Clustering different types of wine using K-Means

Import libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import metrics
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
```

Reading the data

```
In [2]:

df = pd.read_csv("wine.csv")
    print(df.shape)
    df
```

(178, 14)

Out[2]:		Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflavanoid_Pher
	0	14.23	1.71	2.43	15.6	127	2.80	3.06	(
	1	13.20	1.78	2.14	11.2	100	2.65	2.76	(
	2	13.16	2.36	2.67	18.6	101	2.80	3.24	(
	3	14.37	1.95	2.50	16.8	113	3.85	3.49	(
	4	13.24	2.59	2.87	21.0	118	2.80	2.69	(
	•••	•••	•••		•••	***	***	•••	
	173	13.71	5.65	2.45	20.5	95	1.68	0.61	(
	174	13.40	3.91	2.48	23.0	102	1.80	0.75	(
	175	13.27	4.28	2.26	20.0	120	1.59	0.69	(
	176	13.17	2.59	2.37	20.0	120	1.65	0.68	(
	177	14.13	4.10	2.74	24.5	96	2.05	0.76	(

178 rows × 14 columns

Statistical Data analysis

```
In [3]: df.describe()
```

Out[3]:		Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonfla
	count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	
	mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	
	std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	
	min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	
	25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	
	50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	
	75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	
	max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	

One Hot Encoding

```
dummies = pd.get_dummies(df['Customer_Segment'])
df = pd.concat([df, dummies], axis=1)
df.drop('Customer_Segment', axis=1, inplace=True)
df
```

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflavanoid_Pher
0	14.23	1.71	2.43	15.6	127	2.80	3.06	(
1	13.20	1.78	2.14	11.2	100	2.65	2.76	(
2	13.16	2.36	2.67	18.6	101	2.80	3.24	(
3	14.37	1.95	2.50	16.8	113	3.85	3.49	(
4	13.24	2.59	2.87	21.0	118	2.80	2.69	(
•••	•••		•••					
173	13.71	5.65	2.45	20.5	95	1.68	0.61	(
174	13.40	3.91	2.48	23.0	102	1.80	0.75	(
175	13.27	4.28	2.26	20.0	120	1.59	0.69	(
176	13.17	2.59	2.37	20.0	120	1.65	0.68	(
177	14.13	4.10	2.74	24.5	96	2.05	0.76	(
	1 2 3 4 173 174 175	 0 14.23 1 13.20 2 13.16 3 14.37 4 13.24 173 13.71 174 13.40 175 13.27 176 13.17 	0 14.23 1.71 1 13.20 1.78 2 13.16 2.36 3 14.37 1.95 4 13.24 2.59 173 13.71 5.65 174 13.40 3.91 175 13.27 4.28 176 13.17 2.59	0 14.23 1.71 2.43 1 13.20 1.78 2.14 2 13.16 2.36 2.67 3 14.37 1.95 2.50 4 13.24 2.59 2.87 173 13.71 5.65 2.45 174 13.40 3.91 2.48 175 13.27 4.28 2.26 176 13.17 2.59 2.37	0 14.23 1.71 2.43 15.6 1 13.20 1.78 2.14 11.2 2 13.16 2.36 2.67 18.6 3 14.37 1.95 2.50 16.8 4 13.24 2.59 2.87 21.0 173 13.71 5.65 2.45 20.5 174 13.40 3.91 2.48 23.0 175 13.27 4.28 2.26 20.0 176 13.17 2.59 2.37 20.0	0 14.23 1.71 2.43 15.6 127 1 13.20 1.78 2.14 11.2 100 2 13.16 2.36 2.67 18.6 101 3 14.37 1.95 2.50 16.8 113 4 13.24 2.59 2.87 21.0 118 173 13.71 5.65 2.45 20.5 95 174 13.40 3.91 2.48 23.0 102 175 13.27 4.28 2.26 20.0 120 176 13.17 2.59 2.37 20.0 120	0 14.23 1.71 2.43 15.6 127 2.80 1 13.20 1.78 2.14 11.2 100 2.65 2 13.16 2.36 2.67 18.6 101 2.80 3 14.37 1.95 2.50 16.8 113 3.85 4 13.24 2.59 2.87 21.0 118 2.80 173 13.71 5.65 2.45 20.5 95 1.68 174 13.40 3.91 2.48 23.0 102 1.80 175 13.27 4.28 2.26 20.0 120 1.59 176 13.17 2.59 2.37 20.0 120 1.65	1 13.20 1.78 2.14 11.2 100 2.65 2.76 2 13.16 2.36 2.67 18.6 101 2.80 3.24 3 14.37 1.95 2.50 16.8 113 3.85 3.49 4 13.24 2.59 2.87 21.0 118 2.80 2.69 173 13.71 5.65 2.45 20.5 95 1.68 0.61 174 13.40 3.91 2.48 23.0 102 1.80 0.75 175 13.27 4.28 2.26 20.0 120 1.59 0.69 176 13.17 2.59 2.37 20.0 120 1.65 0.68

Scaling data to a similar range

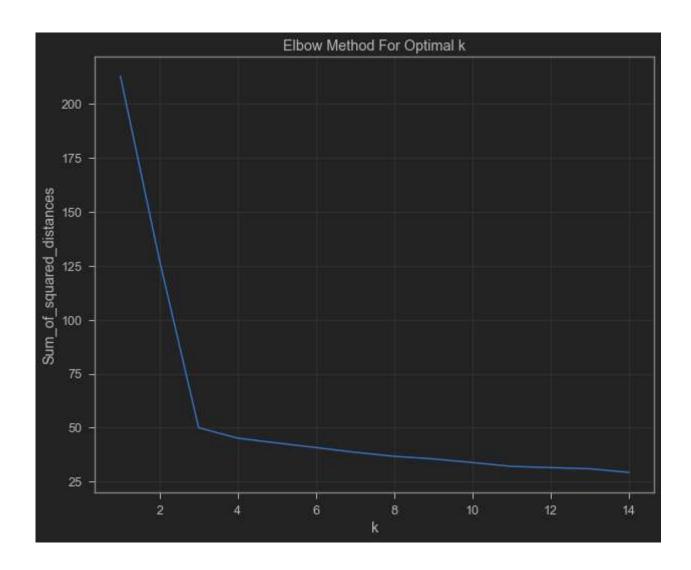
```
In [5]:
         scaler = MinMaxScaler()
         scaled data = scaler.fit transform(df)
         scaled data
Out[5]: array([[0.84210526, 0.1916996 , 0.57219251, ..., 1.
                                                              , 0.
              [0.57105263, 0.2055336 , 0.4171123 , ..., 1.
                                                               , 0.
              [0.56052632, 0.3201581, 0.70053476, ..., 1.
                                                               , 0.
              [0.58947368, 0.69960474, 0.48128342, ..., 0.
                                                               , 0.
              [0.56315789, 0.36561265, 0.54010695, ..., 0.
                                                              , 0.
              1. ],
[0.81578947, 0.66403162, 0.73796791, ..., 0.
                                                               , 0.
```

Using K-Means along with Elbow method using Sum of Squared Distances

```
In [6]: Sum_of_squared_distances = []
K = range(1,15)
for k in K:
    km = KMeans(n_clusters=k)
    km = km.fit(scaled_data)
    Sum_of_squared_distances.append(km.inertia_)
```

Plotting elbow curve

```
In [7]: plt.figure(figsize=(10, 8))
  plt.plot(K, Sum_of_squared_distances, 'bx-')
  plt.xlabel('k')
  plt.ylabel('Sum_of_squared_distances')
  plt.title('Elbow Method For Optimal k')
  plt.show()
```



Optimal Value of k = 3