ASSIGNMENT 3B

1. **Consider the BCP dataset and its class variable with values “R” (Recurrence Occurred) and “N” (No Recurrence Occurred so far). Ignore the attribute that gives the number of years after which recurrence occurred or the number of years for which the patient has been free of recurrence. There are thirty other attribute values given as features measured for every patient. Use only these thirty attributes.**

**CODE**:

data = pd.read\_csv('wpbc.data.csv')

# Considering only 30 attributes

X = data.loc[:,'M\_radius':'fractal dimension']

y = data['Outcome']

X.info()

**Output**:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 194 entries, 0 to 193

Data columns (total 30 columns):

M\_radius 194 non-null float64

M\_texture 194 non-null float64

M\_perimeter 194 non-null float64

M\_area 194 non-null float64

M\_smoothness 194 non-null float64

M\_compactness 194 non-null float64

M\_concavity 194 non-null float64

M\_concave points 194 non-null float64

M\_symmetry 194 non-null float64

M\_fractal dimension 194 non-null float64

SE\_radius 194 non-null float64

SE\_texture 194 non-null float64

SE\_perimeter 194 non-null float64

SE\_area 194 non-null float64

SE\_smoothness 194 non-null float64

SE\_compactness 194 non-null float64

SE\_concavity 194 non-null float64

SE\_concave points 194 non-null float64

SE\_symmetry 194 non-null float64

SE\_fractal dimension 194 non-null float64

radius 194 non-null float64

texture 194 non-null float64

perimeter 194 non-null float64

area 194 non-null float64

smoothness 194 non-null float64

compactness 194 non-null float64

concavity 194 non-null float64

concave points 194 non-null float64

symmetry 194 non-null float64

fractal dimension 194 non-null float64

dtypes: float64(30)

memory usage: 45.5 KB

* 1. **Run k-means algorithm with this dataset for k=4. Run it three different times and for each run show the cluster centers and the SSE values for each cluster and also the total SSE value for the clustering.**

**FOR 1st RUN:**

K\_Means\_1 = KMeans(n\_clusters=4,init='random', random\_state=30).fit(X)

center = K\_Means\_1.cluster\_centers\_

indx = K\_Means\_1.labels\_

clusters\_SSE1 = []

print('Below are SSE values for each cluster of K\_Means\_1 clustering\n')

for i in range(4):

SSE = np.sum(np.sum((X[indx == i] - center[i])\*\*2))

print('SSE of cluster ' + str(i) + ' = \t' + str(SSE))

clusters\_SSE1.append(SSE)

K\_Means\_1\_SSE = K\_Means\_1.inertia\_

print('Total SSE for 1st run \t' + str(K\_Means\_1\_SSE))

print('\n Cluster centers for 1st run:\n' + str(K\_Means\_1.cluster\_centers\_))

**OUTPUT:**

Below are SSE values for each cluster of K\_Means\_1 clustering

SSE of cluster 0 = 2795136.257930213

SSE of cluster 1 = 3744386.7368627535

SSE of cluster 2 = 2462018.7541844267

SSE of cluster 3 = 2567115.4151013456

Total SSE for 1st run 11568657.164078739

Cluster centers for 1st run:

[[2.35563636e+01 2.46300000e+01 1.56118182e+02 1.74590909e+03

1.01784545e-01 1.58554545e-01 2.21609091e-01 1.34490909e-01

1.84372727e-01 5.93345455e-02 1.12993636e+00 1.27262727e+00

7.74863636e+00 1.76058182e+02 5.47018182e-03 2.63345455e-02

3.70290909e-02 1.49672727e-02 1.78472727e-02 3.35472727e-03

3.09372727e+01 3.29990909e+01 2.06809091e+02 2.95145455e+03

1.38245455e-01 3.48081818e-01 4.66627273e-01 2.26745455e-01

2.95609091e-01 8.28236364e-02 2.77272727e+00 1.90909091e+00]

[2.04033333e+01 2.25075000e+01 1.35312500e+02 1.29415833e+03

1.01379375e-01 1.58957500e-01 1.95429375e-01 1.09532292e-01

1.96485417e-01 6.07245833e-02 7.78566667e-01 1.23975417e+00

5.54252083e+00 1.00741042e+02 6.68050000e-03 3.41447917e-02

4.61258333e-02 1.63574583e-02 2.16468750e-02 3.97083333e-03

2.49175000e+01 2.94935417e+01 1.67431250e+02 1.90250000e+03

1.39456458e-01 3.80918750e-01 4.78495833e-01 2.01351667e-01

3.24829167e-01 8.63220833e-02 3.46666667e+00 3.97916667e+00]

[1.41684507e+01 2.16912676e+01 9.32536620e+01 6.25071831e+02

1.05981127e-01 1.39403803e-01 1.31486479e-01 6.66121127e-02

1.94956338e-01 6.65381690e-02 4.01209859e-01 1.22968028e+00

2.89861972e+00 3.54732394e+01 6.96430986e-03 3.17556620e-02

3.89342254e-02 1.39222394e-02 2.09582958e-02 4.15969014e-03

1.67556338e+01 3.02914085e+01 1.12257183e+02 8.63574648e+02

1.53298169e-01 3.99116338e-01 4.44295493e-01 1.64238732e-01

3.39373239e-01 1.01006056e-01 2.32957746e+00 3.01408451e+00]

[1.76814062e+01 2.24221875e+01 1.16160938e+02 9.73425000e+02

1.00433281e-01 1.31261406e-01 1.43283125e-01 8.39731250e-02

1.89348437e-01 6.06314062e-02 6.07687500e-01 1.34821562e+00

4.20525000e+00 6.79087500e+01 6.92489063e-03 2.95979375e-02

3.99081250e-02 1.56422187e-02 1.99180156e-02 3.96907812e-03

2.10351562e+01 3.00875000e+01 1.39135938e+02 1.35689062e+03

1.37841094e-01 3.16809375e-01 3.89692344e-01 1.68736250e-01

3.05900000e-01 8.41370312e-02 3.03125000e+00 3.07812500e+00]]

**FOR 2ND RUN:**

K\_Means\_2 = KMeans(n\_clusters=4,init='random', random\_state=10).fit(X)

center2 = K\_Means\_2.cluster\_centers\_

indx2 = K\_Means\_2.labels\_

clusters\_SSE2 = []

print('Below are SSE values for each cluster of K\_Means\_2 clustering\n')

for i in range(4):

SSE2 = np.sum(np.sum((X[indx2 ==i] - center2[i])\*\*2))

print('SSE of cluster ' + str(i) + ' = \t' + str(SSE2))

clusters\_SSE2.append(SSE2)

K\_Means\_2\_SSE = K\_Means\_2.inertia\_

print('Total SSE for 1st run \t' + str(K\_Means\_2\_SSE))

print('\n Cluster centers for 2nd run:\n' + str(K\_Means\_2.cluster\_centers\_))

**OUTPUT:**

Below are SSE values for each cluster of K\_Means\_2 clustering

SSE of cluster 0 = 2795136.257930213

SSE of cluster 1 = 3852709.6162770456

SSE of cluster 2 = 2369170.0050529144

SSE of cluster 3 = 2555245.9802285926

Total SSE for 1st run 11572261.859488767

Cluster centers for 2nd run:

[[2.35563636e+01 2.46300000e+01 1.56118182e+02 1.74590909e+03

1.01784545e-01 1.58554545e-01 2.21609091e-01 1.34490909e-01

1.84372727e-01 5.93345455e-02 1.12993636e+00 1.27262727e+00

7.74863636e+00 1.76058182e+02 5.47018182e-03 2.63345455e-02

3.70290909e-02 1.49672727e-02 1.78472727e-02 3.35472727e-03

3.09372727e+01 3.29990909e+01 2.06809091e+02 2.95145455e+03

1.38245455e-01 3.48081818e-01 4.66627273e-01 2.26745455e-01

2.95609091e-01 8.28236364e-02 2.77272727e+00 1.90909091e+00]

[2.03908163e+01 2.25608163e+01 1.35212245e+02 1.29207347e+03

1.01381837e-01 1.58956327e-01 1.96634898e-01 1.09641837e-01

1.96969388e-01 6.07328571e-02 7.72785714e-01 1.23892245e+00

5.48583673e+00 9.99775510e+01 6.64687755e-03 3.40967347e-02

4.61548980e-02 1.62364898e-02 2.15271429e-02 3.95559184e-03

2.48708163e+01 2.95769388e+01 1.67048980e+02 1.89610204e+03

1.39212449e-01 3.81024490e-01 4.80308163e-01 2.00777143e-01

3.24944898e-01 8.62879592e-02 3.45714286e+00 4.12244898e+00]

[1.41374286e+01 2.17038571e+01 9.30230000e+01 6.22147143e+02

1.06022286e-01 1.39125286e-01 1.30837714e-01 6.63564286e-02

1.94911429e-01 6.65440000e-02 4.02715714e-01 1.22926143e+00

2.90748571e+00 3.55652857e+01 6.97068571e-03 3.18521714e-02

3.89065714e-02 1.39291286e-02 2.10009857e-02 4.16731429e-03

1.67252857e+01 3.02677143e+01 1.12019429e+02 8.60411429e+02

1.53283857e-01 3.99880857e-01 4.42102571e-01 1.63952143e-01

3.39101429e-01 1.01024000e-01 2.34857143e+00 3.04285714e+00]

[1.76275000e+01 2.23548438e+01 1.15832813e+02 9.67765625e+02

1.00458281e-01 1.31261406e-01 1.42070625e-01 8.34982813e-02

1.89003125e-01 6.07095312e-02 6.04570313e-01 1.34915312e+00

4.19764063e+00 6.73728125e+01 6.94809375e-03 2.94918437e-02

3.98037500e-02 1.56892500e-02 1.99522344e-02 3.97535938e-03

2.09765625e+01 3.00620313e+01 1.38826562e+02 1.34901563e+03

1.38259844e-01 3.16176562e-01 3.90168906e-01 1.68909688e-01

3.06335937e-01 8.43729687e-02 3.00000000e+00 2.92187500e+00]]

**FOR 3rd RUN:**

K\_Means\_3 = KMeans(n\_clusters=4,init='random', random\_state=20).fit(X)

center = K\_Means\_3.cluster\_centers\_

indx = K\_Means\_3.labels\_

clusters\_SSE3 = []

print('Below are SSE values for each cluster of K\_Means\_3 clustering\n')

for i in range(4):

SSE = np.sum(np.sum((X[indx ==i] - center[i])\*\*2))

print('SSE of cluster ' + str(i) + ' = \t' + str(SSE))

clusters\_SSE3.append(SSE)

K\_Means\_3\_SSE = K\_Means\_3.inertia\_

print('Total SSE for 1st run \t' + str(K\_Means\_3\_SSE))

print('\n Cluster centers for 2nd run:\n' + str(K\_Means\_3.cluster\_centers\_))

**OUTPUT:**

Below are SSE values for each cluster of K\_Means\_3 clustering

SSE of cluster 0 = 2462018.7541844267

SSE of cluster 1 = 3642638.9674575753

SSE of cluster 2 = 2672064.0493349275

SSE of cluster 3 = 2795136.257930213

Total SSE for 1st run 11571858.028907144

Cluster centers for 2nd run:

[[1.41684507e+01 2.16912676e+01 9.32536620e+01 6.25071831e+02

1.05981127e-01 1.39403803e-01 1.31486479e-01 6.66121127e-02

1.94956338e-01 6.65381690e-02 4.01209859e-01 1.22968028e+00

2.89861972e+00 3.54732394e+01 6.96430986e-03 3.17556620e-02

3.89342254e-02 1.39222394e-02 2.09582958e-02 4.15969014e-03

1.67556338e+01 3.02914085e+01 1.12257183e+02 8.63574648e+02

1.53298169e-01 3.99116338e-01 4.44295493e-01 1.64238732e-01

3.39373239e-01 1.01006056e-01 2.32957746e+00 3.01408451e+00]

[2.04317021e+01 2.24585106e+01 1.35461702e+02 1.29820426e+03

1.01604255e-01 1.57680000e-01 1.95104468e-01 1.09743404e-01

1.95751064e-01 6.06670213e-02 7.74257447e-01 1.23068511e+00

5.47257447e+00 1.00652553e+02 6.68331915e-03 3.27308511e-02

4.50385106e-02 1.61442128e-02 2.09727660e-02 3.89263830e-03

2.49351064e+01 2.94153191e+01 1.67219149e+02 1.90785106e+03

1.39770426e-01 3.73185106e-01 4.73268085e-01 2.00331489e-01

3.21804255e-01 8.59502128e-02 3.49148936e+00 4.06382979e+00]

[1.77027692e+01 2.24589231e+01 1.16347692e+02 9.75433846e+02

1.00285231e-01 1.32611231e-01 1.44320308e-01 8.42136923e-02

1.89989231e-01 6.06744615e-02 6.13432308e-01 1.35310462e+00

4.27640000e+00 6.84778462e+01 6.91909231e-03 3.06902769e-02

4.07900000e-02 1.58074154e-02 2.04320462e-02 4.02564615e-03

2.10821538e+01 3.01349231e+01 1.39724615e+02 1.36141538e+03

1.37638923e-01 3.23387692e-01 3.94838615e-01 1.69975692e-01

3.08378462e-01 8.44395385e-02 3.02000000e+00 3.03076923e+00]

[2.35563636e+01 2.46300000e+01 1.56118182e+02 1.74590909e+03

1.01784545e-01 1.58554545e-01 2.21609091e-01 1.34490909e-01

1.84372727e-01 5.93345455e-02 1.12993636e+00 1.27262727e+00

7.74863636e+00 1.76058182e+02 5.47018182e-03 2.63345455e-02

3.70290909e-02 1.49672727e-02 1.78472727e-02 3.35472727e-03

3.09372727e+01 3.29990909e+01 2.06809091e+02 2.95145455e+03

1.38245455e-01 3.48081818e-01 4.66627273e-01 2.26745455e-01

2.95609091e-01 8.28236364e-02 2.77272727e+00 1.90909091e+00]]

* 1. **Select the best of the above three clustering’s and explain how you chose the best candidate.**

The best clustering is K\_Means\_1, as the Total SSE for 1st run is 11568657.164078, which is lowest compared to 2nd and 3rd run, which are 11572261.859488767, 11571858.028907144.

* 1. **For the best candidate clustering chosen by you plot the Silhouette coefficient for the clustering. Compute and report the average Silhouette coefficient for each cluster of the chosen clustering**.

Best Clustering candidate = K\_Means\_1

**CODE**:

AvgSilhouetteScoreOfCluster = silhouette\_score(X, K\_Means\_1.labels\_)

print('Mean Silhouette Coefficient of all samples:='+ str(AvgSilhouetteScoreOfCluster))

**OUTPUT**:

mean Silhouette Coefficient of all samples:=0.5008996425644388

**CODE**:

silhouettePerSample = silhouette\_samples(X, K\_Means\_1.labels\_)

%matplotlib inline

plt.scatter(range(len(X)),silhouettePerSample, c=K\_Means\_1.labels\_)

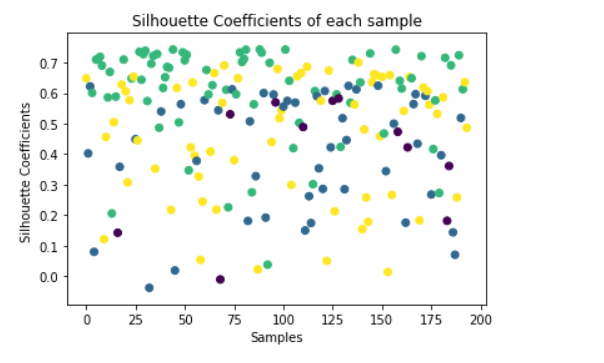
plt.figsize = (35,25)

plt.xlabel("Samples")

plt.ylabel("Silhouette Coefficients ")

plt.title("Silhouette Coefficients of each sample")

**OUTPUT**:



* **Silhouette Coefficient for each cluster**

**CODE**:

for i in range(4):

cluster\_samples = silhouettePerSample[K\_Means\_1.labels\_ == i]

avg\_cluster\_silhouette = np.sum(cluster\_samples, axis=0)/len(cluster\_samples)

print("Silhoutte Coefficient for cluster "+str(i)+" = "+str(avg\_cluster\_silhouette))

**OUTPUT**:

Silhoutte Coefficient for cluster 0 = 0.39235086742978814

Silhoutte Coefficient for cluster 1 = 0.41175448713040125

Silhoutte Coefficient for cluster 2 = 0.6095792912961995

Silhoutte Coefficient for cluster 3 = 0.4658488445544381

* 1. **Now consider the class label for each data point in each cluster (“R” or “N”). To each cluster assign the label that belongs to most of the data points in that cluster. Report the cluster center, its SSE, and its class label, and the fraction of points that have the class label.**

**CODE:**

Class = []

sse = clusters\_SSE1

for i in range(4):

print('\n\nFor Cluster ' + str(i+1))

print('Cluster center: \n' + str(K\_Means\_1.cluster\_centers\_[i]))

print('SSE: \t' + str(sse[i]))

print('CLASS LABEL:= ' + str(labels.idxmax()))

labels = (y[K\_Means\_1.labels\_ == i]).value\_counts()

Class.append(labels.idxmax())

fraction = labels.max()/(labels.max()+labels.min())

print('Fraction of points that have the class label: \t'+ str(fraction))

**OUTPUT**:

For Cluster 1

Cluster center:

[2.35563636e+01 2.46300000e+01 1.56118182e+02 1.74590909e+03

1.01784545e-01 1.58554545e-01 2.21609091e-01 1.34490909e-01

1.84372727e-01 5.93345455e-02 1.12993636e+00 1.27262727e+00

7.74863636e+00 1.76058182e+02 5.47018182e-03 2.63345455e-02

3.70290909e-02 1.49672727e-02 1.78472727e-02 3.35472727e-03

3.09372727e+01 3.29990909e+01 2.06809091e+02 2.95145455e+03

1.38245455e-01 3.48081818e-01 4.66627273e-01 2.26745455e-01

2.95609091e-01 8.28236364e-02 2.77272727e+00 1.90909091e+00]

SSE: 2795136.257930213

CLASS LABEL:= N

Fraction of points that have the class label: 0.5454545454545454

For Cluster 2

Cluster center:

[2.04033333e+01 2.25075000e+01 1.35312500e+02 1.29415833e+03

1.01379375e-01 1.58957500e-01 1.95429375e-01 1.09532292e-01

1.96485417e-01 6.07245833e-02 7.78566667e-01 1.23975417e+00

5.54252083e+00 1.00741042e+02 6.68050000e-03 3.41447917e-02

4.61258333e-02 1.63574583e-02 2.16468750e-02 3.97083333e-03

2.49175000e+01 2.94935417e+01 1.67431250e+02 1.90250000e+03

1.39456458e-01 3.80918750e-01 4.78495833e-01 2.01351667e-01

3.24829167e-01 8.63220833e-02 3.46666667e+00 3.97916667e+00]

SSE: 3744386.7368627535

CLASS LABEL:= N

Fraction of points that have the class label: 0.7291666666666666

For Cluster 3

Cluster center:

[1.41684507e+01 2.16912676e+01 9.32536620e+01 6.25071831e+02

1.05981127e-01 1.39403803e-01 1.31486479e-01 6.66121127e-02

1.94956338e-01 6.65381690e-02 4.01209859e-01 1.22968028e+00

2.89861972e+00 3.54732394e+01 6.96430986e-03 3.17556620e-02

3.89342254e-02 1.39222394e-02 2.09582958e-02 4.15969014e-03

1.67556338e+01 3.02914085e+01 1.12257183e+02 8.63574648e+02

1.53298169e-01 3.99116338e-01 4.44295493e-01 1.64238732e-01

3.39373239e-01 1.01006056e-01 2.32957746e+00 3.01408451e+00]

SSE: 2462018.7541844267

CLASS LABEL:= N

Fraction of points that have the class label: 0.8450704225352113

For Cluster 4

Cluster center:

[1.76814062e+01 2.24221875e+01 1.16160938e+02 9.73425000e+02

1.00433281e-01 1.31261406e-01 1.43283125e-01 8.39731250e-02

1.89348437e-01 6.06314062e-02 6.07687500e-01 1.34821562e+00

4.20525000e+00 6.79087500e+01 6.92489063e-03 2.95979375e-02

3.99081250e-02 1.56422187e-02 1.99180156e-02 3.96907812e-03

2.10351562e+01 3.00875000e+01 1.39135938e+02 1.35689062e+03

1.37841094e-01 3.16809375e-01 3.89692344e-01 1.68736250e-01

3.05900000e-01 8.41370312e-02 3.03125000e+00 3.07812500e+00]

SSE: 2567115.4151013456

CLASS LABEL:= N

Fraction of points that have the class label: 0.734375

* 1. **Now, use the cluster centers and the class labels as a new classifier. Consider each data point again as belonging to your test set. For each data point predict its class label to be the one that belongs to the cluster center that is closest to the data point. Build the confusion matrix for this new classifier and compute its accuracy, precision and recall values.**

**CODE:**

confusion\_matrix(y, Pred)

**OUTPUT:**

array([[148, 0],

[ 46, 0]], dtype=int64)

**For non recurrence class, TP = 148, FP = 46, TN = 0, FN = 0**

**For recurrence calss, TP = 0, FP = 0, TN = 148, FN = 46**

**CODE:**

from sklearn.metrics import classification\_report

print(classification\_report(y, Pred))

accuracy = accuracy\_score(y, Pred)

print('\nAccuracy:= ' + str(accuracy))

precision recall f1-score support

N 0.76 1.00 0.87 148

R 0.00 0.00 0.00 46

avg / total 0.58 0.76 0.66 194

Accuracy:= 0.7628865979381443

* 1. **Compare these performance results with those obtained by you in HW3 Q1. Comment on the possible causes for the differences between the two sets of performance values.**

**FOR HW3 Q1,**

**Mean Accuracy using four fold: 0.763 (std: 0.008)**

**For Non-Recurrence class Precision 0.8154761904761905**

**Recall 0.925675675 F1 Score 0.8670886075949368**

**For Recurrence class Precision 0.5769230769230769**

**Recall 0.326086 F1 Score 0.4166666666666663**

**FOR HW3B,**

**Accuracy: = 0.7628865979381443**

**Class precision recall f1-score support**

**N 0.76 1.00 0.87 148 R 0.00 0.00 0.00 46**

* **Accuracy is nearly same for clustering and Decision Tree algorithm**

**But, there is large difference between the precision and recall values of recurrence class. Since, clustering predicts all recurrence dataset as non-recurrence, precision and recall value is zero for this class, whereas decision tree correctly predicts few of recurrence class datasets.**

1. **Mix the datasets for the red and white wines in one dataset. Perform k-means clustering on this large dataset for the values of k to be: 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, and 14. For each value of k report the lowest total SSE value after selecting the best of the 3-runs for each value of k. Plot the SSE value vs. the value of k. What can you infer from this plot?**

**CODE**:

dataW = pd.read\_csv("winequality-white.csv")

dataR = pd.read\_csv("winequality-red.csv")

data = pd.concat([dataW, dataR], axis=0)

num\_clusters = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]

SSEVal = []

min\_SSE= []

for i in num\_clusters:

print(" For No of clusters(k)= " + str(i))

for k in range(3):

K\_Means = KMeans(n\_clusters=i, init='random').fit(data)

SSE = K\_Means.inertia\_

SSEVal.append(SSE)

min\_sse = min(SSEVal)

min\_SSE.append(min\_sse)

print(" Minimum SSE: "+ str(min\_sse) + "\n")

**OUTPUT**:

For No of clusters(k)= 3

Minimum SSE: 4336387.441652566

For No of clusters(k)= 4

Minimum SSE: 3043516.8905087877

For No of clusters(k)= 5

Minimum SSE: 2398851.170080419

For No of clusters(k)= 6

Minimum SSE: 2046427.9092727543

For No of clusters(k)= 7

Minimum SSE: 1801340.7858560903

For No of clusters(k)= 8

Minimum SSE: 1628570.0455626736

For No of clusters(k)= 9

Minimum SSE: 1487998.6982805221

For No of clusters(k)= 10

Minimum SSE: 1371831.0411858177

For No of clusters(k)= 11

Minimum SSE: 1272185.5117560178

For No of clusters(k)= 12

Minimum SSE: 1185073.8043305934

For No of clusters(k)= 13

Minimum SSE: 1117115.8245017577

For No of clusters(k)= 14

Minimum SSE: 1028438.9697431189

**CODE**:

plt.figure(figsize = (10,8))

plt.bar(np.arange(12),min\_SSE)

plt.xticks(np.arange(12), num\_clusters)

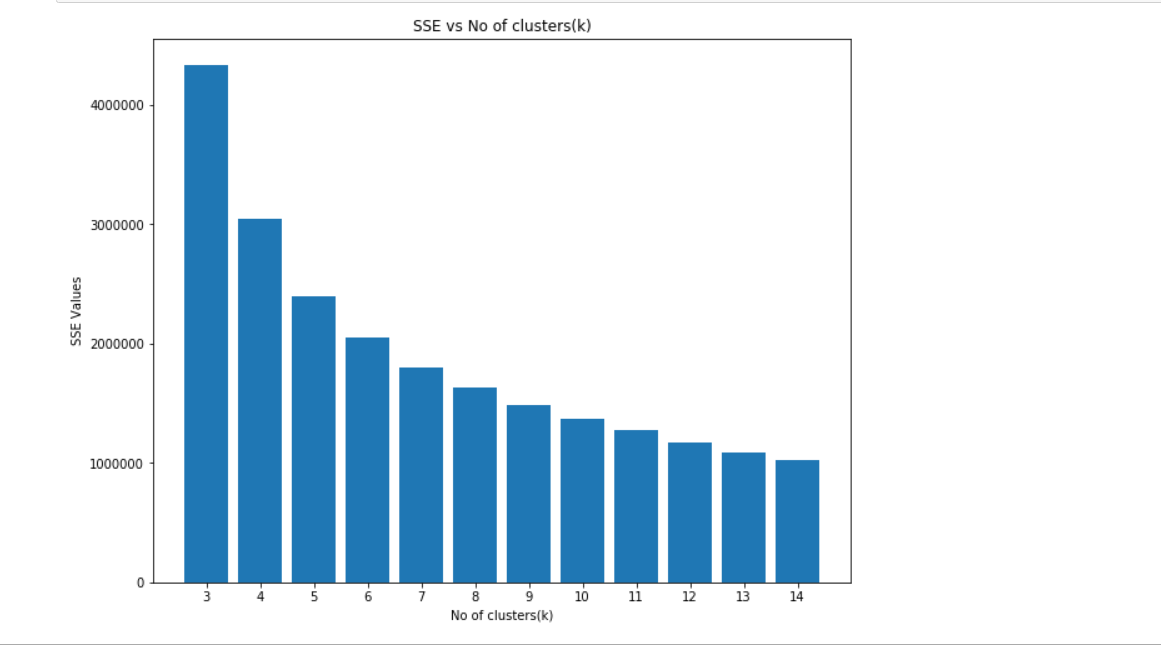
plt.xlabel("No of clusters(k)")

plt.ylabel("SSE Values")

plt.title("SSE vs No of clusters(k)")

plt.show()

**OUTPUT**:



* **As seen from above graph, the SSE value decreases as the number of clusters are increased, this is because as the number of cluster increases the number of datasets per cluster decreases and so the distance between the cluster centre and cluster points.**