ASSIGNMENT 1 & 2

1.Reverse a Given Number

Take the value of the integer and store in a variable, using a while loop, get each digit of the number and store the reversed number in another variable and print the reverse of the number.

```
a=int(input('Enter value: '))
s=0
m=a
while a>0:
  s=int(s*10+(a%10))
  a=int(a/10)
print("Reverse of",m,"is",s)
```

```
Enter value: 523
Reverse of 523 is 325
```

2.Print largest permutation number of a given number

```
def largest_permutation(number):
    digits=[]
    n=number
    while n>0:
        digit=n%10
        digits=[digit]+digits
        n//= 10
    for i in range(len(digits)):
        for j in range(len(digits)-1):
            if digits[j]<digits[j+1]:
                digits[j],digits[j+1]=digits[j+1],digits[j]
    largest_number = 0
    for digit in digits:
        largest_number=largest_number* 10+digit
    return largest_number
number=int(input('Enter value: '))
print("Largest permutation of", number, "is:", largest_permutation(number))
```

```
Enter value: 324
Largest permutation of 324 is: 432
```

3.Find the number of ones in the binary representation of a number

```
a=int(input('Enter value: '))
s=0
c=0
while a>0:
  r=int(a%2)
  if r==1:
    c=c+1
  a=int(a/2)
print(c)
```

```
Enter value: 3
2
```

4.Write a program to print following patterns

a)

```
rows=int(input('Enter range: '))
for i in range(1,rows+1):
  for _ in range(rows-i):
    print(" ", end="")
  for _ in range(i):
    print("*", end="")
  print()
```

```
Enter range: 5
    *
   **
  ***
 ****
*****
```

4. b)

```
rows=int(input('Enter range: '))
for i in range(1,rows+1):
  for _ in range(rows-i):
    print(" ", end="")
  for _ in range(i):
    print("* ", end="")
  print()
```

```
Enter range: 5
    *
   * *
  * * *
 * * * *
* * * * *
```

4. c)

```
rows=int(input('Enter range: '))
for i in range(1,rows+1):
  for _ in range(i):
    print(chr(64+i), end="")
  print()
```

```
Enter range: 5
A
BB
CCC
DDDD
EEEEE
```

5.Check if two numbers are amicable numbers

```
def d_sum(n):
    divisors_sum = 0
    for i in range(1,n):
        if n%i==0:
            divisors_sum+=i
    return divisors_sum
def amicable(num1, num2):
    sum1=d_sum(num1)
    sum2=d_sum(num2)
    return sum1==num2 and sum2==num1
a=int(input("Enter the first number: "))
b=int(input("Enter the second number: "))
if amicable(a, b):
    print("The numbers", a, "and", b, "are amicable.")
else:
    print("The numbers", a, "and", b, "are not amicable.")
```

```
Enter the first number: 220
Enter the second number: 284
The numbers 220 and 284 are amicable.
```

6.Find the cumulative sum of a list where the i-th element is the sum of the first i+1 elements from the original list.

```
a=input("Enter the list elements separated by spaces: ")
a=[int(x) for x in a.split()]
ans=[]
total=0
for i in a:
  total+=i
  ans.append(total)
print("Original List:", a)
print("Cumulative Sum List:", ans)
```

```
Enter the list elements separated by spaces: 1 2 3 4 5
Original List: [1, 2, 3, 4, 5]
Cumulative Sum List: [1, 3, 6, 10, 15]
```

+ Code + Text

7.Given a list of sorted numbers and a variable K, where K is also a number, write a Python program using binary search to find the number in the list which is closest to the given number K

```
n = input("Enter the sorted list of numbers separated by spaces: ")
sorted_list = [int(x) for x in n.split()]
k = int(input("Enter the number K: "))
low = 0
high = len(sorted_list) - 1
ans = None
```

```
while low <= high:
    mid = (low + high) // 2
    if sorted_list[mid] == k:
        ans = sorted_list[mid]
        break
    elif sorted_list[mid] < k:
        low = mid + 1
    else:
        high = mid - 1
    if ans is None or abs(sorted_list[mid] - k) < abs(ans - k):
        ans = sorted_list[mid]
print("Number in the list closest to", k, "is:", ans)
```

```
Enter the sorted list of numbers separated by spaces: 2 5 7 8 12
Enter the number K: 9
Number in the list closest to 9 is: 8
```

8.Given a list of tuples, write a Python program to remove all the duplicated tuples from the given list using the concept of set.

```
t_list=[(1, 2), (3, 4), (1, 2), (5, 6), (3, 4)]
t_set=set(t_list)
t_list1=list(t_set)
print("Original list of tuples:",t_list)
print("List of unique tuples:", t_list1)
```

```
Original list of tuples: [(1, 2), (3, 4), (1, 2), (5, 6), (3, 4)]
List of unique tuples: [(1, 2), (3, 4), (5, 6)]
```

9.Given an unsorted list of some elements (may or may not be integers), Find the frequency of each distinct element in the list using a dictionary.

```
def frequency(a):
    f_dict = {}
    for i in a:
        if i in f_dict:
            f_dict[i]+=1
        else:
            f_dict[i]=1
    return f_dict
unsorted_list = [1, 2, 1, 2, 1, 'a','b','a','a']
f=frequency(unsorted_list)
print("Frequency of each distinct element is:")
for i,freq in f.items():
    print(i, ":", freq)
```

```
Frequency of each distinct element is:
1 : 3
2 : 2
a : 3
b : 1
```

10.Given two words, check whether they are anagrams using dictionary.

```
def anagram(a,b):
  a=a.lower()
  b=b.lower()
  if len(a)!=len(b):
    return False
  f1={}
  f2={}
  for char in a:
        if char in f1:
            f1[char]+=1
        else:
            f1[char]=1
  for char in b:
        if char in f2:
            f2[char]+=1
        else:
            f2[char]=1
  return f1==f2
a=input("Enter the first word: ")
b=input("Enter the second word: ")
if anagram(a,b):
  print(f"{a} and {b} are anagrams.")
else:
    print(f"{a} and {b} are not anagrams.")
```

```
Enter the first word: moon
Enter the second word: mono
moon and mono are anagrams.
```

11.Find common elements in three sorted lists using sets.

```
common = lambda a, b, c: set(a) & set(b) & set(c)
list1 = [1, 2, 3, 4, 5, 8]
list2 = [2, 4, 6, 8, 10]
list3 = [3, 4, 7, 8, 9]
ans = common(list1, list2, list3)
print("Common elements in the three lists:", ans)
```

```
Common elements in the three lists: {8, 4}
```

12.Find Symmetric Pairs in dictionary using loop.

```
def symmetric_pair(d):
    ans=[]
    for key,value in d.items():
        if value in d and d[value]==key:
            ans.append((key, value))
    return ans
d={'a':'b', 'b':'a', 'c':'d', 'd':'e', 'e':'d'}
ans=symmetric_pair(d)
print("Symmetric pairs in the dictionary:",ans)
```

```
Symmetric pairs in the dictionary: [('a', 'b'), ('b', 'a'), ('d', 'e'), ('e', 'd')]
```

KNN CLASSIFICATION IMPLEMENTATION

CODE

```
import pandas as pd
dataset = pd.read_csv("/content/drive/MyDrive/JISNIT/Courses/ML/LectureNotes/data/iris.csv")
print(dataset.head())
X = dataset.iloc[:, 1:5]
y = dataset.iloc[:, 5]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
#from sklearn.preprocessing import StandardScaler
#scaler = StandardScaler()
#scaler.fit(X)
#X_train = scaler.transform(X_train)
#X_test = scaler.transform(X_test)
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
import pickle
output_model_file = 'Knnmodel.pkl'
with open(output_model_file, 'wb') as f:
    pickle.dump(classifier, f)
```

OUTPUT

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
[[ 6  0  0]
 [ 0 10  0]
 [ 0  2 12]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         6
Iris-versicolor       0.83      1.00      0.91        10
 Iris-virginica       1.00      0.86      0.92        14

       accuracy                           0.93        30
      macro avg       0.94      0.95      0.94        30
   weighted avg       0.94      0.93      0.93        30
```

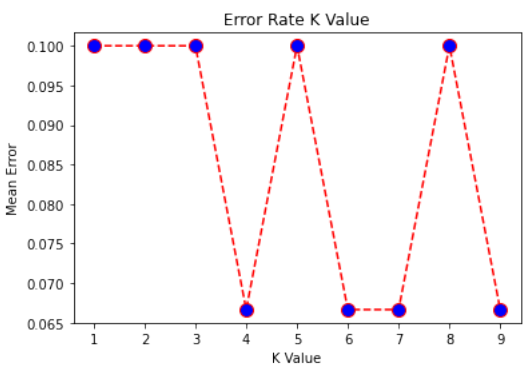
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

CODE

```
import numpy as np
error = []
for i in range(1, 10):
    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))
import matplotlib.pyplot as plt
plt.plot(range(1, 10), 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')
plt.show()
```

OUTPUT



KNN CLASSIFICATION IMPLEMENTATION

CODE

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv("/content/drive/MyDrive/JISNIT/Courses/ML/LectureNotes/data/Titanic.csv")
x = data.drop('Survived', axis = 1)
y = data['Survived']
x.drop(['Name', 'Ticket', 'Cabin'], axis = 1, inplace = True)
print(data.shape)
print(data.isna().sum())
# Missing value Imputation
# data = data.dropna(axis = 0, how ='any')
# Checking for missing value
# print(data.isna().sum())
# print(data.shape)
# numeric value imputation with mean
x['Age'] = x['Age'].fillna(x['Age'].mean())
x['Embarked'] = x['Embarked'].fillna(x['Embarked'].mode()[0])
x = pd.get_dummies(x, columns = ['Sex', 'Embarked'], prefix = ['Sex', 'Embarked'], drop_first = True)
print(x.head())
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
from sklearn.preprocessing import StandardScaler
std_x = StandardScaler()
x_train = std_x.fit_transform(x_train)
x_test = std_x.transform(x_test)
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5)
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

OUTPUT

```
(891, 12)
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       0
dtype: int64
(183, 12)
   PassengerId  Pclass   Age  SibSp  Parch     Fare  Sex_male  Embarked_Q  \
0            1       3  22.0      1      0   7.2500         1           0
1            2       1  38.0      1      0  71.2833         0           0
2            3       3  26.0      0      0   7.9250         0           0
3            4       1  35.0      1      0  53.1000         0           0
4            5       3  35.0      0      0   8.0500         1           0

   Embarked_S
0           1
1           0
2           1
3           1
4           1
[[99 11]
 [19 50]]
              precision    recall  f1-score   support

           0       0.84      0.90      0.87       110
           1       0.82      0.72      0.77        69

    accuracy                           0.83       179
   macro avg       0.83      0.81      0.82       179
weighted avg       0.83      0.83      0.83       179
```

NAIVE BAYESIAN CLASSIFICATION IMPLEMENTATION

CODE

```
# Assigning features and label variables
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy']
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','Mild']

play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']

# Import LabelEncoder
from sklearn import preprocessing
#creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers.
weather_encoded=le.fit_transform(weather)
temp_encoded=le.fit_transform(temp)
target=le.fit_transform(play)

print(weather_encoded)
print(temp_encoded)
print(target)

import numpy as np
zipped=zip(weather_encoded,temp_encoded)
features = np.array(list(zipped)).tolist()
print(features)

#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import CategoricalNB

#Create a Gaussian Classifier
model = CategoricalNB()

# Train the model using the training sets
model.fit(features,target)

#Predict Output
predicted= model.predict([[0, 2]]) # 0:Overcast, 2:Mild
print("Predicted Value:", predicted)
```

OUTPUT

```
[2 2 0 1 1 1 0 2 2 1 2 0 0 1]
[1 1 1 2 0 0 0 2 0 2 2 2 1 2]
[0 0 1 1 1 0 1 0 1 1 1 1 1 0]
[[2, 1], [2, 1], [0, 1], [1, 2], [1, 0], [1, 0], [0, 0], [2, 2], [2, 0], [1, 2], [2, 2], [0, 2], [0, 1], [1, 2]]
Predicted Value: [1]
```

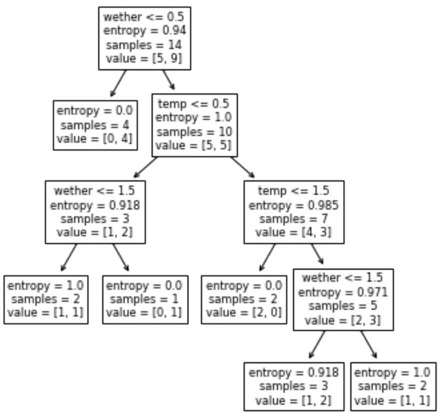
DECISION TREE CLASSIFICATION IMPLEMENTATION

CODE

```
# Assigning features and label variables
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy']
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','Mild']

play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']

# Import LabelEncoder
from sklearn import preprocessing
#creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers.
weather_encoded=le.fit_transform(weather)
temp_encoded=le.fit_transform(temp)
target=le.fit_transform(play)

print(weather_encoded)
print(temp_encoded)
print(target)

import numpy as np
zipped=zip(weather_encoded,temp_encoded)
features = np.array(list(zipped)).tolist()
print(features)

from sklearn import tree
#Create a Gaussian Classifier
model = tree.DecisionTreeClassifier(criterion='entropy')

# Train the model using the training sets
model.fit(features,target)

#Predict Output
predicted= model.predict([[0, 2]]) # 0:Overcast, 2:Mild
print("Predicted Value:", predicted)

from matplotlib import pyplot as plt
fig, ax = plt.subplots(figsize=(6, 6)) #figsize value changes the size of plot
tree.plot_tree(model,ax=ax,feature_names=['wether','temp'])
plt.show()
```

OUTPUT

```
[2 2 0 1 1 1 0 2 2 1 2 0 0 1]
[1 1 1 2 0 0 0 2 0 2 2 2 1 2]
[0 0 1 1 1 0 1 0 1 1 1 1 1 0]
[[2, 1], [2, 1], [0, 1], [1, 2], [1, 0], [1, 0], [0, 0], [2, 2], [2, 0], [1, 2], [2,
Predicted Value: [1]
```



LINEAR REGRESSION

```
from google.colab import drive
drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

The aim is to build a model which predicts sales based on the money spent on different platforms such as TV, radio, and newspaper for marketing.

CODE

```
#Importing the libraries import pandas
as pd
import numpy as np
import matplotlib.pyplot as plt import seaborn
as sns
```

```
#Reading the dataset
dataset = pd.read_csv("advertising.csv")
```

```
dataset.head()
```

OUTPUT

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9

☐ Data Pre-Processing

```
dataset.shape
```

(200, 4)

1. Checking for missing values

```
dataset.isna().sum()
```

OUTPUT

```
TV           0
Radio        0
Newspaper    0
Sales        0
dtype: int64
```

Conclusion: The dataset does not have missing values

2. Checking for duplicate rows

```
dataset.duplicated().any()
```

False

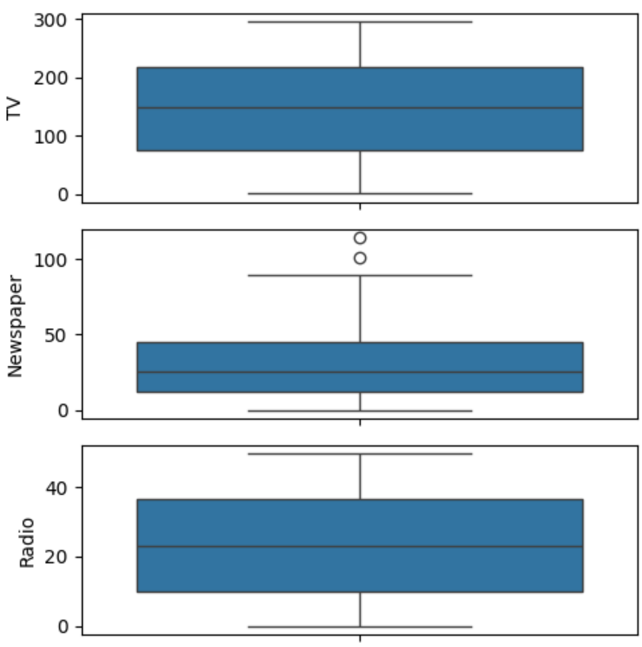
Conclusion: There are no duplicate rows present in the dataset

3. Checking for outliers

CODE

```
fig, axs = plt.subplots(3, figsize = (5,5))
plt1 = sns.boxplot(dataset['TV'], ax = axs[0])
plt2 = sns.boxplot(dataset['Newspaper'], ax = axs[1]) plt3 =
sns.boxplot(dataset['Radio'], ax = axs[2])
plt.tight_layout()
```

OUTPUT



Conclusion: There are not that extreme values present in the dataset

Exploratory Data Analysis

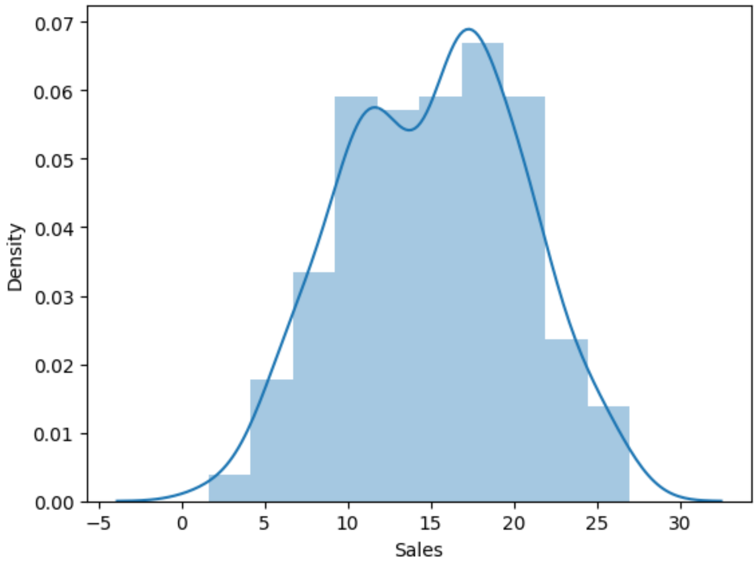
1. Distribution of the target variable

CODE

```
sns.distplot(dataset['Sales']);
```

OUTPUT

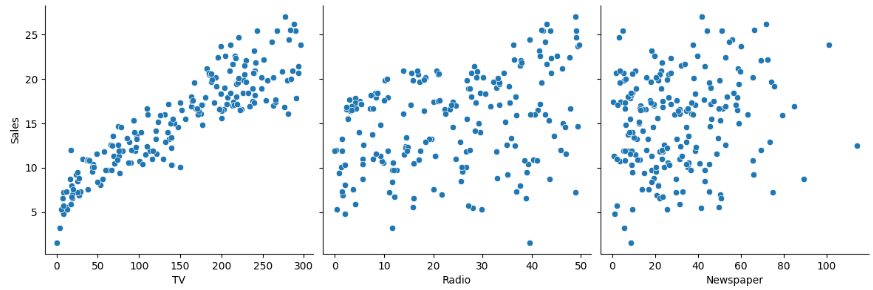
```
<ipython-input-11-e26ae89dfd77>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an
axes-level function for histograms).

For a guide to updating your code to use the new functions, please see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataset['Sales']);
```



Conclusion: It is normally distributed

2. How Sales are related with other variables

```
sns.pairplot(dataset, x_vars=['TV', 'Radio', 'Newspaper'], y_vars='Sales', height=4, aspect=1, kind='scatter') plt.show()
```



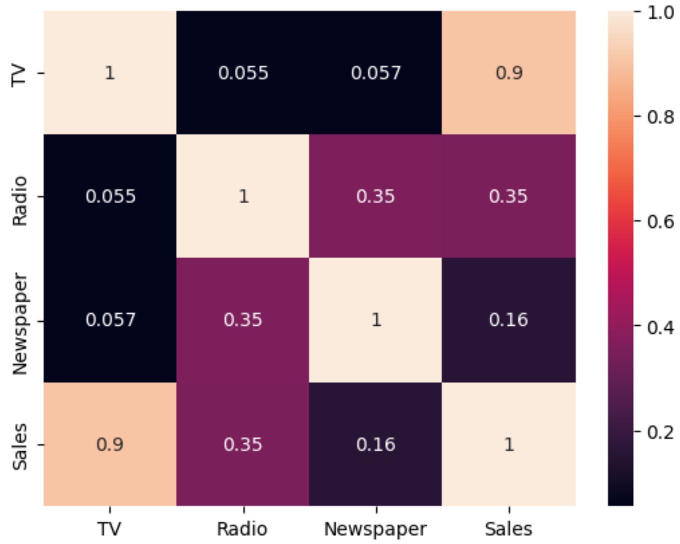
Conclusion: TV is strongly, positively, linearly correlated with the target variable. Bu the Newspaper feature seems to be uncorrelated

3. Heatmap

CODE

```
sns.heatmap(dataset.corr(), annot = True) plt.show()
```

OUTPUT



Conclusion: TV seems to be most correlated with Sales as 0.9 is very close to 1

1. Simple Linear Regression

Simple linear regression has only one x and one y variable. It is an approach for predicting a quantitative response using a single feature.

It establishes the relationship between two variables using a straight line. Linear regression attempts to draw a line that comes closest to the data by finding the slope and intercept that define the line and minimize regression errors.

Formula: Y = β0 + β1X + e

```
Y = Dependent variable / Target variable β0 = Intercept
of the regression line
β1 = Slope of the regression lime which tells whether the line is increasing or decreasing
X = Independent variable / Predictor variable e = Error
```

Equation: Sales = β0 + β1X + TV

CODE

```
from sklearn.model_selection import train_test_split from
sklearn.linear_model import LinearRegression
from sklearn import metrics
```

```
#Setting the value for X and Y x =
dataset[['TV']]
y = dataset['Sales']
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 100)
```

```
slr= LinearRegression()
slr.fit(x_train.values, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
#Printing the model coefficients
print('Intercept: ', slr.intercept_)
print('Coefficient:', slr.coef_)
```

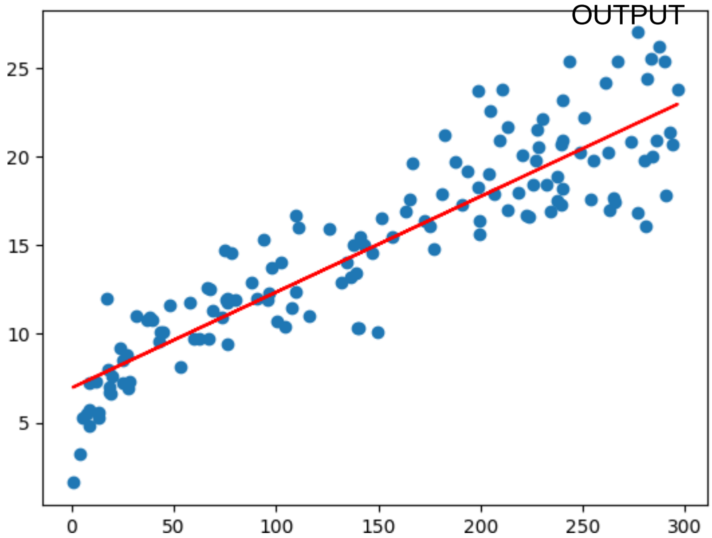
```
Intercept:   6.948683200001357
 Coefficient: [0.05454575]
```

```
print('Regression Equation: Sales = 6.948 + 0.054 * TV')
```

```
Regression Equation: Sales = 6.948 + 0.054 * TV
```

```
#Line of best fit
plt.scatter(x_train, y_train)
plt.plot(x_train, 6.948 + 0.054*x_train, 'r') plt.show()
```

OUTPUT



CODE

```
#Prediction of Test and Training set result y_pred_slr=
slr.predict(x_test.values)
x_pred_slr= slr.predict(x_train.values)
```

```
print("Prediction for test set: {}".format(y_pred_slr))
```

OUTPUT

```
Prediction for test set: [ 7.37414007 19.94148154 14.32326899 18.82329361 20.13239168 18.2287449
 14.54145201 17.72692398 18.75238413 18.77420243 13.34144544 19.46693349
 10.01415451 17.1923756  11.70507285 12.08689312 15.11418241 16.23237035
 15.8669138  13.1068987  18.65965635 14.00690363 17.60692332 16.60328147
 17.03419291 18.96511257 18.93783969 11.05597839 17.03419291 13.66326538
 10.6796127  10.71234015 13.5487193  17.22510305  9.67597085 13.52144643
 12.25053038 16.13418799 19.07965865 17.48692266 18.69783838 16.53237199
 15.92145955 18.86693021 13.5050827  11.84143724  7.87050642 20.51966653
 10.79961336  9.03233096 17.99419817 16.29237067 11.04506924 14.09963141
 18.44147334  9.3759692   7.88687015  8.34505447 17.72692398 11.62325422]
```

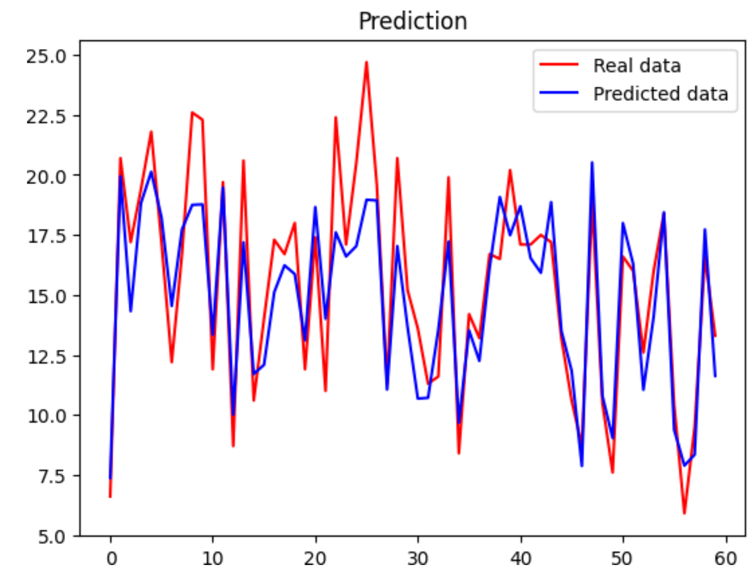
CODE

```
#Actual value and the predicted value
slr_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y_pred_slr}) slr_diff
```

Show hidden output

```
plt.plot(y_test.values, color = 'red', label = 'Real data')
plt.plot(y_pred_slr, color = 'blue', label = 'Predicted data') plt.title('Prediction')
plt.legend() plt.show()
```

OUTPUT




```
#Predict for any value slr.predict([[56]])
```

```
array([10.00324536])
```

Conclusion: The model predicted the Sales of 10.003 in that market

```
# print the R-squared value for the model from
sklearn.metrics import accuracy_score
print('R squared value of the model: {:.2f}'.format(slr.score(x,y)*100))
```

```
R squared value of the model: 81.10
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but LinearRegression was fitted with warnings.warn(
```

Conclusion: 81.10% of the data fit the regression model

CODE

```
# 0 means the model is perfect. Therefore the value should be as close to 0 as possible meanAbErr =
metrics.mean_absolute_error(y_test, y_pred_slr)
meanSqErr = metrics.mean_squared_error(y_test, y_pred_slr)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, y_pred_slr))

print('Mean Absolute Error:', meanAbErr) print('Mean
Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

OUTPUT

```
Mean Absolute Error: 1.6480589869746525 Mean
Square Error: 4.077556371826948
Root Mean Square Error: 2.019296008966231
```

2. Multiple Linear Regression

Multiple linear regression has one y and two or more x variables. It is an extension of Simple Linear regression as it takes more than one predictor variable to predict the response variable.

Multiple Linear Regression is one of the important regression algorithms which models the linear relationship between a single dependent continuous variable and more than one independent variable.

Assumptions for Multiple Linear Regression: 1. A linear relationship should exist between the Target and predictor variables. 2. The regression residuals must be normally distributed. 3. MLR assumes little or no multicollinearity (correlation between the independent variable) in data.

Formula: $Y = \beta0 + \beta1X1 + \beta2X2 + \beta3X3 + ... + \beta nXn + e$

Y = Dependent variable / Target variable β0 = Intercept

of the regression line

β1, β2,..βn = Slope of the regression lime which tells whether the line is increasing or decreasing X1, X2,..Xn = Independent variables /

Predictor variables

e = Error

Equation: Sales = β0 + (β1 * TV) + (β2 * Radio) + (β3 * Newspaper)

CODE

```
#Setting the value for X and Y
x = dataset[['TV', 'Radio', 'Newspaper']] y =
dataset['Sales']
```

```
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.3, random_state=100)
```

```
mlr= LinearRegression()
mlr.fit(x_train.values, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
#Printing the model coefficients
print(mlr.intercept_)
# pair the feature names with the coefficients list(zip(x, mlr.coef_))
```

```
4.334595861728431
[('TV', 0.053829108667250075),
 ('Radio', 0.11001224388558054),
 ('Newspaper', 0.0062899501461303325)]
```

```
#Predicting the Test and Train set result y_pred_mlr=
mlr.predict(x_test.values)
x_pred_mlr= mlr.predict(x_train.values)
```

```
print("Prediction for test set: {}".format(y_pred_mlr))
```

OUTPUT

```
Prediction for test set: [ 9.35221067 20.96344625 16.48851064 20.10971005 21.67148354 16.16054424
 13.5618056  15.39338129 20.81980757 21.00537077 12.29451311 20.70848608
  8.17367308 16.82471534 10.48954832  9.99530649 16.34698901 14.5758119
 17.23065133 12.56890735 18.55715915 12.12402775 20.43312609 17.78017811
 16.73623408 21.60387629 20.13532087 10.82559967 19.12782848 14.84537816
 13.13597397  9.07757918 12.07834143 16.62824427  8.41792841 14.0456697
  9.92050209 14.26101605 16.76262961 17.17185467 18.88797595 15.50165469
 15.78688377 16.86266686 13.03405813 10.47673934 10.6141644  20.85264977
 10.1517568   6.88471443 17.88702583 18.16013938 12.55907083 16.28189561
 18.98024679 11.33714913  5.91026916 10.06159509 17.62383031 13.19628335]
```

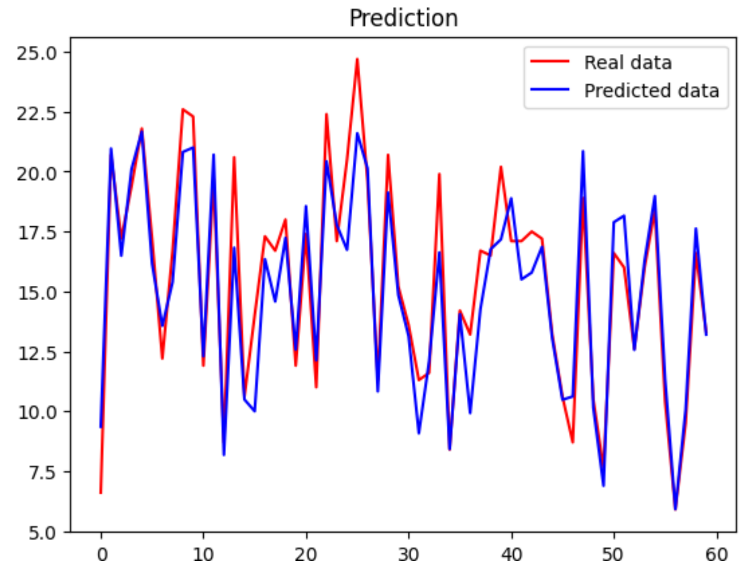
CODE

```
#Actual value and the predicted value
mlr_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y_pred_mlr}) mlr_diff
```

Show hidden output

OUTPUT

```
plt.plot(y_test.values, color = 'red', label = 'Real data')
plt.plot(y_pred_mlr, color = 'blue', label = 'Predicted data') plt.title('Prediction')
plt.legend() plt.show()
```



CODE

```
#Predict for any value
mlr.predict([[56, 55, 67]])
```

OUTPUT

```
array([13.82112602])
```

Conclusion: The model predicted the Sales of 13.82 in that market

```
# print the R-squared value for the model
print('R squared value of the model: {:.2f}'.format(mlr.score(x,y)*100))
```

```
R squared value of the model: 90.11
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but LinearRegression was fitted with warnings.warn(
```

Conclusion: 90.21% of the data fit the multiple regression model

CODE

```
# 0 means the model is perfect. Therefore the value should be as close to 0 as possible meanAbErr =
metrics.mean_absolute_error(y_test, y_pred_mlr)
meanSqErr = metrics.mean_squared_error(y_test, y_pred_mlr)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, y_pred_mlr))

print('Mean Absolute Error:', meanAbErr) print('Mean
Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

OUTPUT

```
Mean Absolute Error: 1.227818356658941
Mean Square Error: 2.6360765623280655
Root Mean Square Error: 1.623599877533891
```

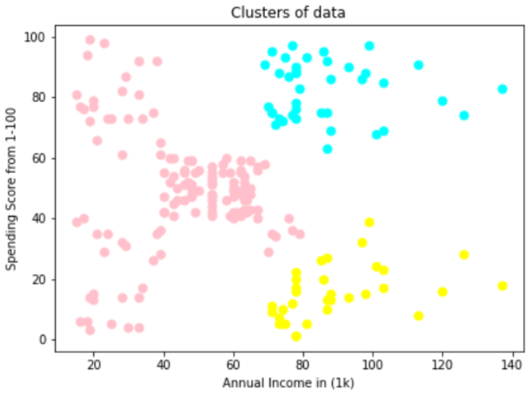
HIERARCHICAL AGGLOMERATIVE CLUSTERING

CODE

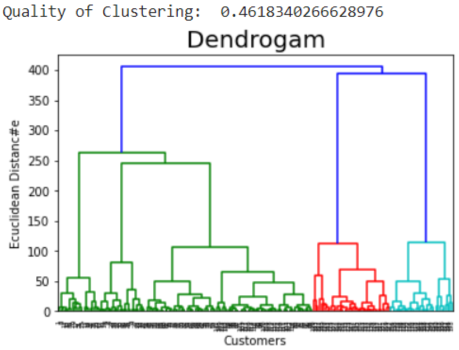
```
from sklearn.cluster import AgglomerativeClustering
import numpy
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import silhouette_score
import scipy.cluster.hierarchy as sch
data = pd.read_csv("/content/drive/MyDrive/JISNIT/Courses/ML/LectureNotes/data/Mall_Customers.csv")
#f1 = data['Age'].values
f2 = data['Annual Income (k$)'].values
f3 = data['Spending Score (1-100)'].values
X = numpy.array(list(zip(f2, f3)))
hc = AgglomerativeClustering(n_clusters = 3, affinity = 'euclidean', linkage = 'ward')
cluster_labels = hc.fit_predict(X)
print(cluster_labels)
n_clusters_ = len(set(cluster_labels)) - (1 if -1 in cluster_labels else 0)
n_noise_ = list(cluster_labels).count(-1)
print("Estimated number of clusters: " , n_clusters_)
print("Estimated number of noise points: " , n_noise_)
plt.figure(figsize=(7,5))
plt.scatter(X[cluster_labels == 0, 0], X[cluster_labels == 0, 1], s = 50, c = 'pink')
plt.scatter(X[cluster_labels == 1, 0], X[cluster_labels == 1, 1], s = 50, c = 'yellow')
plt.scatter(X[cluster_labels == 2, 0], X[cluster_labels == 2, 1], s = 50, c = 'cyan')
plt.scatter(X[cluster_labels == 3, 0], X[cluster_labels == 3, 1], s = 50, c = 'magenta')
plt.scatter(X[cluster_labels == 4, 0], X[cluster_labels == 4, 1], s = 50, c = 'orange')
plt.scatter(X[cluster_labels == 5, 0], X[cluster_labels == 5, 1], s = 50, c = 'blue')
plt.scatter(X[cluster_labels == 6, 0], X[cluster_labels == 6, 1], s = 50, c = 'red')
plt.scatter(X[cluster_labels == 7, 0], X[cluster_labels == 7, 1], s = 50, c = 'black')
plt.scatter(X[cluster_labels == 8, 0], X[cluster_labels == 8, 1], s = 50, c = 'green')
plt.xlabel('Annual Income in (1k)')
plt.ylabel('Spending Score from 1-100')
plt.title('Clusters of data')
plt.show()
if(n_clusters_ > 1):
  sil = silhouette_score(X, cluster_labels, metric='euclidean', sample_size = len(data))
  print("Quality of Clustering: ", sil)
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogam', fontsize = 20)
plt.xlabel('Customers')
plt.ylabel('Ecuclidean Distanc#e')
plt.show()
```

OUTPUT

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 0 2 1 2 1 2 0 2 1 2 1 2 1 2 1 2 0 2 1 2 0 2
 1 2 1 2 1 2 1 2 1 2 1 2 0 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1
 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2]
Estimated number of clusters:  3
Estimated number of noise points:  0
```




```
Quality of Clustering:  0.4618340266628976
```



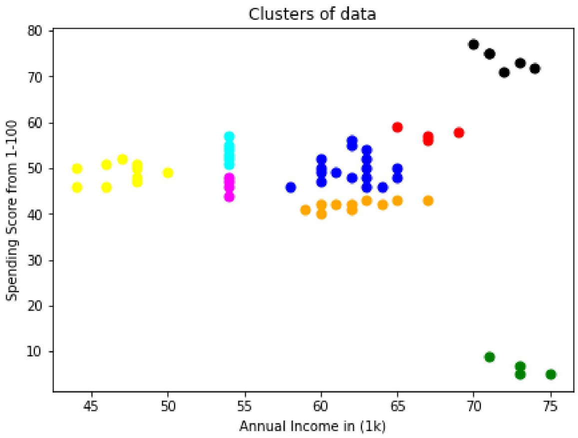
DBSCAN

CODE

```
import numpy
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import silhouette_score
from sklearn.cluster import DBSCAN

def dbscan(D, eps, MinPts):
    labels = [0]*len(D)
    C = 0
    for P in range(0, len(D)):
        if not (labels[P] == 0):
           continue
        NeighborPts = region_query(D, P, eps)
        else:
           C += 1
           grow_cluster(D, labels, P, NeighborPts, C, eps, MinPts)
    return labels

def grow_cluster(D, labels, P, NeighborPts, C, eps, MinPts):
    labels[P] = C
    i = 0
    while i < len(NeighborPts):
        Pn = NeighborPts[i]
        if labels[Pn] == -1:
           labels[Pn] = C
        elif labels[Pn] == 0:
            labels[Pn] = C
            PnNeighborPts = region_query(D, Pn, eps)
            if len(PnNeighborPts) >= MinPts:
                NeighborPts = NeighborPts + PnNeighborPts
        i += 1

def region_query(D, P, eps):
    neighbors = []
    for Pn in range(0, len(D)):
        if numpy.linalg.norm(D[P] - D[Pn]) < eps:
           neighbors.append(Pn)
    return neighbors

data = pd.read_csv("/content/drive/MyDrive/JISNIT/Courses/ML/LectureNotes/data/Mall_Customers.csv")
#f1 = data['Age'].values
f2 = data['Annual Income (k$)'].values
f3 = data['Spending Score (1-100)'].values
X = numpy.array(list(zip(f2, f3)))
cluster_labels = numpy.array(dbscan(X, 3, 4))
#db = DBSCAN(eps = 3, min_samples = 4).fit(X)
#cluster_labels = db.labels_
print(cluster_labels)

n_clusters_ = len(set(cluster_labels)) - (1 if -1 in cluster_labels else 0)
n_noise_ = list(cluster_labels).count(-1)

print("Estimated number of clusters: " , n_clusters_)
print("Estimated number of noise points: " , n_noise_)

plt.figure(figsize=(7,5))
plt.scatter(X[cluster_labels == 0, 0], X[cluster_labels == 0, 1], s = 50, c = 'pink')
plt.scatter(X[cluster_labels == 1, 0], X[cluster_labels == 1, 1], s = 50, c = 'yellow')
plt.scatter(X[cluster_labels == 2, 0], X[cluster_labels == 2, 1], s = 50, c = 'cyan')
plt.scatter(X[cluster_labels == 3, 0], X[cluster_labels == 3, 1], s = 50, c = 'magenta')
plt.scatter(X[cluster_labels == 4, 0], X[cluster_labels == 4, 1], s = 50, c = 'orange')
plt.scatter(X[cluster_labels == 5, 0], X[cluster_labels == 5, 1], s = 50, c = 'blue')
plt.scatter(X[cluster_labels == 6, 0], X[cluster_labels == 6, 1], s = 50, c = 'red')
plt.scatter(X[cluster_labels == 7, 0], X[cluster_labels == 7, 1], s = 50, c = 'black')
plt.scatter(X[cluster_labels == 8, 0], X[cluster_labels == 8, 1], s = 50, c = 'green')
plt.xlabel('Annual Income in (1k)')
plt.ylabel('Spending Score from 1-100')
plt.title('Clusters of data')
plt.show()
if(n_clusters_ > 1):
  sil = silhouette_score(X, cluster_labels,
                         metric='euclidean',
                         sample_size = len(data))
  print("Quality of Clustering: ", sil)
```

OUTPUT

```
[-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1 -1 -1  1  1  1  1 -1 -1  1 -1  1 -1  1  1 -1  1 -1 -1
  1 -1  3  2  2  3  2 -1  2  2 -1  3  2  3 -1 -1 -1  5 -1  4  5  4  4  5
  5  5  4  5  4  5 -1  5  5  4  5  5  4  5  5  5  4  5  5  5  4  6  4  6
  6 -1  6 -1 -1  7 -1 -1 -1  7  8  7 -1  7  8 -1  8  7 -1  7  8 -1 -1  9
 -1 -1 -1 -1 -1  9 -1  9 -1 -1 -1  9 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1 -1 -1]
Estimated number of clusters:  9
Estimated number of noise points:  135
```



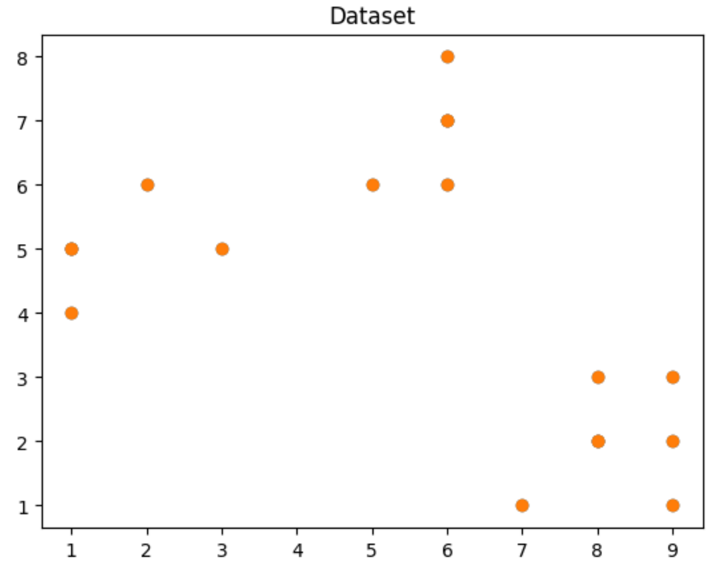

```
Quality of Clustering:  -0.22656703338648562
```

K-MEANS CLUSTERING

CODE

```
from sklearn.cluster import KMeans
from sklearn import metrics
from scipy.spatial.distance import cdist
import numpy as np
import matplotlib.pyplot as plt
x1 = np.array([3, 1, 1, 2, 1, 6, 6, 6, 5, 6, 7, 8, 9, 8, 9, 9, 8])
x2 = np.array([5, 4, 5, 6, 5, 8, 6, 7, 6, 7, 1, 2, 1, 2, 3, 2, 3])
X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)
plt.scatter(X[:,0],X[:,1], label='True Position')
plt.title('Dataset')
plt.scatter(x1, x2)
plt.show()
```

OUTPUT

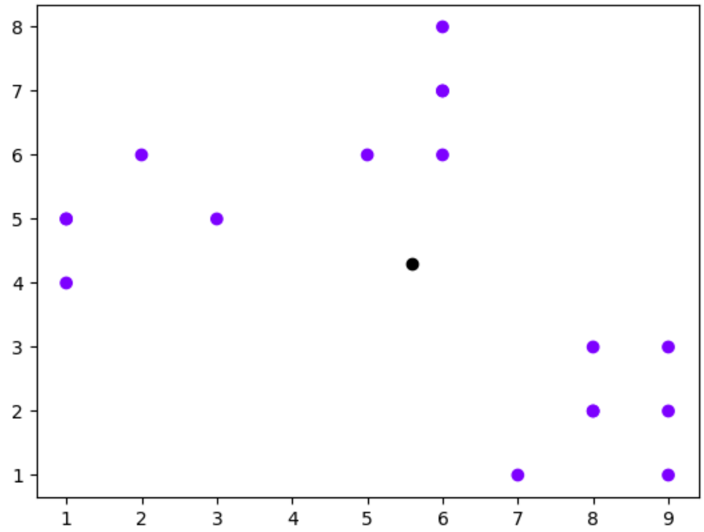


CODE

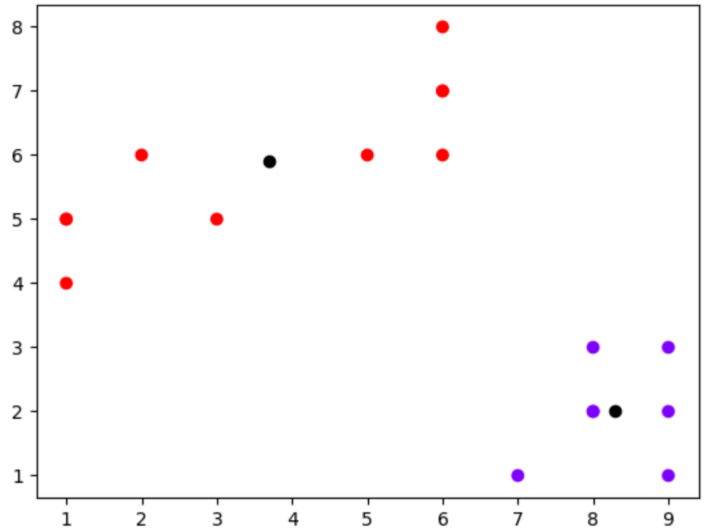
```
import matplotlib.pyplot as plt
distortions = []
mapping1 = {}
for k in range(1, 10):
  kmeanModel = KMeans(n_clusters=k)
  kmeanModel.fit(X)
  sse = sum(np.min(cdist(X, kmeanModel.cluster_centers_,'euclidean'), axis=1)) / X.shape[0]
  print("Value of K = ", k, ", SSE = ", sse)
  distortions.append(sum(np.min(cdist(X, kmeanModel.cluster_centers_,'euclidean'), axis=1)) / X.shape[0])
  mapping1[k] = sum(np.min(cdist(X, kmeanModel.cluster_centers_,'euclidean'), axis=1)) / X.shape[0]
  plt.scatter(X[:,0],X[:,1], c=kmeanModel.labels_, cmap='rainbow')
  plt.scatter(kmeanModel.cluster_centers_[:,0] ,kmeanModel.cluster_centers_[:,1], color='black')
  plt.show()
for key, val in mapping1.items():
    print(f'{key} : {val}')
plt.plot(range(1, 10), distortions, 'bx-')
plt.xlabel('Values of K')
plt.ylabel('Distortion')
plt.title('The Elbow Method using Distortion')
plt.show()
```

OUTPUT

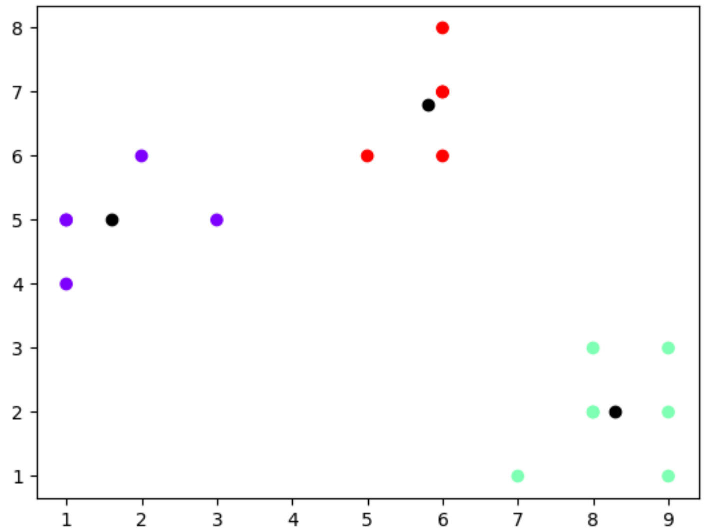
```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
Value of K =  1 , SSE =  3.4577032384495707
```



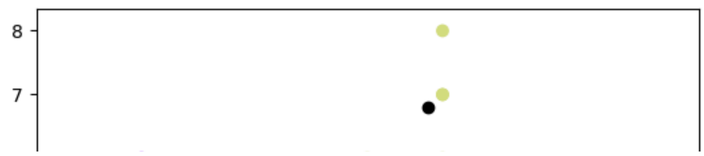
```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
Value of K =  2 , SSE =  1.7687413573405673
```

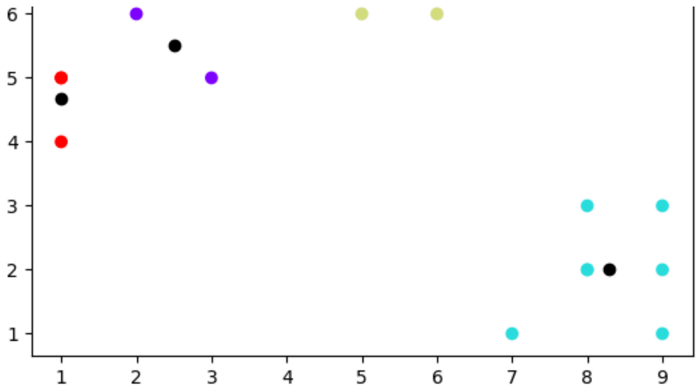


```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
Value of K =  3 , SSE =  0.8819889697423957
```

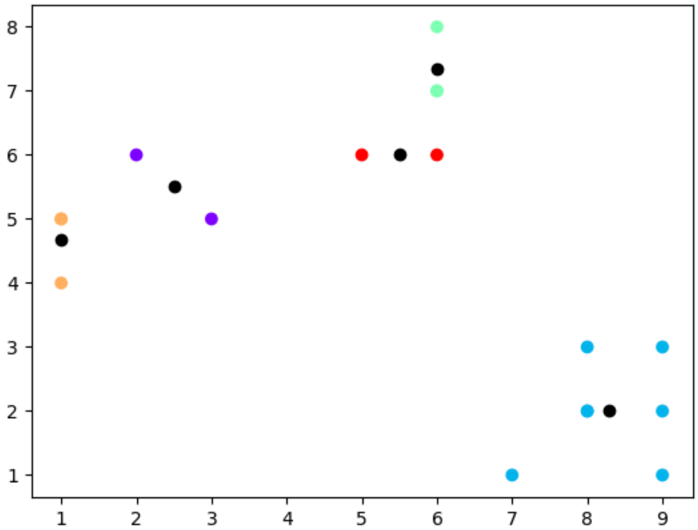


```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
Value of K =  4 , SSE =  0.7587138847606585
```

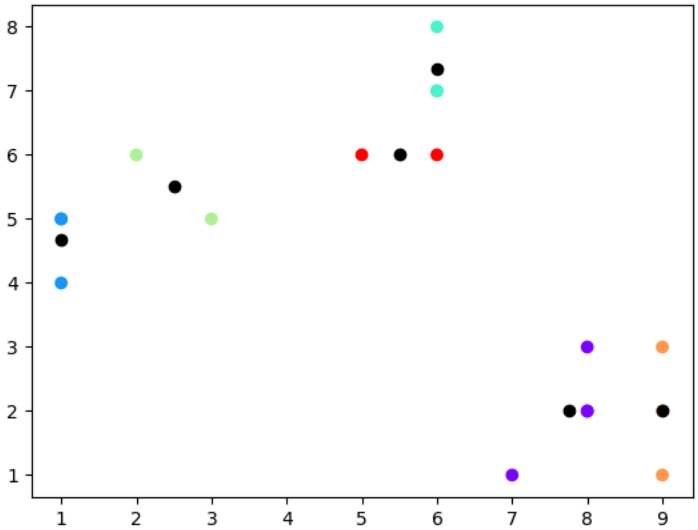




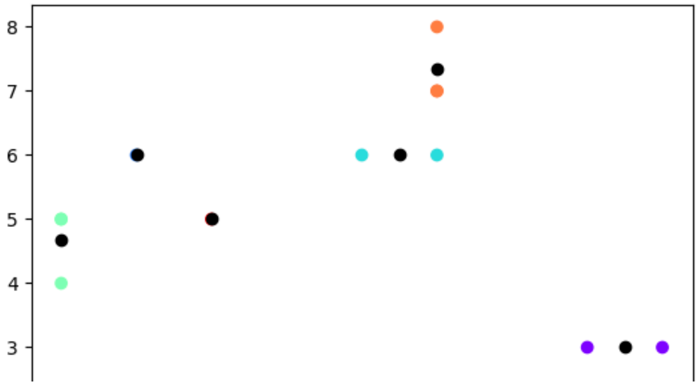
```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
Value of K =  5 , SSE =  0.6760729098960964
```

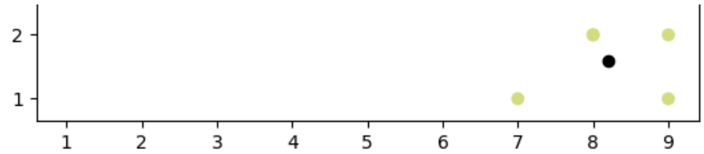


```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
Value of K =  6 , SSE =  0.580097449143775
```

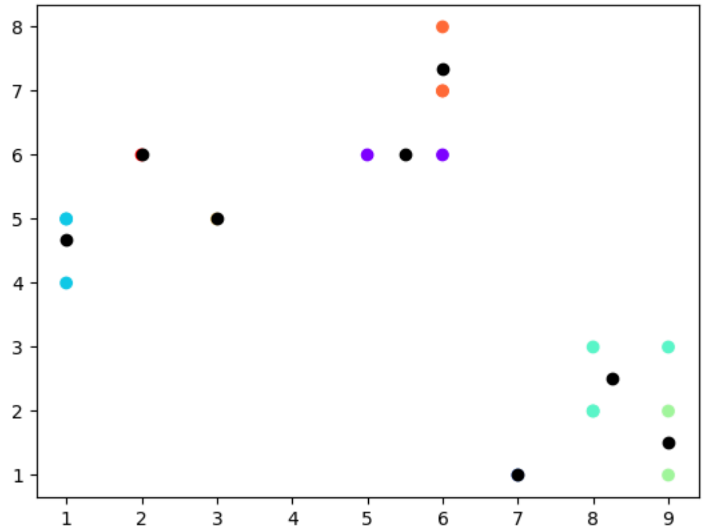


```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
Value of K =  7 , SSE =  0.517480107950963
```

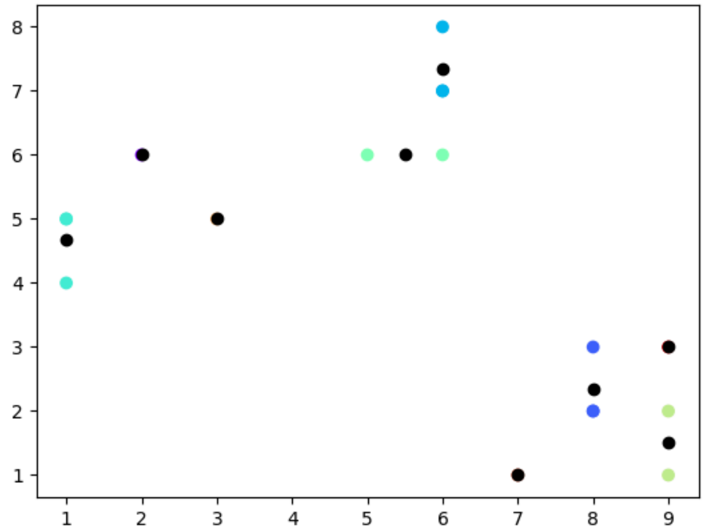




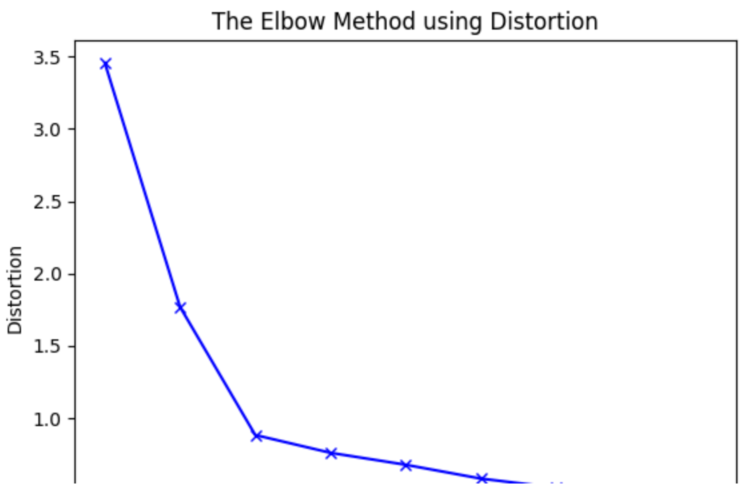

```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
Value of K =  8 , SSE =  0.42618267462691206
```

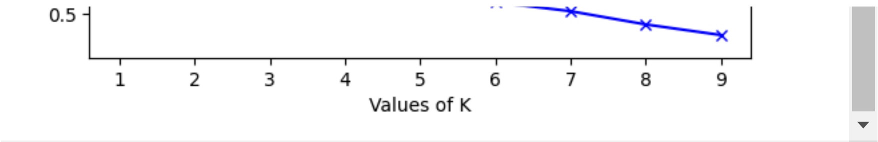



```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarni
  warnings.warn(
Value of K =  9 , SSE =  0.35294117647058826
```




```
1 : 3.4577032384495707
2 : 1.7687413573405673
3 : 0.8819889697423957
4 : 0.7587138847606585
5 : 0.6760729098960964
6 : 0.580097449143775
7 : 0.517480107950963
8 : 0.42618267462691206
9 : 0.35294117647058826
```

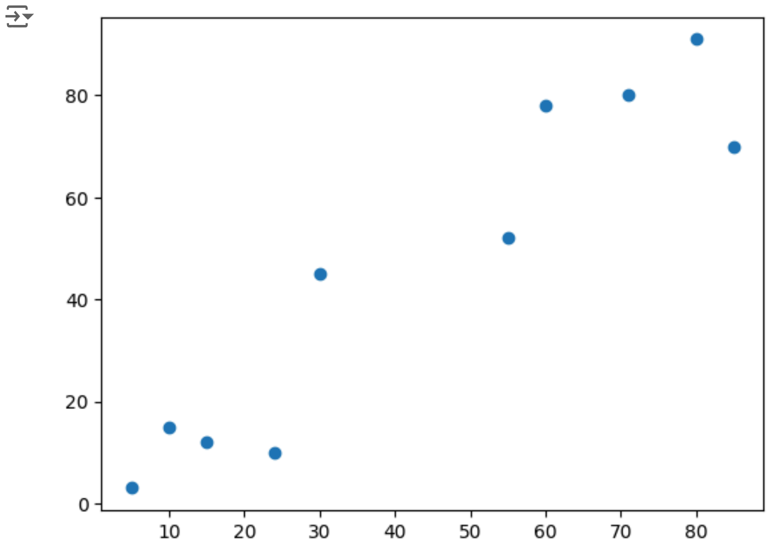




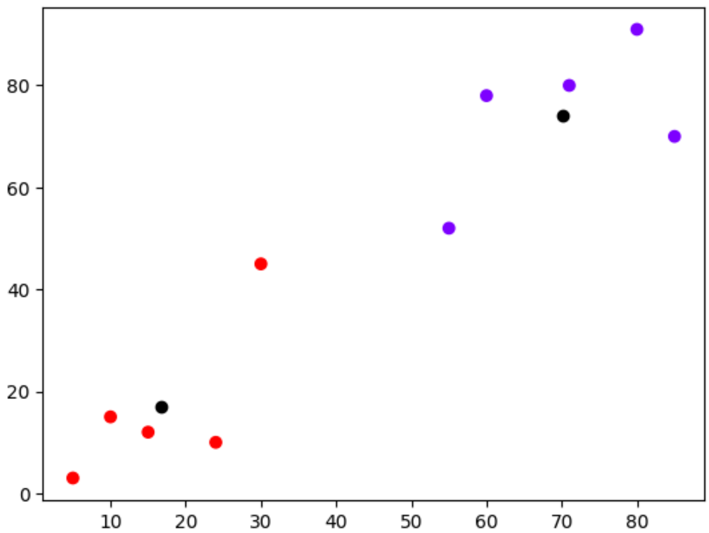
CODE

```
import matplotlib.pyplot as plt
import numpy as np
X = np.array([[5,3], [10,15], [15,12], [24,10], [30,45], [85,70], [71,80], [60,78], [55,52], [80,91],])
plt.scatter(X[:,0],X[:,1], label='True Position')
plt.show()
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
print(kmeans.cluster_centers_)
print(kmeans.labels_)
cluster_labels = kmeans.predict(X)
C = kmeans.cluster_centers_
sil = silhouette_score(X, cluster_labels, metric='euclidean',sample_size = len(X))
print(C)
print("Quality of Clustering: ", sil)
plt.scatter(X[:,0],X[:,1], c=kmeans.labels_, cmap='rainbow')
plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1], color='black')
plt.show()
```

OUTPUT




```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:
  warnings.warn(
[[70.2 74.2]
 [16.8 17. ]]
[1 1 1 1 1 0 0 0 0 0]
[[70.2 74.2]
 [16.8 17. ]]
Quality of Clustering:  0.6586004781412067
```



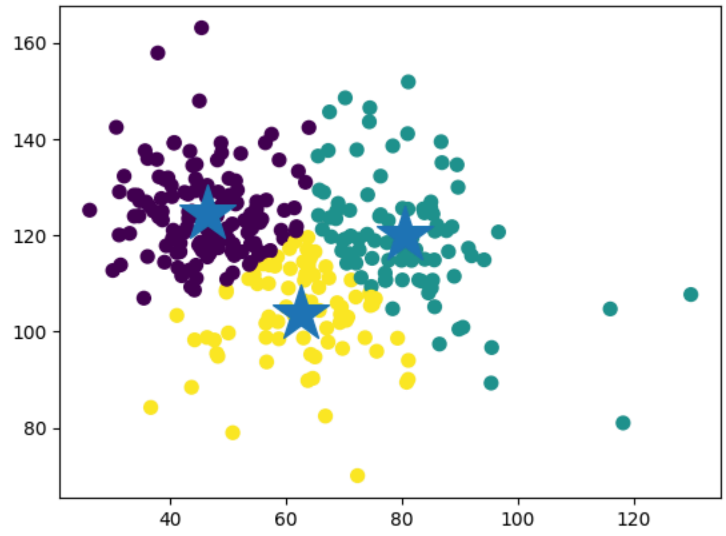
CODE

```
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
data = pd.read_csv("/content/drive/MyDrive/JISNIT/Courses/ML/LectureNotes/data/spinem.csv")
X = data[['pelvic_incidence', 'pelvic_radius', 'thoracic_slope']]
kmeans = KMeans(n_clusters = 3, random_state = 123)
model = kmeans.fit(X)
cluster_labels = kmeans.predict(X)
X['Cluster'] = cluster_labels
print(X)
C = kmeans.cluster_centers_
sil = silhouette_score(X, cluster_labels, metric='euclidean',sample_size = len(data))
print(C)
print("Quality of Clustering: ", sil)
fig = plt.figure()
plt.scatter(X['pelvic_incidence'], X['pelvic_radius'], c=cluster_labels,
            s=50, cmap='viridis');
plt.scatter(C[:, 0], C[:, 1], marker='*', s=1000)
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(X['pelvic_incidence'], X['pelvic_radius'],
           X['thoracic_slope'],
           c=cluster_labels,
          cmap='viridis');
ax.scatter(C[:, 0], C[:, 1],  C[:, 2],
           marker='*',
           c='#050505')
```

OUTPUT

```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:
  warnings.warn(
     pelvic_incidence  pelvic_radius  thoracic_slope  Cluster
0           63.027817      98.672917         14.5386        2
1           39.056951     114.405425         17.5323        0
2           68.832021     105.985135         17.4861        2
3           69.297008     101.868495         12.7074        2
4           49.712859     108.168725         15.9546        2
..                ...            ...             ...      ...
305         47.903565     117.449062         14.7484        0
306         53.936748     114.365845         18.1972        0
307         61.446597     125.670725         13.5565        0
308         45.252792     118.545842         16.0928        0
309         33.841641     123.945244         17.6963        0

[310 rows x 4 columns]
[[ 46.42903837 124.47018491  13.26250567]
 [ 80.49567418 120.00557969  12.78133516]
 [ 62.59438009 103.64870812  13.03697051]]
Quality of Clustering:  0.3539586349354204
<ipython-input-1-cecf937cbdde>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  X['Cluster'] = cluster_labels
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7febd0c27790>
```




```
<Figure size 640x480 with 0 Axes>
```

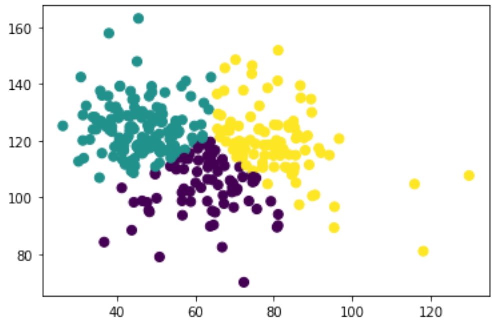
The following code block implements k-means algorithm from the scratch

CODE

```
from sklearn.metrics import pairwise_distances_argmin
def find_clusters_kmeans(X, n_clusters, rseed=2):
    centers = [X[0], X[1], X[2]]
    print(centers)
    while True:
        labels = pairwise_distances_argmin(X, centers)
        new_centers = np.array([X[labels == i].mean(0)
                                for i in range(n_clusters)])
        if np.all(centers == new_centers):
            break
        centers = new_centers
    return centers, labels
X = np.array(X)
centers, cluster_labels = find_clusters_kmeans(X, 3)
plt.scatter(X[:, 0], X[:, 1], c=cluster_labels,
            s=50, cmap='viridis');
sil = silhouette_score(X, cluster_labels, metric='euclidean',sample_size = len(data))
print(centers)
print("Quality of Clustering: ", sil)
```

OUTPUT

```
[array([63.0278175 , 98.67291675, 14.5386    ,  2.        ]), array([ 39.05695098, 11
[[ 62.59438009 103.64870812  13.03697051   2.        ]
 [ 46.42903837 124.47018491  13.26250567   0.        ]
 [ 80.49567418 120.00557969  12.78133516   1.        ]]
Quality of Clustering:  0.3539586349354204
```



APRIORI ALGORITHM (MARKET BASKET ANALYSIS)

CODE

```
dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
           ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
           ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]
```

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)
```

OUTPUT

	Apple	Corn	Dill	Eggs	Ice cream	Kidney Beans	Milk	Nutmeg	Onion	Unicorn	Yogurt
0	False	False	False	True	False	True	True	True	True	False	True
1	False	False	True	True	False	True	False	True	True	False	True
2	True	False	False	True	False	True	True	False	False	False	False
3	False	True	False	False	False	True	True	False	False	True	True
4	False	True	False	True	True	True	False	False	True	False	False

CODE

```
from mlxtend.frequent_patterns import apriori
frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x))
frequent_itemsets
```

OUTPUT

	support	itemsets	length
0	0.8	(Eggs)	1
1	1.0	(Kidney Beans)	1
2	0.6	(Milk)	1
3	0.6	(Onion)	1
4	0.6	(Yogurt)	1
5	0.8	(Eggs, Kidney Beans)	2
6	0.6	(Onion, Eggs)	2
7	0.6	(Milk, Kidney Beans)	2
8	0.6	(Onion, Kidney Beans)	2
9	0.6	(Yogurt, Kidney Beans)	2
10	0.6	(Onion, Eggs, Kidney Beans)	3

CODE

```
frequent_itemsets[(frequent_itemsets['length'] > 2) &
                  (frequent_itemsets['support'] >= 0.6)]
```

	support	itemsets	length
10	0.6	(Onion, Eggs, Kidney Beans)	3

CODE

```
from mlxtend.frequent_patterns import association_rules

association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)
```

OUTPUT

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Eggs)	(Kidney Beans)	0.8	1.0	0.8	1.00	1.00	0.00	inf
1	(Kidney Beans)	(Eggs)	1.0	0.8	0.8	0.80	1.00	0.00	1.0
2	(Onion)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
3	(Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.6
4	(Milk)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
5	(Onion)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
6	(Yogurt)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
7	(Onion, Eggs)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
8	(Onion, Kidney Beans)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
9	(Eggs, Kidney Beans)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.6
10	(Onion)	(Eggs, Kidney Beans)	0.6	0.8	0.6	1.00	1.25	0.12	inf

IMAGE CLASSIFICATION USING KNN, SVC, ANN

CODE

```
import cv2
import os
```

```
def extract_color_histogram(image):
  hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
  bins = [10]
  h, s, v = hsv[:, :, 0], hsv[:, :, 1], hsv[:, :, 2]
  hist = cv2.calcHist([h], [0], None, bins, [0, 180])
  cv2.normalize(hist, hist)
  return hist.flatten()
```

```
train_data_path = '/content/drive/MyDrive/JISNIT/Courses/ML/LectureNotes/data/ImageData/FruitData/Training'
test_data_path = '/content/drive/MyDrive/JISNIT/Courses/ML/LectureNotes/data/ImageData/FruitData/Test'
data_dir_list = list(os.listdir(train_data_path))
print(data_dir_list)
features = []
classLabels = []
for dataset in data_dir_list:
    img_list = os.listdir(train_data_path+'/'+ dataset)
    print ('Loaded the images of dataset-'+'{}\n'.format(dataset))
    for img in img_list:
        image = cv2.imread(train_data_path + '/'+ dataset + '/'+ img )
        label = dataset
        hist = extract_color_histogram(image)
        features.append(hist)
        classLabels.append(label)
```

OUTPUT

```
['Apple', 'Banana']
Loaded the images of dataset-Apple

Loaded the images of dataset-Banana
```

CODE

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
trainFeat, testFeat, trainLabels, testLabels =  train_test_split(features, classLabels, test_size=0.20)
```

```
from sklearn.neighbors import KNeighborsClassifier
print("\n")
print("[INFO] evaluating k-NN...")
k = 9
model = KNeighborsClassifier(n_neighbors = k)
model.fit(trainFeat, trainLabels)
acc = model.score(testFeat, testLabels)
print("[INFO] k-NN classifier: k = ", k)
print("[INFO] accuracy: {:.2f}%".format(acc * 100))
predLabels = model.predict(testFeat)
print(confusion_matrix(testLabels, predLabels))
print(classification_report(testLabels, predLabels))
test_img_list = os.listdir(test_data_path)
for img in test_img_list:
    print(test_data_path + '/'+img)
    image = cv2.imread(test_data_path + '/'+ img )
    hist = extract_color_histogram(image)
    prediction = model.predict([hist])
    print("Predicted Class Label = ",prediction)
```

OUTPUT

```
[INFO] evaluating k-NN...
[INFO] k-NN classifier: k =  9
[INFO] accuracy: 100.00%
[[100   0]
 [  0  97]]
              precision    recall  f1-score   support

       Apple       1.00      1.00      1.00       100
      Banana       1.00      1.00      1.00        97

    accuracy                           1.00       197
   macro avg       1.00      1.00      1.00       197
weighted avg       1.00      1.00      1.00       197

/content/drive/MyDrive/JISNIT/Courses/ML/LectureNotes/data/ImageData/FruitData/Test/banana_test.jpg
Predicted Class Label =  ['Banana']
/content/drive/MyDrive/JISNIT/Courses/ML/LectureNotes/data/ImageData/FruitData/Test/apple_test.jpg
Predicted Class Label =  ['Apple']
```

CODE

```
from sklearn.svm import SVC
print("\n")
print("[INFO] evaluating SVC...")
model = SVC(max_iter=1000, class_weight='balanced')
model.fit(trainFeat, trainLabels)
acc = model.score(testFeat, testLabels)
print("[INFO] SVC classifier")
print("[INFO] accuracy: {:.2f}%".format(acc * 100))
predLabels = model.predict(testFeat)
print(confusion_matrix(testLabels, predLabels))
print(classification_report(testLabels, predLabels))
test_img_list = os.listdir(test_data_path)
for img in test_img_list:
    print(test_data_path + '/'+img)
    image = cv2.imread(test_data_path + '/'+ img )
    hist = extract_color_histogram(image)
    prediction = model.predict([hist])
    print("Predicted Class Label = ",prediction)
```

OUTPUT

```
[INFO] evaluating SVC...
[INFO] SVC classifier
[INFO] accuracy: 100.00%
[[100   0]
 [  0  97]]
              precision    recall  f1-score   support

       Apple       1.00      1.00      1.00       100
      Banana       1.00      1.00      1.00        97

    accuracy                           1.00       197
   macro avg       1.00      1.00      1.00       197
weighted avg       1.00      1.00      1.00       197

/content/drive/MyDrive/JISNIT/Courses/ML/LectureNotes/data/ImageData/FruitData/Test/banana_test.jpg
Predicted Class Label =  ['Banana']
/content/drive/MyDrive/JISNIT/Courses/ML/LectureNotes/data/ImageData/FruitData/Test/apple_test.jpg
Predicted Class Label =  ['Apple']
```

CODE

```
from sklearn.neural_network import MLPClassifier
#Neural Network
print("\n")
print("[INFO] evaluating ANN...")
model = MLPClassifier(hidden_layer_sizes=(10, 10), max_iter=1000, solver='sgd', learning_rate_init=.1)
model.fit(trainFeat, trainLabels)
acc = model.score(testFeat, testLabels)
print("[INFO] Neural Network accuracy: {:.2f}%".format(acc * 100))
predLabels = model.predict(testFeat)
print(confusion_matrix(testLabels, predLabels))
print(classification_report(testLabels, predLabels))
test_img_list = os.listdir(test_data_path)
for img in test_img_list:
    print(test_data_path + '/'+img)
    image = cv2.imread(test_data_path + '/'+ img )
    hist = extract_color_histogram(image)
    prediction = model.predict([hist])
    print("Predicted Class Label = ",prediction)
```

OUTPUT

```
[INFO] evaluating ANN...
[INFO] Neural Network accuracy: 100.00%
[[100   0]
 [  0  97]]
              precision    recall  f1-score   support

       Apple       1.00      1.00      1.00       100
      Banana       1.00      1.00      1.00        97

    accuracy                           1.00       197
   macro avg       1.00      1.00      1.00       197
weighted avg       1.00      1.00      1.00       197

/content/drive/MyDrive/JISNIT/Courses/ML/LectureNotes/data/ImageData/FruitData/Test/banana_test.jpg
Predicted Class Label =  ['Banana']
/content/drive/MyDrive/JISNIT/Courses/ML/LectureNotes/data/ImageData/FruitData/Test/apple_test.jpg
Predicted Class Label =  ['Apple']
```