------------------------------
def preprocess(df, col):
    print("*********************** {} **********************\n".format(col))
    print("lower outlier: {} ****** upper outlier: {}\n".format(lower_outlier(df,col).shape[0],
    plt.figure(figsize=(10,8))
    plt.subplot(2,1,1)
    df[col].plot(kind='box', subplots=True, sharex=False, vert=False)
    plt.subplot(2,1,2)
    df[col].plot(kind='density', subplots=True, sharex=False)
    plt.show()
```

```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
```

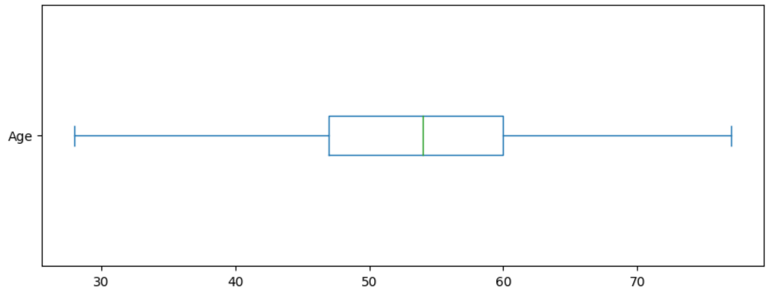
```
for i in range(len(final_numeric_feature)):
    print("IQR => {}: {}".format(final_numeric_feature[i],(outlier_detect(df[final_numeric_featu
    print("Z_Score => {}: {}".format(final_numeric_feature[i],(outlier_detect_normal(df[final_nu
    print("********************************")
```

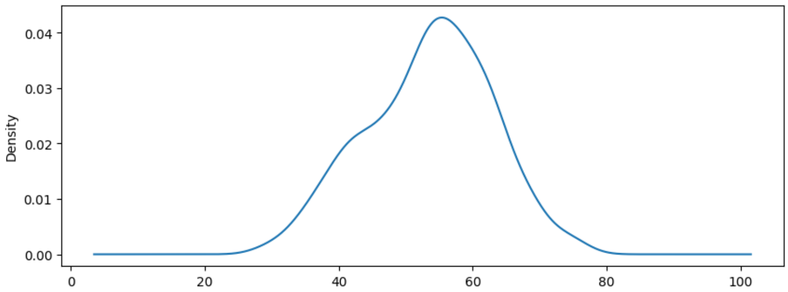
```
IQR => Age: 0
Z_Score => Age: 0
********************************
IQR => RestingBP: 28
Z_Score => RestingBP: 8
********************************
IQR => Cholesterol: 183
Z_Score => Cholesterol: 3
********************************
IQR => MaxHR: 2
Z_Score => MaxHR: 1
********************************
IQR => Oldpeak: 16
Z_Score => Oldpeak: 7
********************************
```

```
for i in range(len(final_numeric_feature)):
    preprocess(df[final_numeric_feature], final_numeric_feature[i])
```

********************** Age **********************

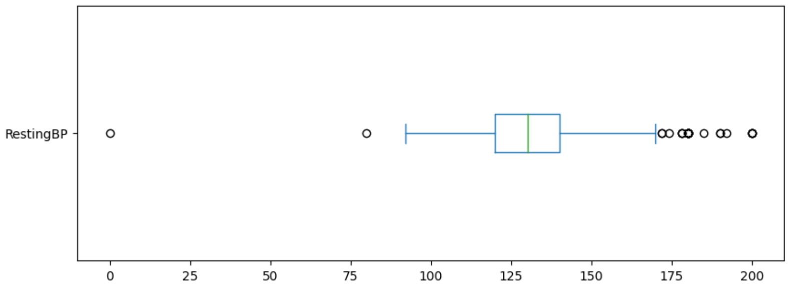
lower outlier: 0 ****** upper outlier: 0

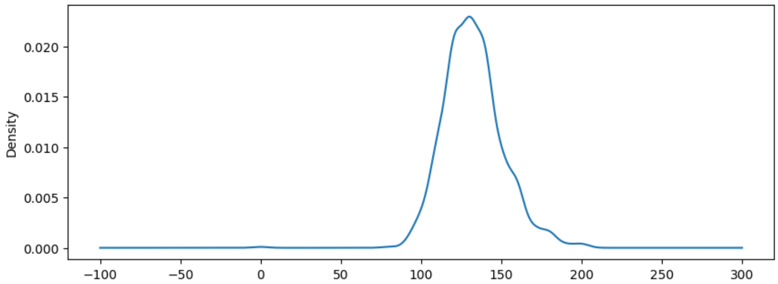




********************** RestingBP **********************

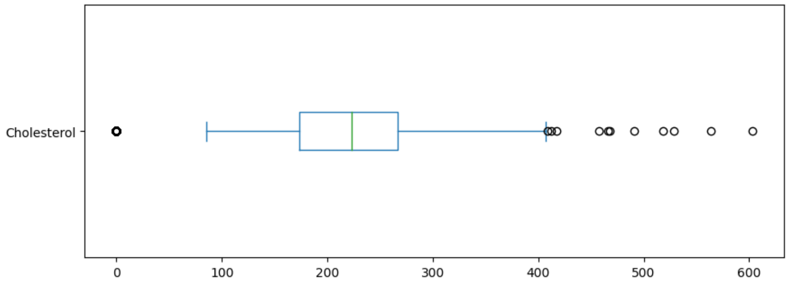
lower outlier: 2 ****** upper outlier: 26

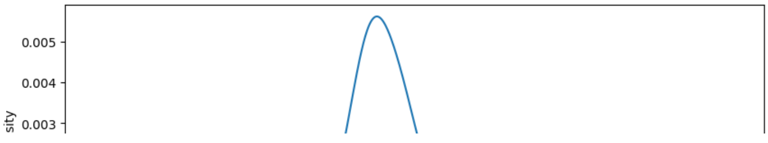


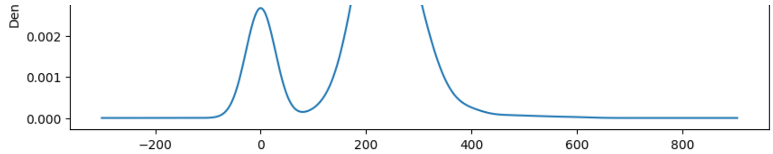


********************** Cholesterol **********************

lower outlier: 172 ****** upper outlier: 11

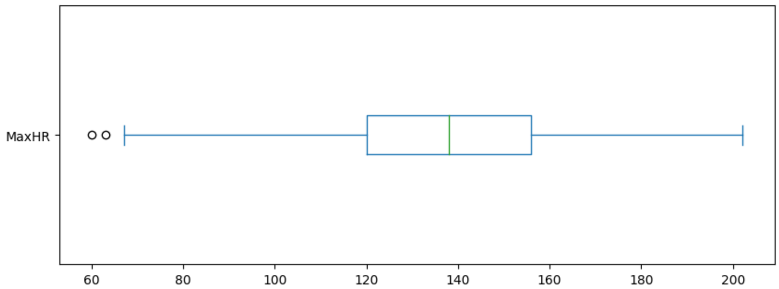


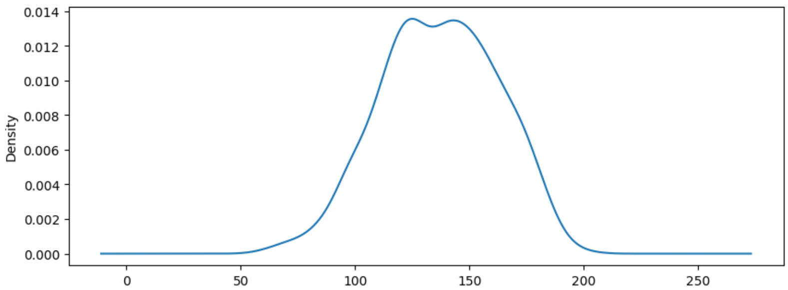




********************** MaxHR ***********************

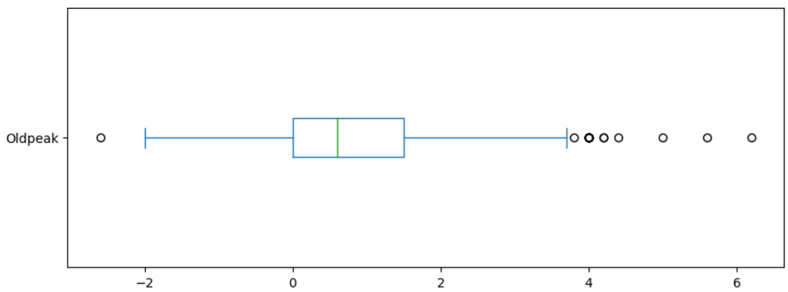
lower outlier: 2 ****** upper outlier: 0

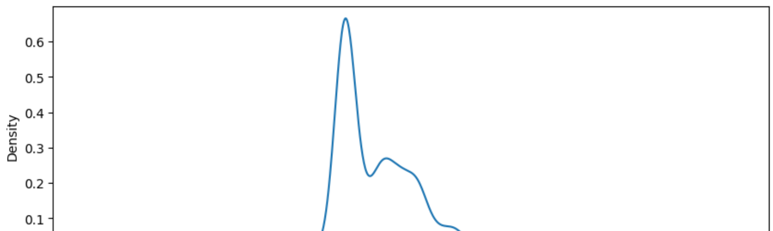




********************** Oldpeak ***********************

lower outlier: 1 ****** upper outlier: 15





```
outlier = []
for i in range(len(final_numeric_feature)):
    if outlier_detect(df[final_numeric_feature],final_numeric_feature[i]).shape[0] !=0:
        outlier.append(final_numeric_feature[i])

outlier
```

```
['RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']
```

```
for i in range(len(outlier)):
    replace_upper(df, outlier[i])

print("\n********************************\n")
for i in range(len(outlier)):
    replace_lower(df, outlier[i])
```

```
outlier replace with upper bound - RestingBP
outlier replace with upper bound - Cholesterol
outlier replace with upper bound - MaxHR
outlier replace with upper bound - Oldpeak

********************************

outlier replace with lower bound - RestingBP
outlier replace with lower bound - Cholesterol
outlier replace with lower bound - MaxHR
outlier replace with lower bound - Oldpeak
```

```
for i in range(len(final_numeric_feature)):
    print("IQR => {}: {}".format(final_numeric_feature[i],(outlier_detect(df,final_numeric_featu
    print("Z_Score => {}: {}".format(final_numeric_feature[i],(outlier_detect_normal(df,final_nu
    print("********************************")
```

```
IQR => Age: 0
Z_Score => Age: 0
********************************
IQR => RestingBP: 0
Z_Score => RestingBP: 0
********************************
IQR => Cholesterol: 0
Z_Score => Cholesterol: 0
********************************
IQR => MaxHR: 0
Z_Score => MaxHR: 0
********************************
```

```
IQR => Oldpeak: 0
Z_Score => Oldpeak: 1
********************************
```

2.Implement the techniques to deal with missing values.

```
# Check for missing values in the dataset

missing_values = df.isna().sum()

print(f"It seems like there are no missing values \n{missing_values}")
```

```
It seems like there are no missing values
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

CO-2. ASSIGNMENT

3.Implement distance measuring techniques for two features of your dataset:
(a) Euclidean (b)Minkowski (c) Manhattan (d) Jaccard (e) Cosine (f) Simple matching coefficient (g)hamming

```
# (a) Euclidean distance
feature1 = df['RestingBP'].values
feature2 = df['Cholesterol'].values

euclidean_distance = np.linalg.norm(feature1 - feature2)

print("Euclidean Distance between feature 1 and feature 2:", euclidean_distance)
```

```
Euclidean Distance between feature 1 and feature 2: 3622.403516227175
```

```
# (b) Minkowski distance
from scipy.spatial import distance
f1 = df['RestingBP'].values
f2 = df['Cholesterol'].values

p=2;

minkowski_distance = distance.minkowski(f1, f2, p)

print(f"Minkowski Distance (p={p}) between feature 1 and feature 2:", minkowski_distance)
```

```
Minkowski Distance (p=2) between feature 1 and feature 2: 3622.403516227175
```

```
# (c) Manhattan distance
ff1 = df['Cholesterol'].values
ff2 = df['MaxHR'].values

manhattan_distance = distance.cityblock(ff1, ff2)
```

```
manhattan_distance = distance.cityblock(f1, f2)

print("Manhattan Distance between feature 1 and feature 2:", manhattan_distance)
```

```
Manhattan Distance between feature 1 and feature 2: 93335.375
```

```
# (d) Jaccard distance
set1 = df['Cholesterol']
set2 = df['MaxHR']

jaccard_distance = distance.jaccard(set1, set2)

print("Jaccard Distance between set 1 and set 2:", jaccard_distance)
```

```
Jaccard Distance between set 1 and set 2: 1.0
```

```
# (e) Cosine distance
from sklearn.metrics.pairwise import cosine_distances

vector1 = df['Cholesterol'].values.reshape(1, -1)
vector2 = df['RestingBP'].values.reshape(1, -1)

cosine_distance = cosine_distances(vector1, vector2)

print("Cosine Distance between vector 1 and vector 2:", cosine_distance[0][0])
```

```
Cosine Distance between vector 1 and vector 2: 0.09653752602018406
```

```
# (f) Simple matching coefficient distance
from sklearn.metrics import pairwise_distances

vector1 = df['Oldpeak'].values.reshape(1, -1)
vector2 = df['HeartDisease'].values.reshape(1, -1)

smc = 1 - pairwise_distances(vector1, vector2, metric='hamming')

print("Simple Matching Coefficient between vector 1 and vector 2:", smc[0][0])
```

```
Simple Matching Coefficient between vector 1 and vector 2: 0.3311546840958606
```

```
# (g) Hamming distance
from sklearn.metrics.pairwise import pairwise_distances

column1 = df['Oldpeak'].astype(str)
column2 = df['HeartDisease'].astype(str)

hamming_distance = pairwise_distances(column1.str.count('1').values.reshape(-1, 1),
                                       column2.str.count('1').values.reshape(-1, 1),
                                       metric='manhattan')

print("Hamming Distance between column 1 and column 2:", hamming_distance[0][0])
```

```
Hamming Distance between column 1 and column 2: 0.0
```

4. Implement any data reduction technique.

```
df.head()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	Fasting
0	40	M	ATA	140	289.00	
1	49	F	NAP	160	180.00	
2	37	M	ATA	130	283.00	

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data = df
data[final_categorical_feature]= df[final_categorical_feature].apply(lambda col: le.fit_transfor
data.head(5)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel
  and should_run_async(code)
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	Fasting
0	40	1	1	140	289.00	
1	49	0	2	160	180.00	
2	37	1	1	130	283.00	
3	48	0	0	138	214.00	
4	54	1	2	150	195.00	

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
x = data.drop("HeartDisease", axis = 1)
y = data['HeartDisease']
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state =100 ,stratify=y, test_si
print(y_train.value_counts())
```

```
1    355
0    287
Name: HeartDisease, dtype: int64
```

```
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.ensemble import ExtraTreesClassifier, GradientBoostingClassifier, StackingClassifie
sfs = SFS(GradientBoostingClassifier(n_estimators=100, random_state=0),
          k_features = 7,
          forward= True,
          floating = False,
          verbose= 2,
          scoring= 'accuracy',
          cv = 4,
          n_jobs= -1).fit(x_train, y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  11 out of  11 | elapsed:    6.8s finished

[2023-11-03 13:31:22] Features: 1/7 -- score: 0.8130628881987577[Parallel(n_jobs=-1)]: Usir
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    3.1s finished

[2023-11-03 13:31:25] Features: 2/7 -- score: 0.8364615683229814[Parallel(n_jobs=-1)]: Usir
[Parallel(n_jobs=-1)]: Done   9 out of   9 | elapsed:    3.0s finished

[2023-11-03 13:31:28] Features: 3/7 -- score: 0.8457880434782609[Parallel(n_jobs=-1)]: Usir
[Parallel(n_jobs=-1)]: Done   8 out of   8 | elapsed:    3.3s finished

[2023-11-03 13:31:31] Features: 4/7 -- score: 0.8598020186335403[Parallel(n_jobs=-1)]: Usir
[Parallel(n_jobs=-1)]: Done   7 out of   7 | elapsed:    3.7s finished
```

```
[2023-11-03 13:31:35] Features: 5/7 -- score: 0.866022903726708[Parallel(n_jobs=-1)]: Using
[Parallel(n_jobs=-1)]: Done   6 out of   6 | elapsed:    2.3s finished

[2023-11-03 13:31:37] Features: 6/7 -- score: 0.8707201086956522[Parallel(n_jobs=-1)]: Usin
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    2.0s finished

[2023-11-03 13:31:39] Features: 7/7 -- score: 0.876950698757764
```

```
print("Best features: ",sfs.k_feature_names_)
print("Best score: ",sfs.k_score_)
```

```
Best features:  ('Sex', 'Cholesterol', 'FastingBS', 'RestingECG', 'ExerciseAngina', 'Oldpea
Best score:  0.876950698757764
```

```
x_train_new = x_train[['Sex','Cholesterol','FastingBS','RestingECG','ExerciseAngina','Oldpeak','
x_test_new = x_test[['Sex','Cholesterol','FastingBS','RestingECG','ExerciseAngina','Oldpeak','ST
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `sho
  and should_run_async(code)
```

CO-3. ASSIGNMENT

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import ExtraTreesClassifier, GradientBoostingClassifier, StackingClassifie
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, f1_score,confusion_matrix, recall_score, precision_s
from sklearn.metrics import average_precision_score
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `sho
  and should_run_async(code)
```

5.Implement various knn classification algorithms and do prediction for unknown data.

```
from sklearn.neighbors import KNeighborsClassifier
KNN = KNeighborsClassifier(n_neighbors=8)
KNN.fit(x_train_new, y_train)

y_test_pred_KNN = KNN.predict(x_test_new)
y_train_pred_KNN = KNN.predict(x_train_new)

test_acc_KNN = accuracy_score(y_test, y_test_pred_KNN)
train_acc_KNN = accuracy_score(y_train, y_train_pred_KNN)
scores_KNN = cross_val_score(KNN, x_train_new , y_train , cv = 10, scoring = 'accuracy' )

precision_score_KNN = precision_score(y_test, y_test_pred_KNN)
recall_score_KNN = recall_score(y_test, y_test_pred_KNN)
f1_score_KNN = f1_score(y_test, y_test_pred_KNN)
conf_KNN = confusion_matrix(y_test, y_test_pred_KNN)
```

```
print("Tain set Accuracy: ", train_acc_KNN)
print("Test set Accuracy: ", test_acc_KNN)
print("cv:  %s\n"% scores_KNN.mean())
print("************************************************")
print("precision_score: ", precision_score_KNN)
print("recall_score: ", recall_score_KNN)
print("f1_score: ", f1_score_KNN)
print("************************************************")
print("\nReport:\n%s\n"%classification_report(y_test, y_test_pred_KNN))
```

```
Tain set Accuracy:  0.7772585669781932
Test set Accuracy:  0.6847826086956522
cv:  0.6867788461538462

************************************************
precision_score:  0.8173076923076923
recall_score:  0.5555555555555556
f1_score:  0.6614785992217899
************************************************

Report:
              precision    recall  f1-score   support

           0       0.60      0.85      0.71       123
           1       0.82      0.56      0.66       153

    accuracy                           0.68       276
   macro avg       0.71      0.70      0.68       276
weighted avg       0.72      0.68      0.68       276


/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `shc
  and should_run_async(code)
```

6.Implement a decision tree classification algorithm.

```
DT = DecisionTreeClassifier(max_depth=5, min_samples_leaf=5, random_state=0)
DT.fit(x_train_new, y_train)

y_test_pred_DT = DT.predict(x_test_new)
y_train_pred_DT = DT.predict(x_train_new)

test_acc_DT = accuracy_score(y_test, y_test_pred_DT)
train_acc_DT = accuracy_score(y_train, y_train_pred_DT)
scores_DT = cross_val_score(DT, x_train_new , y_train , cv = 10, scoring = 'accuracy' )

precision_score_DT = precision_score(y_test, y_test_pred_DT)
recall_score_DT = recall_score(y_test, y_test_pred_DT)
f1_score_DT = f1_score(y_test, y_test_pred_DT)
conf_DT = confusion_matrix(y_test, y_test_pred_DT)


print("Tain set Accuracy: ", train_acc_DT)
print("Test set Accuracy: ", test_acc_DT)
print("cv:  %s\n"% scores_DT.mean())
print("************************************************")
print("precision_score: ", precision_score_DT)
print("recall_score: ", recall_score_DT)
print("f1_score: ", f1_score_DT)
print("************************************************")
print("\nReport:\n%s\n"%classification_report(y_test, y_test_pred_DT))
```

```
Tain set Accuracy:  0.8956386292834891
Test set Accuracy:  0.8333333333333334
```

```
cv:  0.8738461538461539

************************************************
precision_score:  0.8590604026845637
recall_score:  0.8366013071895425
f1_score:  0.847682119205298
************************************************

Report:
               precision    recall  f1-score   support

           0       0.80      0.83      0.82       123
           1       0.86      0.84      0.85       153

    accuracy                           0.83       276
   macro avg       0.83      0.83      0.83       276
weighted avg       0.83      0.83      0.83       276
```

7..Implement a support vector machine algorithm.

```
SVM = SVC(C=10)
SVM.fit(x_train_new, y_train)

y_test_pred_SVM = SVM.predict(x_test_new)
y_train_pred_SVM = SVM.predict(x_train_new)

test_acc_SVM = accuracy_score(y_test, y_test_pred_SVM)
train_acc_SVM = accuracy_score(y_train, y_train_pred_SVM)
scores_SVM = cross_val_score(SVM, x_train_new , y_train , cv = 10, scoring = 'accuracy' )

precision_score_SVM = precision_score(y_test, y_test_pred_SVM, average='macro')
recall_score_SVM = recall_score(y_test, y_test_pred_SVM, average='macro')
f1_score_SVM = f1_score(y_test, y_test_pred_SVM, average='macro')
conf_SVM = confusion_matrix(y_test, y_test_pred_SVM)


print("Tain set Accuracy: ", train_acc_SVM)
print("Test set Accuracy: ", test_acc_SVM)
print("cv:  %s\n"% scores_SVM.mean())
print("************************************************")
print("precision_score: ", precision_score_SVM)
print("recall_score: ", recall_score_SVM)
print("f1_score: ", f1_score_SVM)
print("************************************************")
print("\nReport:\n%s\n"%classification_report(y_test, y_test_pred_SVM))
```

```
Tain set Accuracy:  0.6355140186915887
Test set Accuracy:  0.6014492753623188
cv:  0.5996153846153847

************************************************
precision_score:  0.6525734112960226
recall_score:  0.6261756735214411
f1_score:  0.5910560344827587
************************************************

Report:
               precision    recall  f1-score   support

           0       0.53      0.85      0.66       123
           1       0.77      0.40      0.53       153

    accuracy                           0.60       276
   macro avg       0.65      0.63      0.59       276
```

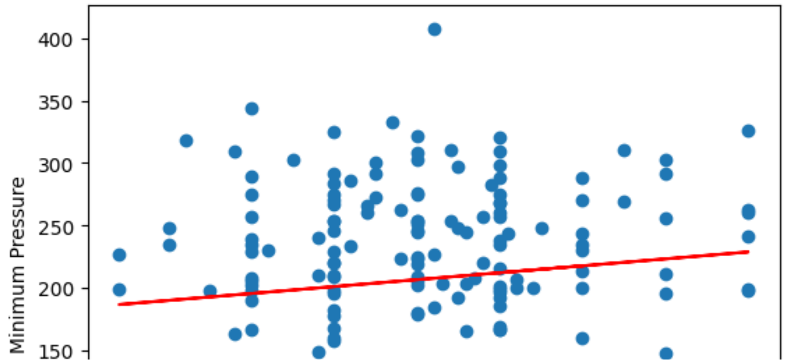
```
weighted avg       0.67      0.60      0.58       276
```

8.Implement regression algorithms:

(a)linear regression

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

X = data['RestingBP'].values.reshape(-1, 1)
y = data['Cholesterol'].values.reshape(-1, 1)

# 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)

# Create a linear regression model
reg = LinearRegression()

# Fit the model to the training data
reg.fit(X_train, y_train)

# Predict the target values of the test set
y_pred = reg.predict(X_test)


# Calculate performance metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

# Print the results
print('Mean Absolute Error (MAE):', mae)
print('Mean Squared Error (MSE):', mse)
print('Root Mean Squared Error (RMSE):', rmse)
print('R-squared (R^2):', r2)


# Plot the true vs predicted values
plt.scatter(X_test, y_test)
plt.xlabel('Maximum Wind')
plt.ylabel('Minimum Pressure')

# Add a regression line to the plot
plt.plot(X_test, y_pred, color='red')

plt.show()
```

```
Mean Absolute Error (MAE): 76.42114166894726
Mean Squared Error (MSE): 9691.53615613595
Root Mean Squared Error (RMSE): 98.44559998362521
R-squared (R^2): -0.021625100942510356
```



(b)logistic regression

```
LR = LogisticRegression(C=2, penalty='l1', random_state=0, solver='liblinear')
LR.fit(x_train_new, y_train)

y_test_pred_LR = LR.predict(x_test_new)
y_train_pred_LR = LR.predict(x_train_new)

test_acc_LR = accuracy_score(y_test, y_test_pred_LR)
train_acc_LR = accuracy_score(y_train, y_train_pred_LR)
scores_LR = cross_val_score(LR, x_train_new , y_train , cv = 10, scoring = 'accuracy' )

precision_score_LR = precision_score(y_test, y_test_pred_LR)
recall_score_LR = recall_score(y_test, y_test_pred_LR)
f1_score_LR = f1_score(y_test, y_test_pred_LR)
conf_LR = confusion_matrix(y_test, y_test_pred_LR)


print("Tain set Accuracy: ", train_acc_LR)
print("Test set Accuracy: ", test_acc_LR)
print("cv:  %s\n"% scores_LR.mean())
print("************************************************")
print("precision_score: ", precision_score_LR)
print("recall_score: ", recall_score_LR)
print("f1_score: ", f1_score_LR)
print("************************************************")
print("\nReport:\n%s\n"%classification_report(y_test, y_test_pred_LR))
```

```
Tain set Accuracy:  0.8504672897196262
Test set Accuracy:  0.8115942028985508
cv:  0.8505048076923079

************************************************
precision_score:  0.8435374149659864
recall_score:  0.8104575163398693
f1_score:  0.8266666666666667
************************************************

Report:
              precision    recall  f1-score   support

           0       0.78      0.81      0.79       123
           1       0.84      0.81      0.83       153

    accuracy                           0.81       276
   macro avg       0.81      0.81      0.81       276
weighted avg       0.81      0.81      0.81       276
```

CO-4. ASSIGNMENT

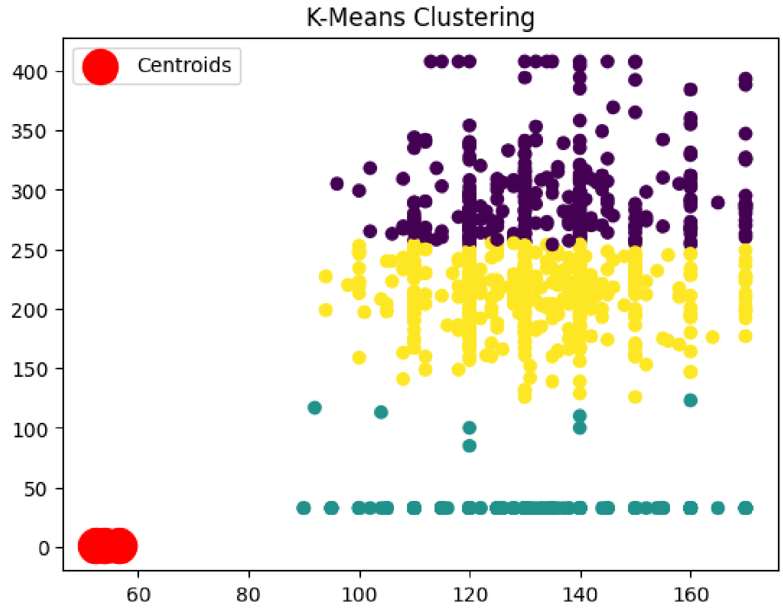
9.Implement k-means/k-medoid clustering algorithms and do prediction for unknown data.

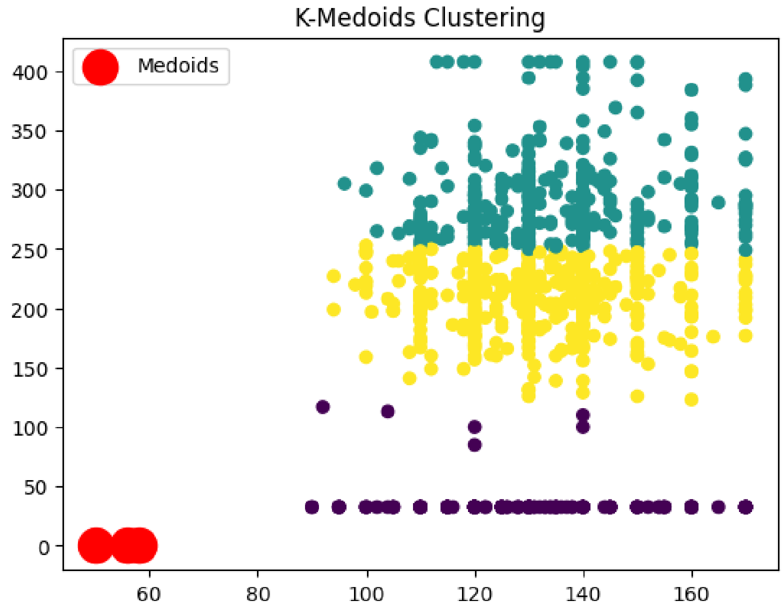
```
pip install scikit-learn-extra
```

```
Collecting scikit-learn-extra
  Downloading scikit_learn_extra-0.3.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 25.2 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: scikit-learn>=0.23.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packa
Installing collected packages: scikit-learn-extra
Successfully installed scikit-learn-extra-0.3.0
```

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn_extra.cluster import KMedoids
import matplotlib.pyplot as plt


X = data

# Perform K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

# Perform K-Medoids clustering
kmedoids = KMedoids(n_clusters=3, random_state=42)
kmedoids.fit(X)

# Predict clusters for the data points
kmeans_labels = kmeans.predict(X)
kmedoids_labels = kmedoids.predict(X)

# Visualize the clusters for K-Means
plt.scatter(X['RestingBP'], X['Cholesterol'], c=kmeans_labels, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], s=300, c='red', label='C
plt.title('K-Means Clustering')
plt.legend()
plt.show()

# Visualize the clusters for K-Medoids
plt.scatter(X['RestingBP'], X['Cholesterol'], c=kmedoids_labels, cmap='viridis')
plt.scatter(kmedoids.cluster_centers_[:,0], kmedoids.cluster_centers_[:,1], s=300, c='red', labe
plt.title('K-Medoids Clustering')
plt.legend()
plt.show()

# Predict clusters for unknown data points
unknown_data = np.array([[140, 150, 130, 170,190,166,125,190,210,170,180,160], [289.00, 280.00,
kmeans_prediction = kmeans.predict(unknown_data)
kmedoids_prediction = kmedoids.predict(unknown_data)
```

```
print("K-Means Prediction for Unknown Data:", kmeans_prediction)
print("K-Medoids Prediction for Unknown Data:", kmedoids_prediction)
```



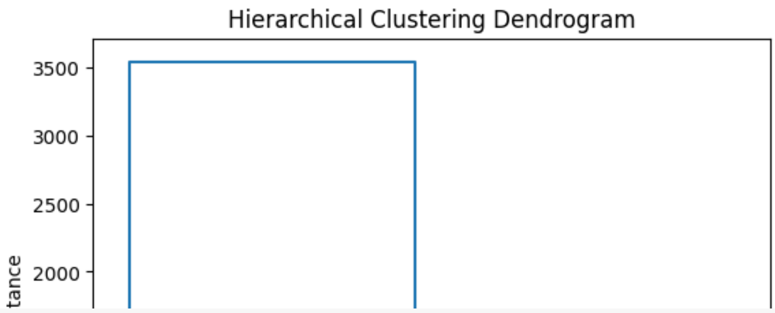



```
K-Means Prediction for Unknown Data: [2 0]
K-Medoids Prediction for Unknown Data: [2 1]
```

10. Implement hierarchical clustering algorithms and do prediction for unknown data.

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
import matplotlib.pyplot as plt


X = df
```

```
# Perform hierarchical clustering
linkage_matrix = linkage(X, method='ward', metric='euclidean')

# Create a dendrogram
dendrogram(linkage_matrix)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()

# Determine the number of clusters using the dendrogram
num_clusters = 3  # Adjust this based on the dendrogram

# Perform clustering to assign data points to clusters
clusters = fcluster(linkage_matrix, t=num_clusters, criterion='maxclust')

# Visualize the clusters for the Iris dataset
plt.scatter(X['RestingBP'], X['Cholesterol'], c=clusters, cmap='viridis')
plt.title('Hierarchical Clustering ')
plt.xlabel('RestingBP')
plt.ylabel('Cholesterol')
plt.show()
```




```
# Predict clusters for unknown data points
unknown_data = np.array([[140, 150, 130, 170], [289.00, 280.00, 290.00, 283.00]])  # Replace wit

# Rebuild the linkage matrix with the unknown data points
linkage_matrix_unknown = linkage(unknown_data, method='ward', metric='euclidean')

# Assign clusters to the unknown data points
unknown_clusters = fcluster(linkage_matrix, t=num_clusters, criterion='maxclust')

print("Clusters for Unknown Data:", unknown_clusters)
```

```
Clusters for Unknown Data: [2 3 2 3 3 2 3 3 3 2 3 3 3 3 3 2 3 3 3 3 3 3 3 2 3 3 2 2 2 3 2 3
 3 3 3 3 2 3 3 3 2 3 3 2 2 3 3 3 3 3 3 3 2 2 3 3 3 2 3 2 2 3 3 2 3 3 3 3
 3 3 2 3 3 3 2 3 3 3 3 3 2 3 2 3 3 2 3 2 3 3 3 3 3 3 2 3 2 2 3 3 2 3 3 3 3
 3 2 2 3 3 2 2 3 3 3 3 3 2 3 2 3 3 3 3 2 3 2 3 2 3 3 3 3 3 2 2 2 3 2 2 3 3
 3 2 3 3 3 3 2 2 3 3 2 3 2 3 3 3 3 2 3 3 3 3 3 3 2 3 3 2 3 2 3 3 3 3 2 3 2
 3 2 3 2 2 2 3 3 3 3 3 2 3 2 2 3 3 3 3 3 3 2 3 3 3 2 2 2 3 3 3 2 3 3 3 3 2
 3 2 3 3 3 3 2 3 2 2 2 3 2 3 2 2 2 3 2 3 2 2 2 3 3 2 3 2 2 2 2 2 3 2 2 3 3
 2 2 3 3 3 3 2 2 3 3 3 3 3 3 3 2 3 3 3 2 3 3 3 3 2 2 3 3 3 2 3 3 3 3 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 2 3 3 3 3 1 3 1 1 2 3 1 1 1 1 3 3 3 1 1 1 1 1 1 1 1 1 3
 3 3 1 3 3 1 1 1 2 1 3 1 1 1 1 1 2 1 3 1 1 3 1 1 3 3 1 1 1 2 1 1 3 1 2 1 1
 1 2 1 1 3 3 3 3 3 3 3 1 2 3 2 2 3 2 2 3 2 3 3 3 3 3 3 1 3 3 3 3 3 1 1 3 3
 1 2 3 2 2 3 3 3 2 2 3 3 3 2 3 3 3 1 1 2 3 3 3 3 2 3 2 3 3 3 3 3 3 3 3 3 3
 3 2 3 3 2 3 3 2 3 2 2 3 3 2 3 3 3 3 3 3 2 3 2 3 2 3 3 3 2 3 2 2 3 3 3 3 3
 2 3 3 3 2 3 3 3 3 2 2 2 3 2 3 2 3 2 2 3 3 2 2 2 2 3 2 2 3 3 3 2 2 3 3 3 3
 2 3 2 3 3 3 3 3 2 3 2 3 3 2 3 2 3 3 3 2 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 2
 3 2 3 3 3 3 3 3 2 3 2 3 3 3 2 3 2 3 3 3 2 3 3 3 2 3 3 3 3 3 3 2 3 3 3 3 2
 2 3 3 2 2 3 3 2 2 3 2 3 3 3 2 2 3 2 3 2 3 3 2 3 2 2 2 2 3 2 3 2 3 2 3 2 2
 3 2 3 3 3 3 3 3 2 3 3 2 3 3 3 3 3 3 3 2 2 3 3 3 3 3 3 3 2 3 3 3 2 3 3 3 3
 3 3 3 2 3 3 3 2 3 3 2 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 3 3 2 2 3 3 3 3 3
 2 2 2 3 2 2 2 2 3 2 3 2 3 3 3 3 3 3 3 3 3 3 3 2 3 3 2 2 2 3 2 3 3 3 3 3 3
 3 3 3 2 2 2 3 2 2 3 3 2 3 2 3 3 2 3 3 2 3 2 2 3 2 3 2 2 3 3 3 2 3 2 3 3 3
 3 3 3 2 3 3 3 2 3 3 3 3 2 3 3 3 3 3 2 3 3 3 3 3 3 2 3 3 3 3]
```

11. Implement DBSCAN clustering algorithms and do prediction for unknown data.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
data_1=data[['Sex','Cholesterol','FastingBS','RestingECG','ExerciseAngina','Oldpeak','ST_Slope']]
data_scaled = scaler.fit_transform(data_1)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `shc
  and should_run_async(code)
```

```
from sklearn.cluster import DBSCAN

# Create a DBSCAN model
dbscan = DBSCAN(eps=0.5, min_samples=5)
```

```
# Fit the model to the data
clusters = dbscan.fit_predict(data_scaled)

# Add the cluster labels to the original dataset
data['Cluster'] = clusters
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `sho
  and should_run_async(code)
```

```
data_1
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel
  and should_run_async(code)
```

	Sex	Cholesterol	FastingBS	RestingECG	ExerciseAngin
0	1	289.00	0	1	
1	0	180.00	0	1	
2	1	283.00	0	2	
3	0	214.00	0	1	
4	1	195.00	0	1	
...	...	...	...	...	
913	1	264.00	0	1	
914	1	193.00	1	1	
915	1	131.00	0	1	
916	0	236.00	0	0	
917	1	175.00	0	1	

918 rows × 7 columns

```
unknown_data = pd.DataFrame({
    'Sex':[1],'Cholesterol': [195.0],'FastingBS': [0],'RestingECG': [1],'ExerciseAngina': [0],'O
})

# Scale the unknown data using the same scaler
unknown_data_scaled = scaler.transform(unknown_data)

# Predict the cluster for the unknown data
unknown_cluster = dbscan.fit_predict(unknown_data_scaled)

print("Cluster for unknown data:", unknown_cluster)
```

```
Cluster for unknown data: [-1]
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `sho
  and should_run_async(code)
```

12. Implement apriori algorithm to get association rules.

```
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
import pandas as pd
```

```
# Sample transaction data (replace with your own dataset)
data = pd.DataFrame({
    'TransactionID': [1, 2, 3, 4, 5],
    'Items': ['A, B, D', 'B, C', 'A, C, D', 'A, D', 'B, C']
})

# Split items in the 'Items' column and create binary columns
items_df = data['Items'].str.get_dummies(', ')

# Concatenate the binary columns with the original DataFrame
data = pd.concat([data, items_df], axis=1)

# Drop the original 'Items' column
data.drop('Items', axis=1, inplace=True)

# Apply Apriori algorithm
frequent_itemsets = apriori(data.drop('TransactionID', axis=1), min_support=0.5, use_colnames=Tr

# Generate association rules
rules = association_rules(frequent_itemsets, metric='lift', min_threshold=1.0)

# Display association rules
print("Association Rules:")
print(rules)
```

```
Association Rules:
  antecedents consequents  antecedent support  consequent support  support  \
0         (A)         (D)                 0.60                0.60     0.60
1         (D)         (A)                 0.60                0.60     0.60

   confidence  lift  leverage  conviction  zhangs_metric
0        1.00  1.67      0.24         inf           1.00
1        1.00  1.67      0.24         inf           1.00
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `shc
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: Deprecat
  warnings.warn(
```

13. Implement backpropagation neural network algorithm.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow import keras
from tensorflow.keras import layers
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `shc
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/tensorflow/python/framework/dtypes.py:35: Deprecati
  from tensorflow.tsl.python.lib.core import pywrap_ml_dtypes
```

```
model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(x_train_new.shape[1],)),
    layers.Dense(32, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `sho
  and should_run_async(code)
```

```
model.fit(x_train_new, y_train, epochs=50, batch_size=32, validation_data=(x_test_new, y_test))
```

```
21/21 [==============================] - 0s 9ms/step - loss: 0.4931 - accuracy: 0.7695 -
Epoch 19/50
21/21 [==============================] - 0s 13ms/step - loss: 0.4674 - accuracy: 0.8100 -
Epoch 20/50
21/21 [==============================] - 0s 10ms/step - loss: 0.4581 - accuracy: 0.7991 -
Epoch 21/50
21/21 [==============================] - 0s 9ms/step - loss: 0.4623 - accuracy: 0.7991 -
Epoch 22/50
21/21 [==============================] - 0s 9ms/step - loss: 0.4769 - accuracy: 0.7882 -
Epoch 23/50
21/21 [==============================] - 0s 8ms/step - loss: 0.4417 - accuracy: 0.8240 -
Epoch 24/50
21/21 [==============================] - 0s 10ms/step - loss: 0.4457 - accuracy: 0.8100 -
Epoch 25/50
21/21 [==============================] - 0s 8ms/step - loss: 0.4347 - accuracy: 0.8209 -
Epoch 26/50
21/21 [==============================] - 0s 8ms/step - loss: 0.4758 - accuracy: 0.7866 -
Epoch 27/50
21/21 [==============================] - 0s 9ms/step - loss: 0.4512 - accuracy: 0.8162 -
Epoch 28/50
21/21 [==============================] - 0s 8ms/step - loss: 0.5740 - accuracy: 0.7414 -
Epoch 29/50
21/21 [==============================] - 0s 18ms/step - loss: 0.5028 - accuracy: 0.7913 -
Epoch 30/50
21/21 [==============================] - 0s 19ms/step - loss: 0.4236 - accuracy: 0.8271 -
Epoch 31/50
21/21 [==============================] - 0s 21ms/step - loss: 0.4226 - accuracy: 0.8255 -
Epoch 32/50
21/21 [==============================] - 0s 4ms/step - loss: 0.4506 - accuracy: 0.8100 -
Epoch 33/50
21/21 [==============================] - 0s 5ms/step - loss: 0.4174 - accuracy: 0.8271 -
Epoch 34/50
21/21 [==============================] - 0s 4ms/step - loss: 0.4072 - accuracy: 0.8318 -
Epoch 35/50
21/21 [==============================] - 0s 5ms/step - loss: 0.4104 - accuracy: 0.8442 -
Epoch 36/50
21/21 [==============================] - 0s 4ms/step - loss: 0.4247 - accuracy: 0.8209 -
Epoch 37/50
21/21 [==============================] - 0s 5ms/step - loss: 0.6093 - accuracy: 0.7227 -
Epoch 38/50
21/21 [==============================] - 0s 5ms/step - loss: 0.4236 - accuracy: 0.8224 -
Epoch 39/50
21/21 [==============================] - 0s 4ms/step - loss: 0.4235 - accuracy: 0.8178 -
Epoch 40/50
21/21 [==============================] - 0s 21ms/step - loss: 0.4123 - accuracy: 0.8224 -
Epoch 41/50
21/21 [==============================] - 0s 5ms/step - loss: 0.4060 - accuracy: 0.8302 -
Epoch 42/50
21/21 [==============================] - 0s 4ms/step - loss: 0.4548 - accuracy: 0.8084 -
Epoch 43/50
21/21 [==============================] - 0s 5ms/step - loss: 0.5064 - accuracy: 0.7866 -
Epoch 44/50
21/21 [==============================] - 0s 4ms/step - loss: 0.4151 - accuracy: 0.8271 -
Epoch 45/50
21/21 [==============================] - 0s 5ms/step - loss: 0.3954 - accuracy: 0.8442 -
Epoch 46/50
21/21 [==============================] - 0s 5ms/step - loss: 0.4380 - accuracy: 0.8115 -
```

```
loss, accuracy = model.evaluate(x_test_new, y_test)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
9/9 [==============================] - 0s 3ms/step - loss: 0.4157 - accuracy: 0.8116
Accuracy: 81.16%
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `shc
  and should_run_async(code)
```

14. Make a comparison tables for classification and clustering algorithms, for what you implemented here:

(a)Write unknown data:

```
unknown_data = pd.DataFrame({
    'Sex':[1],'Cholesterol': [195.0],'FastingBS': [0],'RestingECG': [1],'ExerciseAngina': [0],'O
})
```

(b)Compare performance of classification algorithms:

Algorithm name	Accuracy	Sensitivity	F-measure	Precision	Recall	Predicted value for unknown data
KNN	0.77	0.56	0.66	0.82	0.56	[1,264.00,0,1,0,1.20,1]
Decision tree	0.68	0.56	0.66	0.82	0.56	[0,192.00,0,1,0,1,1]
SVM	0.60	0.40	0.53	0.77	0.40	[1,190.00,1,1,0,1.20,1]
logistic regression	0.81	0.81	0.83	0.84	0.81	[1,224.00,0,1,0,1.20,1]

(c) Compare performance of clustering algorithms you implemented. Conclude which clustering algorithm is the best for your data.

```
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import silhouette_score

# Assuming you have your data stored in X

# K-means clustering
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(data[['Cholesterol','Oldpeak']])
kmeans_labels = kmeans.labels_
kmeans_silhouette_score = silhouette_score(data[['Cholesterol','Oldpeak']], kmeans_labels)

# Agglomerative clustering
agg = AgglomerativeClustering(n_clusters=3)
agg.fit(data[['Cholesterol','Oldpeak']])
agg_labels = agg.labels_
agg_silhouette_score = silhouette_score(data[['Cholesterol','Oldpeak']], agg_labels)

# Printing the results
print("Comparison of Clustering Algorithms:")
print(f"K-means Silhouette Score: {kmeans_silhouette_score}")
print(f"Agglomerative Clustering Silhouette Score: {agg_silhouette_score}")
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `shc
  and should_run_async(code)
Comparison of Clustering Algorithms:
K-means Silhouette Score: 0.6412359569219986
Agglomerative Clustering Silhouette Score: 0.6331864431898627
```

(d) Use different distance measures as in CO2's 3rd assignment and make a table to compare the performance of clustering algorithms you implemented. Conclude which clustering algorithm is the best for your data.

```
import numpy as np
from scipy.spatial.distance import cdist
from scipy.spatial.distance import cityblock, cosine, hamming

# Assuming you have already initialized X and the clustering algorithms
X=data[['Cholesterol','Oldpeak']]
# Calculate distances for K-means
kmeans_distances = {
    'Euclidean': cdist(X, kmeans.cluster_centers_, 'euclidean'),
    'Minkowski': cdist(X, kmeans.cluster_centers_, 'minkowski', p=3),
    'Manhattan': cdist(X, kmeans.cluster_centers_, 'cityblock'),
    'Jaccard': cdist(X, kmeans.cluster_centers_, 'jaccard'),
    'Cosine': cdist(X, kmeans.cluster_centers_, 'cosine'),
    'Simple matching coefficient': cdist(X, kmeans.cluster_centers_, 'hamming')
}

# Calculate distances for Agglomerative clustering
agg_distances = {
    'Euclidean': cdist(X, np.array([np.mean(X, axis=0)]), 'euclidean'),
    'Minkowski': cdist(X, np.array([np.mean(X, axis=0)]), 'minkowski', p=3),
    'Manhattan': cdist(X, np.array([np.mean(X, axis=0)]), 'cityblock'),
    'Jaccard': cdist(X, np.array([np.mean(X, axis=0)]), 'jaccard'),
    'Cosine': cdist(X, np.array([np.mean(X, axis=0)]), 'cosine'),
    'Simple matching coefficient': cdist(X, np.array([np.mean(X, axis=0)]), 'hamming')
}

# Create a table to compare the performance of clustering algorithms using different distance me
print("Comparison Table for Clustering Algorithms with Different Distance Measures:")
print("{:<30} {:<15} {:<15}".format('Distance Measure', 'K-means', 'Agglomerative'))
for key in kmeans_distances:
    print("{:<30} {:<15} {:<15}".format(key, np.mean(kmeans_distances[key]), np.mean(agg_distanc

kmeans_avg_distance = np.mean([np.mean(kmeans_distances[key]) for key in kmeans_distances])
agg_avg_distance = np.mean([np.mean(agg_distances[key]) for key in agg_distances])

if kmeans_avg_distance < agg_avg_distance:
    print("K-means clustering is better for this data based on average distance.")
elif kmeans_avg_distance > agg_avg_distance:
    print("Agglomerative clustering is better for this data based on average distance.")
else:
    print("Both clustering algorithms perform equally well on this data based on average distanc
```

```
Comparison Table for Clustering Algorithms with Different Distance Measures:
Distance Measure               K-means         Agglomerative
Euclidean                      114.69836183068112 73.93444631498534
Minkowski                      114.67886375312288 73.91572636510742
Manhattan                      115.53680360449827 74.77196816514065
Jaccard                        1.0             1.0
Cosine                         0.0001881248529648123 0.00014330835913913814
Simple matching coefficient    1.0             1.0
Agglomerative clustering is better for this data based on average distance.
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `shc
  and should_run_async(code)
```

15. Write any deep learning program of your choice.

```
data
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel
  and should_run_async(code)
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	Fasti
0	40	M	ATA	140	289	
1	49	F	NAP	160	180	
2	37	M	ATA	130	283	
3	48	F	ASY	138	214	
4	54	M	NAP	150	195	
...	...	...	...	...	...	
913	45	M	TA	110	264	
914	68	M	ASY	144	193	
915	57	M	ASY	130	131	
916	57	F	ATA	130	236	
917	38	M	NAP	138	175	

918 rows × 12 columns

```
import numpy as np
import pandas as pd

from sklearn.feature_selection import SelectKBest ,chi2 ,f_classif
from sklearn.preprocessing import StandardScaler , MinMaxScaler

from sklearn.linear_model import LogisticRegression , Lasso , RidgeClassifier
#from lazypredict.Supervised import LazyClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score , confusion_matrix
from sklearn.model_selection import train_test_split

import tensorflow as tf
import keras
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `sho
  and should_run_async(code)
```

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=(x_train_new.shape[1],)),
    keras.layers.Dense(256 , activation = "relu"),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(128 , activation = "relu"),
```

```
    keras.layers.Dropout(0.25),


    keras.layers.Dense(1 , activation = "sigmoid"),
])
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `shc
  and should_run_async(code)
```

```
model.compile(optimizer="adam", loss='binary_crossentropy', metrics=['accuracy'])
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `shc
  and should_run_async(code)
```

```
earlystopping = EarlyStopping(monitor='val_loss',
                                        mode='min',
                                        verbose=1,
                                        patience=20)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `shc
  and should_run_async(code)
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 7)                 0

 dense_3 (Dense)             (None, 256)               2048

 dropout (Dropout)           (None, 256)               0

 dense_4 (Dense)             (None, 128)               32896

 dropout_1 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 1)                 129

=================================================================
Total params: 35073 (137.00 KB)
Trainable params: 35073 (137.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `shc
  and should_run_async(code)
```

```
history = model.fit(x_train_new, y_train, validation_data=(x_test_new, y_test), epochs=300, batc
```

```
Epoch 4/300
11/11 [==============================] - 1s 61ms/step - loss: 3.0441 - accuracy: 0.5312 -
```

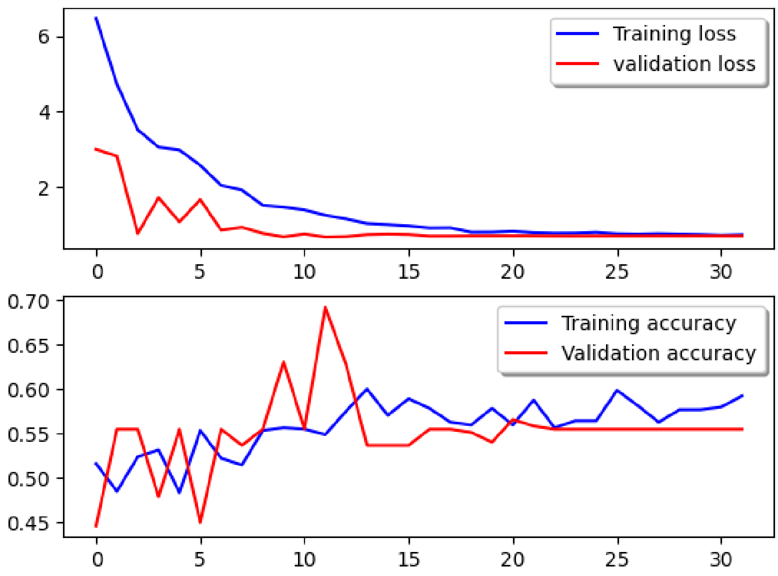
```
Epoch 9/300
11/11 [==============================] - 0s 13ms/step - loss: 1.4980 - accuracy: 0.5530 -
Epoch 10/300
11/11 [==============================] - 0s 12ms/step - loss: 1.4515 - accuracy: 0.5561 -
Epoch 11/300
11/11 [==============================] - 0s 18ms/step - loss: 1.3763 - accuracy: 0.5545 -
Epoch 12/300
11/11 [==============================] - 0s 18ms/step - loss: 1.2350 - accuracy: 0.5483 -
Epoch 13/300
11/11 [==============================] - 0s 18ms/step - loss: 1.1372 - accuracy: 0.5748 -
Epoch 14/300
11/11 [==============================] - 0s 14ms/step - loss: 1.0164 - accuracy: 0.5997 -
Epoch 15/300
11/11 [==============================] - 0s 14ms/step - loss: 0.9871 - accuracy: 0.5701 -
Epoch 16/300
11/11 [==============================] - 0s 13ms/step - loss: 0.9537 - accuracy: 0.5888 -
Epoch 17/300
11/11 [==============================] - 0s 8ms/step - loss: 0.8982 - accuracy: 0.5779 -
Epoch 18/300
11/11 [==============================] - 0s 9ms/step - loss: 0.9025 - accuracy: 0.5623 -
Epoch 19/300
11/11 [==============================] - 0s 9ms/step - loss: 0.7869 - accuracy: 0.5592 -
Epoch 20/300
11/11 [==============================] - 0s 9ms/step - loss: 0.7887 - accuracy: 0.5779 -
Epoch 21/300
11/11 [==============================] - 0s 8ms/step - loss: 0.8117 - accuracy: 0.5592 -
Epoch 22/300
11/11 [==============================] - 0s 7ms/step - loss: 0.7748 - accuracy: 0.5872 -
Epoch 23/300
11/11 [==============================] - 0s 9ms/step - loss: 0.7587 - accuracy: 0.5561 -
Epoch 24/300
11/11 [==============================] - 0s 10ms/step - loss: 0.7620 - accuracy: 0.5639 -
Epoch 25/300
11/11 [==============================] - 0s 8ms/step - loss: 0.7831 - accuracy: 0.5639 -
Epoch 26/300
11/11 [==============================] - 0s 9ms/step - loss: 0.7421 - accuracy: 0.5981 -
Epoch 27/300
11/11 [==============================] - 0s 8ms/step - loss: 0.7330 - accuracy: 0.5810 -
Epoch 28/300
11/11 [==============================] - 0s 9ms/step - loss: 0.7448 - accuracy: 0.5623 -
Epoch 29/300
11/11 [==============================] - 0s 7ms/step - loss: 0.7332 - accuracy: 0.5763 -
Epoch 30/300
11/11 [==============================] - 0s 8ms/step - loss: 0.7237 - accuracy: 0.5763 -
Epoch 31/300
11/11 [==============================] - 0s 8ms/step - loss: 0.7035 - accuracy: 0.5794 -
Epoch 32/300
11/11 [==============================] - 0s 9ms/step - loss: 0.7185 - accuracy: 0.5919 -
```

```
model.evaluate(x_test_new  ,y_test)
```

```
9/9 [==============================] - 0s 4ms/step - loss: 0.6869 - accuracy: 0.5543
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `sho
  and should_run_async(code)
[0.6868999600410461, 0.554347813129425]
```

```
fig, ax = plt.subplots(2,1)
ax[0].plot(history.history['loss'], color='b', label="Training loss")
ax[0].plot(history.history['val_loss'], color='r', label="validation loss",axes =ax[0])
legend = ax[0].legend(loc='best', shadow=True)

ax[1].plot(history.history['accuracy'], color='b', label="Training accuracy")
ax[1].plot(history.history['val_accuracy'], color='r',label="Validation accuracy")
legend = ax[1].legend(loc='best', shadow=True)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel
  and should_run_async(code)
```




```
Y_pred = model.predict(x_test_new)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `shc
  and should_run_async(code)
9/9 [==============================] - 0s 5ms/step
```

```
y_pred_binary = (Y_pred > 0.5).astype(int)
y_pred_binary.reshape(276)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `shc
  and should_run_async(code)
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

```
ann_cm = confusion_matrix(y_test, y_pred_binary)
ann_acc = round(accuracy_score(y_pred_binary,y_test) * 100, 2)
print(ann_cm)
print(ann_acc,'%')
```

```
[[  0 123]
 [  0 153]]
55.43 %
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `shc
  and should_run_async(code)
```