In [1]:

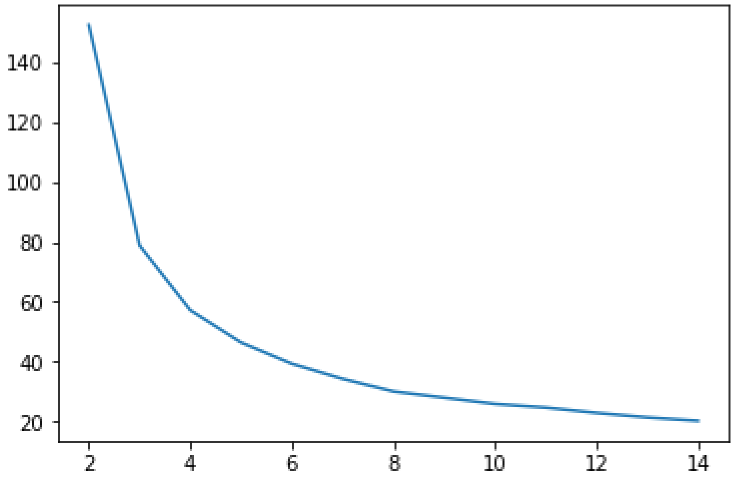
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
import random
from scipy.spatial.distance import cdist
from string import ascii_lowercase
import matplotlib.pyplot as plt
```

In [2]:

```
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
t_target = pd.Series(iris.target)
```

In [3]:

```
sum_of_squared_distances = []
K = range(2,15)
for k in K:
    km = KMeans(n_clusters=k)
    km = km.fit(iris.data)
    sum_of_squared_distances.append(km.inertia_)
plt.plot(K,sum_of_squared_distances)
plt.show()
```



Optimal number of clusters = 4

In [4]:

```
def SSE(center,cluster):
    sum_sse = 0
    for i in range(len(cluster)):
        sse = sum([x**2 for x in (cluster[i]-center)])
        sum_sse += sse
    return np.array([sum_sse])
```

In [5]:

```
def bisect_kmeans(k,Data):

    clusters= []
    final_clusters = []
    j=0 # Class label of current cluster
    Sse=[]
    centers_list = []
    while len(final_clusters) < k:

        # Splitting data into 2 clusters with KMeans algorithm
        kmeans = KMeans(n_clusters=2, random_state=random.randint(0,1000000)).
fit(Data)
        cluster_points = list(zip(Data,kmeans.labels_)) # Appending appropriat
e cluster label to each point
        centers = kmeans.cluster_centers_
        centers_list+=[x for x in centers]

        # Separating the data into 2 clusters by class
        clusters=[]
        clusters.append([x for x in cluster_points if x[-1]==0])
        clusters.append([x for x in cluster_points if x[-1]==1])
        final_clusters+=clusters
        if len(final_clusters) == k:
            break

        # Calculating SSE for each cluster
        cluster_classes = [x for x in range(len(final_clusters))]
        sse = []
        for i in range(len(final_clusters)):
            sse.append( (SSE(centers_list[i],[x[0] for x in final_clusters[i
]]),cluster_classes[i]) )

        # Finding the cluster with the highest SSE
        max_sse = max(sse)
        Data = final_clusters[max_sse[1]]
        Data = [p[0] for p in Data]

        final_clusters.pop(max_sse[1])
        centers_list.pop(max_sse[1])

    classes=[]
    cluster_classes = [x for x in range(len(final_clusters))]
    for idx,cluster in enumerate(final_clusters):
        for point in range(len(cluster)):
            classes.append(cluster_classes[idx])

    final_data = [x[0] for x in [x[i] for x in final_clusters for i in range(l
en(x))]]
    final_clusters = list(zip(final_data,classes))


    return final_clusters,centers_list
```

3 Clusters

```
In [6]:  clusters_3,centers_3 = bisect_kmeans(3,iris.data)
```

```
In [7]:  centers_3
```

```
Out[7]: [array([5.00566038, 3.36981132, 1.56037736, 0.29056604]),
          array([5.94745763, 2.76610169, 4.45423729, 1.45423729]),
          array([6.85      , 3.07368421, 5.74210526, 2.07105263])]
```

```
In [8]:  pd.Series([x[1] for x in clusters_3]).value_counts()
```

```
Out[8]: 1    59
        0    53
        2    38
        dtype: int64
```

```
In [9]:  pd.Series([x[1] for x in clusters_3]).value_counts().sum()
```

```
Out[9]: 150
```

4 Clusters

```
In [10]:  clusters_4,centers_4 = bisect_kmeans(4,iris.data)
```

```
In [11]:  centers_4
```

```
Out[11]: [array([5.00566038, 3.36981132, 1.56037736, 0.29056604]),
           array([6.85      , 3.07368421, 5.74210526, 2.07105263]),
           array([5.596, 2.664, 4.052, 1.252]),
           array([6.20588235, 2.84117647, 4.75      , 1.60294118])]
```

```
In [12]:  pd.Series([x[1] for x in clusters_4]).value_counts()
```

```
Out[12]: 0    53
         1    38
         3    34
         2    25
         dtype: int64
```

```
In [13]:  pd.Series([x[1] for x in clusters_4]).value_counts().sum()
```

```
Out[13]: 150
```