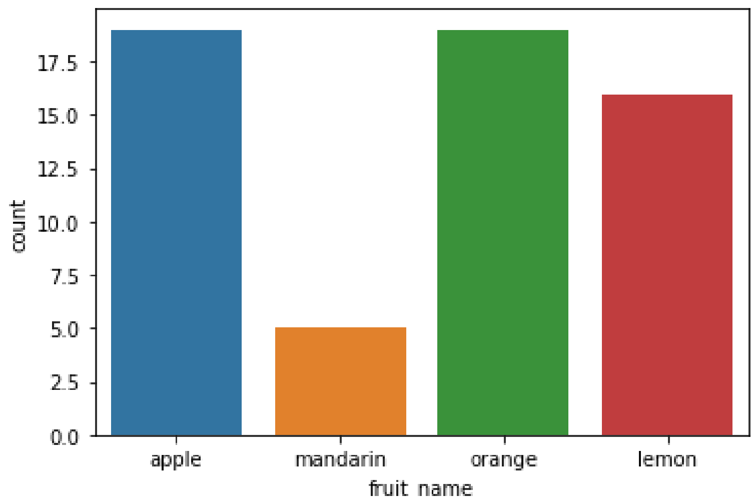
Practical no – 1

On the fruit dataset, compare the performance of Logistic Regression, SVM, KNN on the basis of their accuracy.

sns.countplot(df['fruit_name'],label='Count') # count plot

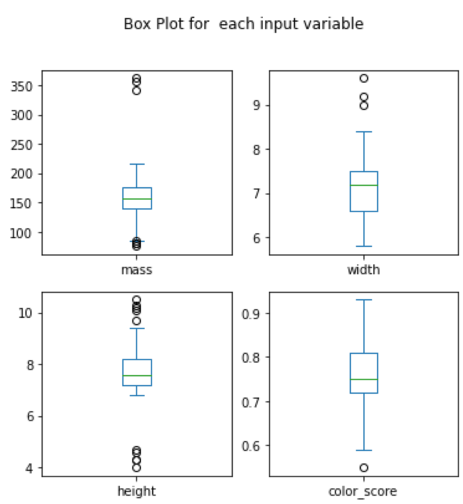
plt.show()



df.drop('fruit_label',axis=1).plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False, figsize=(6,6), title='Box Plot for each input variable')

plt.savefig('fruits_box')

plt.show()



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
feature_names = ['mass', 'width', 'height', 'color_score']
x=df[feature_names]
y=df['fruit_label']
x_train, x_test, y_train, y_test = train_test_split(x,y, random_state=0)
print(x_train[:3]) # to check output
scaler = MinMaxScaler()
x_train=scaler.fit_transform(x_train)
x_test= scaler.transform(x_test)
print("\nAfter scaling\n")
print(x_train[:3])
```

```
    mass  width  height  color_score
42   154    7.2     7.2         0.82
48   174    7.3    10.1         0.72
7     76    5.8     4.0         0.81

After scaling

[[0.27857143 0.41176471 0.49230769 0.72972973]
 [0.35       0.44117647 0.93846154 0.45945946]
 [0.         0.         0.         0.7027027 ]]
```

```
from sklearn.linear_model import LogisticRegression   # machine learning lib/model

feature_names = ['mass', 'width', 'height', 'color_score']

x=df[feature_names]

y=df['fruit_label']


x_train, x_test, y_train, y_test = train_test_split(x,y, random_state=0)


scaler = MinMaxScaler()

x_train=scaler.fit_transform(x_train)

x_test= scaler.transform(x_test)


#logistic regression

logreg = LogisticRegression()  # machine learning algorithm

logreg.fit(x_train, y_train)


#print score of train data

print('Accuracy of Logistic regression classifier on training set:{:.2f}'
     .format(logreg.score(x_train, y_train)))


#print score of test data

print('Accuracy of Logistic regression classifier on test set:{:.2f}'
     .format(logreg.score(x_test, y_test)))
```

Accuracy of Logistic regression classifier on training set:0.75

Accuracy of Logistic regression classifier on test set:0.47

```
from sklearn.neighbors import KNeighborsClassifier
```

```
# KNN method
knn = KNeighborsClassifier()
knn.fit(x_train, y_train)

#print score of train data
print('Accuracy of KNN classifier on training set:{:.2f}'
     .format(knn.score(x_train, y_train)))

#print score of test data
print('Accuracy of KNN Classifier on test set:{:.2f}'
     .format(knn.score(x_test, y_test)))
Accuracy of KNN classifier on training set:0.95
Accuracy of KNN Classifier on test set:1.00
from sklearn.svm import SVC

# SVM classifier
svm = SVC()
svm.fit(x_train, y_train)

#print score of train data
print('Accuracy of SVM classifier on training set:{:.2f}'
     .format(svm.score(x_train, y_train)))

#print score of test data
```

```
print('Accuracy of SVM Classifier on test set:{:.2f}'
     .format(svm.score(x_test, y_test)))
```

Accuracy of SVM classifier on training set:0.91

Accuracy of SVM Classifier on test set:0.80

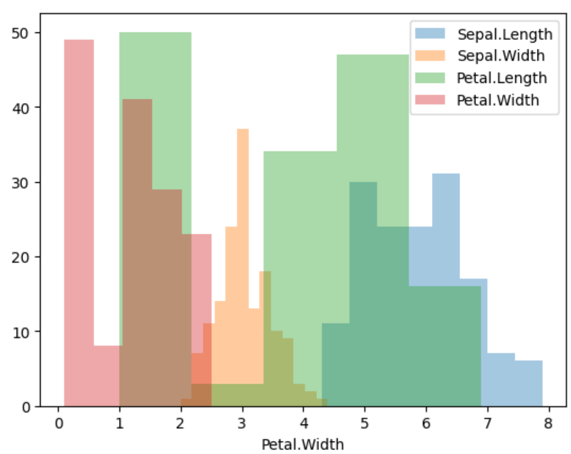
```
data = {'Training Accuracy (in %)':[75,95,91],'Testing Accuracy (in %)':[47,100,80]}


df1 = pd.DataFrame(data, index =['Logistic Regression','K-Nearest Neighbour (KNN)','Support Vector Machine (SVM)'])
  df1
```

	Training Accuracy (in %)	Testing Accuracy (in %)
Logistic Regression	75	47
K-Nearest Neighbour (KNN)	95	100
Support Vector Machine (SVM)	91	80

Practical No – 2

2) On the iris dataset, perform KNN algorithm and discuss result

```
import pandas as pd
iris = pd.read_csv("iris.csv")
iris.dtypes
import matplotlib.pyplot as plt # mostly used for visualization purposes
import numpy as np
import seaborn as sns

sns.distplot(iris['Sepal.Length'], kde=False,label='Sepal.Length')
sns.distplot(iris['Sepal.Width'], kde=False,label='Sepal.Width')
sns.distplot(iris['Petal.Length'], kde=False,label='Petal.Length')
sns.distplot(iris['Petal.Width'], kde=False,label='Petal.Width')
plt.legend()
```




```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

```
feature_names = ['Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width']
x=iris[feature_names]
y=iris['Species']

x_train, x_test, y_train, y_test = train_test_split(x,y, random_state=0)

print(x_train[:3]) # to check output

scaler = MinMaxScaler()
x_train=scaler.fit_transform(x_train)
x_test= scaler.transform(x_test)

print("\nAfter scaling\n")
print(x_train[:3]) # to check output
```

```
     Sepal.Length  Sepal.Width  Petal.Length  Petal.Width
61            5.9          3.0           4.2          1.5
92            5.8          2.6           4.0          1.2
112           6.8          3.0           5.5          2.1

After scaling

[[0.44444444 0.41666667 0.53448276 0.58333333]
 [0.41666667 0.25       0.5        0.45833333]
 [0.69444444 0.41666667 0.75862069 0.83333333]]
```

```
from sklearn.neighbors import KNeighborsClassifier

# KNN method
knn = KNeighborsClassifier()
knn.fit(x_train, y_train)

#print score of train data
print('Accuracy of KNN classifier on training set:{:.2f}'
    .format(knn.score(x_train, y_train)))

#print score of test data
print('Accuracy of KNN Classifier on test set:{:.2f}'
    .format(knn.score(x_test, y_test)))
```

```
Accuracy of KNN classifier on training set:0.96
Accuracy of KNN Classifier on test set:0.97
```

Practical No – 3

3) Implement apriori algorithm on online retail dataset and discuss result

```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
/kaggle/input/online-retail-ii-uci/online_retail_II.csv
transaction_df= pd.read_csv("../input/online-retail-ii-uci/online_retail_II.csv")
transaction_df
transaction_df = transaction_df[transaction_df.Country=='France']
transaction_filtered = transaction_df[['Invoice','Description','Quantity']].copy()
transaction_filtered
transaction_filtered.sort_values(by='Quantity', ascending=True)
transaction_filtered = transaction_filtered[transaction_filtered.Quantity > 0 ]
transaction_filtered.sort_values(by='Quantity', ascending=True)
transaction_filtered['Quantity']= [1]*len(transaction_filtered)
invoice = list(transaction_filtered.Invoice)
index_no = [invoice[index] for index in np.arange(len(invoice)) if not
invoice[index].isnumeric()]
transaction_filtered[transaction_filtered['Invoice'].isin(index_no)]
transaction_filtered=
transaction_filtered[~transaction_filtered['Invoice'].isin(index_no)]
invoice = list(transaction_filtered.Invoice)
index_no = [index for index in np.arange(len(invoice)) if not
invoice[index].isnumeric()]
transaction_filtered.iloc[index_no,:]
```

```
temp_df = transaction_filtered[transaction_filtered.Description !=
transaction_filtered.Description]

temp_df

for invoice in list(temp_df.Invoice):

    if len(transaction_filtered[transaction_filtered.Invoice == invoice]) > 1:

        print((str)(invoice))

        temp = transaction_filtered[transaction_filtered.Invoice ==
invoice].groupby(['Invoice']).agg({'Description':lambda x: list(x)})

        if len(list(set(temp)))>0 :

            print(temp)

transaction_filtered.dropna(axis=0, inplace=True)

transaction_filtered

def return_one(x):

    return 1

table = pd.pivot_table(transaction_filtered, values='Quantity', index=['Invoice'],

                    columns=['Description'], aggfunc=return_one, fill_value=0)

table

frequent_itemsets = apriori(table, min_support=0.01, use_colnames=True)

frequent_itemsets

rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)

rules

rules.sort_values(by=['support','confidence'], ascending=False)
```

Practical No – 4

4)Implement Naïve Bayes Classifier and K-Nearest Neighbor Classifier on Data set of your choice. Test and Compare for Accuracy and Precision.

```
df=pd.read_csv('fruit_data.csv')

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import MinMaxScaler


feature_names = ['mass', 'width', 'height', 'color_score']

x=df[feature_names]

y=df['fruit_label']


x_train, x_test, y_train, y_test = train_test_split(x,y, random_state=0)


print(x_train[:3]) # to check output


scaler = MinMaxScaler()

x_train=scaler.fit_transform(x_train)

x_test= scaler.transform(x_test)


print("\nAfter scaling\n")

print(x_train[:3]) # to check output
```

```
    mass  width  height  color_score
42   154    7.2     7.2         0.82
48   174    7.3    10.1          0.72
```

```
7    76    5.8    4.0         0.81

After scaling

[[0.27857143 0.41176471 0.49230769 0.72972973]
 [0.35       0.44117647 0.93846154 0.45945946]
 [0.        0.         0.         0.7027027 ]]

from sklearn.neighbors import KNeighborsClassifier

# KNN method
knn = KNeighborsClassifier()
knn.fit(x_train, y_train)

#print score of train data
print('Accuracy of KNN classifier on training set:{:.2f}'
     .format(knn.score(x_train, y_train)))

#print score of test data
print('Accuracy of KNN Classifier on test set:{:.2f}'
     .format(knn.score(x_test, y_test)))
Accuracy of KNN classifier on training set:0.95
Accuracy of KNN Classifier on test set:1.00
from sklearn.naive_bayes import GaussianNB
```

```
# Gaussian Naive bayes
gnb = GaussianNB()
gnb.fit(x_train, y_train)

#print score of train data
print('Accuracy of GNB classifier on training set:{:.2f}'
     .format(gnb.score(x_train, y_train)))

#print score of test data
print('Accuracy of GNB Classifier on test set:{:.2f}'
     .format(gnb.score(x_test, y_test)))
```

Accuracy of GNB classifier on training set:0.86

Accuracy of GNB Classifier on test set:0.67

apply confusin matrix

confusion matrix for KNN

```
import sklearn
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming you have already trained your classification models, e.g., K-Nearest Neighbors (knn)
y_pred_knn = knn.predict(x_test)  # Make predictions on the test data
```

Create the confusion matrix for K-Nearest Neighbors

confusion_knn = confusion_matrix(y_test, y_pred_knn)

Create a heatmap of the confusion matrix

plt.figure(figsize=(8, 6))

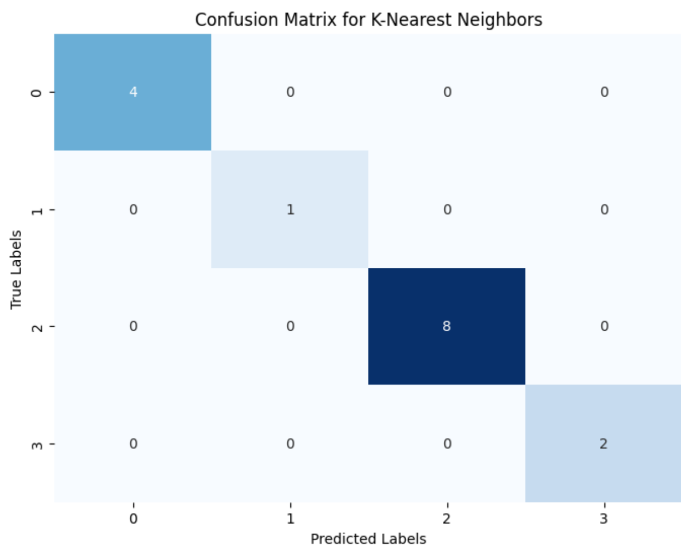
sns.heatmap(confusion_knn, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.title('Confusion Matrix for K-Nearest Neighbors')

plt.show()



confusion matrix for naive bayes

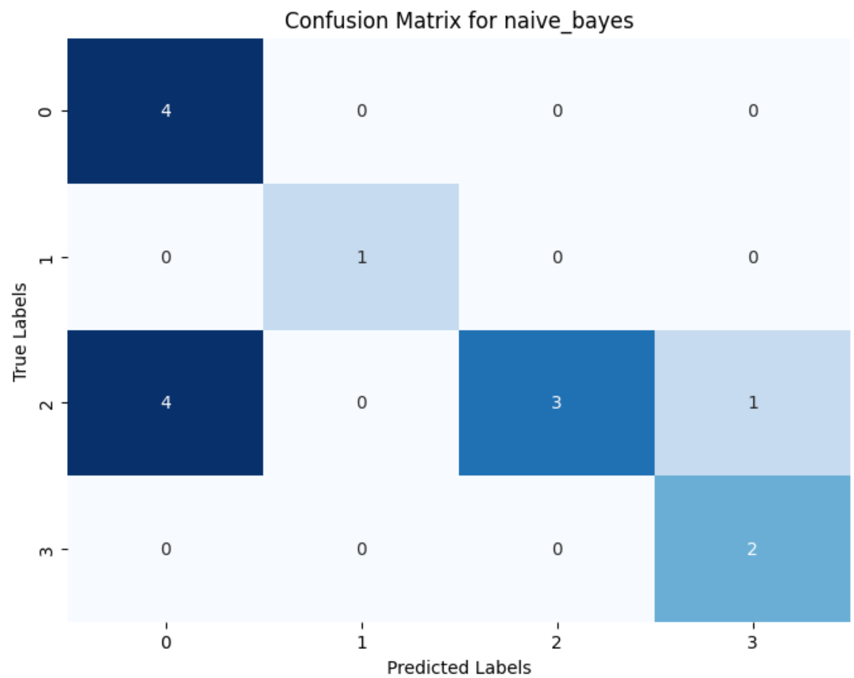
from sklearn.metrics import confusion_matrix

import seaborn as sns

import matplotlib.pyplot as plt

Assuming you have already trained your classification models

```
y_pred_gnb = gnb.predict(x_test)  # Make predictions on the test data

# Create the confusion matrix
confusion_gnb = confusion_matrix(y_test, y_pred_gnb)


# Create a heatmap of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_gnb, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for naive_bayes')
plt.show()
```



Practical No – 5

5) Implement K-means Clustering on a proper dataset of your choice

K-Means Clustering : Perform clustering for the crime data and identify the number of clusters formed and draw inferences. Refer to crime_data.csv dataset.

crime = pd.read_csv("crime_data.csv")

crime['State'] = crime.iloc[:,0]

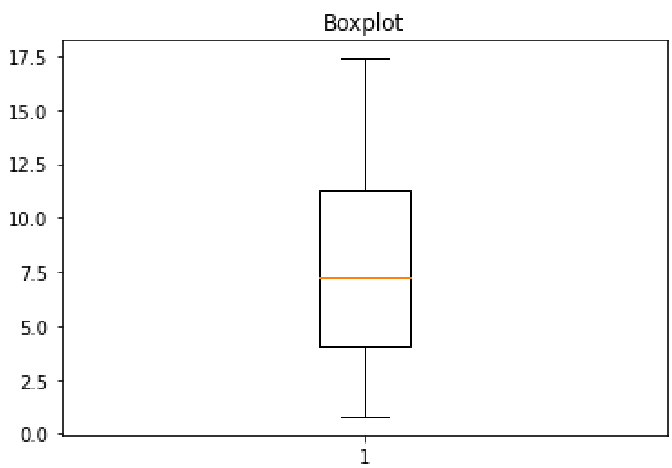
crime = crime.iloc[:, [5,1,2,3,4]]

crime.isna().sum()

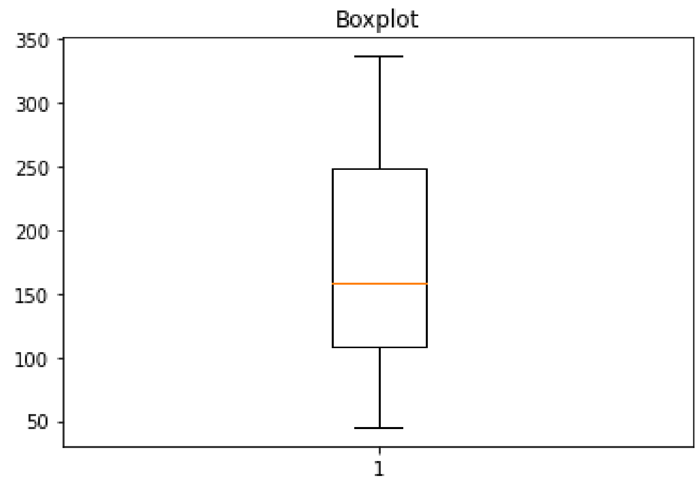
crime1 = crime.duplicated()

sum(crime1)

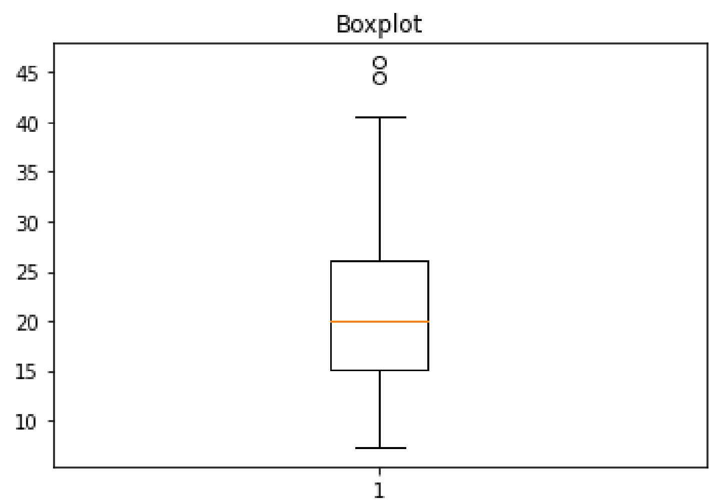
plt.boxplot(crime.Murder);plt.title('Boxplot');plt.show()



plt.boxplot(crime.Assault);plt.title('Boxplot');plt.show()



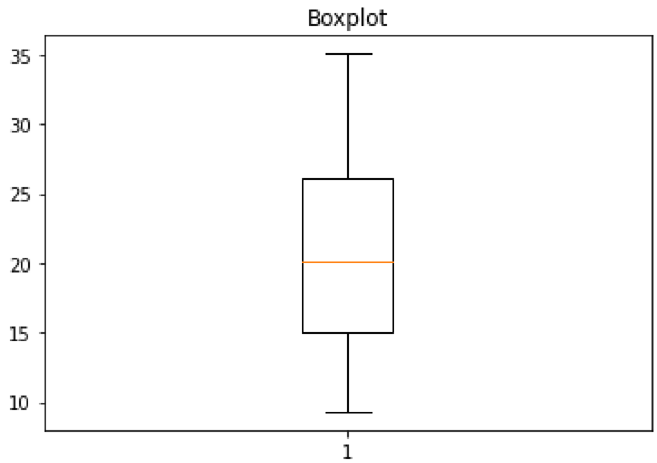
plt.boxplot(crime.Rape);plt.title('Boxplot');plt.show()



from scipy.stats.mstats import winsorize

crime['Rape'] = winsorize(crime.Rape, limits=[0.07, 0.093])

plt.boxplot(crime['Rape']);plt.title('Boxplot');plt.show()

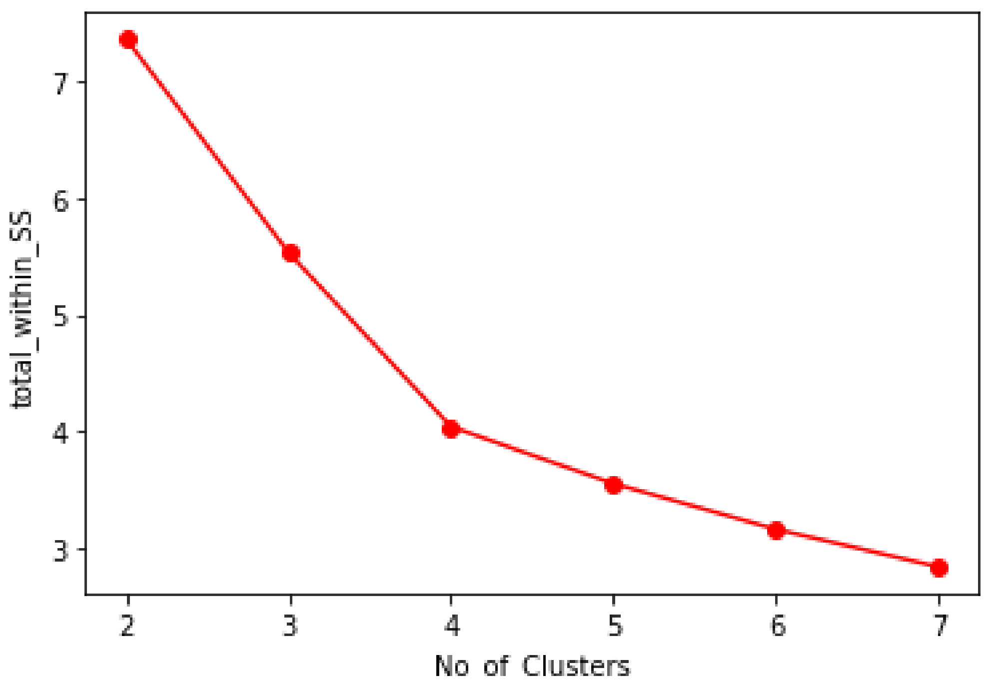



```
(crime == 0).all()
State       False
Murder      False
Assault     False
UrbanPop    False
Rape        False
dtype: bool

from sklearn.cluster import KMeans

TWSS = []
k = list(range(2, 8))

for i in k:
    kmeans = KMeans(n_clusters = i)
    kmeans.fit(df_norm)
    TWSS.append(kmeans.inertia_)
```

TWSS

[7.358376498536079,
 5.532071995078602,
 4.0407678952238815,
 3.5539811127025747,
 3.1628651131109455,
 2.8417637970747243]

plt.plot(k, TWSS, 'ro-');plt.xlabel("No_of_Clusters");plt.ylabel("total_within_SS")



model = KMeans(n_clusters = 4)

model.fit(df_norm)

KMeans(n_clusters=4)

crime.iloc[:, 1:6].groupby(crime.clust).mean()