NAME: ISHA CHAUHAN

ROLL NO: 20HCS4125

PRACTICALS

Ques 1. i. Read the data from the file people.csv

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
```

```
In [2]: data = pd.read_csv("people.csv")
        data.head()
```

Out[2]:

	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

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.

```
In [ ]: def ruleset(data):
            data['Rule1'] = data['Age'].apply(lambda x: x in range(0, 150))
            data['Rule2'] = data.apply(lambda x: x.Age > x.yearsmarried, axis=1)
            data['Rule3'] = data['status'].apply(lambda x: x in {'married', 'single', 'widowed'})
            data['Rule4'] = data.apply(lambda x: (x.Age < 18 and x.agegroup == 'child') or (18 <= x.Age <= 65 and x.agegroup == 'adult')
```

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

```
In [ ]: ruleset(data)

        data
```

Out[8]:

	Age	agegroup	height	status	yearsmarried	Rule1	Rule2	Rule3	Rule4
0	21	adult	6.0	single	-1	True	True	True	True
1	2	child	3.0	married	0	True	True	True	True
2	18	adult	5.7	married	20	True	False	True	True
3	221	elderly	5.0	widowed	2	False	True	True	True
4	34	child	-7.0	married	3	True	True	True	False

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

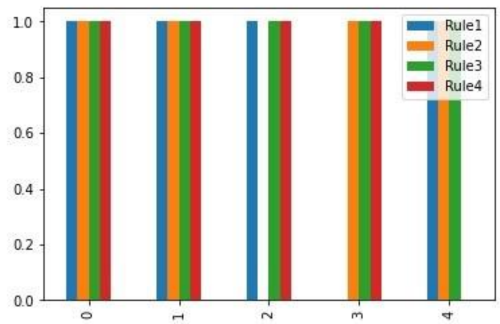
```
In [ ]: summary = data.loc[:, 'Rule1':'Rule4'].replace({True:1, False:0})

        summary
```

Out[9]:

	Rule1	Rule2	Rule3	Rule4
0	1	1	1	1
1	1	1	1	1
2	1	0	1	1
3	0	1	1	1
4	1	1	1	0

v. Visualize the results obtained in part (iii)

```
In [ ]: summary.plot(kind='bar')
        plt.show()
```



Ques 2 : Perform the following preprocessing tasks on the dirty_iris dataset.

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
```

```
In [2]: from google.colab import drive
        drive.mount('/content/drive/')

        Mounted at /content/drive/
```

Importing the dataset.

```
In [3]: path = '/content/drive/MyDrive/College Work/Data Mining/dirty_iris.csv'

        data = pd.read_csv(path)
        data.head()
```

Out[3]:

	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

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

In [6]:
```
complete_observations = data.isnull().sum(axis=1).value_counts().iloc[0]

print(f'Complete Observations: {complete_observations}')
print(f'Percentage: {complete_observations / len(data) * 100} %')
```

```
Complete Observations: 96
Percentage: 64.0 %
```

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

In [7]:
```
# data.fillna(value='NA', inplace=True)
```

---

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

In [8]:
```
data.dropna(inplace=True)
```

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

In [15]:
```
def check_species(data):
    x = data['Species'].apply(lambda x: x in {'setosa', 'versicolor', 'virginica'})
    violations = len(data) - np.sum(x)

    if violations == 0:
        print('No Violation.')
    else:
        print('Violation: Invalid Species Name.')
        print(f'Violations: {violations}')

    return violations
```

In [16]:
```
species_violations = check_species(data)
```

```
No Violation.
```

All measured numerical properties of an iris should be positive.

In [17]:
```
def check_all_positive(data):
    x = data.loc[:, 'Sepal.Length':'Petal.Width'].apply(lambda x: x > 0).values
    x = x.reshape(-1)
    violations = len(data) * 4 - np.sum(x)

    if violations == 0:
        print('No Violation.')
    else:
        print('Violation: Non-positive Numerical Property.')
        print(f'Violations: {violations}')

    return violations
```

In [18]:
```
non_positive_violations = check_all_positive(data)
```

```
Violation: Non-positive Numerical Property.
Violations: 3
```

The petal length of an iris is at least 2 times its petal width.

In [19]:
```
def check_petal_length(data):
    x = data['Petal.Length'] >= 2 * data['Petal.Width']
    violations = x.value_counts().loc[False]

    if violations == 0:
        print('No Violation.')
    else:
        print('Violation: Petal Length is less than twice its Petal Width.')
        print(f'Violations: {violations}')

    return violations
```

In [20]:
```
petal_length_violations = check_petal_length(data)
```

```
Violation: Petal Length is less than twice its Petal Width.
Violations: 2
```

The sepal length of an iris cannot exceed 30 cms.

In [21]:
```
def check_sepal_length(data):
    x = data['Sepal.Length'] <= 30
    violations = x.value_counts().loc[False]

    if violations == 0:
        print('No Violation.')
    else:
        print('Violation: Sepal Length exceeded the value of 30cms.')
        print(f'Violations: {violations}')

    return violations
```

In [22]:
```
sepal_length_violations = check_sepal_length(data)
```

```
Violation: Sepal Length exceeded the value of 30cms.
Violations: 1
```

The sepals of an iris are longer than its petals.

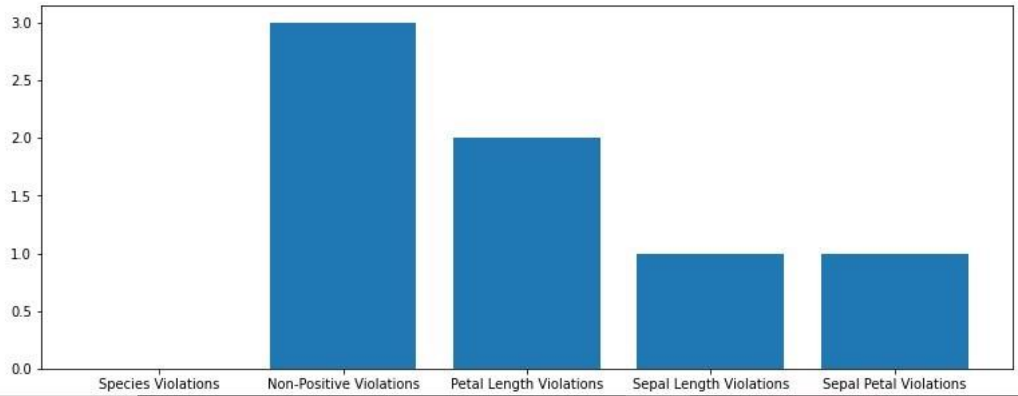
In [23]:
```
def check_sepal_petal_length(data):
    x = data['Sepal.Length'] > data['Petal.Length']
    violations = x.value_counts().loc[False]

    if violations == 0:
        print('No Violation.')
    else:
        print('Violation: Sepal Length are less than Petal Length.')
        print(f'Violations: {violations}')

    return violations
```

In [24]:
```
sepal_petal_violations = check_sepal_petal_length(data)
```

```
Violation: Sepal Length are less than Petal Length.
Violations: 1
```

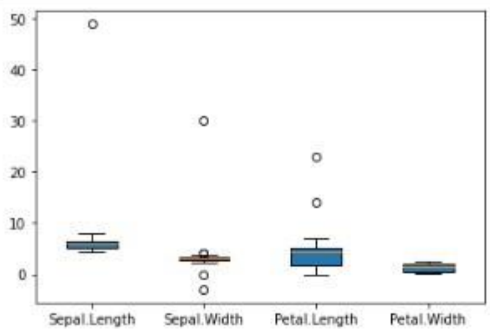
iv. Determine how often each rule is broken (violatedEdits). Also summarize and plot the result.

```
In [25]: rule_break_frequency = {
             'Species Violations': species_violations,
             'Non-Positive Violations': non_positive_violations,
             'Petal Length Violations': petal_length_violations,
             'Sepal Length Violations': sepal_length_violations,
             'Sepal Petal Violations': sepal_petal_violations
         }

         fig = plt.figure(figsize=(13, 5))
         plt.bar(rule_break_frequency.keys(), rule_break_frequency.values())
         plt.show()
```



Find outliers in sepal length using boxplot.

```
In [27]: x = [data[col] for col in data.columns[:-1]]

         box = plt.boxplot(x, labels=data.columns[:-1], patch_artist=True)
         plt.show()
```




```
In [28]: print(box.keys())

         dict_keys(['whiskers', 'caps', 'boxes', 'medians', 'fliers', 'means'])
```

```
In [29]: outliers = [item.get_ydata() for item in box['fliers']]

         print(f'Outliers in Sepal Length: {outliers[0]}')

         Outliers in Sepal Length: [49.]
```

Ques 3: 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.

In [ ]:
```
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_wine, load_iris
```

Wine Dataset

In [ ]:
```
data = load_wine()

X = data.data
```

Mean and standard deviation along the columns.

In [ ]:
```
X.mean(axis=0)
```

Out[3]:
```
array([1.30006180e+01, 2.33634831e+00, 2.36651685e+00, 1.94949438e+01,
       9.97415730e+01, 2.29511236e+00, 2.02926966e+00, 3.61853933e-01,
       1.59089888e+00, 5.05808988e+00, 9.57449438e-01, 2.61168539e+00,
       7.46893258e+02])
```

In [ ]:
```
X.std(axis=0)
```

Out[4]:
```
array([8.09542915e-01, 1.11400363e+00, 2.73572294e-01, 3.33016976e+00,
       1.42423077e+01, 6.24090564e-01, 9.96048950e-01, 1.24103260e-01,
       5.70748849e-01, 2.31176466e+00, 2.27928607e-01, 7.07993265e-01,
       3.14021657e+02])
```

Standardizing the dataset.

In [ ]:
```
sc = StandardScaler()
X = sc.fit_transform(X)
```

In [ ]:
```
X.mean(axis=0)
```

Out[6]:
```
array([ 7.84141790e-15,  2.44498554e-16, -4.05917497e-15, -7.11041712e-17,
       -2.49488320e-17, -1.95536471e-16,  9.44313292e-16, -4.17892936e-16,
       -1.54059038e-15, -4.12903170e-16,  1.39838203e-15,  2.12688793e-15,
       -6.98567296e-17])
```

In [ ]:
```
X.std(axis=0)
```

Out[7]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

Iris Dataset

In [ ]:
```
data = load_iris()

X = data.data
```

Mean and standard deviation along the columns.

```
In [ ]: X.mean(axis=0)
Out[9]: array([5.84333333, 3.05733333, 3.758     , 1.19933333])
```

```
In [ ]: X.std(axis=0)
Out[10]: array([0.82530129, 0.43441097, 1.75940407, 0.75969263])
```

Standardizing the dataset.

```
In [ ]: sc = StandardScaler()
        X = sc.fit_transform(X)
```

```
In [ ]: X.mean(axis=0)
Out[13]: array([-1.69031455e-15, -1.84297022e-15, -1.69864123e-15, -1.40924309e-15])
```

```
In [ ]: X.std(axis=0)
Out[14]: array([1., 1., 1., 1.])
```

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

```
In [ ]: import numpy as np
        import pandas as pd
        from mlxtend.preprocessing import TransactionEncoder
        from mlxtend.frequent_patterns import apriori, association_rules
```

```
In [ ]: dataset = [
            ['A', 'B', 'C', 'D', 'F', 'H'],
            ['B', 'E', 'F', 'H'],
            ['A', 'C', 'E'],
            ['B', 'C', 'D', 'F', 'H'],
            ['A', 'B', 'C', 'D', 'E'],
            ['C','D','F','H'],
            ['A','C','D','H'],
            ['E','H']
        ]
```

```
In [ ]: encoder = TransactionEncoder()
        transactions = encoder.fit_transform(dataset)

        data = pd.DataFrame(transactions, columns=encoder.columns_)
        data
```

Out[5]:

	A	B	C	D	E	F	H
0	True	True	True	True	False	True	True
1	False	True	False	False	True	True	True
2	True	False	True	False	True	False	False
3	False	True	True	True	False	True	True
4	True	True	True	True	True	False	False
5	False	False	True	True	False	True	True
6	True	False	True	True	False	False	True
7	False	False	False	False	True	False	True

```
In [ ]: frequent_itemsets = apriori(data, min_support=0.5, use_colnames=True)
        frequent_itemsets
```

Out[6]:

	support	itemsets
0	0.500	(A)
1	0.500	(B)
2	0.750	(C)
3	0.625	(D)
4	0.500	(E)
5	0.500	(F)
6	0.750	(H)
7	0.500	(C, A)
8	0.625	(C, D)
9	0.500	(C, H)
10	0.500	(D, H)
11	0.500	(F, H)
12	0.500	(C, D, H)

```
In [ ]: association_rules(frequent_itemsets, metric='confidence', min_threshold=0.75)
```

Out[7]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(A)	(C)	0.500	0.750	0.500	1.000000	1.333333	0.12500	inf
1	(C)	(D)	0.750	0.625	0.625	0.833333	1.333333	0.15625	2.25
2	(D)	(C)	0.625	0.750	0.625	1.000000	1.333333	0.15625	inf
3	(D)	(H)	0.625	0.750	0.500	0.800000	1.066667	0.03125	1.25
4	(F)	(H)	0.500	0.750	0.500	1.000000	1.333333	0.12500	inf
5	(C, D)	(H)	0.625	0.750	0.500	0.800000	1.066667	0.03125	1.25
6	(C, H)	(D)	0.500	0.625	0.500	1.000000	1.600000	0.18750	inf
7	(D, H)	(C)	0.500	0.750	0.500	1.000000	1.333333	0.12500	inf
8	(D)	(C, H)	0.625	0.500	0.500	0.800000	1.600000	0.18750	2.50

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

```
In [ ]: frequent_itemsets = apriori(data, min_support=0.6, use_colnames=True)
        frequent_itemsets
```

Out[8]:

	support	itemsets
0	0.750	(C)
1	0.625	(D)
2	0.750	(H)
3	0.625	(C, D)


```
In [ ]: association_rules(frequent_itemsets, metric='confidence', min_threshold=0.6)
```

```
In [ ]: association_rules(frequent_itemsets, metric='confidence', min_threshold=0.6)
```

Out[9]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(C)	(D)	0.750	0.625	0.625	0.833333	1.333333	0.15625	2.25
1	(D)	(C)	0.625	0.750	0.625	1.000000	1.333333	0.15625	inf

Apply apriori algorithm and find association rules on Grocery Purchase Dataset.

```
In [ ]: import numpy as np
        import pandas as pd
        from mlxtend.frequent_patterns import apriori, association_rules
```

Importing the dataset.

```
In [ ]: from google.colab import drive
        drive.mount('/content/drive/')

        Mounted at /content/drive/
```

```
In [ ]: path = '/content/drive/MyDrive/College Work/Data Mining/Grocery Products Purchase.csv'

        data = pd.read_csv(path)
        data.head()
```

Out[12]:

	Product 1	Product 2	Product 3	Product 4	Product 5	Product 6	Product 7	Product 8	Product 9	Product 10	...	Product 23	Product 24	Product 25	Product 26	Product 27	Product 28
0	citrus fruit	semi-finished bread	margarine	ready soups	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
1	tropical fruit	yogurt	coffee	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
2	whole milk	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
3	pip fruit	yogurt	cream cheese	meat spreads	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
4	other vegetables	whole milk	condensed milk	long life bakery product	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 32 columns

Getting sparse matrix of transactions.

```
In [ ]: transactions = pd.get_dummies(data.unstack().dropna()).groupby(level=1).sum()
        transactions.head()
```


Getting sparse matrix of transactions.

```
In [ ]: transactions = pd.get_dummies(data.unstack().dropna()).groupby(level=1).sum()
        transactions.head()
```


Out[13]:

	Instant food products	UHT-milk	abrasive cleaner	artif. sweetener	baby cosmetics	baby food	bags	baking powder	bathroom cleaner	beef	...	turkey	vinegar	waffles	whipped/sour cream	whisky	white bread	white wine	wh r
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	

5 rows × 169 columns

Minimum Support = 0.1

```
In [ ]: frequent_itemsets = apriori(transactions, min_support=0.1, use_colnames=True)
        frequent_itemsets
```

Out[14]:

	support	itemsets
0	0.110524	(bottled water)
1	0.193493	(other vegetables)
2	0.183935	(rolls/buns)
3	0.108998	(root vegetables)
4	0.174377	(soda)
5	0.104931	(tropical fruit)
6	0.255516	(whole milk)

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:

```
In [ ]: import numpy as np
        from sklearn.datasets import load_iris
        from sklearn.naive_bayes import GaussianNB
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.model_selection import train_test_split, cross_val_score
        from sklearn.metrics import accuracy_score, classification_report
```

```
In [ ]: X, y = load_iris(return_X_y=True)
```

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

```
In [ ]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=100)
```

Naive Bayes Classifier

```
In [ ]: gnb = GaussianNB()
        gnb.fit(X_train, y_train)

        y_pred = gnb.predict(X_test)
```

```
In [ ]: print(f'Accuracy Score: {accuracy_score(y_test, y_pred) * 100} %')

        Accuracy Score: 94.73684210526315 %
```

K-Nearest Neighbors Classifier

```
In [ ]: knn = KNeighborsClassifier()    # default k=5
        knn.fit(X_train, y_train)

        y_pred = knn.predict(X_test)
```



```
In [ ]: print(f'Accuracy Score: {accuracy_score(y_test, y_pred) * 100} %')

        Accuracy Score: 97.36842105263158 %
```

```
In [ ]: print(classification_report(y_test, y_pred))

                      precision    recall  f1-score   support

                   0       1.00      1.00      1.00        14
                   1       0.91      1.00      0.95        10
                   2       1.00      0.93      0.96        14

            accuracy                           0.97        38
           macro avg       0.97      0.98      0.97        38
        weighted avg       0.98      0.97      0.97        38
```

Decision Tree Classifier

```
In [ ]: dtree = DecisionTreeClassifier()    # default criteria='gini'
        dtree.fit(X_train, y_train)

        y_pred = dtree.predict(X_test)
```

```
In [ ]: print(f'Accuracy Score: {accuracy_score(y_test, y_pred) * 100} %')

        Accuracy Score: 94.73684210526315 %
```

```
In [ ]: print(classification_report(y_test, y_pred))
```

In [ ]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       0.90      0.90      0.90        10
           2       0.93      0.93      0.93        14

    accuracy                           0.95        38
   macro avg       0.94      0.94      0.94        38
weighted avg       0.95      0.95      0.95        38
```

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

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=100)
```

Naive Bayes Classifier

In [ ]:
```
gnb = GaussianNB()
gnb.fit(X_train, y_train)

y_pred = gnb.predict(X_test)
```

In [ ]:
```
print(f'Accuracy Score: {accuracy_score(y_test, y_pred) * 100} %')
```

```
Accuracy Score: 96.0 %
```


K-Nearest Neighbors Classifier

In [ ]:
```
knn = KNeighborsClassifier()    # default k=5
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
```

In [ ]:
```
print(f'Accuracy Score: {accuracy_score(y_test, y_pred) * 100} %')
```

```
Accuracy Score: 98.0 %
```

In [ ]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       0.92      1.00      0.96        12
           2       1.00      0.94      0.97        18

    accuracy                           0.98        50
   macro avg       0.97      0.98      0.98        50
weighted avg       0.98      0.98      0.98        50
```

Decision Tree Classifier

In [ ]:
```
dtree = DecisionTreeClassifier()    # default criteria='gini'
dtree.fit(X_train, y_train)

y_pred = dtree.predict(X_test)
```

```
In [ ]: print(f'Accuracy Score: {accuracy_score(y_test, y_pred) * 100} %')
```

```
Accuracy Score: 96.0 %
```

```
In [ ]: print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       0.92      0.92      0.92        12
           2       0.94      0.94      0.94        18

    accuracy                           0.96        50
   macro avg       0.95      0.95      0.95        50
weighted avg       0.96      0.96      0.96        50
```

5.2 a) Training set is chosen by hold out method.

```
In [ ]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=100)
```


Naive Bayes Classifier

```
In [ ]: gnb = GaussianNB()
        gnb.fit(X_train, y_train)

        y_pred = gnb.predict(X_test)
```

```
In [ ]: print(f'Accuracy Score: {accuracy_score(y_test, y_pred) * 100} %')
```

```
Accuracy Score: 95.55555555555556 %
```

K-Nearest Neighbors Classifier

```
In [ ]: knn = KNeighborsClassifier()    # default k=5
        knn.fit(X_train, y_train)

        y_pred = knn.predict(X_test)
```

```
In [ ]: print(f'Accuracy Score: {accuracy_score(y_test, y_pred) * 100} %')
```

```
Accuracy Score: 97.77777777777777 %
```

```
In [ ]: print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       0.92      1.00      0.96        11
           2       1.00      0.94      0.97        18

    accuracy                           0.98        45
```

Decision Tree Classifier

In [ ]:
```
dtree = DecisionTreeClassifier()     # default criteria='gini'
dtree.fit(X_train, y_train)

y_pred = dtree.predict(X_test)
```

In [ ]:
```
print(f'Accuracy Score: {accuracy_score(y_test, y_pred) * 100} %')
```

```
Accuracy Score: 95.55555555555556 %
```

In [ ]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       0.91      0.91      0.91        11
           2       0.94      0.94      0.94        18

    accuracy                           0.96        45
   macro avg       0.95      0.95      0.95        45
weighted avg       0.96      0.96      0.96        45
```

5.2 b) Training set is chosen by Random Subsampling.

In [ ]:
```
from sklearn.model_selection import ShuffleSplit
```


5.2 b) Training set is chosen by Random Subsampling.

In [ ]:
```
from sklearn.model_selection import ShuffleSplit
```


In [ ]:
```
rs = ShuffleSplit(n_splits=10, test_size=0.25, random_state=100)

accuracy_gnb = []
accuracy_knn = []
accuracy_dtree = []
```

In [ ]:
```
for train_index, test_index in rs.split(X):
    X_train = np.array([X[index] for index in train_index])
    X_test = np.array([X[index] for index in test_index])
    y_train = np.array([y[index] for index in train_index])
    y_test = np.array([y[index] for index in test_index])

    y_pred = GaussianNB().fit(X_train, y_train).predict(X_test)
    accuracy_gnb.append(accuracy_score(y_test, y_pred))

    y_pred = KNeighborsClassifier().fit(X_train, y_train).predict(X_test)
    accuracy_knn.append(accuracy_score(y_test, y_pred))

    y_pred = DecisionTreeClassifier().fit(X_train, y_train).predict(X_test)
    accuracy_dtree.append(accuracy_score(y_test, y_pred))
```

In [ ]:
```
print(f'Mean accuracy of Gaussian Naive Bayes: {sum(accuracy_gnb) / len(accuracy_gnb) * 100} %')
print(f'Mean accuracy of K-Nearest Neighbors: {sum(accuracy_knn) / len(accuracy_knn) * 100} %')
print(f'Mean accuracy of Decision Tree Classifier: {sum(accuracy_dtree) / len(accuracy_dtree) * 100} %')
```

```
Mean accuracy of Gaussian Naive Bayes: 96.05263157894737 %
Mean accuracy of K-Nearest Neighbors: 96.84210526315789 %
Mean accuracy of Decision Tree Classifier: 95.52631578947366 %
```

5.2 c) Training set is chosen by Cross Validation.

In [ ]:
```
dtree = DecisionTreeClassifier()
knn = KNeighborsClassifier()
gnb = GaussianNB()
```

In [ ]:
```
accuracy_dtree = cross_val_score(dtree, X, y, cv=5)
accuracy_knn = cross_val_score(knn, X, y, cv=5)
accuracy_gnb = cross_val_score(gnb, X, y, cv=5)
```

In [ ]:
```
print(f'Mean accuracy of Gaussian Naive Bayes: {sum(accuracy_gnb) / len(accuracy_gnb) * 100} %')
print(f'Mean accuracy of K-Nearest Neighbors: {sum(accuracy_knn) / len(accuracy_knn) * 100} %')
print(f'Mean accuracy of Decision Tree Classifier: {sum(accuracy_dtree) / len(accuracy_dtree) * 100} %')
```

```
Mean accuracy of Gaussian Naive Bayes: 95.33333333333334 %
Mean accuracy of K-Nearest Neighbors: 97.33333333333334 %
Mean accuracy of Decision Tree Classifier: 96.66666666666669 %
```

5.3 Data is scaled to standard format.

In [ ]:
```
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=100)
```

In [ ]:
```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Naive Bayes Classifier

In [ ]:
```
gnb = GaussianNB()
gnb.fit(X_train, y_train)

y_pred = gnb.predict(X_test)
```

In [ ]:
```
print(f'Accuracy Score: {accuracy_score(y_test, y_pred) * 100} %')
```

```
Accuracy Score: 94.73684210526315 %
```

K-Nearest Neighbors Classifier

In [ ]:
```
knn = KNeighborsClassifier()    # default k=5
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
```

In [ ]:
```
print(f'Accuracy Score: {accuracy_score(y_test, y_pred) * 100} %')
```

```
Accuracy Score: 97.36842105263158 %
```

In [ ]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       0.91      1.00      0.95        10
           2       1.00      0.93      0.96        14

    accuracy                           0.97        38
   macro avg       0.97      0.98      0.97        38
weighted avg       0.98      0.97      0.97        38
```

Decision Tree Classifier

```
In [ ]: dtree = DecisionTreeClassifier()    # default criteria='gini'
        dtree.fit(X_train, y_train)

        y_pred = dtree.predict(X_test)
```

```
In [ ]: print(f'Accuracy Score: {accuracy_score(y_test, y_pred) * 100} %')
```

```
Accuracy Score: 94.73684210526315 %
```

```
In [ ]: print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       0.90      0.90      0.90        10
           2       0.93      0.93      0.93        14

    accuracy                           0.95        38
   macro avg       0.94      0.94      0.94        38
weighted avg       0.95      0.95      0.95        38
```

Ques6. Use Simple Kmeans, DBScan, Hierachical clustering algorithms for clustering. Compare the performance of clusters by changing the parameters involved in the algorithms

```
In [ ]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt

        from sklearn.datasets import load_iris
        from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
```

```
In [ ]: data = load_iris(as_frame=True).frame

        data.head()
```

Out[9]:

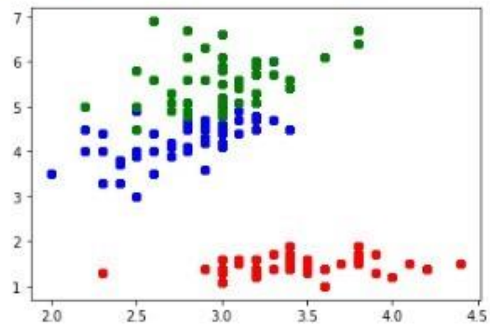
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Plotting Sepal Width and Petal Length

```
In [ ]: for index in range(150):
            if index <= 49:
                plt.plot(data.values[index:, 1], data.values[index:, 2], 'ro')
            elif index > 49 and index <= 99:
                plt.plot(data.values[index:, 1], data.values[index:, 2], 'bo')
            elif index > 99:
                plt.plot(data.values[index:, 1], data.values[index:, 2], 'go')

        plt.show()
```

Plotting Sepal Width and Petal Length

In [ ]:
```
for index in range(150):
    if index <= 49:
        plt.plot(data.values[index:, 1], data.values[index:, 2], 'ro')
    elif index > 49 and index <= 99:
        plt.plot(data.values[index:, 1], data.values[index:, 2], 'bo')
    elif index > 99:
        plt.plot(data.values[index:, 1], data.values[index:, 2], 'go')

plt.show()
```



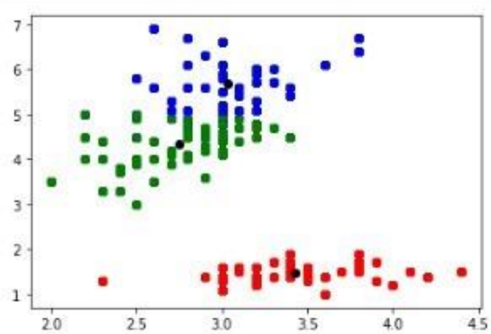
K-Means Clustering

K-Means Clustering ¶

In [ ]:
```
k_cluster = KMeans(n_clusters=3)

k_cluster.fit(data.values[:, 1:3])
```

Out[18]: KMeans(n_clusters=3)

In [ ]:
```
for index in range(150):
    if k_cluster.labels_[index] == 0:
        plt.plot(data.values[index:, 1], data.values[index:, 2], 'go')
    elif k_cluster.labels_[index] == 1:
        plt.plot(data.values[index:, 1], data.values[index:, 2], 'ro')
    elif k_cluster.labels_[index] == 2:
        plt.plot(data.values[index:, 1], data.values[index:, 2], 'bo')

plt.plot(k_cluster.cluster_centers_[:, 0], k_cluster.cluster_centers_[:, 1], 'o', c='black')
plt.show()
```

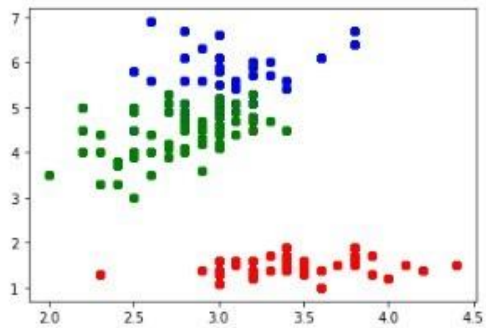


Hierarchical Agglomerative Clustering

In [ ]:
```
agg_cluster = AgglomerativeClustering(n_clusters=3)

agg_cluster.fit(data.values[:, 1:3])
```

Out[22]: AgglomerativeClustering(n_clusters=3)

In [ ]:
```
for index in range(150):
    if agg_cluster.labels_[index] == 0:
        plt.plot(data.values[index:, 1], data.values[index:, 2], 'go')
    elif agg_cluster.labels_[index] == 1:
        plt.plot(data.values[index:, 1], data.values[index:, 2], 'ro')
    elif agg_cluster.labels_[index] == 2:
        plt.plot(data.values[index:, 1], data.values[index:, 2], 'bo')

plt.show()
```



DBSCAN Clustering

In [ ]:
```
db_cluster = DBSCAN()

db_cluster.fit(data.values[:, 1:3])
```

Out[24]: DBSCAN()

In [ ]:
```
for index in range(150):
    if db_cluster.labels_[index] == 0:
        plt.plot(data.values[index:, 1], data.values[index:, 2], 'go')
    elif db_cluster.labels_[index] == 1:
        plt.plot(data.values[index:, 1], data.values[index:, 2], 'ro')
    elif db_cluster.labels_[index] == 2:
        plt.plot(data.values[index:, 1], data.values[index:, 2], 'bo')

plt.show()
```