B.Sc. H Computer Science

Semester VI

Data Mining

Lab Record

Name: shubham singh fartyal
Roll. No. – 19/78054

Table of Contents

-

Q1. Create a file "people.txt" with the following

Age	agegroup	height	status	Years married
21	adult	6.0	single	-1
2	child	3	married	0
18	adult	5.7	married	20
221	elderly	5	widowed	2
34	child	-7	married	3

i) **i) Read the data from the file "people.txt".**

ii) Create a ruleset E that contain rules to check for the following conditions:

1. The age should be in the range 0-150.
2. The age should be greater than yearsmarried.
3. The status should be married or single or widowed.
4. If age is less than 18 the agegroup should be child, if age is between 18 and 65 the agegroup should be adult, if age is more than 65 the agegroup should be elderly.

iii) Check whether ruleset E is violated by the data in the file people.txt.

iv) Summarize the results obtained in part (iii) v)

Visualize the results obtained in part (iii)

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

df = pd.read_csv("people.txt",delimiter=' ')
 df
```

	Age	agegroup	height	status	yearsmarried
0	21	adult	6.0	single	-1
1	2	child	3.0	married	0
2	18	adult	5.7	married	20
3	221	elderly	5.0	widowed	2
4	34	child	-7.0	married	3

-

ii) Create a ruleset E that contain rules to check for the following conditions:

1. **The age should be in the range 0-150.**
2. **The age should be greater than years married.**
3. **The status should be married or single or widowed.**
4. **If age is less than 18 the age group should be child, if age is between 18 and 65 the age group should be adult, if age is more than 65 the age group should be elderly.**

Ruleset.py:

```
def age_check(df):
    errors = df["Age"][(df["Age"]>150)|(df["Age"]<0)].shape[0]
return errors, "Checking if age is in range 0-150"
 def age_check2(df):     n = df.shape[0]
errors = 0     for i in range(n):         if
df["Age"][i]<df["yearsmarried"][i]:
            errors+=1     return errors, "Checking if age is
greater than years married"
 def status_check(df):     errors =
df.shape[0]df[df["status"].isin(['single','married','widowed'])].shape[0]
return errors, "Checking if status contains values only from single,
married, widowed"
 def agegroup_check(df):
    n = df.shape[0]     errors = 0     for i in range(n):
if df["Age"][i]<18 and df["agegroup"][i]!="child":
            errors+=1         elif df["Age"][i]>=18
and df["Age"][i]<65 and df["agegroup"][i]!="adult":
            errors+=1         elif df["Age"][i]>=65 and
df["agegroup"][i]!="elderly":
            errors+=1     return errors, "Checking if age group lies
correctly according to age"
```

-

```
rules = []
rules.append(ruleset.age_check)
rules.append(ruleset.age_check2)
rules.append(ruleset.status_check)
rules.append(ruleset.agegroup_check)
 violations = [] rule = list(map(lambda x: "rule"+str(x),
range(1,len(rules)+1))) for i in range(len(rules)):
    violation,desc = rules[i](df)
print(f"{rule[i]}: {desc}\nviolations:{violation}")
violations.append(violation)
```

Output

```
rule1: Checking if age is in range 0-150
violations:1
rule2: Checking if age is greater than years married
violations:1
rule3: Checking if status contains values only from single, married, widowed
violations:0
rule4: Checking if age group lies correctly according to age
violations:1
```

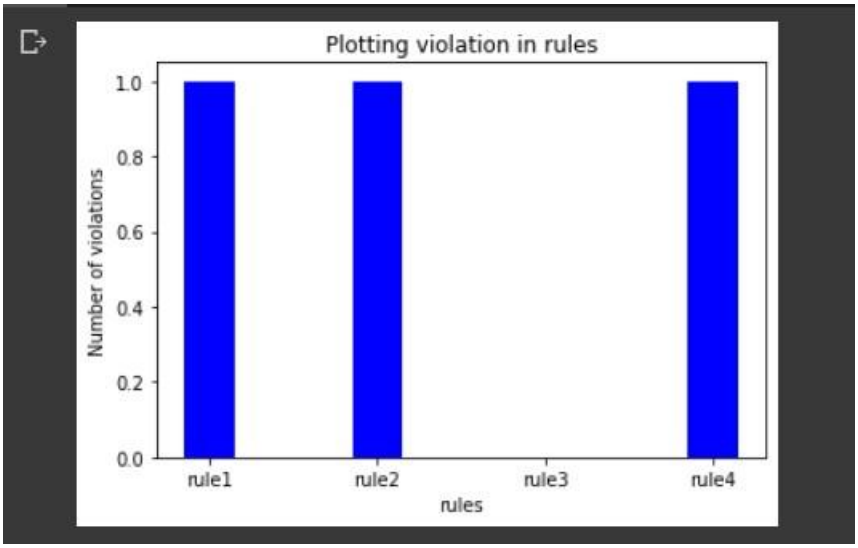
Q1.

v) Visualize the results obtained in part (iii)

-

```
fig = plt.figure() ax = fig.add_subplot()
ax.bar(rule,violations,0.3,color = 'blue')
plt.ylabel("Number of violations")
plt.xlabel("rules") plt.title("Plotting
violation in rules") plt.show()
```

Output:



Q2. Perform the following preprocessing tasks on the dirty_iris datasetii.

i) Calculate the number and percentage of observations that are complete.

ii) Replace all the special values in data with NA.

iii) Define these rules in a separate text file and read them.

(Use editfile function in R (package editrules). Use similar function in Python).

Print the resulting constraint object.

– Species should be one of the following values: setosa, versicolor or virginica.

-

- All measured numerical properties of an iris should be positive.
- The petal length of an iris is at least 2 times its petal width.
- The sepal length of an iris cannot exceed 30 cm.
- The sepals of an iris are longer than its petals.

iv)Determine how often each rule is broken (violatedEdits). Also summarize and plot the result.

v) Find outliers in sepal length using boxplot and boxplot.stats

```
import numpy as np import
pandas as pd import
matplotlib.pyplot as plt import
ruleset
  dataset =
pd.read_csv("dirty_iris.csv")

dataset.head(10)
```

Output

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	6.4	3.2	4.5	1.5	versicolor
1	6.3	3.3	6.0	2.5	virginica
2	6.2	NaN	5.4	2.3	virginica
3	5.0	3.4	1.6	0.4	setosa
4	5.7	2.6	3.5	1.0	versicolor
5	5.3	NaN	NaN	0.2	setosa
6	6.4	2.7	5.3	NaN	virginica
7	5.9	3.0	5.1	1.8	virginica
8	5.8	2.7	4.1	1.0	versicolor
9	4.8	3.1	1.6	0.2	setosa

i) Calculate the number and percentage of observations that are complete.

-

```
new_n = dataset.dropna().shape[0]
n = dataset.shape[0]
print(f"Number of complete records:{new_n}")
print("Percentage of complete records:{:.2f}%".format(float(new_n*100/n)))
```

```
Number of complete records:96
Percentage of complete records:64.00%
```

ii) Replace all the special values in data with NA.

```
[5] dataset.fillna("NA",inplace=True)
```

```
[6] dataset
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	6.4	3.2	4.5	1.5	versicolor
1	6.3	3.3	6.0	2.5	virginica
2	6.2	NA	5.4	2.3	virginica
3	5.0	3.4	1.6	0.4	setosa
4	5.7	2.6	3.5	1.0	versicolor
...	...	...	...	...	...
145	6.7	3.1	5.6	2.4	virginica
146	5.6	3.0	4.5	1.5	versicolor
147	5.2	3.5	1.5	0.2	setosa
148	6.4	3.1	NA	1.8	virginica
149	5.8	2.6	4.0	NA	versicolor

150 rows × 5 columns

-

iii) Define these rules in a separate text file and read them.

(Use editfile function in R (package editrules). Use similar function in Python).

Print the resulting constraint object.

- Species should be one of the following values: setosa, versicolor or virginica.
- All measured numerical properties of an iris should be positive.
- The petal length of an iris is at least 2 times its petal width.
- The sepal length of an iris cannot exceed 30 cm.
- The sepals of an iris are longer than its petals.

Ruleset:

-

```
def check_species(dataset):     n = dataset.shape[0]     species =
dataset["Species"][dataset["Species"]!='NA']     correct_n =
species[species.isin(["setosa","versicolor","virginica"])].shape[0]     return n-
correct_n, "Checking if all species consist of setosa, versicolor and virginica"
 def
check_positive(dataset):
    violation_n = 0     for i in
dataset.values:         for j in i:
if isinstance(j,float):
if j<0:
                    violation_n+=1
    return violation_n, "Checking if there all the length values are greater than
0"  def check_petal_length(dataset):
    n = dataset.shape[0]     correct_n = 0     for i in range(n):         if
dataset["Petal.Width"][i]=='NA' or dataset["Petal.Length"][i]=='NA':
            continue         elif
dataset["Petal.Width"][i]*2>dataset["Petal.Length"][i]:
            continue
else:
            correct_n+=1
    return n-correct_n, "Checking if petal length is at least twice of petal width"
```

--

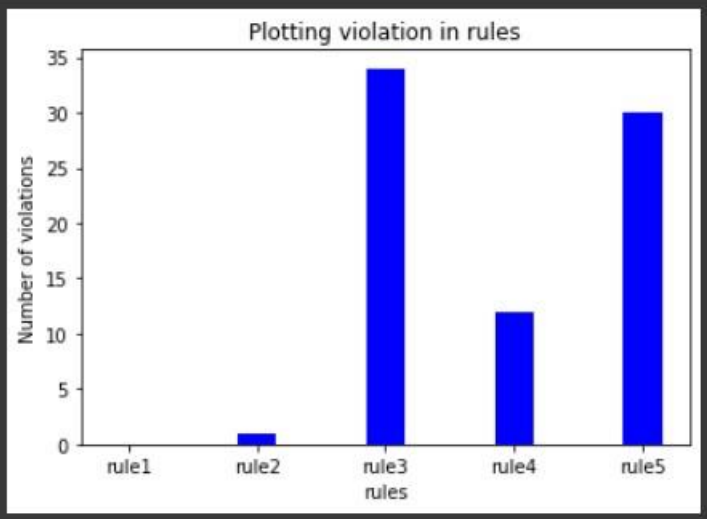
```
 def
check_sepal_length(dataset):
    n      =      dataset.shape[0]                          sl           =
dataset["Sepal.Length"][dataset["Sepal.Length"]!='NA']      correct_n =
sl[sl<=30].shape[0]        return  n-correct_n,  "Checking  if  all  sepal
lengths are below 30 cm"
 def
check_sepal_length2(dataset):
    n = dataset.shape[0]     correct_n = 0     for i in range(n):         if
dataset["Sepal.Length"][i]=='NA' or dataset["Petal.Length"][i]=='NA':
            continue         elif
dataset["Petal.Length"][i]>=dataset["Sepal.Length"][i]:
            continue
else:
            correct_n+=1     return n-correct_n, "Checking if sepal length is
more than petal length"
```

```
rules = []

[8] rules.append(ruleset.check_species)
    rules.append(ruleset.check_positive)
    rules.append(ruleset.check_petal_length)
    rules.append(ruleset.check_sepal_length)
    rules.append(ruleset.check_sepal_length2)

violations = []
rule = list(map(lambda x: "rule"+str(x), range(1,len(rules)+1)))
for i in range(len(rules)):
    violation,desc = rules[i](dataset)
    print(f"{rule[i]}: {desc}\nviolations:{violation}")
    violations.append(violation)

rule1: Checking if all species consist of setosa, versicolor and virginica
violations:0
rule2: Checking if there all the length values are greater than 0
violations:1
rule3: Checking if petal length is at least twice of petal width
violations:34
rule4: Checking if all sepal lengths are below 30 cm
violations:12
rule5: Checking if sepal length is more than petal length
violations:30
```

iv) Determine how often each rule is broken (violated Edits). Also summarize and plot the result.

--

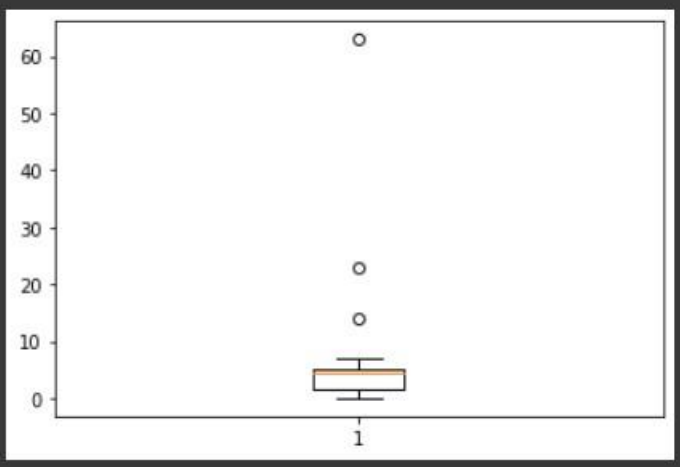
```
[ ]  fig = plt.figure()
     ax = fig.add_subplot()
     ax.bar(rule,violations,0.3,color = 'blue')
     plt.ylabel("Number of violations")
     plt.xlabel("rules")
     plt.title("Plotting violation in rules")
     plt.show()
```



vi) Find outliers in sepal length using boxplot

--

```
[ ]  fig = plt.figure()
     ax = fig.add_subplot()
     ax.boxplot(dataset["Petal.Length"][dataset["Petal.Length"]!='NA'])
     plt.show()
```



Q3. Load the data from wine dataset. Check whether all attributes are standardized or not (mean is 0 and standard deviation is 1). If not, standardize the attributes. Do the same with Iris dataset.

```
----------------------------------------------------------------------------------------------------------------------------------
-
```

```
import pandas as pd import
```

-

```
numpy as np
```

-

-

```
# In[2]: wd =
```

```
---------------------------------------------------------------------------------------------------------------------------
-
```

```
pd.read_csv('winequalityN.csv')
```

-

```
-----------------------------------------------------------------------------------------------------------------------------
-
```

```
# In[3]:
```

```
----------------------------------------------------------------------------------------------------------------------------
-
```

```
wd.head(9)
```

-

```
----------------------------------------------------------------------------------------------------------------------------
-
```

```
# In[4]:
```

```
-----------------------------------------------------------------------------------------------------------------------------
-
```

```
print(f'Number of empty records: {wd.isna().sum().sum()}')
```

```
-----------------------------------------------------------------------------------------------------------------------------------
-
```

```
# In[5]: wd.dropna(inplace=True)
```

-

-

```
# In[6]: wd_des =
```

```
-------------------------------------------------------------------------------------------------------------------------------
-
```

```
wd.describe()
```

-

```
----------------------------------------------------------------------------------------------------------------------------
-
```

```
# In[7]: for i in wd_des.columns:   if
```

```
-----------------------------------------------------------------------------------------------------------------------------------
-
```

```
wd_des[i]['mean']!=0 or wd_des[i]['std']!=1 :
```

```
----------------------------------------------------------------------------------------------------------------------------------
-
```

```
    print('standardization needed')
```

-

```
break
```

-

-

```
# In[8]:
```

```
----------------------------------------------------------------------------------------------------------------------------------
-
```

```
X = wd.iloc[:,:-1].values y
```

```
----------------------------------------------------------------------------------------------------------------------------
-
```

```
= wd.iloc[:,-1].values
```

-

```
-------------------------------------------------------------------------------------------------------------------------------
-
```

```
# In[9]:
```

```
----------------------------------------------------------------------------------------------------------------------------------
-
```

```
from sklearn.compose import ColumnTransformer from
```

-

```
sklearn.preprocessing import OneHotEncoder ct =
```

```
----------------------------------------------------------------------------------------------------------------------------------
-
```

```
ColumnTransformer(transformers =
```

```
----------------------------------------------------------------------------------------------------------------------------
-
```

```
[('encoder',OneHotEncoder(),[0])],remainder = 'passthrough')
```

```
-----------------------------------------------------------------------------------------------------------------------------
-
```

```
X = np.array(ct.fit_transform(X))
```

-

-

```
# In[10]: from sklearn.preprocessing import
```

-

```
StandardScaler ss = StandardScaler() X =
```

```
----------------------------------------------------------------------------------------------------------------------------
-
```

```
ss.fit_transform(X)
```

-

```
# In[11]:
X

# In[12]: ir =
pd.read_csv('iris.csv')

# In[13]:
ir.head(5)

# In[14]:
print(f'Number of empty records: {ir.isna().sum().sum()}')
# In[15]: ir_des =
ir.describe()

# In[16]: for i in ir_des.columns:   if
ir_des[i]['mean']!=0 or ir_des[i]['std']!=1 :
    print('standardization
needed')     break
# In[17]:
X2 = ir.iloc[:,:-1].values y2
= ir.iloc[:,-1].values

# In[18]: ss =
StandardScaler()
X2 = ss.fit_transform(X2)


# In[19]:
X2
```

-

Output:

```
wd.head(9)
```

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	white	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	white	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	white	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
5	white	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
6	white	6.2	0.32	0.16	7.0	0.045	30.0	136.0	0.9949	3.18	0.47	9.6	6
7	white	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
8	white	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6

```
print(f'Number of empty records: {wd.isna().sum().sum()}')
```

```
Number of empty records: 38
```

```
[5] wd.dropna(inplace=True)
```

```
[6] wd_des = wd.describe()
```

```
[7] for i in wd_des.columns:
      if wd_des[i]['mean']!=0 or wd_des[i]['std']!=1 :
        print('standardization needed')
        break
```

```
standardization needed
```

```
[8] X = wd.iloc[:,:-1].values
    y = wd.iloc[:,-1].values
```

```
[9] from sklearn.compose import ColumnTransformer
    from sklearn.preprocessing import OneHotEncoder
    ct = ColumnTransformer(transformers = [('encoder',OneHotEncoder(),[0])],remainder = 'passthrough')
    X = np.array(ct.fit_transform(X))
```

```
[10] from sklearn.preprocessing import StandardScaler
     ss = StandardScaler()
     X = ss.fit_transform(X)
```

-

```
[11] X

     array([[-0.5719307 ,  0.5719307 , -0.16778609, ..., -1.35916011,
             -0.5449872 , -1.41892232],
            [-0.5719307 ,  0.5719307 , -0.70715516, ...,  0.50839916,
             -0.27635393, -0.83218392],
            [-0.5719307 ,  0.5719307 ,  0.67979387, ...,  0.25939126,
             -0.61214551, -0.32926528],
            ...,
            [ 1.74846359, -1.74846359, -0.70715516, ...,  1.25542287,
              1.46976231,  0.42511267],
            [ 1.74846359, -1.74846359, -1.01536606, ...,  2.18920251,
              1.20112905, -0.24544551],
            [ 1.74846359, -1.74846359, -0.93831333, ...,  1.06866695,
              0.86533746,  0.42511267]])

[12] ir = pd.read_csv('iris.csv')

[13] ir.head(5)
```

	Sepal_length	Sepal_width	Petal_length	Petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
print(f'Number of empty records: {ir.isna().sum().sum()}')

Number of empty records: 0

[15] ir_des = ir.describe()

[16] for i in ir_des.columns:
       if ir_des[i]['mean']!=0 or ir_des[i]['std']!=1 :
         print('standardization needed')
         break

     standardization needed
```

-

Q4. Run Apriori algorithm to find frequent item sets and association rules

1.1 Use minimum support as 50% and minimum confidence as 75%

1.2 Use minimum support as 60% and minimum confidence as 60 %

-

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from mlxtend.frequent_patterns import apriori, association_rules
```

porting dataset

```
#importing the data
df=pd.read_csv("GroceryData.csv")

df.head()
```

	Customer ID	Bathing Soap	Beverages	Processed Food	Detergent	Utensils	Hair Colour	Skincare Products	Dish Soap
0	1001	0	0	1	1	0	0	1	0
1	1002	1	0	1	1	0	1	1	1
2	1003	1	0	1	0	0	1	1	1
3	1004	0	1	1	0	0	0	1	1
4	1005	0	0	1	0	0	0	1	0

```
df.drop(['Customer ID'],inplace=True,axis=1)
```

```
freq_items = apriori(df, min_support=0.5,use_colnames=True)
freq_items
```

	support	itemsets
0	0.571429	(Bathing Soap)
1	0.571429	(Processed Food)
2	0.714286	(Skincare Products)
3	0.523810	(Dish Soap)
4	0.523810	(Processed Food, Skincare Products)

```
association_rules(freq_items, metric="confidence", min_threshold=0.75)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Processed Food)	(Skincare Products)	0.571429	0.714286	0.52381	0.916667	1.283333	0.115646	3.428571

-

```
#df2
df2=pd.read_csv("grocery_data2.csv")

df2.head()
```

	id	whole milk	rolls/buns	soda	yogurt	root vegetables	tropical fruit	bottled water	sausage	citrus fruit
0	0	0	0	1	1	1	1	1	1	1
1	1	0	0	1	0	0	0	0	0	0
2	2	0	0	1	0	0	1	0	0	0
3	3	0	1	1	0	0	0	0	1	1
4	4	1	1	0	1	1	0	0	1	1

```
[ ] df2.drop(['id'],inplace=True,axis=1)
```

```
[ ] freq_items2 = apriori(df2, min_support=0.6,use_colnames=True)
    freq_items2
```

	support	itemsets
0	0.774	(rolls/buns)
1	0.730	(soda)
2	0.766	(sausage)
3	0.766	(citrus fruit)
4	0.677	(sausage, rolls/buns)
5	0.601	(rolls/buns, citrus fruit)

```
[ ] association_rules(freq_items2, metric="confidence", min_threshold=0.6)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(sausage)	(rolls/buns)	0.766	0.774	0.677	0.883812	1.141876	0.084116	1.945124
1	(rolls/buns)	(sausage)	0.774	0.766	0.677	0.874677	1.141876	0.084116	1.867175
2	(rolls/buns)	(citrus fruit)	0.774	0.766	0.601	0.776486	1.013689	0.008116	1.046913
3	(citrus fruit)	(rolls/buns)	0.766	0.774	0.601	0.784595	1.013689	0.008116	1.049188

-

Q5. Use Naive bayes, K-nearest, and Decision tree classification algorithms and build classifiers. Divide the data set into training and test set. Compare the accuracy of the different classifiers under the following situations:

5.1 a) Training set = 75% Test set = 25% b) Training set = 66.6% (2/3rd of total), Test set = 33.3%

5.2 Training set is chosen by i) hold out method ii) Random subsampling iii) CrossValidation. Compare the accuracy of the classifiers obtained.

5.3 Data is scaled to standard format.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import classification_report
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
import matplotlib.pyplot as plt
import math
import seaborn as sns
```

```
from google.colab import drive
drive.mount("/content/drive/",force_remount=True)
```

```
Mounted at /content/drive/
```

```
#importing the data
path="/content/drive/MyDrive/Iris.csv"
df=pd.read_csv(path)
df
```

	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

-

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Species'] = le.fit_transform(df['Species'])
df.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	0
1	2	4.9	3.0	1.4	0.2	0
2	3	4.7	3.2	1.3	0.2	0
3	4	4.6	3.1	1.5	0.2	0
4	5	5.0	3.6	1.4	0.2	0

```
x=df.values[:,:-1]
y=df.values[:,-1]
```

```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    sns.set(font_scale=1)
    plt.figure(figsize=(10,5))
    labels = [0,1,2]
    # representing A in heatmap format
    cmap1=sns.light_palette("#2ecc71")
    sns.heatmap(C, annot=True, cmap=cmap1, fmt=".0f", xticklabels=labels, yticklabels=labels,annot_kws={"size":14})
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")
    plt.show()
```

-

Hold Out Method

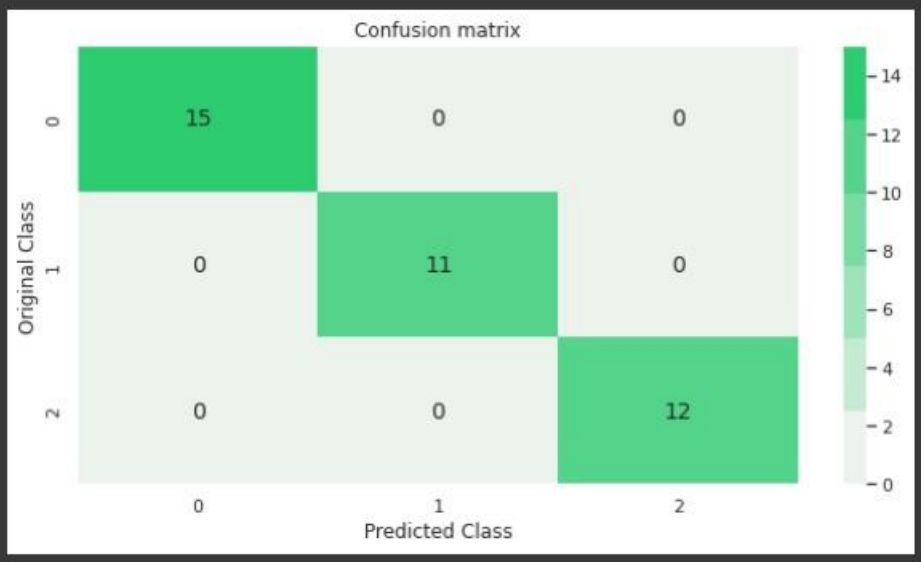
```
[ ]  def model_evaluations(X_train,y_train,X_test,y_test,ls):
         nb = GaussianNB()
         knn = KNeighborsClassifier(round(math.sqrt(X_train.shape[0])))
         dt = DecisionTreeClassifier()
         nb.fit(X_train,y_train)
         knn.fit(X_train,y_train)
         dt.fit(X_train,y_train)
         y_pred = nb.predict(X_test)
         ls.append(accuracy_score(y_test,y_pred)*100)
         print(f"\nmodel: Naive bayes \n\t\tAccuracy:{accuracy_score(y_test,y_pred)*100}")
         print(f"\n\t\tPrecision Score:{precision_score(y_test,y_pred,average='macro')*100}")
         print(f"\n\t\tRecall:{recall_score(y_test,y_pred,average='macro')*100}")
         print(f"\n\t\tF1 Score:{f1_score(y_test,y_pred,average='macro')*100}")
         print('\n============================================================\n')
         print(classification_report(y_pred,y_test))
         print('\n============================================================\n')
         plot_confusion_matrix(y_test,y_pred)
         y_pred = knn.predict(X_test)
         ls.append(accuracy_score(y_test,y_pred)*100)
         print(f"\n\nmodel: K-Nearest Neighbors \n\t\tAccuracy:{accuracy_score(y_test,y_pred)}")
         print(f"\n\t\tPrecision Score:{precision_score(y_test,y_pred,average='macro')*100}")
         print(f"\n\t\tRecall:{recall_score(y_test,y_pred,average='macro')*100}")
         print(f"\n\t\tF1 Score:{f1_score(y_test,y_pred,average='macro')*100}")
         print('\n============================================================\n')
         print(classification_report(y_pred,y_test))
         print('\n============================================================\n')
         plot_confusion_matrix(y_test,y_pred)
         y_pred = dt.predict(X_test)
         ls.append(accuracy_score(y_test,y_pred)*100)
         print(f"\n\nmodel:Decision Tree \n\t\tAccuracy:{accuracy_score(y_test,y_pred)}")
         print(f"\n\t\tPrecision Score:{precision_score(y_test,y_pred,average='macro')*100}")
         print(f"\n\t\tRecall:{recall_score(y_test,y_pred,average='macro')*100}")
         print(f"\n\t\tF1 Score:{f1_score(y_test,y_pred,average='macro')*100}")
         print('\n============================================================\n')
         print(classification_report(y_pred,y_test))
         print('\n============================================================\n')
         plot_confusion_matrix(y_test,y_pred)
```

-

a) Training set = 75% Test set = 25%

```
[ ]  seed = 42
     ls1 = []
     X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.25,random_state=seed)
     print("accuracy score for models with train set = 0.75 and test set = 0.25 ")
     model_evaluations(X_train, y_train, X_test, y_test,ls1)
```

```
    accuracy                           1.00        38
   macro avg       1.00      1.00      1.00        38
weighted avg       1.00      1.00      1.00        38


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

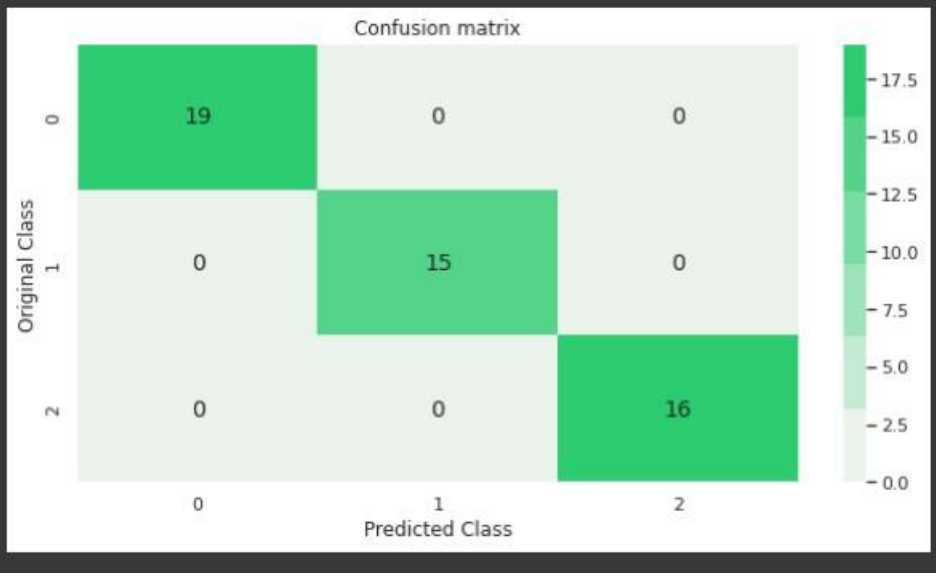


-

b) Training set = 66.6% (2/3rd of total), Test set = 33.3%

```
[ ]  ls2=[]
     X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.33,random_state=seed)
     print("accuracy score for models with train set = 0.667 and test set = 0.333 ")
     model_evaluations(X_train, y_train, X_test, y_test, ls2)
```

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

                  precision    recall  f1-score   support

              0.0       1.00      1.00      1.00        19
              1.0       1.00      1.00      1.00        15
              2.0       1.00      1.00      1.00        16

         accuracy                           1.00        50
        macro avg       1.00      1.00      1.00        50
     weighted avg       1.00      1.00      1.00        50


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



-

```
def random_subsampling(x_train,y_train,x_test,y_test):
  D=DecisionTreeClassifier()
  nb = GaussianNB()
  knn = KNeighborsClassifier(round(math.sqrt(x_train.shape[0])))
  D.fit(x_train,y_train)
  nb.fit(x_train,y_train)
  knn.fit(x_train,y_train)
  p1=D.predict(x_test)
  p2=nb.predict(x_test)
  p3=knn.predict(x_test)

  rs_results1.append(accuracy_score(y_test,p1))
  rs_results2.append(accuracy_score(y_test,p2))
  rs_results3.append(accuracy_score(y_test,p3))

  print(f"\nmodel: Decision Tree \n\t\tAccuracy:{accuracy_score(y_test,p1)*100}")
  print(f"\n\t\tPrecision Score:{precision_score(y_test,p1,average='macro')*100}")
  print(f"\n\t\tRecall:{recall_score(y_test,p1,average='macro')*100}")
  print(f"\n\t\tF1 Score:{f1_score(y_test,p1,average='macro')*100}")
  print('\n============================================================\n')
  print('\nConfusion Matrix :')
  print(confusion_matrix(yrs_test,p1))
  print('\n============================================================\n')
  print(f"\nmodel: Naive bayes \n\t\tAccuracy:{accuracy_score(y_test,p2)*100}")
  print(f"\n\t\tPrecision Score:{precision_score(y_test,p2,average='macro')*100}")
  print(f"\n\t\tRecall:{recall_score(y_test,p2,average='macro')*100}")
  print(f"\n\t\tF1 Score:{f1_score(y_test,p2,average='macro')*100}")
  print('\n============================================================\n')
  print('\nConfusion Matrix :')
  print(confusion_matrix(yrs_test,p2))
  print('\n============================================================\n')
  print(f"\nmodel: K-Nearest Neighbours \n\t\tAccuracy:{accuracy_score(y_test,p3)*100}")
  print(f"\n\t\tPrecision Score:{precision_score(y_test,p3,average='macro')*100}")
  print(f"\n\t\tRecall:{recall_score(y_test,p3,average='macro')*100}")
  print(f"\n\t\tF1 Score:{f1_score(y_test,p3,average='macro')*100}")
  print('\n============================================================\n')
  print('\nConfusion Matrix :')
  print(confusion_matrix(yrs_test,p3))
  print('\n============================================================\n')
```

-

Random Subsampling

```
[ ] ls = ['Naive bayes', 'K-Nearest Neighbours','Decision Tree']
    dict = {'Accuracy with 75% train and 25% test': ls1, 'Accuracy with 66.6% train and 33.3% test': ls2}
    data = pd.DataFrame(dict,index=ls)
    print('Dataframe of accuaracy with different classifiers using hold out method')
    data
```

```
Dataframe of accuaracy with different classifiers using hold out method
```

	Accuracy with 75% train and 25% test	Accuracy with 66.6% train and 33.3% test
Naive bayes	100.0	100.0
K-Nearest Neighbours	100.0	100.0
Decision Tree	100.0	100.0

Cross-Validation Method

```
[ ] from sklearn.model_selection import cross_val_score

    print('='*35)
    print('\nUsing Cross-Validation Method:\n')
    print('='*35)
    DT = cross_val_score(DecisionTreeClassifier(), x,y )
    print("DecisionTree :",DT.mean())

    KNN = cross_val_score(KNeighborsClassifier(), x,y )
    print("KNeighborsClassifier :",KNN.mean())

    NB = cross_val_score(GaussianNB(), x,y)
    print("GaussianNB : ",NB.mean())
```

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

    Using Cross-Validation Method:

    ===================================
    DecisionTree : 0.9333333333333333
    KNeighborsClassifier : 0.8733333333333333
    GaussianNB :  0.9933333333333334
```

-

(a) Train size : 75% , test size : 25%

```
print('\nRandom Subsampling method for train size 75% and test size 25%:\n')

rsna1 = []

for i in range(1,11):
  xrs_train,xrs_test,yrs_train,yrs_test=train_test_split(x,y,test_size=0.25,random_state=i)
  print('For Random state '+str(i)+' : ')
  random_subsampling(xrs_train,yrs_train,xrs_test,yrs_test)

result_ac = np.array(rs_results1)
net_ac = result_ac.sum()/10
print('\nNet Accuracy of Decision Tree = {a} %'.format(a = net_ac*100))
rsna1.append(net_ac*100)
result_ac = np.array(rs_results2)
net_ac = result_ac.sum()/10
print('\nNet Accuracy of Naive Bayes = {a} %'.format(a = net_ac*100))
rsna1.append(net_ac*100)
result_ac = np.array(rs_results3)
net_ac = result_ac.sum()/10
print('\nNet Accuracy of K-Nearest Neighbours = {a} %'.format(a = net_ac*100))
rsna1.append(net_ac*100)
```

```
Random Subsampling method for train size 75% and test size 25%:

For Random state 1 :

model: Decision Tree
                Accuracy:97.36842105263158

                Precision Score:96.66666666666667

                Recall:97.91666666666666

                F1 Score:97.17034521788342

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

-

(b) Train size : 66.6% , test size : 33.3%

```
rs_results1=list()
rs_results2=list()
rs_results3=list()
```

```
[ ]  print('\nRandom Subsampling method for train size 66.6% and test size 33.3%:\n')

     rsna2 = []

     for i in range(1,11):
       xrs_train,xrs_test,yrs_train,yrs_test=train_test_split(x,y,test_size=0.33,random_state=i)
       print('For Random state '+str(i)+' : ')
       random_subsampling(xrs_train,yrs_train,xrs_test,yrs_test)

     result_ac = np.array(rs_results1)
     net_ac = result_ac.sum()/10
     print('\nNet Accuracy of Decision Tree = {a} %'.format(a = net_ac*100))
     rsna2.append(net_ac*100)
     result_ac = np.array(rs_results2)
     net_ac = result_ac.sum()/10
     print('\nNet Accuracy of Naive Bayes = {a} %'.format(a = net_ac*100))
     rsna2.append(net_ac*100)
     result_ac = np.array(rs_results3)
     net_ac = result_ac.sum()/10
     print('\nNet Accuracy of K-Nearest Neighbours = {a} %'.format(a = net_ac*100))
     rsna2.append(net_ac*100)
```

```
Confusion Matrix :
[[17  0  0]
 [ 0 18  1]
 [ 0  0 14]]

============================================================

For Random state 2 :

model: Decision Tree
                Accuracy:100.0

                Precision Score:100.0
```

-

Standardizing the data

```
[ ]  df.describe()
```

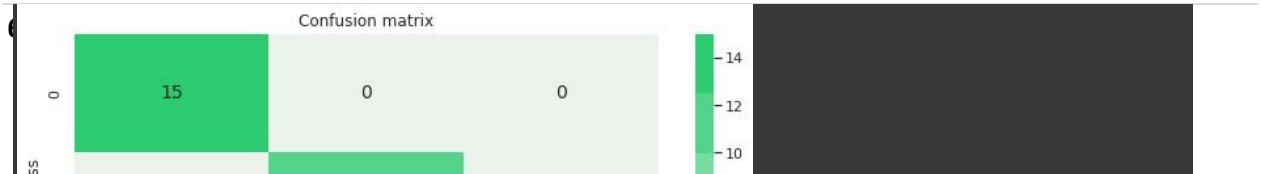
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
count	150.000000	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667	1.000000
std	43.445368	0.828066	0.433594	1.764420	0.763161	0.819232
min	1.000000	4.300000	2.000000	1.000000	0.100000	0.000000
25%	38.250000	5.100000	2.800000	1.600000	0.300000	0.000000
50%	75.500000	5.800000	3.000000	4.350000	1.300000	1.000000
75%	112.750000	6.400000	3.300000	5.100000	1.800000	2.000000
max	150.000000	7.900000	4.400000	6.900000	2.500000	2.000000

```
[ ]  seed = 42
     stand_ls=[]
     X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=seed)
     print("accuracy score for models with train set = 0.75 and test set = 0.25 and all the data is standardized")
     model_evaluations(X_train, y_train, X_test, y_test, stand_ls)
```

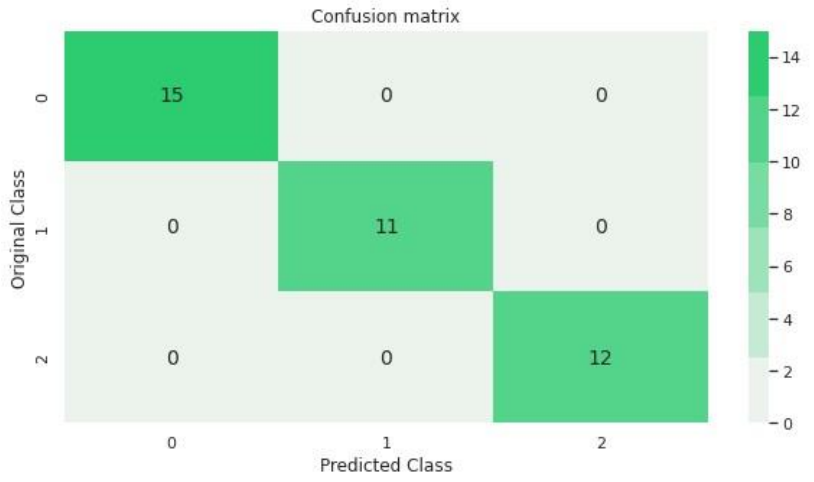
```
accuracy score for models with train set = 0.75 and test set = 0.25 and all the data is standardized

model: Naive bayes
                Accuracy:100.0

                Precision Score:100.0

                Recall:100.0

                F1 Score:100.0
```

```
accuracy score for models with train set = 0.75 and test set = 0.25 and all the
data is standardized

model: Naive bayes
              Accuracy:100.0


              Precision Score:100.0


              Recall:100.0


              F1 Score:100.0


============================================================


              precision    recall  f1-score   support
0      1.00      1.00      1.00        15
1      1.00      1.00      1.00        11            2       1.00
1.00      1.00        12

    accuracy                           1.00        38
macro avg       1.00      1.00      1.00        38
weighted avg       1.00      1.00      1.00        38



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

-


Confusion matrix
15
0
0
14
12
10

```
              Precision Score:100.0

              Recall:100.0

              F1 Score:100.0

============================================================


              precision    recall  f1-score   support
0       1.00      1.00      1.00        15
1       1.00      1.00      1.00        11            2       1.00
1.00      1.00        12

    accuracy                           1.00        38
macro avg       1.00      1.00      1.00        38
weighted avg       1.00      1.00      1.00        38


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




```
ls = ['Naive bayes', 'K-Nearest Neighbours','Decision Tree']
dict = {'Accuracy': stand_ls}
data = pd.DataFrame(dict,index=ls)
print('Dataframe of accuaracy with different classifiers after Standarization')
data
```

```
Dataframe of accuaracy with different classifiers after Standarization
```

	Accuracy
Naive bayes	100.0
K-Nearest Neighbours	100.0
Decision Tree	100.0

-

Q6. Use Simple Kmeans, DBScan, Hierachical clustering algorithms for clustering. Compare the performance of clusters by changing the parameters involved in the algorithms.

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

from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
import scipy.cluster.hierarchy as sch
import seaborn as sns

from IPython.display import display
```

```
from google.colab import drive
drive.mount("/content/drive/",force_remount=True)
```

```
Mounted at /content/drive/
```

```
#importing the data
path="/content/drive/MyDrive/Mall_Customers.csv"
dataset=pd.read_csv(path)
dataset
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
...	...	...	...	...	...

-

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Genre                   200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

inating Pre-defined Labels

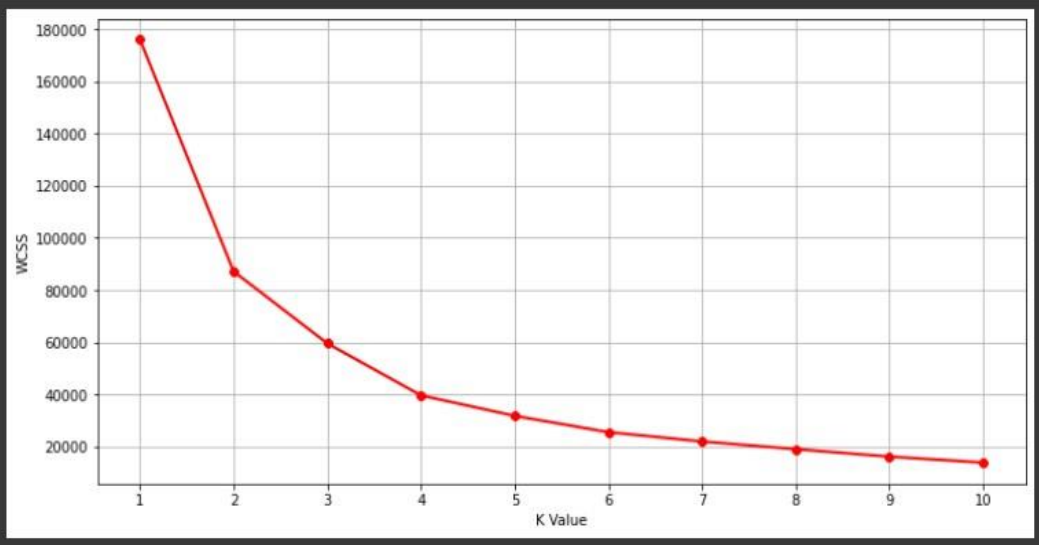
```
df = dataset.iloc[:, :-1].copy()
df
```

	CustomerID	Genre	Age	Annual Income (k$)
0	1	Male	19	15
1	2	Male	21	15
2	3	Female	20	16
3	4	Female	23	16
4	5	Female	31	17
...	...	...	...	...
195	196	Female	35	120
196	197	Female	45	126
197	198	Male	32	126
198	199	Male	32	137
199	200	Male	30	137

-

K-MEANS CLUSTERING

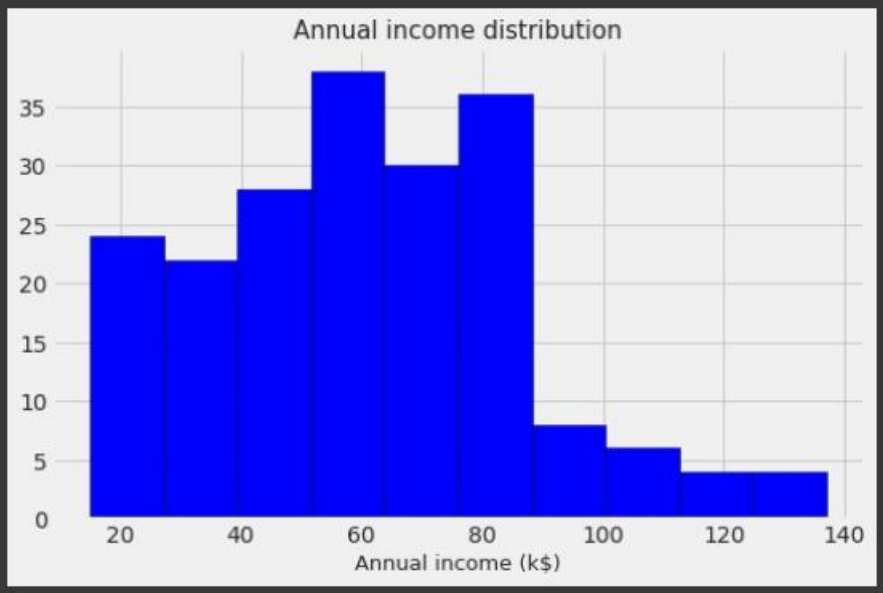
```
[ ]  #elbow method
     from sklearn.cluster import KMeans
     wcss = []
     for k in range(1,11):
         kmeans = KMeans(n_clusters=k, init="k-means++")
         kmeans.fit(df.iloc[:,1:])
         wcss.append(kmeans.inertia_)
     plt.figure(figsize=(12,6))
     plt.grid()
     plt.plot(range(1,11),wcss, linewidth=2, color="red", marker ="8")
     plt.xlabel("K Value")
     plt.xticks(np.arange(1,11,1))
     plt.ylabel("WCSS")
     plt.show()
```



-

Hierachical clustering algorithms for clustering

```
[ ]  plt.figure(figsize=(8,5))
     plt.title("Annual income distribution",fontsize=15)
     plt.xlabel ("Annual income (k$)",fontsize=13)
     plt.grid(True)
     plt.hist(dataset['Annual Income (k$)'],color='blue',edgecolor='k')
     plt.show()
```




```
[ ]  plt.figure(figsize=(8,5))
     plt.title("Spending Score distribution",fontsize=15)
     plt.xlabel ("Spending Score (1-100)",fontsize=14)
     plt.grid(True)
     plt.hist(dataset['Spending Score (1-100)'],color='brown',edgecolor='k')
     plt.show()
```

-


Spending Score distribution
35
30

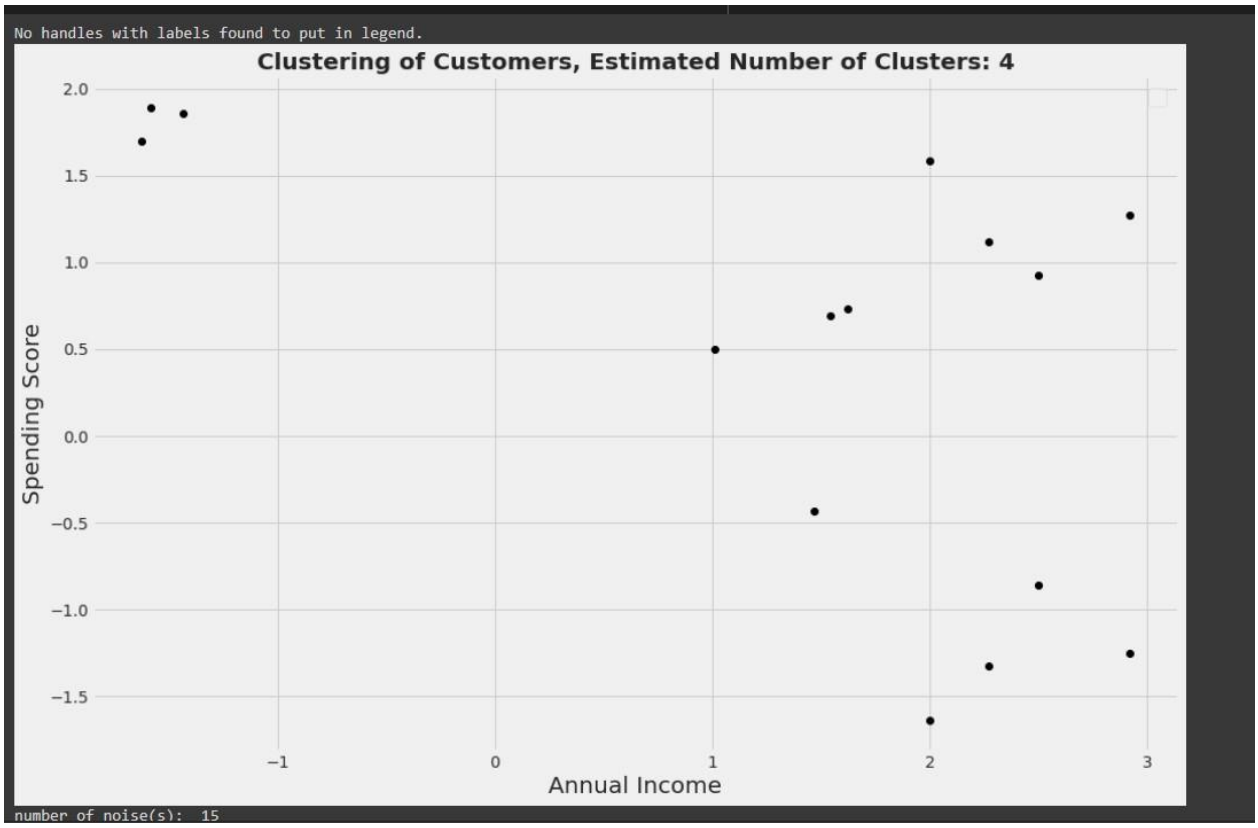
-

```
Clus_dataSet = dataset[['Annual Income (k$)','Spending Score (1-100)']]
Clus_dataSet = np.nan_to_num(Clus_dataSet)
Clus_dataSet = np.array(Clus_dataSet, dtype=np.float64)
Clus_dataSet = StandardScaler().fit_transform(Clus_dataSet)
# Compute DBSCAN
db = DBSCAN(eps=0.4, min_samples=5).fit(Clus_dataSet)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_
dataset['Clus_Db']=labels
realClusterNum=len(set(labels)) - (1 if -1 in labels else 0)
clusterNum = len(set(labels))
# A sample of clusters
print(dataset[['Annual Income (k$)','Spending Score (1-100)']].head())
# Number of Labels
print("number of labels: ", set(labels))
```

```
   Annual Income (k$)  Spending Score (1-100)
0                  15                      39
1                  15                      81
2                  16                       6
3                  16                      77
4                  17                      40
number of labels:  {0, 1, 2, 3, -1}
```

```
# Black removed and is used for noise instead
plt.figure(figsize=(15,10))
unique_labels = set(labels)
colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:

        # Black used for noise.
        col = [0, 0, 0, 1]
class_member_mask = (labels == k)
xy = Clus_dataSet[class_member_mask & core_samples_mask]
plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=14)
```

-


No handles with labels found to put in legend.
Clustering of Customers, Estimated Number of Clusters: 4
Spending Score
Annual Income
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1
0
1
2
3
number of noise(s): 15