Data Mining
Practical File

Name: Akash Kamat

University Roll No.: 20009570006

College Roll No.: 2002073

B.Sc. (Hons) Computer Science

6^{th} Semester

Index

Q1 - Create a file “people.txt” with the following data:

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 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.

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)

Source Code –

i)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import ruleset as rs
df = pd.read_table('people.txt', delim_whitespace='True')
print(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)

```
def age_range(x):
  rule1 = lambda y: y in range(151)
  return x['Age'].apply(rule1).rename('age_range')

def age_check(x):
  rule2 = lambda y: y[0] > y[1]
  return x[['Age', 'yearsmarried']].apply(rule2,
axis=1).rename('age_check')

def status_check(x):
  rule3 = lambda y: y in ['single', 'married', 'widowed']
  return x ['status'].apply(rule3).rename('status_check')

def age_group(x):
  def rule4(y):
    if (y[0] in range(18) and y[1] == 'child') or (y[0] in
range(18, 66) and y[1] == 'adult') or (y[0] > 65 and y[1] ==
'elderly'):
      return True
    else:
      return False
  return (x[['Age', 'agegroup']].apply(rule4,
axis=1).rename('age_group'))
```

iii)

```
E = {'Rule1': rs.age_range, 'Rule2': rs.age_check,'Rule3':
rs.status_check,'Rule4': rs.age_group}
result = []
for i in E.keys():
    result.append(E[i](df))
print(result)
```

```
[0     True
1     True
2     True
3    False
4     True
Name: age_range, dtype: bool, 0     True
1     True
2    False
3     True
4     True
Name: age_check, dtype: bool, 0    True
1    True
2    True
3    True
4    True
Name: status_check, dtype: bool, 0     True
1     True
2     True
3     True
4    False
Name: age_group, dtype: bool]
```

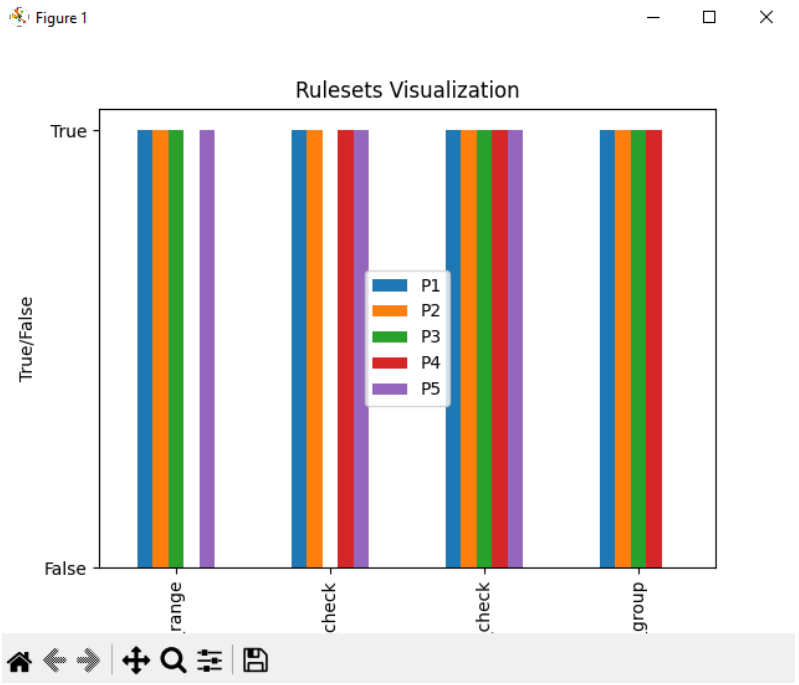
iv)

```
result = pd.DataFrame(result)
print(result.describe())
```

	0	1	2	3	4
count	4	4	4	4	4
unique	1	1	2	2	2
top	True	True	True	True	True
freq	4	4	3	3	3

v)

```
result.astype(int).plot(kind='bar', title='Rulesets Visualization')
plt.xlabel('Rules')
plt.ylabel('True/False')
plt.legend(['P1', 'P2', 'P3', 'P4', 'P5'], loc='center')
plt.yticks([0, 1], [False, True])
plt.show()
```



Q2 - Perform the following preprocessing tasks on the dirty_iris dataset.

1. Calculate the number and percentage of observations that are complete.

2. Replace all the special values in data with NA.

3. 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.

4. Determine how often each rule is broken (violatedEdits). Also summarize and plot the result. Find outliers in sepal length using boxplot and boxplot.stats.

Source Code –

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import ruleset as rs

df = pd.read_csv('dirty_iris.csv', delim_whitespace='True')

i)

```
null_values = df.isnull().sum()
print(null_values)
print(df.shape)
a = 0
for i in null_values:
    print('Percentage of complete observations: ',
null_values.index[a], (150-i)/150*100)
    a = a + 1
df.isnull().sum(axis=1).sum()
```

```
Unnamed: 0      0
Sepal.Length    8
Sepal.Width    11
Petal.Length    0
Petal.Width     0
Species         0
dtype: int64
(150, 6)
Percentage of complete observations:  Unnamed: 0 100.0
Percentage of complete observations:  Sepal.Length 94.66666666666667
Percentage of complete observations:  Sepal.Width 92.66666666666666
Percentage of complete observations:  Petal.Length 100.0
Percentage of complete observations:  Petal.Width 100.0
Percentage of complete observations:  Species 100.0
```

ii)

```
na_values = ['Inf']
df.replace(na_values, 'NA')
print(df)
```

iii)

```
import numpy as np
def species_check(df):
  species = set(['setosa', 'versicolor', 'virginica'])
  func = lambda r: r in species
  x = np.array([func(xi) for xi in df['Species']])
  if False in x:
    print('Violation: Invalid Species')
    print(str(len(x) - np.sum(x)) + ' violations')
  else:
    print('No violations')
  return (len(x) - np.sum(x))
def positive_check(df):
  func = lambda r: r > 0
  a = np.array([func(df[xi]) for xi in df.columns[:-1]])
  a = a.reshape(a.shape[0] * a.shape[1])
  if False in a:
    print('Violation: Non-positive values')
    print(str(len(a) - np.sum(a)) + ' violations')
  else:
    print('No violations')
  return (len(a) - np.sum(a))
def check_petal(df):
  a = np.array(df['Petal.Length'] > (2 * df['Petal.Width']))
  if False in a:
    print('Violation: Petal length less than twice the petal
width')
    print(str(len(a) - np.sum(a)) + ' violations')
  else:
    print('No violations')
  return (len(a) - np.sum(a))
def sepal_check(df):
  a = np.array(df['Sepal.Length'] <= 30)
  if False in a:
    print('Violation: Sepal length greater than 30')
    print(str(len(a) - np.sum(a)) + ' violations')
  else:
```

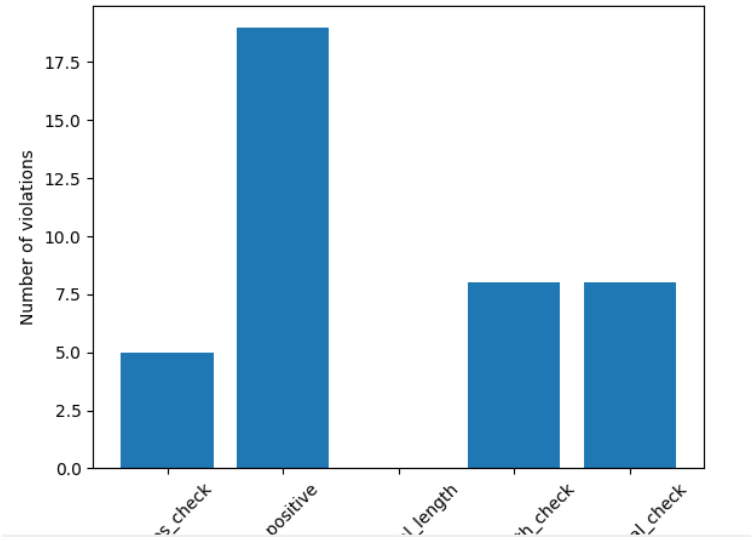
```
    print('No violations')
  return (len(a) - np.sum(a))
def sepal_petal_check(df):
  a = np.array(df['Sepal.Length'] > df['Petal.Length'])
  if False in a:
    print('Violation: Sepal length greater than Petal length')
    print(str(len(a) - np.sum(a)) + ' violations')
  else:
    print('No violations')
  return (len(a) - np.sum(a))
```

iv)

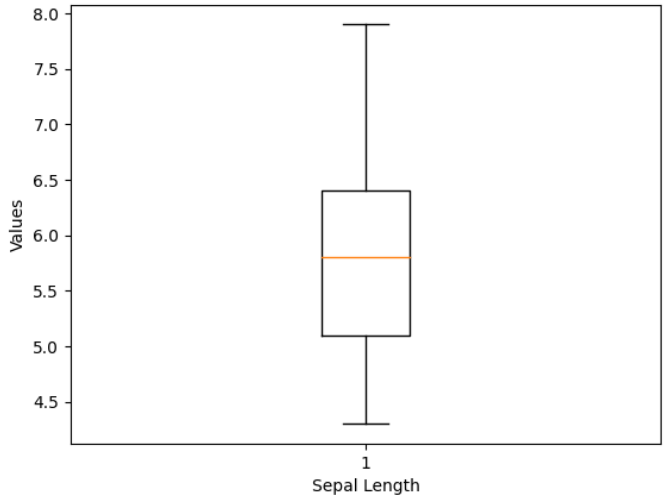
```
rules = {'species_check': rs.species_check, 'all_positive':
rs.positive_check, 'check_petal_length': rs.check_petal,
'sepal_length_check':rs.sepal_check, 'sepal_petal_check':
rs.sepal_petal_check}
result = []
for i in rules.keys():
    result.append(rules[i](df))
result = np.array(result)
print('Total violations: ', result.sum())

plt.bar(rules.keys(), result)
plt.xticks(rotation=45)
plt.xlabel('Rules')
plt.ylabel('Number of violations')
plt.show()
```

```
Violation: Invalid Species
5 violations
Violation: Non-positive values
19 violations
No violations
Violation: Sepal length greater than 30
8 violations
Violation: Sepal length greater than Petal length
8 violations
Total violations:  40
```



v)



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.

Source Code –

i) Iris Dataset:

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

```
dfwine = pd.read_csv('wine_dataset.csv')
dfiris = pd.read_csv('dirty_iris.csv')
x = dfiris.iloc[:,:-1]
y = dfiris.iloc[:,-1]
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.2)
std_s = StandardScaler()
std_s.fit(x_train)
x_train_std = std_s.transform(x_train)
x_test_std = std_s.transform(x_test)
print(x_train[0:5])
print(x_train_std[0:5])
print(x_test[0:5])
print(x_test_std[0:5])
```

```
     Unnamed: 0  Sepal.Length  Sepal.Width  Petal.Length  Petal.Width
70           71           5.9          3.2           4.8          1.8
84           85           5.4          3.0           4.5          1.5
100         101           6.3          3.3           6.0          2.5
102         103           7.1          3.0           5.9          2.1
101         102           5.8          2.7           5.1          1.9
[[-0.08590103  0.12958721  0.34569064  0.63325079  0.83360089]
 [ 0.24433634 -0.48055259 -0.0946304   0.46171558  0.43925618]
 [ 0.62175047  0.61769906  0.56585115  1.31939159  1.75373853]
 [ 0.66892724  1.59392274 -0.0946304   1.26221319  1.22794559]
 [ 0.64533886  0.00755925 -0.75511195  0.80478599  0.96504912]]
     Unnamed: 0  Sepal.Length  Sepal.Width  Petal.Length  Petal.Width
78           79           6.0          2.9           4.5          1.5
137         138           6.4          3.1           5.5          1.8
5             6           5.4          NaN           1.7          0.4
17           18           5.1          3.5           1.4          0.3
95           96           5.7          3.0           4.2          1.2
[[ 0.10280604  0.25161518 -0.31479091  0.46171558  0.43925618]
 [ 1.49452066  0.73972702  0.12553012  1.03349959  0.83360089]
 [-1.61914595 -0.48055259         nan -1.13927963 -1.0066744 ]
 [-1.33608535 -0.84663647  1.00617219 -1.31081483 -1.13812263]
 [ 0.50380856 -0.11446871 -0.0946304_  0.29018038  0.04491148]]
```

ii) Wine Dataset:

```
dfwine = pd.read_csv('wine_dataset.csv')
dfiris = pd.read_csv('dirty_iris.csv')
xwine = dfwine.iloc[:,:-1]
ywine = dfwine.iloc[:,-1]
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
xwine_train, xwine_test, ywine_train, ywine_test =
train_test_split(xwine, ywine, test_size=0.2)
obj2 = StandardScaler()
obj2.fit(xwine_train)
xwine_train_std = obj2.transform(xwine_train)
xwine_test_std = obj2.transform(xwine_test)
print(xwine_train[0:5])
print(xwine_train_std[0:5])
print(xwine_test[0:5])
print(xwine_test_std[0:5])
```

```
      fixed_acidity  volatile_acidity  citric_acid  ...  sulphates  alcohol  quality
6277            6.6             0.285         0.49  ...       0.54      8.9        6
2603            5.8             0.360         0.26  ...       0.55     11.3        6
6274            5.7             0.210         0.37  ...       0.62     10.6        6
5248            7.0             0.290         0.37  ...       0.47     12.3        6
197            11.5             0.300         0.60  ...       0.97     10.1        6

[5 rows x 12 columns]
[[-0.46623466 -0.34288173  1.19899915  1.27798284 -0.59761219  1.51360756
   0.37326473  0.88788688 -0.86883273  0.05025476 -1.32728647  0.2118334 ]
 [-1.08629999  0.10838355 -0.38553083 -0.44912457 -0.5129965   0.53847191
   0.65613031 -1.21221986  0.73234589  0.11710247  0.67791685  0.2118334 ]
 [-1.16380816 -0.79414701  0.37228786 -0.1932568  -0.45658605  1.57096848
   0.42630202 -0.46266408  0.42442692  0.58503645  0.09306588  0.2118334 ]
 [-0.156202   -0.31279738  0.37228786 -0.8116039  -0.59761219  0.19430639
   0.17879464 -1.38779149  0.23967554 -0.41767921  1.51341824  0.2118334 ]
 [ 3.33166548 -0.25262867  1.95681783 -0.72631464  0.30495513 -1.06763385
  -1.57143613  1.15124432 -0.68408135  2.92470633 -0.32468481  0.2118334 ]]
      fixed_acidity  volatile_acidity  citric_acid  ...  sulphates  alcohol  quality
478             9.6             0.68         0.24  ...       0.60     10.2        5
988             7.7             0.39         0.12  ...       0.49      9.4        5
6275            6.5             0.25         0.32  ...       0.52      9.6        6
1727            6.1             0.30         0.56  ...       0.57     10.9        7
4476            6.5             0.15         0.55  ...       0.40      9.3        5

[5 rows x 12 columns]
[[ 1.85901032  2.03378208 -0.52331604 -0.68367002  0.8690597  -1.4691603
  -1.55375704  1.38759074 -0.49932997  0.45134103 -0.24113467 -0.93732962]
 [ 0.38635517  0.28888966 -1.35002733 -0.79028158  1.15111199 -0.66610741
  -1.57143613  0.42869956 -0.37616239 -0.28398379 -0.90953578 -0.93732962]
 [-0.54374283 -0.55347219  0.02782482  0.95814813 -0.3155599   0.59583283
   0.21415284  0.56375465 -0.2529948  -0.08344066 -0.7424355   0.2118334 ]
 [-0.85377549 -0.25262867  1.6812474  -0.55573613 -0.34376513  0.93999836
   1.11578687 -0.7732908   0.48601071  0.2507979   0.3437163   1.36099641]
 [-0.54374283 -1.15515923  1.61235479  0.10525559 -0.3155599   2.54610412
   0.8152422   0.04379253 -1.54625445 -0.88561319 -0.99308592 -0.93732962]]
```

Q4 - Run Apriori algorithm to find frequent itemsets and association rules

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

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

Source Code –

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
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']]
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.column_)
```

i) minimum support as 50% and minimum confidence as 75%

```
frequent_itemsets = apriori(df, min_support=0.5, use_colnames=True)
from mlxtend.frequent_patterns import association_rules
association_rules(frequent_itemsets, metric='confidence', min_threshold=0.75)
print(frequent_itemsets)
```

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

ii) minimum support as 60% and minimum confidence as 60%

```
frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)
from mlxtend.frequent_patterns import association_rules
```

```
association_rules(frequent_itemsets, metric='confidence', min_threshold=0.6)
print(frequent_itemsets)
```

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

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) Cross-Validation. Compare the accuracy of the classifiers obtained.

5.3 Data is scaled to standard format.

Source Code –

```
import pandas as pd
import sklearn as sk
import numpy as np
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
```

```
dfwine = pd.read_csv('wine_dataset.csv')
dfwine.dropna(inplace=True)
dfiris = pd.read_csv('dirty_iris.csv')
dfiris.set_index('Species', inplace=True)
xiris = dfiris.values[:,:-1]
yiris = dfiris.values[:,-1]
x_train, x_test, y_train, y_test = train_test_split(xiris, yiris,
test_size=0.3, random_state=3)
DTclassifier = DecisionTreeClassifier()
DTclassifier.fit(x_train, y_train)
preds = DTclassifier.predict(x_test)
accuracy_score(y_test, preds)
confusion_matrix(y_test, preds)
```

5.1 a)

```
x_train, x_test, y_train, y_test = train_test_split(xwine, ywine,
test_size=0.25, random_state=0)
DTclassifier = DecisionTreeClassifier()
DTclassifier.fit(x_train, y_train)
preds = DTclassifier.predict(x_test)
print('Accuracy: ', accuracy_score(y_test, preds))
```

5.1 b)

```
x_train, x_test, y_train, y_test = train_test_split(xwine, ywine,
test_size=0.33, random_state=3)
DTclassifier = DecisionTreeClassifier()
DTclassifier.fit(x_train, y_train)
preds = DTclassifier.predict(x_test)
print('Accuracy: ', accuracy_score(y_test, preds))
```

5.2 a) Holdout method

```
x_train, x_test, y_train, y_test = train_test_split(xwine, ywine,
test_size=0.25, random_state=0)
DTclassifier = DecisionTreeClassifier()
DTclassifier.fit(x_train, y_train)
preds = DTclassifier.predict(x_test)
print('Accuracy: ', accuracy_score(y_test, preds))
```

5.2 b) Random Sub-sampling

```
x_train, x_test, y_train, y_test = train_test_split(xwine, ywine,
test_size=0.33, random_state=3)
DTclassifier = DecisionTreeClassifier()
DTclassifier.fit(x_train, y_train)
preds = DTclassifier.predict(x_test)
print('Accuracy: ', accuracy_score(y_test, preds))
```

5.3

scaler = MinMaxScaler()

print(scaler.fit(dfiris))

print(dfiris)

Species	Unnamed: 0	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Setosa	1	5.1	3.5	1.4	0.2
setosa	2	4.9	3.0	1.4	0.2
setosa	3	4.7	3.2	1.3	0.2
setosa	4	4.6	3.1	1.5	0.2
setosa	5	NaN	3.6	1.4	0.2
...	...	...	...	...	...
virginica	146	6.7	3.0	5.2	2.3
virginica	147	6.3	2.5	5.0	1.9
virginica	148	6.5	3.0	5.2	2.0
virginica	149	6.2	3.4	5.4	2.3
virginica	150	NaN	3.0	5.1	1.8

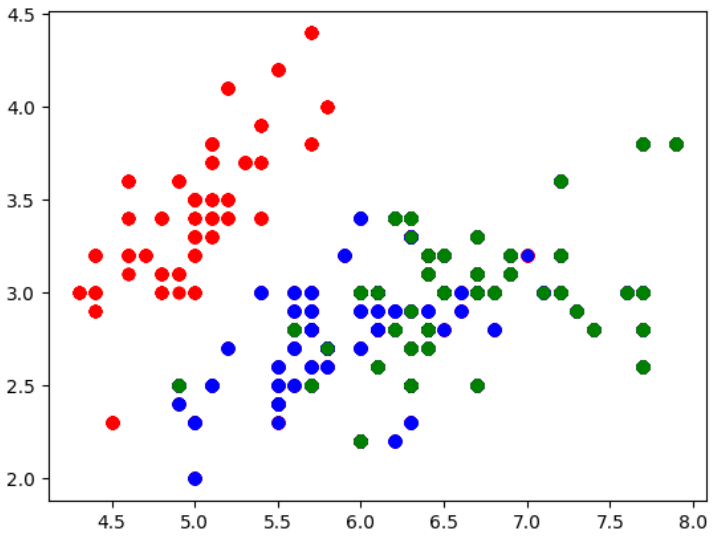
[150 rows x 5 columns]

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

Source Code –

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans, hierarchical, DBSCAN
iris = pd.read_csv('dirty_iris.csv')
```

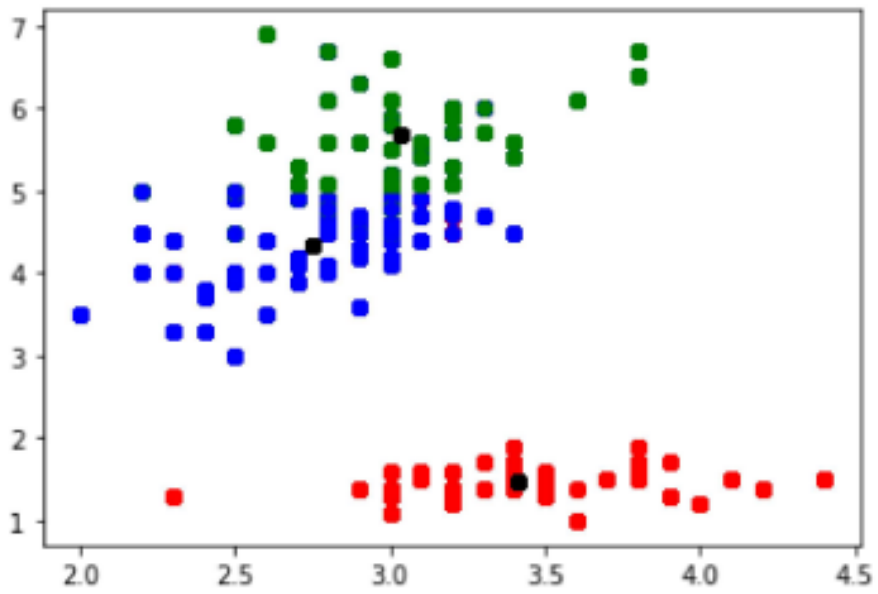
```
sep_length = iris.values[:,0]
pet_length = iris.values[:,1]
for i in range(150):
    if i <= 49:
        plt.plot(iris.values[i:,1], iris.values[i:,2], 'ro')
    if i > 49 and i <= 99:
        plt.plot(iris.values[i:,1], iris.values[i:,2], 'bo')
    if i > 99:
        plt.plot(iris.values[i:,1], iris.values[i:,2], 'go')
plt.show()
```



a) Clustering using KMeans

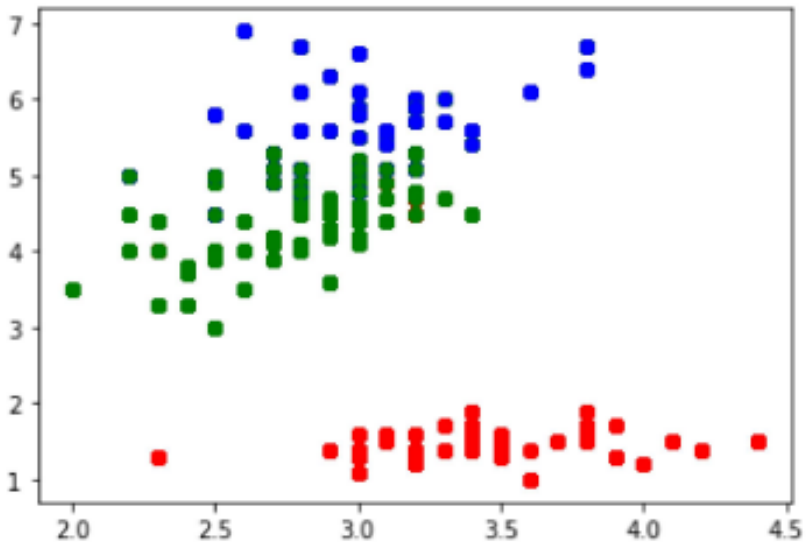
```
estimator1 = KMeans(n_clusters=3)
estimator1.fit(iris.values[:,1:3])
for i in range(149):
    if estimator1.labels_[i] == 0:
        plt.plot(iris.values[i:,1], iris.values[i:,2], 'go')
        plt.plot(estimator1.cluster_centers_[:,0],
estimator1.cluster_centers_[:,1], 'o', c='black')
    elif estimator1.labels_[i] == 1:
        plt.plot(iris.values[i:,1], iris.values[i:,2], 'ro')
        plt.plot(estimator1.cluster_centers_[:,0],
estimator1.cluster_centers_[:,1], 'o', c='black')
    elif estimator1.labels_[i] == 0:
        plt.plot(iris.values[i:,1], iris.values[i:,2], 'bo')
        plt.plot(estimator1.cluster_centers_[:,0],
```

```
estimator1.cluster_centers_[:,1], 'o', c='black')
plt.show()
```



b) Clustering using hierarchical

```
estimator2 = hierarchical.AgglomerativeClustering(n_clusters=3)
estimator2.fit(iris.values[:,1:3])
for i in range(149):
    if estimator2.labels_[i] == 0:
        plt.plot(iris.values[i:,1], iris.values[i:,2], 'go')
    elif estimator2.labels_[i] == 1:
        plt.plot(iris.values[i:,1], iris.values[i:,2], 'ro')
    elif estimator2.labels_[i] == 2:
        plt.plot(iris.values[i:,1], iris.values[i:,2], 'bo')
plt.show()
```



c) Clustering using DBScan

```
estimator3 = DBSCAN(n_clusters=3)
estimator3.fit(iris.values[:,1:3])
for i in range(149):
    if estimator3.labels_[i] == 0:
        plt.plot(iris.values[i:,1], iris.values[i:,2], 'go')
    elif estimator3.labels_[i] == 1:
        plt.plot(iris.values[i:,1], iris.values[i:,2], 'ro')
    elif estimator3.labels_[i] == 2:
        plt.plot(iris.values[i:,1], iris.values[i:,2], 'bo')
plt.show()
```