22nd august ML(clustering)

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
data=pd.read_csv('student_clustering.csv')
data
```

Out[2]:

	cgpa	ML
0	5.13	88
1	5.90	113
2	8.36	93
3	8.27	97
4	5.45	110
...	...	...
195	4.68	89
196	8.57	118
197	5.85	112
198	6.23	108
199	8.82	117

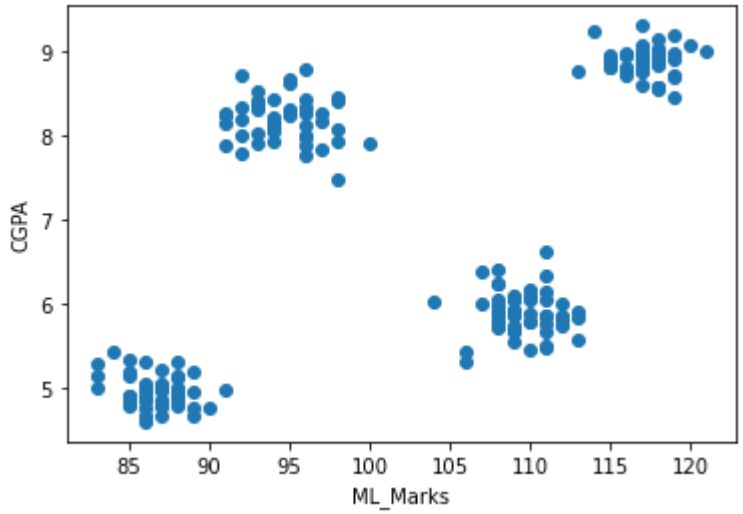
200 rows × 2 columns

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   cgpa    200 non-null    float64
 1   ML      200 non-null    int64  
dtypes: float64(1), int64(1)
memory usage: 3.2 KB
```

In [4]:
```
from matplotlib import pyplot as plt
```

In [10]:
```
plt.scatter(data['ML'],data['cgpa'])
plt.xlabel('ML_Marks')
plt.ylabel('CGPA')
plt.show()
```



In [11]:
```
from sklearn.cluster import KMeans
```

In [13]:
```
# within cluster sum of square WCSS/Elbow method
wcss=[]

for i in range(1,11):
    km = KMeans(n_clusters=i)
    km.fit_predict(data)
    wcss.append(km.inertia_ )
```

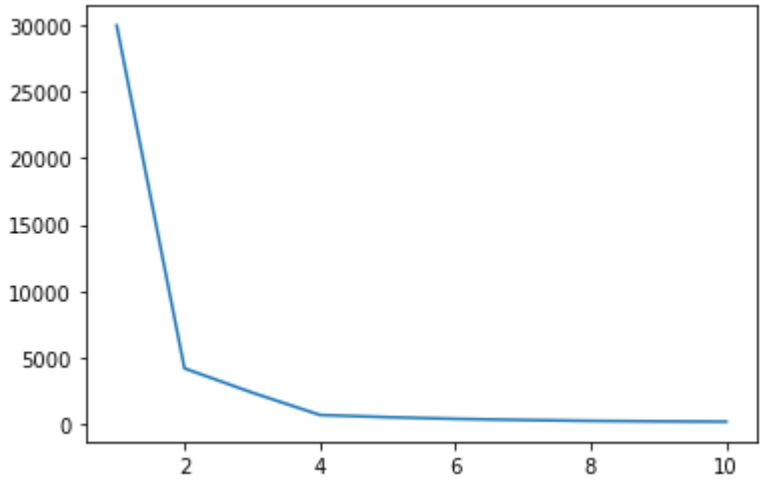
```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1036: Use
rWarning: KMeans is known to have a memory leak on Windows with MKL, when there
are less chunks than available threads. You can avoid it by setting the environ
ment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

In [14]:
```
wcss
```

Out[14]:
```
[29957.898288,
 4184.14127,
 2362.713349,
 681.9696599999999,
 514.1616803171114,
 388.8524026875982,
 302.5473746759043,
 235.30768874397904,
 201.05722088123395,
 177.00660502770498]
```

In [17]:
```
plt.plot(range(1,11),wcss)
```

Out[17]:
```
[<matplotlib.lines.Line2D at 0x1b10565e3d0>]
```



In [20]:
```
x=data.iloc[:,:].values
x
```

```
       [  8.81, 116.  ],
       [  4.88,  86.  ],
       [  8.23,  95.  ],
       [  6.61, 111.  ],
       [  8.54, 118.  ],
       [  6.04, 110.  ],
       [  8.35,  93.  ],
       [  5.01,  86.  ],
       [  8.97, 119.  ],
       [  6.24, 108.  ],
       [  8.33,  92.  ],
       [  8.91, 117.  ],
       [  4.67,  86.  ],
       [  6.1 , 109.  ],
       [  5.15,  85.  ],
       [  4.97,  88.  ],
       [  8.68, 119.  ],
       [  9.06, 120.  ],
       [  5.8 , 110.  ],
       [  8.9 , 117.  ],
```

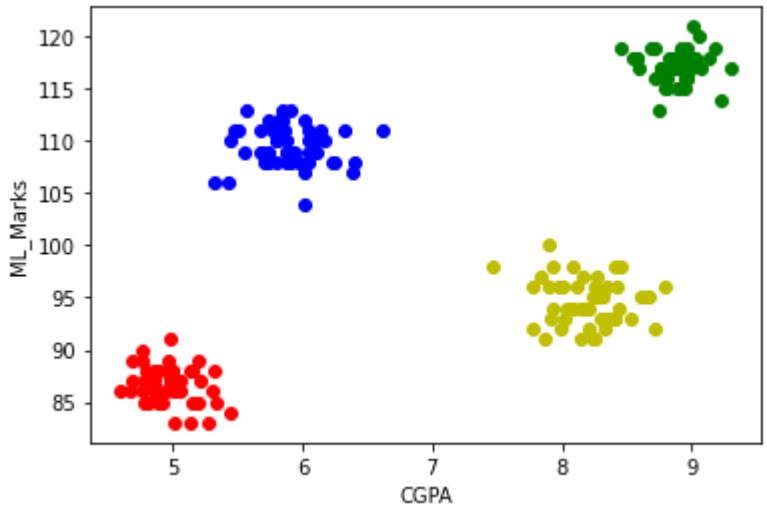
In [23]:
```
km=KMeans(n_clusters=4)
y_means=km.fit_predict(x)
y_means
```

Out[23]:
```
array([0, 1, 2, 2, 1, 1, 2, 3, 1, 2, 0, 1, 2, 0, 1, 2, 1, 2, 1, 1, 2, 0,
       2, 0, 0, 2, 0, 3, 2, 1, 3, 1, 3, 1, 2, 2, 3, 1, 0, 1, 0, 2, 2, 0,
       3, 3, 2, 1, 3, 1, 0, 0, 3, 2, 3, 1, 1, 3, 1, 3, 1, 2, 2, 3, 0, 3,
       2, 0, 1, 2, 1, 3, 2, 0, 1, 3, 1, 3, 0, 2, 2, 3, 1, 0, 3, 0, 3, 1,
       3, 1, 3, 3, 2, 0, 2, 2, 3, 2, 0, 3, 1, 0, 0, 3, 0, 0, 2, 0, 3, 3,
       2, 3, 1, 1, 2, 3, 2, 1, 3, 0, 0, 1, 2, 3, 2, 0, 2, 1, 0, 2, 2, 1,
       0, 0, 1, 3, 1, 0, 2, 2, 2, 0, 1, 0, 0, 3, 0, 3, 1, 0, 3, 0, 3, 3,
       0, 2, 1, 3, 1, 2, 0, 3, 1, 2, 3, 0, 1, 0, 0, 3, 3, 1, 3, 0, 0, 2,
       3, 1, 0, 3, 3, 1, 1, 1, 2, 0, 2, 2, 3, 1, 2, 2, 0, 0, 2, 0, 3, 1,
       1, 3])
```

In [24]:
```
x[y_means == 3,1]
```

Out[24]:
```
array([115., 119., 117., 118., 118., 116., 116., 119., 116., 115., 115.,
       117., 118., 113., 116., 118., 117., 121., 116., 117., 117., 117.,
       114., 118., 118., 119., 118., 118., 117., 118., 117., 119., 118.,
       118., 117., 117., 117., 116., 118., 119., 117., 119., 120., 117.,
       115., 115., 117., 116., 118., 117.])
```

In [31]:
```
plt.scatter(x[y_means == 0,0],x[y_means == 0,1],color="r")
plt.scatter(x[y_means == 1,0],x[y_means == 1,1],color="b")
plt.scatter(x[y_means == 2,0],x[y_means == 2,1],color="y")
plt.scatter(x[y_means == 3,0],x[y_means == 3,1],color="g")
plt.xlabel('CGPA')
plt.ylabel('ML_Marks')
plt.show()
```



cluster data

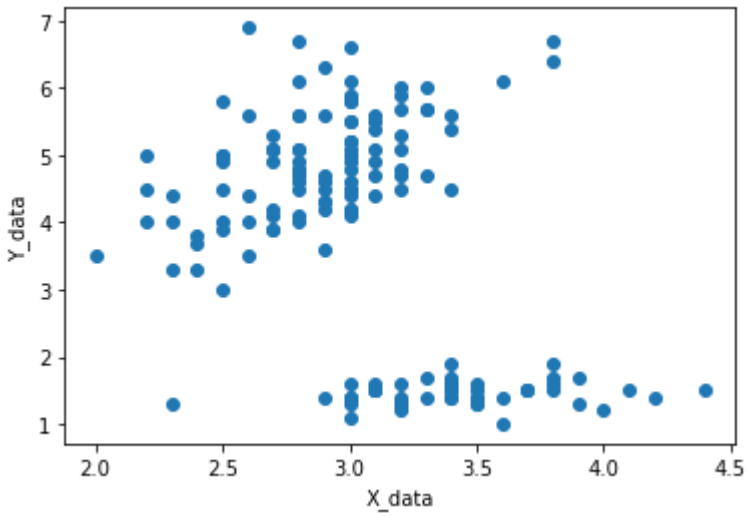
In [33]:
```
df=pd.read_csv('cluster.csv')
df
```

Out[33]:

	X	Y
0	3.5	1.4
1	3.0	1.4
2	3.2	1.3
3	3.1	1.5
4	3.6	1.4
...	...	...
145	3.0	5.2
146	2.5	5.0
147	3.0	5.2
148	3.4	5.4
149	3.0	5.1

150 rows × 2 columns

In [34]:
```
plt.scatter(df['X'],df['Y'])
plt.xlabel('X_data')
plt.ylabel('Y_data')
plt.show()
```



In [54]:
```
# within cluster sum of square WCSS/Elbow method
wcss=[]

for i in range(1,11):
    km = KMeans(n_clusters=i)
    km.fit_predict(df)
    wcss.append(km.inertia_ )
```

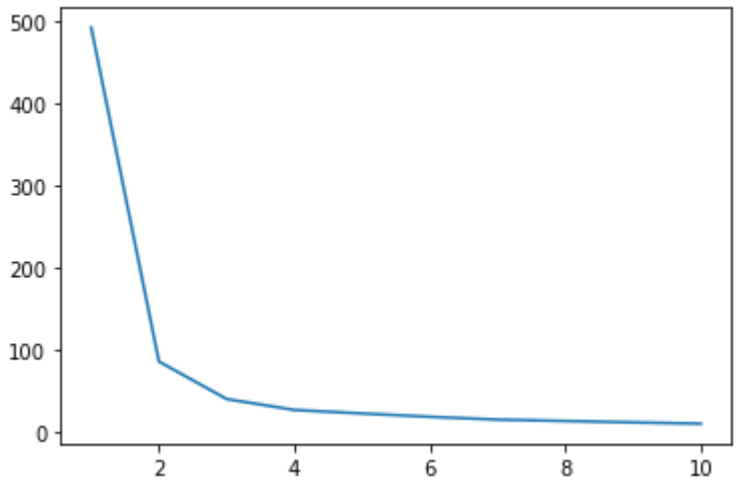
```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1036: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

In [55]:
```
wcss
```

Out[55]:
```
[491.87633333333326,
 86.35692216280452,
 40.80747409220727,
 27.55509523809524,
 23.236,
 19.215913208086533,
 15.808740080609047,
 14.137640882509306,
 12.436960544357607,
 10.860997113997119]
```

In [56]:
```
plt.plot(range(1,11),wcss)
```

Out[56]:
```
[<matplotlib.lines.Line2D at 0x1b108a6b9d0>]
```



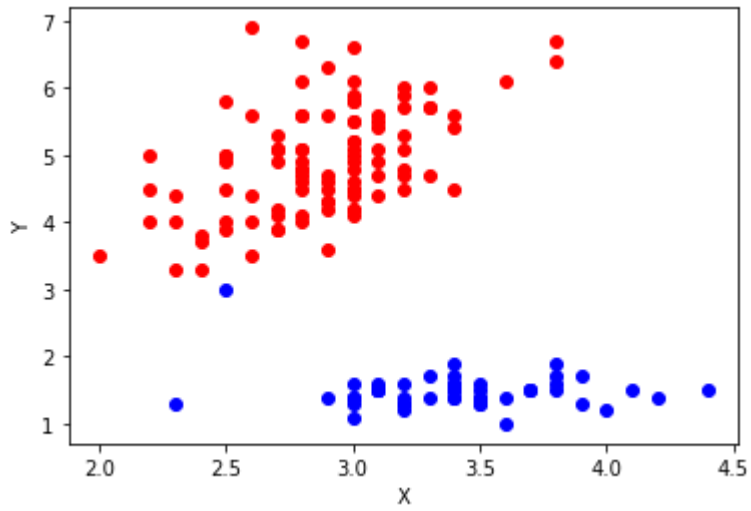
In [67]:
```
x=df.iloc[:,:].values
x
```

```
       [2.3, 1.3],
       [3.2, 1.3],
       [3.5, 1.6],
       [3.8, 1.9],
       [3. , 1.4],
       [3.8, 1.6],
       [3.2, 1.4],
       [3.7, 1.5],
       [3.3, 1.4],
       [3.2, 4.7],
       [3.2, 4.5],
       [3.1, 4.9],
       [2.3, 4. ],
       [2.8, 4.6],
       [2.8, 4.5],
       [3.3, 4.7],
       [2.4, 3.3],
       [2.9, 4.6],
       [2.7, 3.9],
       [2. , 3.5],
```

In [68]:
```
km=KMeans(n_clusters=2)
y_means=km.fit_predict(x)
y_means
```

Out[68]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [71]:
```
plt.scatter(x[y_means == 0,0],x[y_means == 0,1],color="r")
plt.scatter(x[y_means == 1,0],x[y_means == 1,1],color="b")
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



In [ ]:

In [ ]: