In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import preprocessing
from scipy.optimize import curve_fit
from sklearn.cluster import KMeans
```

In [15]:

```
# dataset
urban_df = pd.read_csv('urban population.csv')

'''choose for clustering because its very long dataset but we only choose t
    we also remove all of the nul values to make error free clusters'''
Df_urban = urban_df[["2019","2020"]].dropna()
# make arrary
X = Df_urban.values
```

Out[15]:

	Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962
0	Aruba	ABW	Urban population (% of total population)	SP.URB.TOTL.IN.ZS	50.776000	50.761000	50.746000
1	Africa Eastern and Southern	AFE	Urban population (% of total population)	SP.URB.TOTL.IN.ZS	14.704688	14.944459	15.185608
2	Afghanistan	AFG	Urban population (% of total population)	SP.URB.TOTL.IN.ZS	8.401000	8.684000	8.976000
3	Africa Western and Central	AFW	Urban population (% of total population)	SP.URB.TOTL.IN.ZS	14.670329	15.053577	15.449282
4	Angola	AGO	Urban population (% of total population)	SP.URB.TOTL.IN.ZS	10.435000	10.798000	11.204000
...	...	...	...	...	...	...	...
258	Samoa	WSM	Urban population (% of total population)	SP.URB.TOTL.IN.ZS	18.926000	18.986000	19.061000
259	Yemen, Rep.	YEM	Urban population (% of total population)	SP.URB.TOTL.IN.ZS	9.100000	9.459000	9.831000
260	South Africa	ZAF	Urban population (% of total population)	SP.URB.TOTL.IN.ZS	46.619000	46.793000	46.906000
261	Zambia	ZMB	Urban population (% of total population)	SP.URB.TOTL.IN.ZS	18.145000	18.951000	19.785000
262	Zimbabwe	ZWE	Urban population (% of total population)	SP.URB.TOTL.IN.ZS	12.608000	12.821000	13.082000

263 rows × 66 columns

In [3]:

```
'''desribe the choosen dataset'''
Df_urban.describe()
```

Out[3]:

	2019	2020
count	262.000000	262.000000
mean	60.231410	60.558663
std	22.725521	22.663185
min	13.250000	13.345000
25%	42.201414	42.493195
50%	60.172500	61.063159
75%	79.904000	80.314584
max	100.000000	100.000000

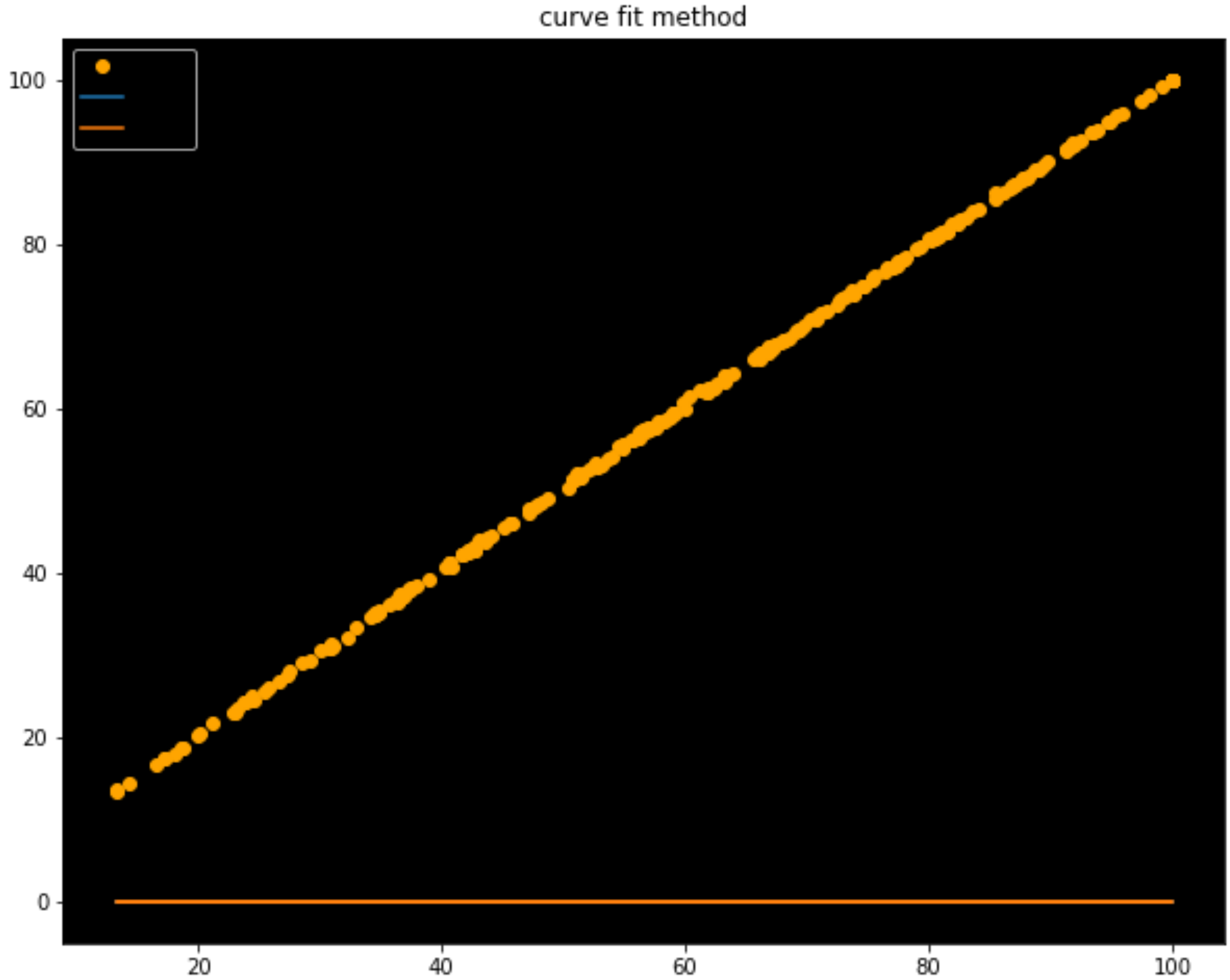
In [4]:

```
'''Min max scaler is used for normaliazation for this we use simple library

scaler = preprocessing.MinMaxScaler()
names = Df_urban.columns
de= scaler.fit_transform(Df_urban)
scaled_df = pd.DataFrame(de,columns=names)
```

In [9]:

```
'''fitting the dataset using curve fit '''
from scipy.optimize import curve_fit

'''split in x and y'''

x = Df_urban["2019"]
y = Df_urban["2020"]

def temp(x, A, B):
    y = A*np.exp(-1*B*x**2)
    return y

'''that above function is guassian function that is for fit the model usin
param, cov = curve_fit(temp,x, y)
fitA = param[0]
fitB = param[1]
fit_y = temp(X, fitA, fitB)

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

plt.rcParams['axes.facecolor'] = 'black'
plt.plot(x, y,'o', label='data', color="orange")
plt.title("curve fit method")
plt.plot(x, fit_y, '-', label='fit')
plt.legend()
plt.show()
```

```
E:\Files\lib\site-packages\scipy\optimize\minpack.py:833: OptimizeWarning: Covariance of the parameters could not be estimated
  warnings.warn('Covariance of the parameters could not be estimated',
```



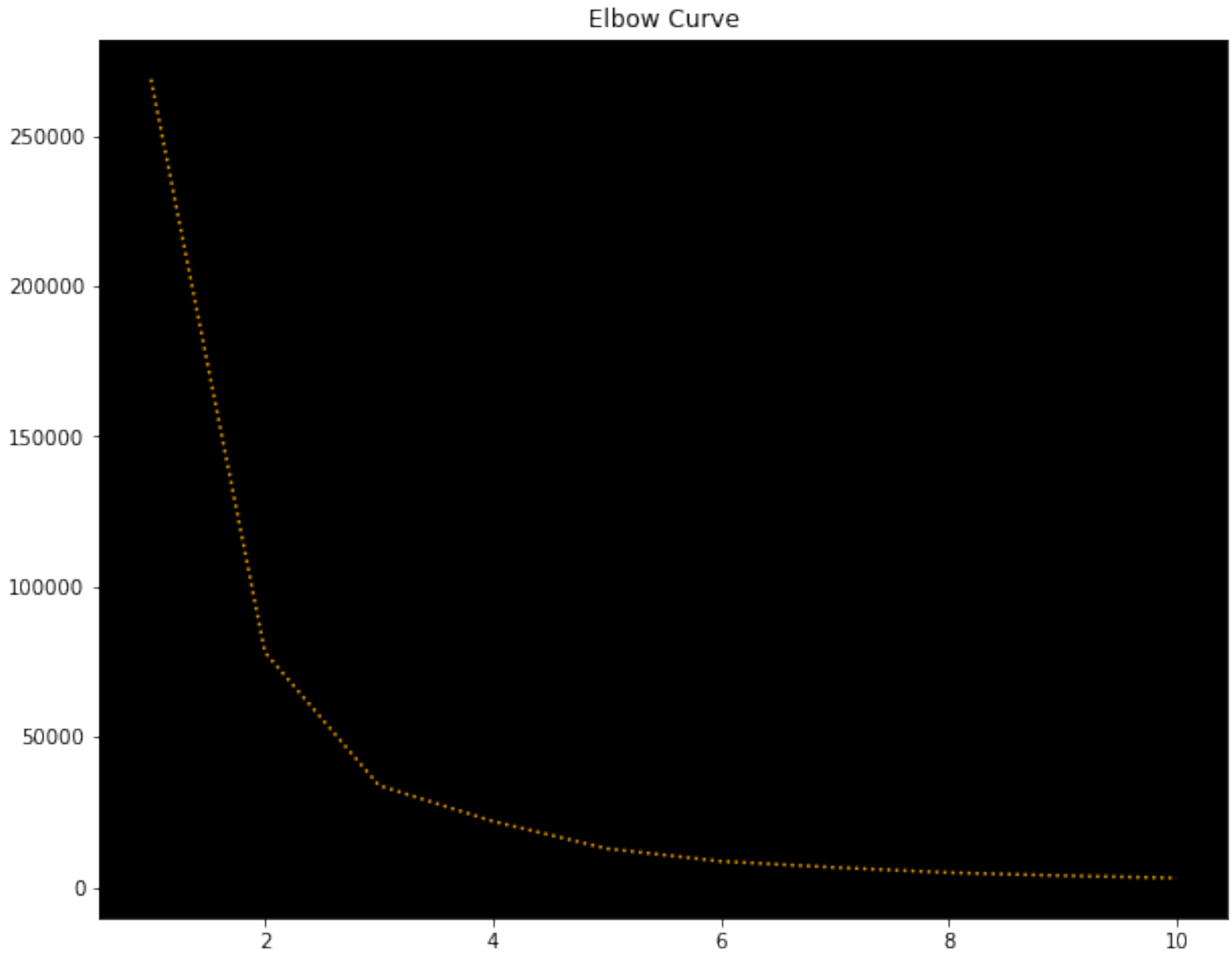
In [11]:

```
# future values prediction
'''finding possible num of cluster in given dataset using elbow graph'''
wcss = []
for i in range(1,11):
    kmeans = KMeans(n_clusters = i, init = "k-means++")
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
```

```
E:\Files\lib\site-packages\sklearn\cluster\_kmeans.py:881: 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=2.
  warnings.warn(
```

In [12]:

```
plt.figure(figsize=(10,8))
plt.rcParams['axes.facecolor'] = 'black'
plt.title("Elbow Curve")
plt.plot(range(1,11),wcss, linestyle ="dotted", color="orange")
plt.show()
```



In [13]:

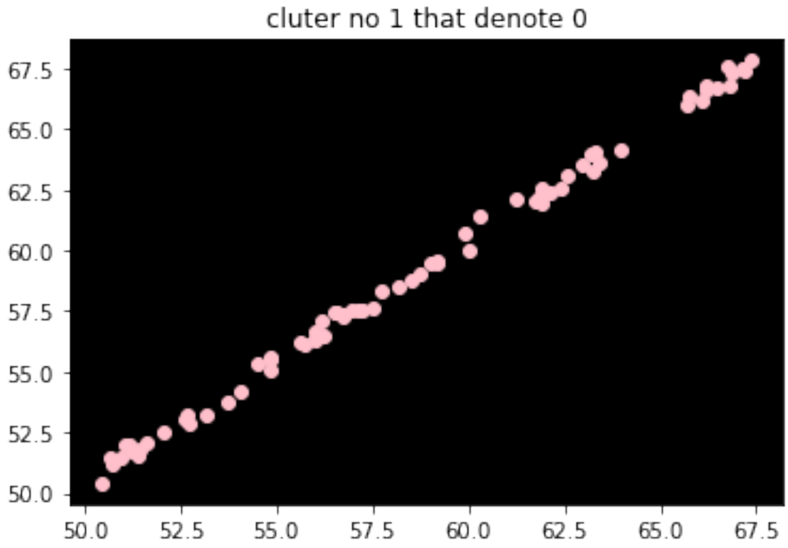
```
'''5 clusters easily make from this dataset as we determine in elbow graph
# use buitlin function for kmean cluster of sklearn.cluster
kmeans = KMeans(n_clusters = 5, init = "k-means++")
y_kmeans = kmeans.fit_predict(X)
y_kmeans
```

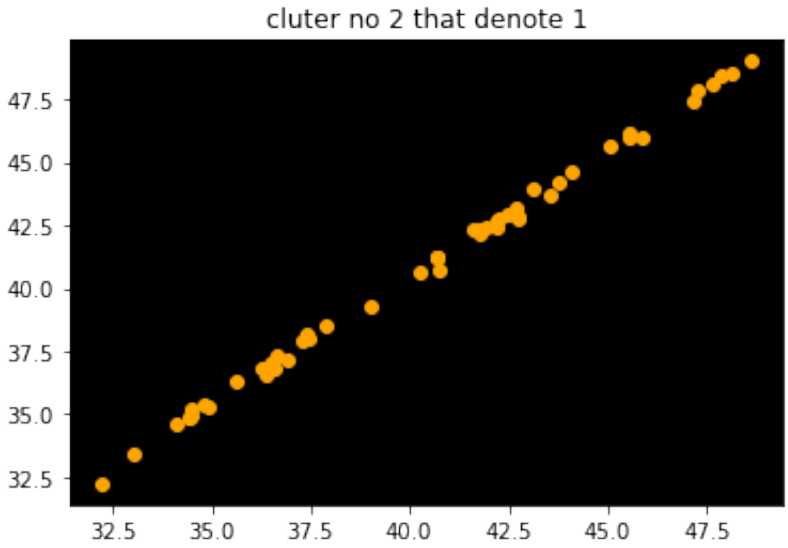
Out[13]:

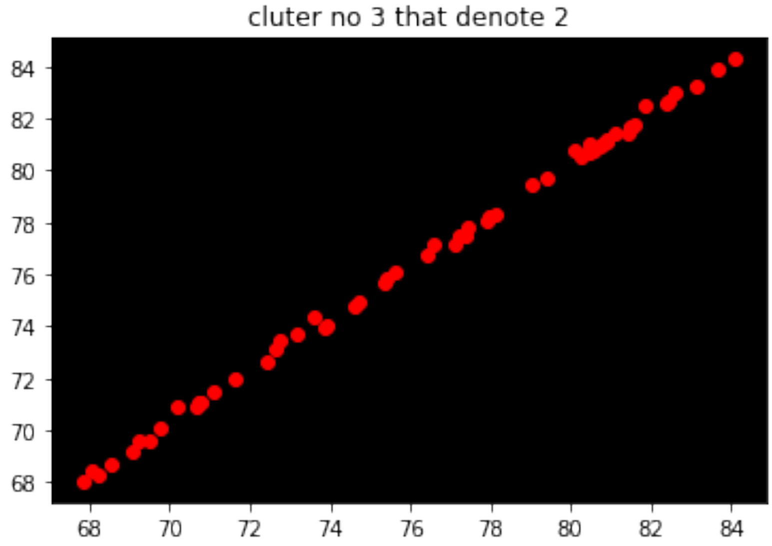
```
array([1, 1, 4, 1, 0, 0, 3, 0, 3, 3, 0, 3, 4, 3, 0, 0, 4, 3, 1, 4, 1, 2,
       3, 2, 1, 2, 1, 3, 2, 3, 4, 2, 1, 2, 1, 2, 0, 2, 4, 3, 0, 0, 0, 1,
       0, 2, 4, 0, 2, 0, 2, 3, 3, 0, 2, 2, 2, 2, 3, 2, 2, 0, 1, 0, 0, 2,
       0, 1, 2, 2, 2, 4, 2, 1, 3, 0, 2, 1, 4, 3, 2, 0, 0, 3, 1, 0, 1, 2,
       2, 1, 3, 0, 3, 4, 2, 3, 0, 1, 0, 0, 2, 0, 0, 1, 1, 0, 1, 0, 1, 0,
       2, 2, 3, 3, 2, 0, 3, 3, 0, 4, 1, 4, 0, 4, 2, 3, 2, 1, 3, 0, 2, 4,
       2, 1, 1, 4, 4, 1, 0, 4, 0, 2, 3, 2, 3, 0, 3, 1, 1, 1, 0, 2, 2, 0,
       0, 1, 3, 4, 0, 0, 2, 3, 1, 0, 1, 4, 2, 2, 0, 2, 4, 0, 0, 3, 2, 4,
       3, 3, 2, 3, 0, 1, 2, 2, 1, 2, 4, 0, 1, 3, 0, 0, 0, 2, 1, 2, 0, 3,
       0, 2, 4, 1, 2, 1, 1, 3, 4, 1, 2, 3, 1, 0, 1, 4, 1, 0, 2, 0, 0, 0,
       3, 4, 3, 0, 0, 3, 4, 0, 0, 1, 0, 4, 0, 2, 4, 0, 4, 1, 1, 0, 2, 2,
       0, 1, 4, 2, 0, 3, 2, 0, 0, 3, 1, 3, 1, 4, 0, 4, 1, 0, 1, 1])
```

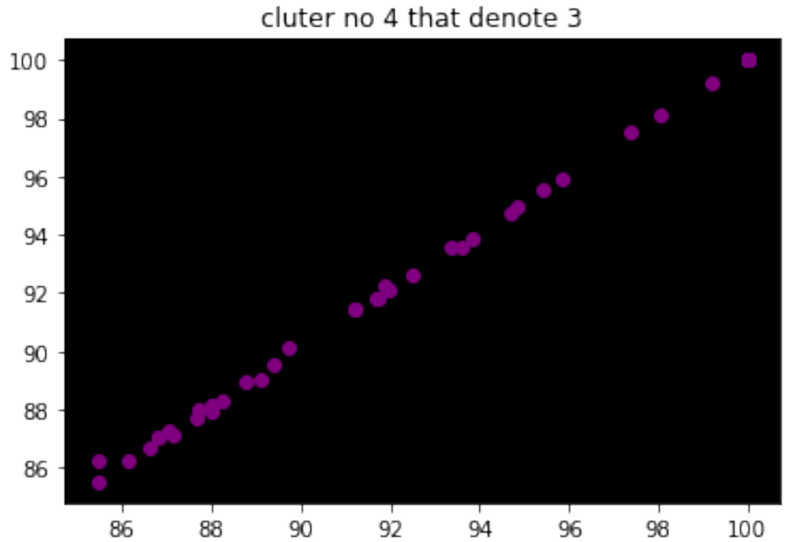
In [14]:

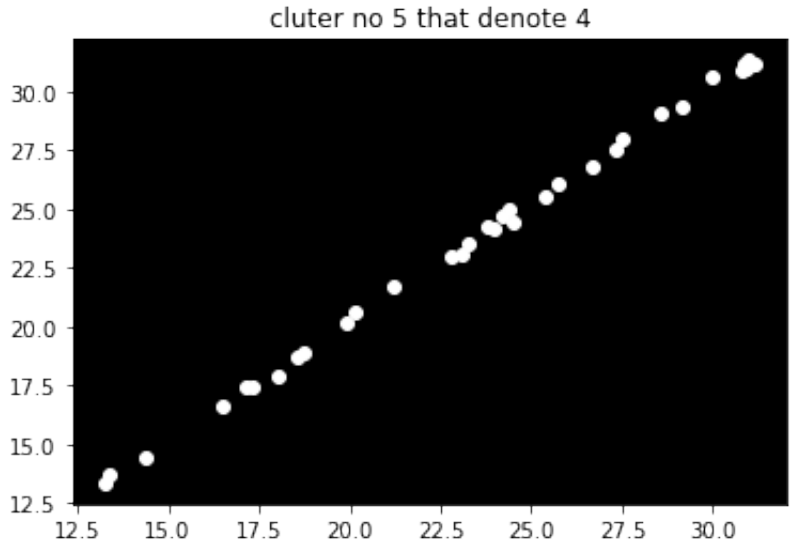
```
# Interpretation of the results.
'''according to prediction of k mean most of values are 0 then 2 then 3 and
first cluster is of 0
second cluster is of 2
third cluster is of 3
fourth cluster is of 1'''

c = ["pink","orange", "red","purple","white"]
for i in range(5):
#     plt.figure(figsize=(10,8))
    string = f"cluter no {i+1} that denote {i}"
    plt.title(string)
    plt.scatter(X[y_kmeans==i,0], X[y_kmeans==i,1], color=c[i])
    plt.show()
```





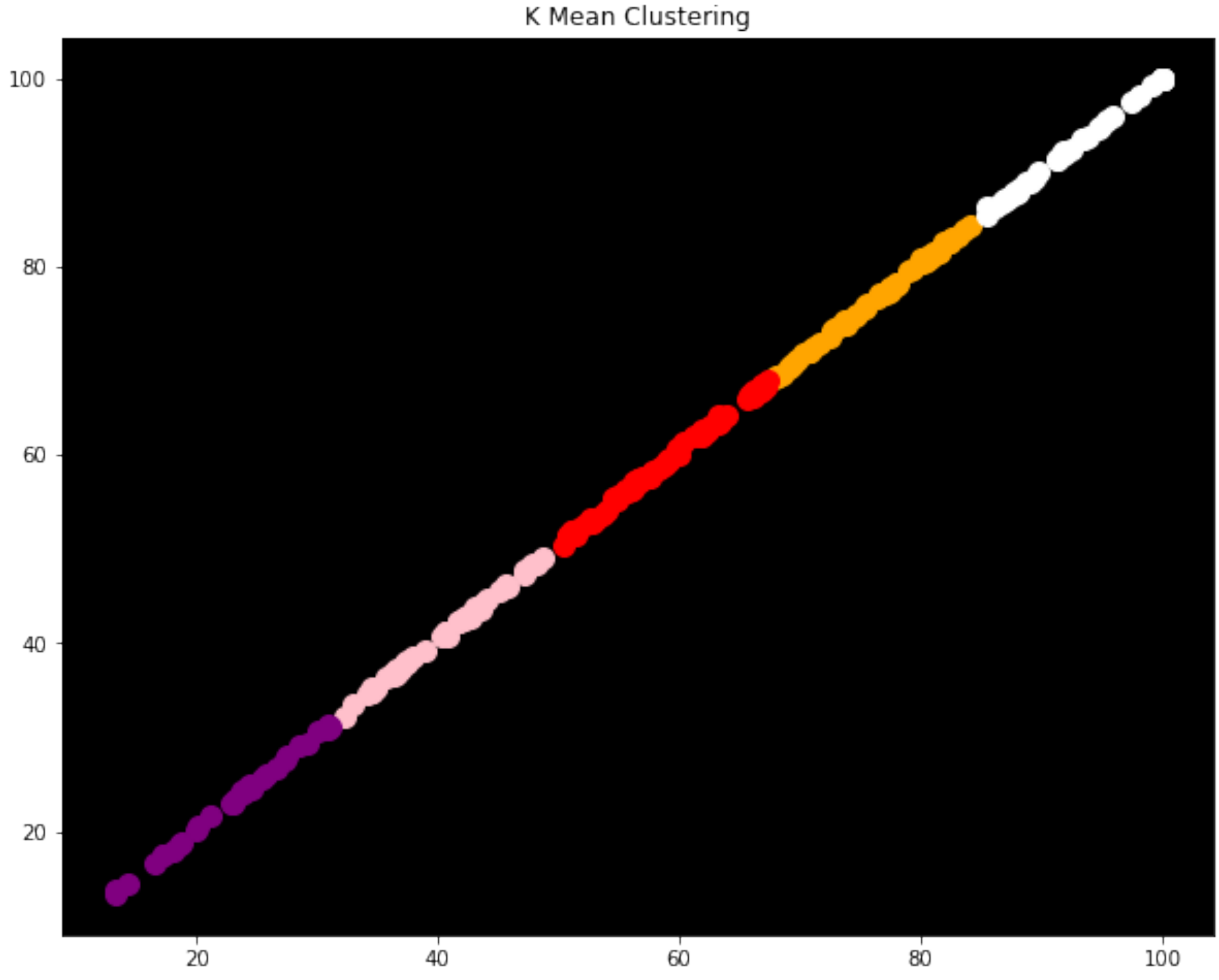

cluter no 3 that denote 2
84
82
80
78
76
74
72
70
68
68 70 72 74 76 78 80 82 84


cluter no 4 that denote 3
100
98
96
94
92
90
88
86
86 88 90 92 94 96 98 100


cluter no 5 that denote 4
30.0
27.5
25.0
22.5
20.0
17.5
15.0
12.5
12.5 15.0 17.5 20.0 22.5 25.0 27.5 30.0

In [27]:
```
'''now we plotting all in single frame to make different clusters finally'


plt.figure(figsize = (10,8))
plt.title("K Mean Clustering")
plt.scatter(X[y_kmeans==0,0], X[y_kmeans==0,1],s=100, c="pink")
plt.scatter(X[y_kmeans==1,0], X[y_kmeans==1,1],s=100, c="orange")
plt.scatter(X[y_kmeans==2,0], X[y_kmeans==2,1],s=100, c="red")
plt.scatter(X[y_kmeans==3,0], X[y_kmeans==3,1],s=100, c="purple")
plt.scatter(X[y_kmeans==4,0], X[y_kmeans==4,1],s=100, c="white")
```

Out[27]:
```
<matplotlib.collections.PathCollection at 0x1e3e28e6dc0>
```



In [ ]: