Web Mining Lab Digital Assignment 4

Devang Mehrotra

18BCE0763

Q1

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xlrd
from sklearn.cluster import AgglomerativeClustering
```

In [6]:

```
dataset = pd.read_excel('Absenteeism_at_work.xls',header=None)
```

In [7]:

```
dataset.head()
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9	...	11	12	13	14	15	16	17	18	19	20
0	11	26	7	3	1	289	36	13	33	239554	...	0	1	2	1	0	1	90	172	30	4
1	36	0	7	3	1	118	13	18	50	239554	...	1	1	1	1	0	0	98	178	31	0
2	3	23	7	4	1	179	51	18	38	239554	...	0	1	0	1	0	0	89	170	31	2
3	7	7	7	5	1	279	5	14	39	239554	...	0	1	2	1	1	0	68	168	24	4
4	11	23	7	5	1	289	36	13	33	239554	...	0	1	2	1	0	1	90	172	30	2

5 rows × 21 columns

Handling Missing values:

In [8]:

```
dataset.fillna(method ='ffill', inplace = True)
```

In [9]:

```
X = dataset.iloc[:,:].values
X
```

Out[9]:

```
array([[ 11,  26,   7, ..., 172,  30,   4],
       [ 36,   0,   7, ..., 178,  31,   0],
       [  3,  23,   7, ..., 170,  31,   2],
       ...,
       [  4,   0,   0, ..., 170,  34,   0],
       [  8,   0,   0, ..., 170,  35,   0],
       [ 35,   0,   0, ..., 175,  25,   0]], dtype=int64)
```

In [10]:

```
#No missing values
```

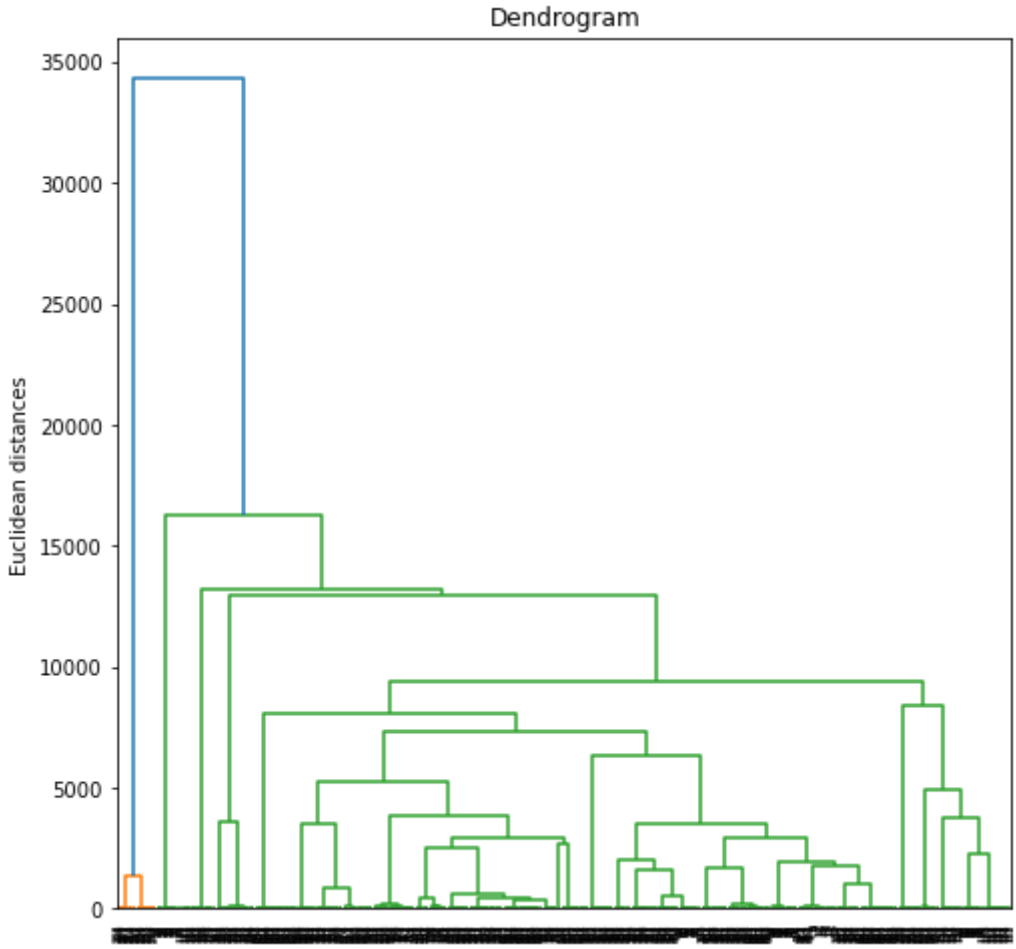
1) Single Linkage

In [11]:

```
#PCA to lower down 20 columns to 2 columns
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X_principal = pca.fit_transform(X)
X_principal = pd.DataFrame(X_principal)
X_principal.columns = ['P1', 'P2']
```

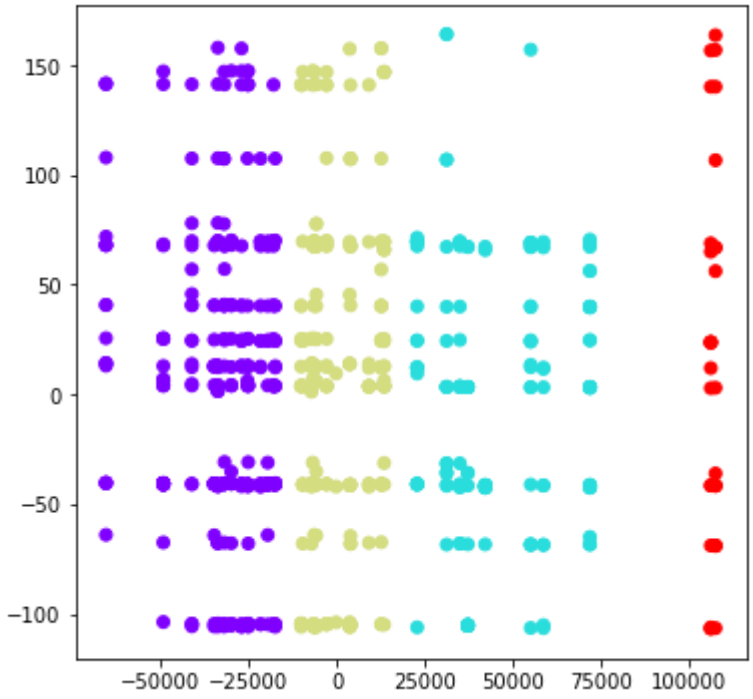
In [12]:

```
import scipy.cluster.hierarchy as shc
plt.figure(figsize =(8, 8))
plt.title('Dendrogram')
plt.ylabel('Euclidean distances')
Dendrogram = shc.dendrogram((shc.linkage(X_principal, method ='single')))
plt.show()
```



In [15]:

```
ac2 = AgglomerativeClustering(n_clusters = 4, affinity = 'euclidean', linkage = 'ward')
plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],c = ac2.fit_predict(X_principal), cmap ='r
plt.show()
```



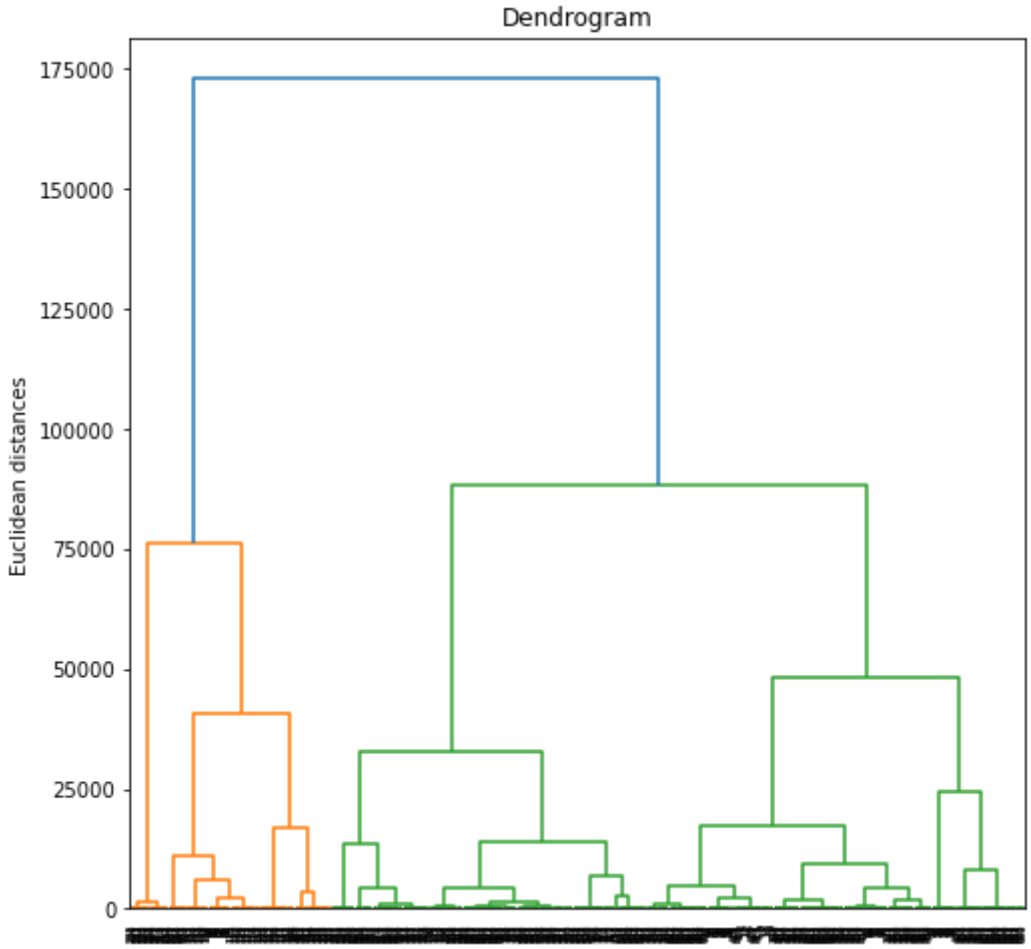
2) Complete Linkage

In [16]:

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X_principal = pca.fit_transform(X)
X_principal = pd.DataFrame(X_principal)
X_principal.columns = ['P1', 'P2']
```

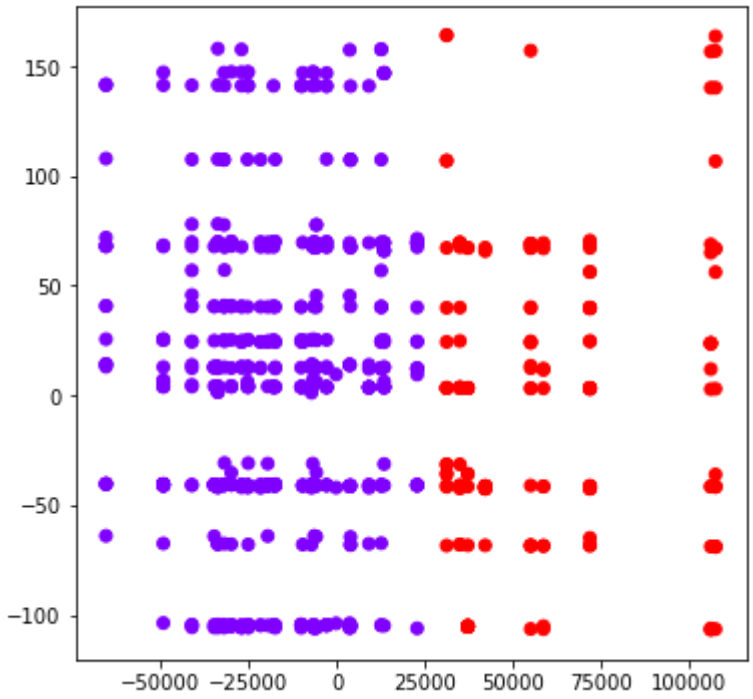
In [17]:

```
import scipy.cluster.hierarchy as shc
plt.figure(figsize =(8, 8))
plt.title('Dendrogram')
plt.ylabel('Euclidean distances')
Dendrogram = shc.dendrogram((shc.linkage(X_principal, method ='complete')))
plt.show()
```



In [18]:

```
ac2 = AgglomerativeClustering(n_clusters = 2, affinity = 'euclidean', linkage = 'complete')
plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],c = ac2.fit_predict(X_principal), cmap ='r
plt.show()
```



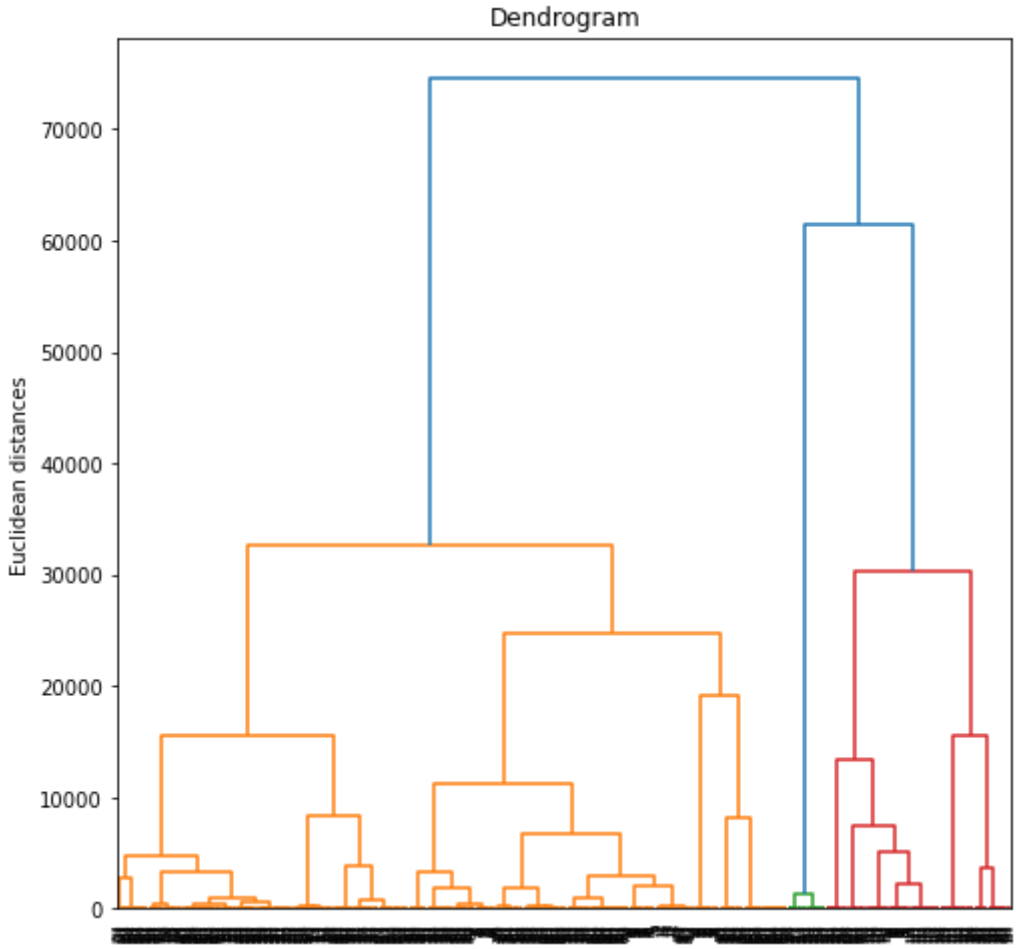
3) Average Linkage

In [19]:

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X_principal = pca.fit_transform(X)
X_principal = pd.DataFrame(X_principal)
X_principal.columns = ['P1', 'P2']
```

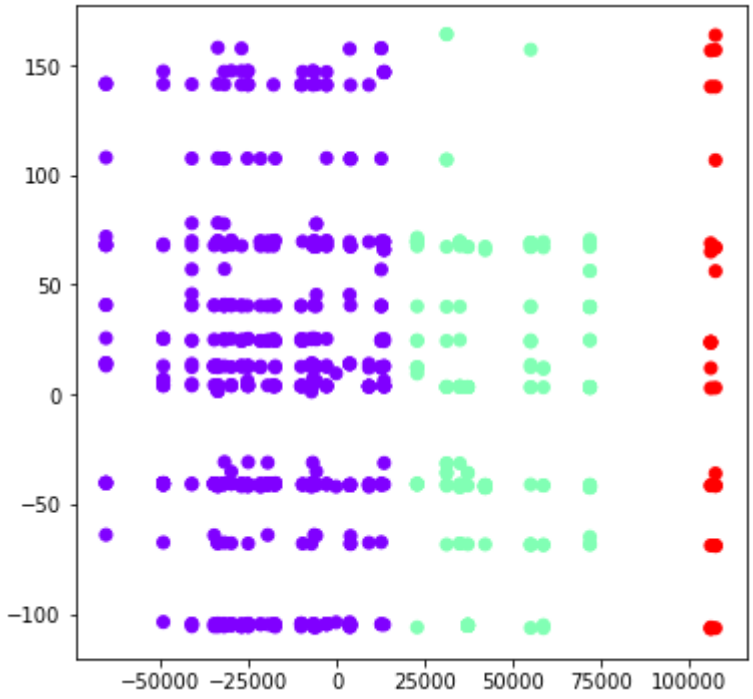
In [20]:

```
import scipy.cluster.hierarchy as shc
plt.figure(figsize =(8, 8))
plt.title('Dendrogram')
plt.ylabel('Euclidean distances')
Dendrogram = shc.dendrogram((shc.linkage(X_principal, method ='average')))
plt.show()
```



In [21]:

```
ac2 = AgglomerativeClustering(n_clusters = 3, affinity = 'euclidean', linkage = 'average')
plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],c = ac2.fit_predict(X_principal), cmap ='r
plt.show()
```



Q2

In [95]:

```
import itertools
def generateC1(dataSet):
    productDict = {}
    returneSet = []
    for data in dataSet:
        for product in data:
            if product not in productDict:
                productDict[product] = 11
            else:
                productDict[product] = productDict[product] +1

    for key in productDict:
        tempArray = []
        tempArray.append(key)
        returneSet.append(tempArray)
        returneSet.append(productDict[key])
        tempArray = []
    return returneSet
```

In [96]:

```
def generateFrequentItemSet(CandidateList, noOfTransactions,minimumSupport, dataSet, father
    frequentItemsArray = []
    for i in range(len(CandidateList)):
        if i%2 != 0:
            support = (CandidateList[i] * 1.0 /noOfTransactions) * 100
            if support >= minimumSupport:
                frequentItemsArray.append(CandidateList[i-1])
                frequentItemsArray.append(CandidateList[i])
            else:
                eleminatedItemsArray.append(CandidateList[i-1])
    for k in frequentItemsArray:
        fatherFrequentArray.append(k)
    if len(frequentItemsArray) == 2 or len(frequentItemsArray)== 0:
         #print("This will be returned")
        returnArray = fatherFrequentArray
        return returnArray
    else:
        generateCandidateSets(dataSet, eleminatedItemsArray,frequentItemsArray, noOfTransac
```

In [97]:

```
def generateCandidateSets(dataSet, eleminatedItemsArray,frequentItemsArray, noOfTransaction
    onlyElements = []
    arrayAfterCombinations = []
    candidateSetArray = []
    for i in range(len(frequentItemsArray)):
        if i%2 == 0:
            onlyElements.append(frequentItemsArray[i])
    for item in onlyElements:
        tempCombinationArray = []
        k = onlyElements.index(item)
        for i in range(k + 1, len(onlyElements)):
            for j in item:
                if j not in tempCombinationArray:
                    tempCombinationArray.append(j)
            for m in onlyElements[i]:
                if m not in tempCombinationArray:
                    tempCombinationArray.append(m)

    tempCombinationArray = []
    arrayAfterCombinations.append(tempCombinationArray)
    sortedCombinationArray = []
    uniqueCombinationArray = []
    for i in arrayAfterCombinations:
        sortedCombinationArray.append(sorted(i))
    for i in sortedCombinationArray:
        if i not in uniqueCombinationArray:
            uniqueCombinationArray.append(i)

    arrayAfterCombinations = uniqueCombinationArray
    for item in arrayAfterCombinations:
        count = 0
        for transaction in dataSet:
            if set(item).issubset(set(transaction)):
                count = count + 1
        if count != 0:
            candidateSetArray.append(item)
            candidateSetArray.append(count)
    generateFrequentItemSet(candidateSetArray,noOfTransactions, minimumSupport, dataSet,fat
```

In [98]:

```
def generateAssociationRule(freqSet):
    associationRule = []
    for item in freqSet:
        if isinstance(item, list):
            if len(item) != 0:
                length = len(item) - 1
                while length > 0:
                    combinations =list(itertools.combinations(item, length))
                    temp = []
                    LHS = []
                    for RHS in combinations:
                        LHS = set(item) - set(RHS)
                        temp.append(list(LHS))
                        temp.append(list(RHS))
                        #print(temp)
                        associationRule.append(temp)
                        temp = []
                        length = length - 1
    return associationRule
```

In [99]:

```
def aprioriOutput(rules, dataSet, minimumSupport,minimumConfidence):
    returnAprioriOutput = []
    for rule in rules:
        supportOfX = 0
        supportOfXinPercentage = 0
        supportOfXandY = 0
        supportOfXandYinPercentage = 0
        for transaction in dataSet:
            if set(rule[0]).issubset(set(transaction)):
                supportOfX = supportOfX + 1
            if set(rule[0] +rule[1]).issubset(set(transaction)):
                supportOfXandY = supportOfXandY + 1
        supportOfXinPercentage = (supportOfX * 1.0 /noOfTransactions) * 100
        supportOfXandYinPercentage = (supportOfXandY * 1.0 /noOfTransactions) * 100
        confidence = (supportOfXandYinPercentage /supportOfXinPercentage) * 100
        if confidence >= minimumConfidence:
            supportOfXAppendString = "Support Of X: " +str(round(supportOfXinPercentage, 2)
            supportOfXandYAppendString = "Support of X & Y: "+ str(round(supportOfXandYinPe
            confidenceAppendString = "Confidence: " + str(round(confidence))
            returnAprioriOutput.append(supportOfXAppendString)

            returnAprioriOutput.append(supportOfXandYAppendString)
            returnAprioriOutput.append(confidenceAppendString)
            returnAprioriOutput.append(rule)
    return returnAprioriOutput
```

In [106]:

```
fileName = "weblog1.txt"
with open(fileName,'r') as fp:
    lines = fp.readlines()
for line in lines:
    line = line.rstrip()
    dataSet.append(line.split(","))
```

In [107]:

```
dataSet
```

Out[107]:

```
[['IP', 'Time', 'URL', 'Staus'],
 ['10.128.2.1', '[29/Nov/2017:06:58:55', 'GET /login.php HTTP/1.1', '20
0'],
 ['10.128.2.1', '[29/Nov/2017:06:59:02', 'POST /process.php HTTP/1.1', '30
2'],
 ['10.128.2.1', '[29/Nov/2017:06:59:03', 'GET /home.php HTTP/1.1', '200'],
 ['10.131.2.1',
  '[29/Nov/2017:06:59:04',
  'GET /js/vendor/moment.min.js HTTP/1.1',
  '200'],
 ['10.130.2.1',
  '[29/Nov/2017:06:59:06',
  'GET /bootstrap-3.3.7/js/bootstrap.js HTTP/1.1',
  '200'],
 ['10.130.2.1',
  '[29/Nov/2017:06:59:19',
  'GET /profile.php?user=bala HTTP/1.1',
  '200'],
```

In [109]:

```
minimumSupport = input('Enter minimum Support: ')
minimumConfidence = input('Enter minimum Confidence: ')
print("\n")
minimumSupport = int(minimumSupport)
minimumConfidence = int(minimumConfidence)
nonFrequentSets = []
allFrequentItemSets = []
tempFrequentItemSets = []
dataSet = []
eleminatedItemsArray = []
noOfTransactions = 0
fatherFrequentArray = []
something = 0

noOfTransactions = len(dataSet)
firstCandidateSet = generateC1(dataSet)
frequentItemSet = generateFrequentItemSet(firstCandidateSet,noOfTransactions, minimumSuppor
associationRules =generateAssociationRule(fatherFrequentArray)
AprioriOutput = aprioriOutput(associationRules, dataSet,minimumSupport, minimumConfidence)
counter = 1
if len(AprioriOutput) == 0:
    print("There are no association rules for this support and confidence.")
else:
    for i in AprioriOutput:
        if counter == 4:
            print(str(i[0]) + "------>" + str(i[1]))
            counter = 0
        else:
            print(i, end=' ')
        counter = counter + 1
```

```
Enter minimum Support: 10
Enter minimum Confidence: 22


Support of X: 70.78 Support of X & Y: 18 Confidence: 26 ['200']----->['10.12
8.2.1']
Support of X: 26.59 Support of X & Y: 18 Confidence: 69 ['10.128.2.1']----->
['200']
Support of X: 20.52 Support of X & Y: 20 Confidence: 100 ['GET /login.php HT
TP/1.1']----->['200']
Support of X: 70.78 Support of X & Y: 20 Confidence: 29 ['200']----->['GET /
login.php HTTP/1.1']
Support of X: 70.78 Support of X & Y: 18 Confidence: 26 ['200']----->['10.12
8.2.1']
Support of X: 25.34 Support of X & Y: 18 Confidence: 71 ['10.130.2.1']----->
['200']
Support of X: 70.78 Support of X & Y: 18 Confidence: 26 ['200']----->['10.13
0.2.1']
Support of X: 26.23 Support of X & Y: 18 Confidence: 71 ['10.130.2.1']----->
['200']
Support of X: 16.49 Support of X & Y: 13 Confidence: 82 ['GET /login.php HTT
P/1.1']----->['302']
Support of X: 21.85 Support of X & Y: 13 Confidence: 62 ['302']----->['GET /
login.php HTTP/1.1']
```

In [ ]:

```
```

Q3

In [110]:

```
data1=pd.read_csv("covid_19_india.csv")
```

In [111]:

```
data1.head()
```

Out[111]:

	Sno	Date	Time	State/UnionTerritory	ConfirmedIndianNational	ConfirmedForeignNational
0	1	30/01/20	6:00 PM	Kerala	1	0
1	2	31/01/20	6:00 PM	Kerala	1	0
2	3	01/02/20	6:00 PM	Kerala	2	0
3	4	02/02/20	6:00 PM	Kerala	3	0
4	5	03/02/20	6:00 PM	Kerala	3	0

In [112]:

```
activecases=data1["Confirmed"]-data1["Cured"]
```

In [113]:

```
data1=data1.assign(active_cases=activecases.values)
data1
```

Out[113]:

	Sno	Date	Time	State/UnionTerritory	ConfirmedIndianNational	ConfirmedForeignNatic
0	1	30/01/20	6:00 PM	Kerala	1	
1	2	31/01/20	6:00 PM	Kerala	1	
2	3	01/02/20	6:00 PM	Kerala	2	
3	4	02/02/20	6:00 PM	Kerala	3	
4	5	03/02/20	6:00 PM	Kerala	3	
...	...	...	...	...	...	
7081	7082	07/10/20	8:00 AM	Telengana	-	
7082	7083	07/10/20	8:00 AM	Tripura	-	
7083	7084	07/10/20	8:00 AM	Uttarakhand	-	
7084	7085	07/10/20	8:00 AM	Uttar Pradesh	-	
7085	7086	07/10/20	8:00 AM	West Bengal	-	

7086 rows × 10 columns

In [114]:

```
data1.groupby(data1["State/UnionTerritory"]).count()
```

Out[114]:

State/UnionTerritory	Sno	Date	Time	State/UnionTerritory	ConfirmedIndianNational	ConfirmedF
Andaman and Nicobar Islands	196	196	196	196	196	
Andhra Pradesh	210	210	210	210	210	
Arunachal Pradesh	188	188	188	188	188	
Assam	190	190	190	190	190	
Bihar	200	200	200	200	200	
Cases being reassigned to states	60	60	60	60	60	
Chandigarh	203	203	203	203	203	
Chhattisgarh	203	203	203	203	203	
Dadar Nagar Haveli	37	37	37	37	37	
Dadra and Nagar Haveli and Daman and Diu	118	118	118	118	118	
Daman & Diu	1	1	1	1	1	
Delhi	220	220	220	220	220	
Goa	196	196	196	196	196	
Gujarat	202	202	202	202	202	
Haryana	218	218	218	218	218	
Himachal Pradesh	201	201	201	201	201	
Jammu and Kashmir	213	213	213	213	213	
Jharkhand	190	190	190	190	190	
Karnataka	213	213	213	213	213	
Kerala	252	252	252	252	252	
Ladakh	215	215	215	215	215	
Madhya Pradesh	201	201	201	201	201	
Maharashtra	213	213	213	213	213	
Manipur	198	198	198	198	198	
Meghalaya	177	177	177	177	177	
Mizoram	197	197	197	197	197	
Nagaland	144	144	144	144	144	
Odisha	206	206	206	206	206	
Puducherry	204	204	204	204	204	
Punjab	213	213	213	213	213	

State/UnionTerritory	Sno	Date	Time	State/UnionTerritory	ConfirmedIndianNational	ConfirmedF
Rajasthan	219	219	219	219	219	
Sikkim	137	137	137	137	137	
Tamil Nadu	215	215	215	215	215	
Telangana	45	45	45	45	45	
Telangana* **	1	1	1	1	1	
Telengana	173	173	173	173	173	
Telengana* **	1	1	1	1	1	
Tripura	184	184	184	184	184	
Unassigned	3	3	3	3	3	
Uttar Pradesh	218	218	218	218	218	
Uttarakhand	207	207	207	207	207	
West Bengal	204	204	204	204	204	

In [115]:

```
data2=pd.read_csv("StatewiseTestingDetails.csv")
```

In [116]:

```
data2["Negative"]=data2["TotalSamples"]-data2["Positive"]
```

In [117]:

```
data2
```

Out[117]:

	Date	State	TotalSamples	Negative	Positive
0	2020-04-17	Andaman and Nicobar Islands	1403.0	1391.0	12.0
1	2020-04-24	Andaman and Nicobar Islands	2679.0	2652.0	27.0
2	2020-04-27	Andaman and Nicobar Islands	2848.0	2815.0	33.0
3	2020-05-01	Andaman and Nicobar Islands	3754.0	3721.0	33.0
4	2020-05-16	Andaman and Nicobar Islands	6677.0	6644.0	33.0
...	...	...	...	...	...
5961	2020-10-02	West Bengal	3314598.0	NaN	NaN
5962	2020-10-03	West Bengal	3355726.0	NaN	NaN
5963	2020-10-04	West Bengal	3397988.0	NaN	NaN
5964	2020-10-05	West Bengal	3438128.0	NaN	NaN
5965	2020-10-06	West Bengal	3480510.0	NaN	NaN

5966 rows × 5 columns

In [118]:

```
data2["Positive"].fillna(0, inplace = True)
data2["Negative"].fillna(0, inplace = True)
data2
```

Out[118]:

	Date	State	TotalSamples	Negative	Positive
0	2020-04-17	Andaman and Nicobar Islands	1403.0	1391.0	12.0
1	2020-04-24	Andaman and Nicobar Islands	2679.0	2652.0	27.0
2	2020-04-27	Andaman and Nicobar Islands	2848.0	2815.0	33.0
3	2020-05-01	Andaman and Nicobar Islands	3754.0	3721.0	33.0
4	2020-05-16	Andaman and Nicobar Islands	6677.0	6644.0	33.0
...	...	...	...	...	...
5961	2020-10-02	West Bengal	3314598.0	0.0	0.0
5962	2020-10-03	West Bengal	3355726.0	0.0	0.0
5963	2020-10-04	West Bengal	3397988.0	0.0	0.0
5964	2020-10-05	West Bengal	3438128.0	0.0	0.0
5965	2020-10-06	West Bengal	3480510.0	0.0	0.0

5966 rows × 5 columns

In [120]:

```
data2.groupby(data2["State"]).count()
```

Out[120]:

State	Date	State	TotalSamples	Negative	Positive
Andaman and Nicobar Islands	145	145	145	145	145
Andhra Pradesh	180	180	180	180	180
Arunachal Pradesh	170	170	170	170	170
Assam	161	161	161	161	161
Bihar	181	181	181	181	181
Chandigarh	180	180	180	180	180
Chhattisgarh	174	174	174	174	174
Dadra and Nagar Haveli and Daman and Diu	150	150	150	150	150
Delhi	181	181	181	181	181
Goa	175	175	175	175	175
Gujarat	181	181	181	181	181
Haryana	184	184	184	184	184
Himachal Pradesh	180	180	180	180	180
Jammu and Kashmir	181	181	181	181	181
Jharkhand	177	177	177	177	177
Karnataka	183	183	183	183	183
Kerala	189	189	189	189	189
Ladakh	121	121	121	121	121
Madhya Pradesh	184	184	184	184	184
Maharashtra	183	183	183	183	183
Manipur	127	127	127	127	127
Meghalaya	142	142	142	142	142
Mizoram	175	175	175	175	175
Nagaland	178	178	178	178	178
Odisha	184	184	184	184	184
Puducherry	171	171	171	171	171
Punjab	183	183	183	183	183
Rajasthan	184	184	184	184	184
Sikkim	150	150	150	150	150
Tamil Nadu	183	183	183	183	183
Telangana	116	116	116	116	116
Tripura	164	164	164	164	164
Uttar Pradesh	181	181	181	181	181

State	Date	State	TotalSamples	Negative	Positive
Uttarakhand	183	183	183	183	183
West Bengal	185	185	185	185	185

Agglomerative Hierarchical Clustering

1) Number of Active Cases

In [146]:

```
X=data1.iloc[:,[8,9]]
X
```

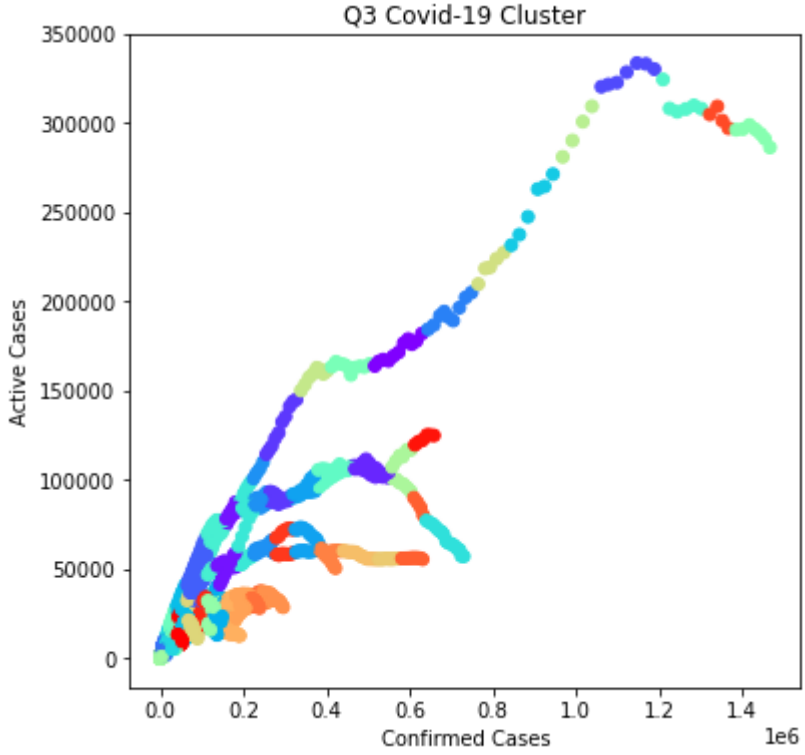
Out[146]:

	Confirmed	active_cases
0	1	1
1	1	1
2	2	2
3	3	3
4	3	3
...	...	...
7081	204748	27740
7082	27545	4922
7083	52329	9091
7084	420937	50184
7085	277049	33306

7086 rows × 2 columns

In [147]:

```
#42 clusters as there are total 42 states+UT's present in dataset
from sklearn.cluster import AgglomerativeClustering
ac2 = AgglomerativeClustering(n_clusters = 42, affinity = 'euclidean')
plt.figure(figsize =(6, 6))
plt.scatter(X["Confirmed"],X['active_cases'],c = ac2.fit_predict(X), cmap ='rainbow')
plt.title('Q3 Covid-19 Cluster')
plt.xlabel('Confirmed Cases')
plt.ylabel('Active Cases')
plt.show()
```



2) Death Rate

In [144]:

```
X=data1.iloc[:,[7,8]]
X
```

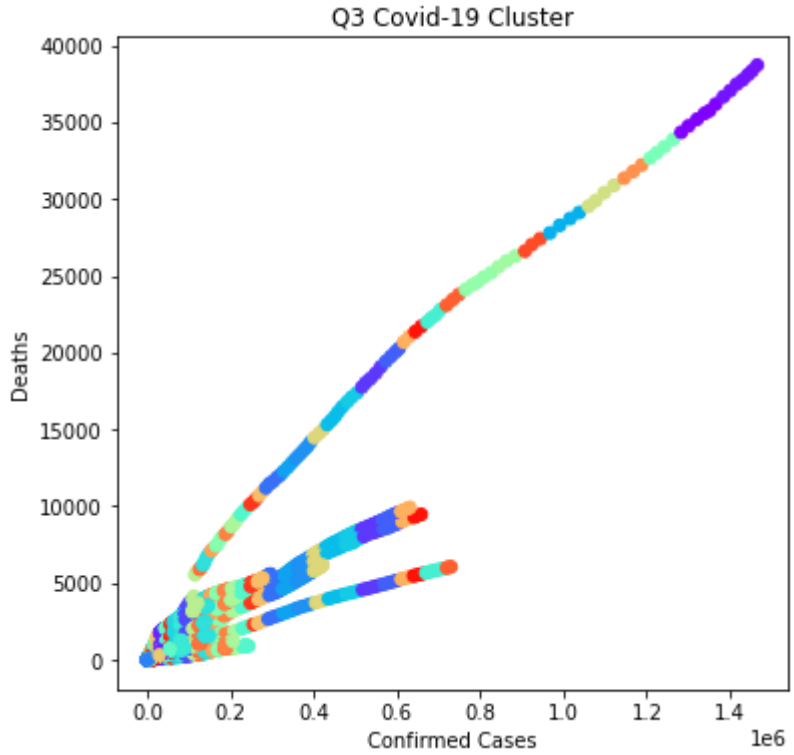
Out[144]:

	Deaths	Confirmed
0	0	1
1	0	1
2	0	2
3	0	3
4	0	3
...	...	...
7081	1189	204748
7082	301	27545
7083	677	52329
7084	6153	420937
7085	5318	277049

7086 rows × 2 columns

In [145]:

```
from sklearn.cluster import AgglomerativeClustering
ac2 = AgglomerativeClustering(n_clusters = 42, affinity = 'euclidean')
plt.figure(figsize =(6, 6))
plt.scatter(X["Confirmed"],X['Deaths'],c = ac2.fit_predict(X), cmap ='rainbow')
plt.title('Q3 Covid-19 Cluster')
plt.xlabel('Confirmed Cases')
plt.ylabel('Deaths')
plt.show()
```



3) Recovery Rate

In [142]:

```
X=data1.iloc[:,[6,8]]
X
```

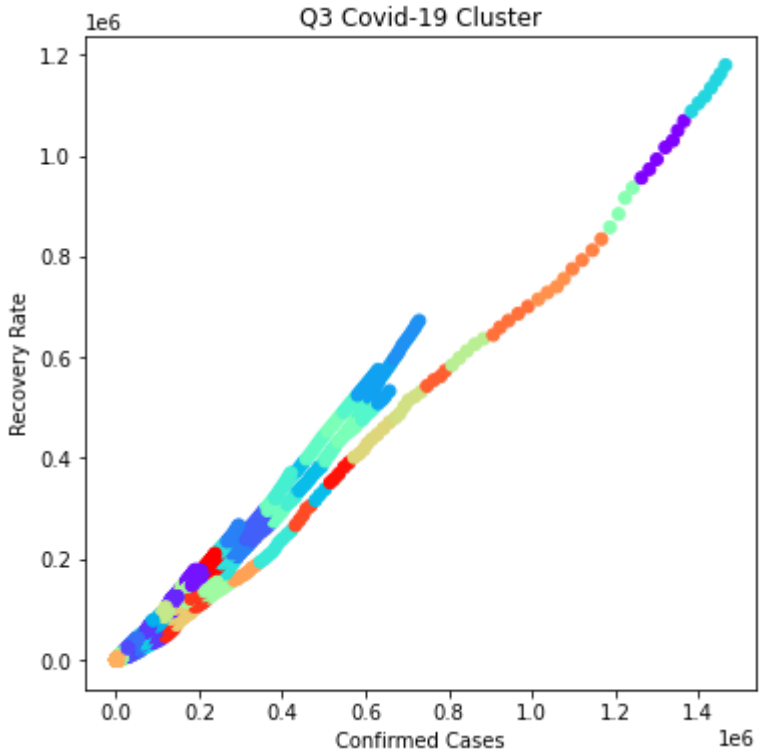
Out[142]:

	Cured	Confirmed
0	0	1
1	0	1
2	0	2
3	0	3
4	0	3
...	...	...
7081	177008	204748
7082	22623	27545
7083	43238	52329
7084	370753	420937
7085	243743	277049

7086 rows × 2 columns

In [143]:

```
from sklearn.cluster import AgglomerativeClustering
ac2 = AgglomerativeClustering(n_clusters = 42, affinity = 'euclidean')
plt.figure(figsize =(6, 6))
plt.scatter(X["Confirmed"],X['Cured'],c = ac2.fit_predict(X), cmap ='rainbow')
plt.title('Q3 Covid-19 Cluster')
plt.xlabel('Confirmed Cases')
plt.ylabel('Recovery Rate')
plt.show()
```



4) Testing-Confirm Cases Ratio

In [140]:

```
X=data2.iloc[:,[2,4]]
X
```

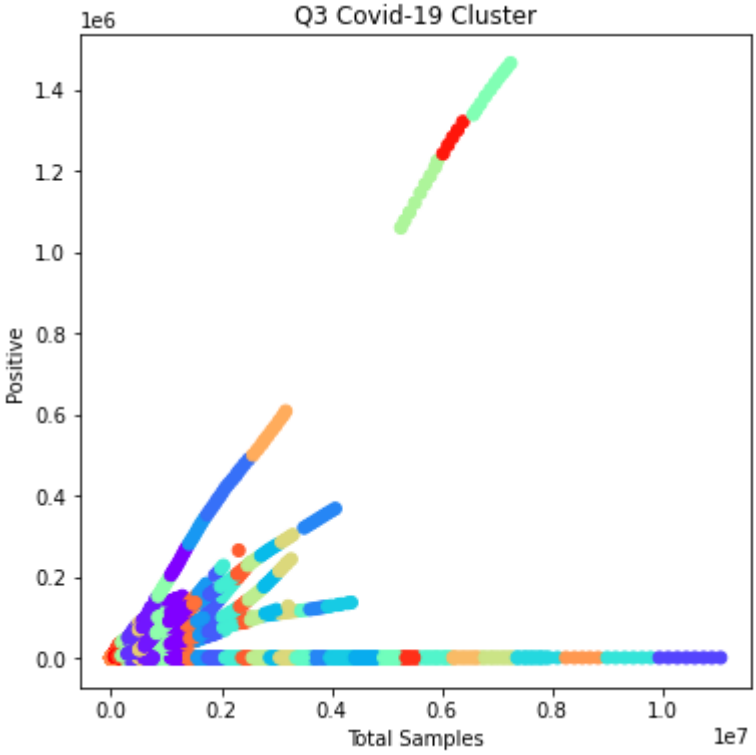
Out[140]:

	TotalSamples	Positive
0	1403.0	12.0
1	2679.0	27.0
2	2848.0	33.0
3	3754.0	33.0
4	6677.0	33.0
...	...	...
5961	3314598.0	0.0
5962	3355726.0	0.0
5963	3397988.0	0.0
5964	3438128.0	0.0
5965	3480510.0	0.0

5966 rows × 2 columns

In [141]:

```
#35 clusters as there are 35 States+UT's in the other dataset
from sklearn.cluster import AgglomerativeClustering
ac2 = AgglomerativeClustering(n_clusters = 35, affinity = 'euclidean')
plt.figure(figsize =(6,6))
plt.scatter(X["TotalSamples"],X['Positive'],c = ac2.fit_predict(X), cmap ='rainbow')
plt.title('Q3 Covid-19 Cluster')
plt.xlabel('Total Samples')
plt.ylabel('Positive')
plt.show()
```



K-Means Clustering

1) Number of Active Cases

In [138]:

```
X=data1.iloc[:,[8,9]]
X
```

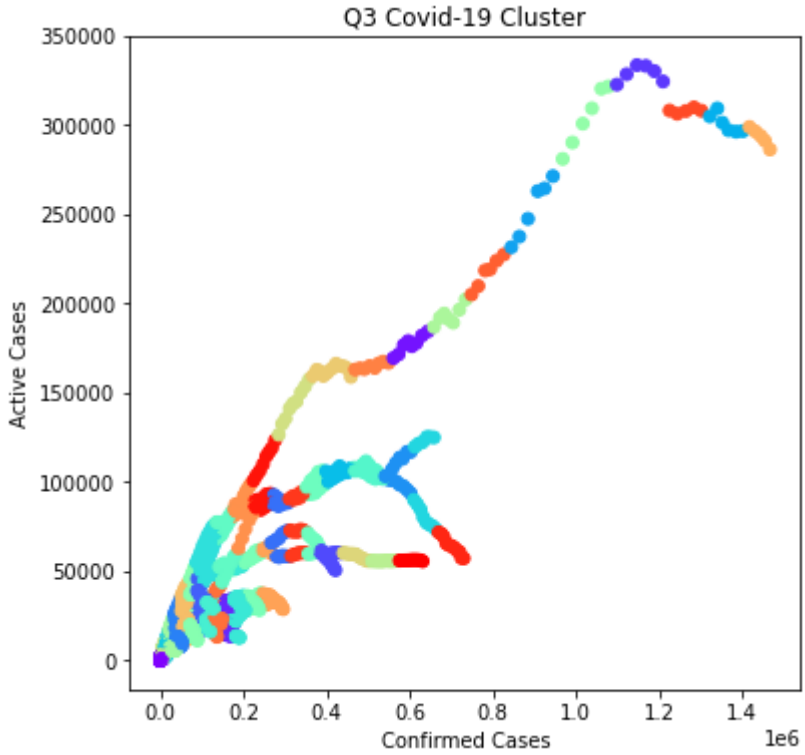
Out[138]:

	Confirmed	active_cases
0	1	1
1	1	1
2	2	2
3	3	3
4	3	3
...	...	...
7081	204748	27740
7082	27545	4922
7083	52329	9091
7084	420937	50184
7085	277049	33306

7086 rows × 2 columns

In [139]:

```
from sklearn.cluster import KMeans
y_kmeans = KMeans(n_clusters = 42, init = 'k-means++', random_state = 0)
plt.figure(figsize =(6, 6))
plt.scatter(X["Confirmed"],X['active_cases'],c=y_kmeans.fit_predict(X), cmap ='rainbow')
plt.title('Q3 Covid-19 Cluster')
plt.xlabel('Confirmed Cases')
plt.ylabel('Active Cases')
plt.show()
```



2) Death Rate

In [132]:

```
X=data1.iloc[:,[7,8]]
X
```

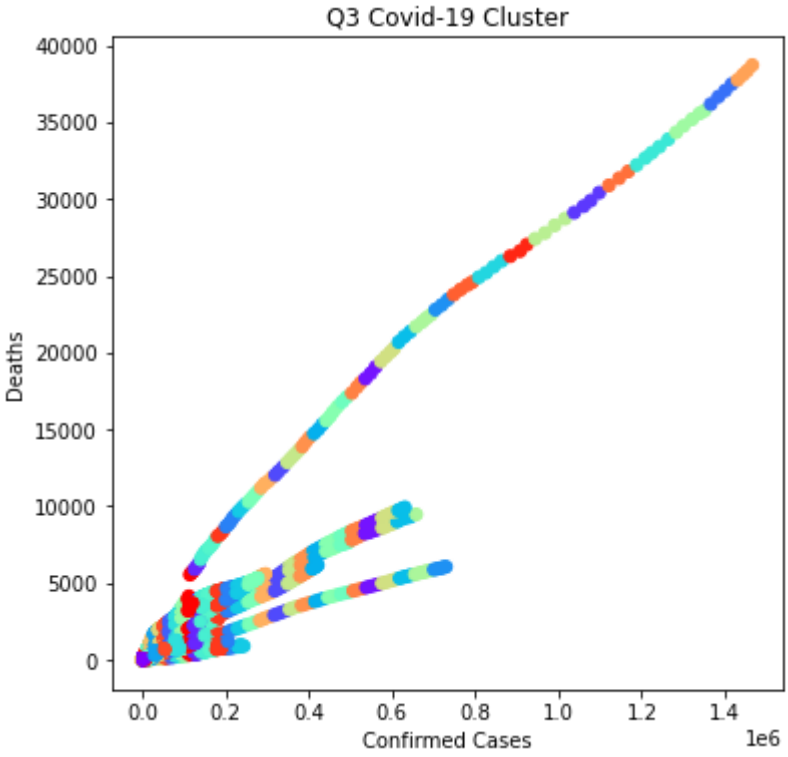
Out[132]:

	Deaths	Confirmed
0	0	1
1	0	1
2	0	2
3	0	3
4	0	3
...	...	...
7081	1189	204748
7082	301	27545
7083	677	52329
7084	6153	420937
7085	5318	277049

7086 rows × 2 columns

In [133]:

```
y_kmeans = KMeans(n_clusters = 42, init = 'k-means++', random_state = 0)
plt.figure(figsize =(6, 6))
plt.scatter(X["Confirmed"],X['Deaths'],c=y_kmeans.fit_predict(X), cmap ='rainbow')
plt.title('Q3 Covid-19 Cluster')
plt.xlabel('Confirmed Cases')
plt.ylabel('Deaths')
plt.show()
```



3) Recovery Rate

In [134]:

```
X=data1.iloc[:,[6,8]]
X
```

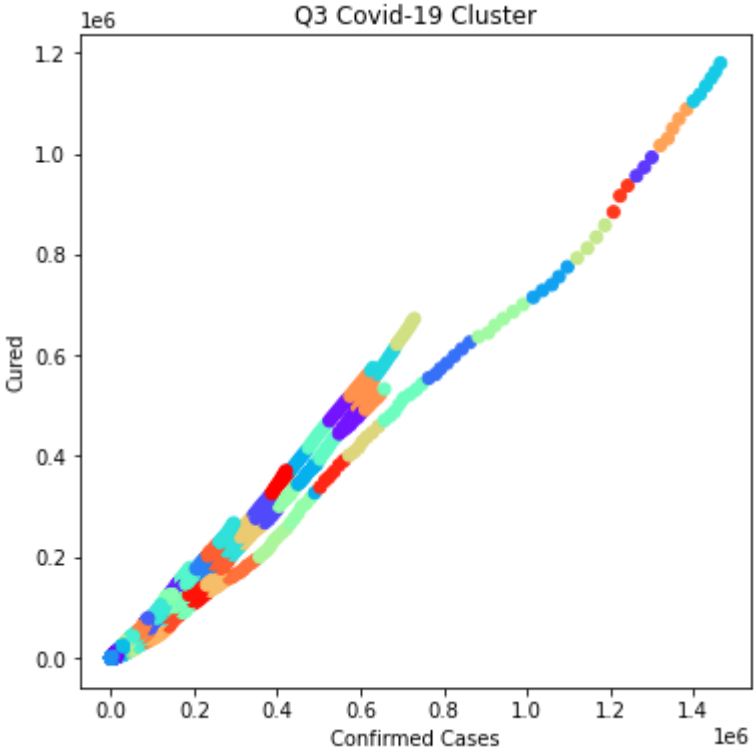
Out[134]:

	Cured	Confirmed
0	0	1
1	0	1
2	0	2
3	0	3
4	0	3
...	...	...
7081	177008	204748
7082	22623	27545
7083	43238	52329
7084	370753	420937
7085	243743	277049

7086 rows × 2 columns

In [135]:

```
y_kmeans = KMeans(n_clusters = 42, init = 'k-means++', random_state = 0)
plt.figure(figsize =(6, 6))
plt.scatter(X["Confirmed"],X['Cured'],c=y_kmeans.fit_predict(X), cmap ='rainbow')
plt.title('Q3 Covid-19 Cluster')
plt.xlabel('Confirmed Cases')
plt.ylabel('Cured')
plt.show()
```



4) Testing-Confirm Cases Ratio

In [136]:

```
X=data2.iloc[:,[2,4]]
X
```

Out[136]:

	TotalSamples	Positive
0	1403.0	12.0
1	2679.0	27.0
2	2848.0	33.0
3	3754.0	33.0
4	6677.0	33.0
...	...	...
5961	3314598.0	0.0
5962	3355726.0	0.0
5963	3397988.0	0.0
5964	3438128.0	0.0
5965	3480510.0	0.0

5966 rows × 2 columns

In [137]:

```
y_kmeans = KMeans(n_clusters = 35, init = 'k-means++', random_state = 0)
plt.figure(figsize =(6, 6))
plt.scatter(X["TotalSamples"],X['Positive'],c=y_kmeans.fit_predict(X), cmap ='rainbow')
plt.title('Q3 Covid-19 Cluster')
plt.xlabel('Total Samples')
plt.ylabel('Positive')
plt.show()
```