Automatic Clustering – Elbow Method

Knowledge Discovery

Hanif Izzudin Rahman

1. Convert that data into the numerical values

```python
data = pd.read_excel("hepatitis_new.xlsx", header=None)
data.drop(0, inplace=True, axis=1)
data.drop(0, inplace=True, axis=0)
data.columns = data.iloc[0]
data.drop(1, inplace=True, axis=0)
data.columns = [c.replace(' ', '_') for c in data.columns]
data = data.replace(to_replace=['no', 'yes'], value=[0, 1])
data.CLASS = data.CLASS.replace(to_replace=['Live', 'Die'], value=[0, 1])
data = data.replace(to_replace=['?'], value=np.nan)
data = data.reset_index()
X_temp = data.drop(columns=['CLASS'])
X_temp
```

	index	Age	Sex	Steroid	Antivirals	Fatique	Malaise	Anorexia	Liver_Big	Liver_Firm	Spleen_Palpable	Speiders	Ascites	Varices	Bilirubin	Alk_Phosp
0	2	30	1	0.0	1	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	
1	3	50	0	0.0	1	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	0.9	
2	4	78	0	1.0	1	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.7	
3	5	31	0	NaN	0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.7	
4	6	34	0	1.0	1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
150	152	46	0	1.0	1	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	7.6	
151	153	44	0	1.0	1	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.9	
152	154	61	0	0.0	1	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0.8	
153	155	53	1	0.0	1	0.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	1.5	
154	156	43	0	1.0	1	0.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	1.0	1.2	

155 rows × 20 columns

```python
y = data['CLASS'].values
y
```

```
array([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, 1, 1, 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, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0,
       1], dtype=int64)
```

2. Impute the missing data with the mean values of same attribute in the same class

```
X = data.groupby("CLASS").transform(lambda x: x.fillna(x.mean()))
X
```

Age	Sex	Steroid	Antivirals	Fatique	Malaise	Anorexia	Liver_Big	Liver_Firm	Spleen_Palpable	Speiders	Ascites	Varices	Bilirubin	Alk_Phosphate	SGOT
30	1	0.000000	1	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	85.000000	18.(
50	0	0.000000	1	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	0.9	135.000000	42.(
78	0	1.000000	1	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.7	96.000000	32.(
31	0	0.540984	0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.7	46.000000	52.(
34	0	1.000000	1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	101.313725	200.(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
46	0	1.000000	1	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	7.6	122.375000	242.(
44	0	1.000000	1	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.9	126.000000	142.(
61	0	0.000000	1	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0.8	75.000000	20.(
53	1	0.000000	1	0.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	1.5	81.000000	19.(
43	0	1.000000	1	0.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	1.0	1.2	100.000000	19.(

20 columns

3. Hide the class label of the supervised data

4. Normalize Data

```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler().fit(X)

X = scaler.transform(X)
X
```

```
array([[-1.7209121 , -0.89419175,  2.94745653, ...,  0.30720513,
         0.26151157, -0.90748521],
       [-1.69856259,  0.70257923, -0.33927557, ..., -0.48942799,
         0.26151157, -0.90748521],
       [-1.67621309,  2.93805862, -0.33927557, ...,  0.30720513,
         0.26151157, -0.90748521],
       ...,
       [ 1.67621309,  1.58080328, -0.33927557, ...,  0.46653176,
         0.26151157,  1.10194633],
       [ 1.69856259,  0.94209488,  2.94745653, ...,  0.46653176,
        -0.75812043,  1.10194633],
       [ 1.7209121 ,  0.14370939, -0.33927557, ..., -1.1267345 ,
        -1.08753999,  1.10194633]])
```

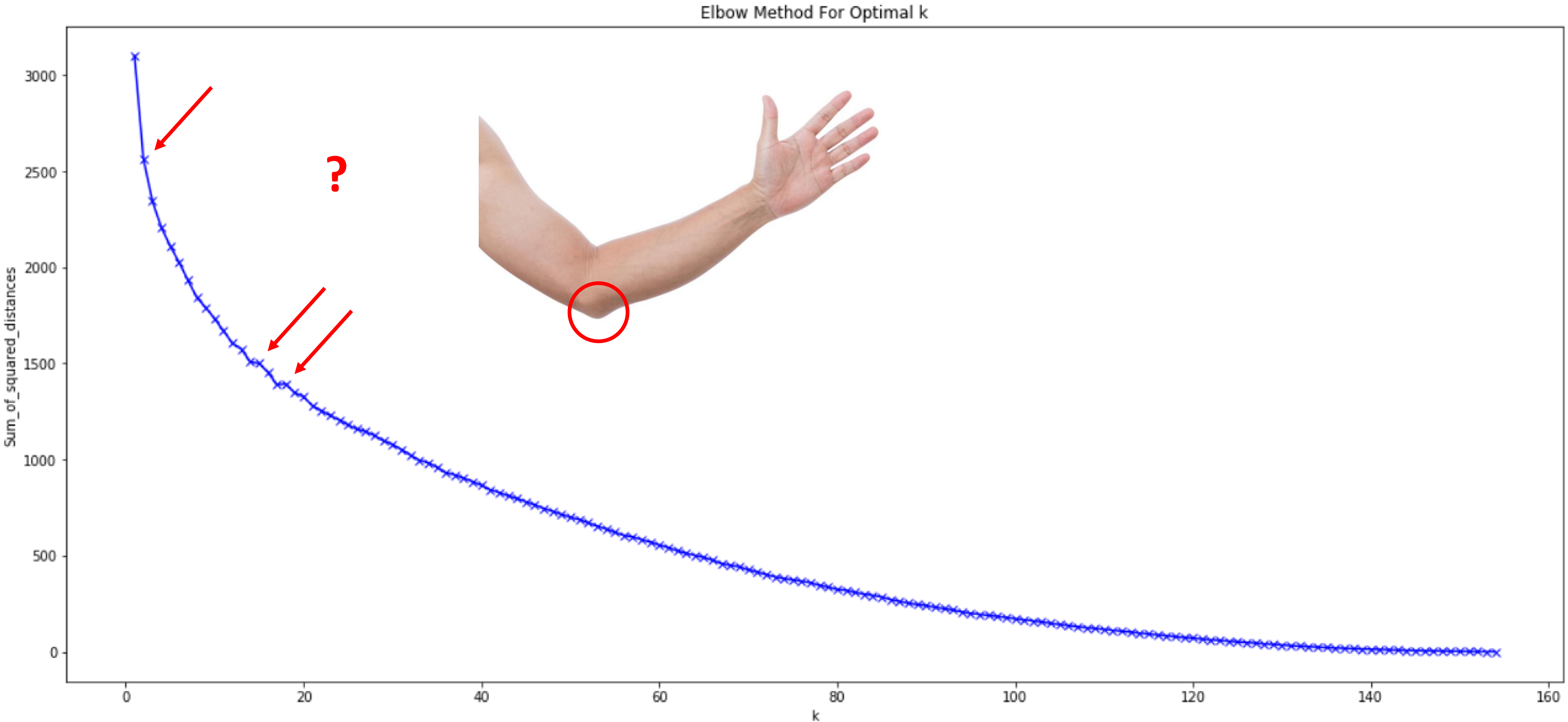
```python
from sklearn.cluster import KMeans

Sum_of_squared_distances = []
K = range (1,len(y))
for k in K:
    kmeans = KMeans(n_clusters=k, random_state=0).fit(X)
    Y = Sum_of_squared_distances.append(kmeans.inertia_)
    #print(kmeans)
    print("K:", k, ", Sum_of_squared_distances:", kmeans.inertia_)
    #print(Sum_of_squared_distances)
```

```
K: 1 , Sum_of_squared_distances: 3100.0
K: 2 , Sum_of_squared_distances: 2563.1880351556065
K: 3 , Sum_of_squared_distances: 2344.4084991183263
K: 4 , Sum_of_squared_distances: 2206.809317329538
K: 5 , Sum_of_squared_distances: 2110.2154065429067
K: 6 , Sum_of_squared_distances: 2026.183904664447
K: 7 , Sum_of_squared_distances: 1932.5104090968682
K: 8 , Sum_of_squared_distances: 1842.1888752072994
K: 9 , Sum_of_squared_distances: 1790.3183684827495
K: 10 , Sum_of_squared_distances: 1730.8361247386383
K: 11 , Sum_of_squared_distances: 1669.4937009163148
K: 12 , Sum_of_squared_distances: 1604.5200690111237
K: 13 , Sum_of_squared_distances: 1574.7286439186155
K: 14 , Sum_of_squared_distances: 1506.8996167161922
K: 15 , Sum_of_squared_distances: 1499.2873642735221
K: 16 , Sum_of_squared_distances: 1453.1393319666813
K: 17 , Sum_of_squared_distances: 1389.3728038949735
K: 18 , Sum_of_squared_distances: 1392.2171550277717
K: 19 , Sum_of_squared_distances: 1347.2949483729005
```

5. Apply the automatic clustering. How many clusters are created? - Elbow Method

Choose The Elbow



Get Elbow – KneeLocator (kneed)

```
from kneed import KneeLocator

k_opt = KneeLocator(K, Sum_of_squared_distances, curve="convex", direction="decreasing")
print('Optimal k is: ',k_opt.elbow)
```

```
Optimal k is:  17
```

6. Compare the clusters and the original classes of the dataset

```
# Optimal K

kmeans = KMeans(n_clusters=k_opt.elbow, random_state=0).fit(X)
kmeans.labels_
```

```
<kneed.knee_locator.KneeLocator object at 0x00000243F7406A08>

array([14,  0, 12,  1,  1,  1,  4,  1, 12,  1,  0,  9,  9, 12,  9, 15, 12,
       12,  1,  0, 14,  4,  1,  1, 14, 12,  9,  6,  0,  9,  9,  9, 14, 14,
       12, 12, 12,  4,  1,  8,  9,  1,  1, 12,  1, 12,  1,  6,  1,  9,  1,
        1,  1,  9, 12, 11, 12,  1,  0,  6,  1,  1, 13,  4, 12,  1,  1,  8,
       12, 12,  1,  2, 12, 11,  1,  9,  4, 14,  2,  1, 12,  1,  1,  6,  6,
       11,  7,  3,  3, 11,  4, 13,  5,  5,  2, 11,  2,  2,  5,  2,  3,  5,
        5, 13,  3,  5, 15, 10,  0, 15, 16,  9,  5,  4,  5,  6,  5,  5,  2,
        2,  2,  8,  5,  5,  5,  2,  8,  7, 15, 15,  5, 13,  8,  9, 16,  7,
        2,  7,  3,  5,  3, 15,  7, 15,  8,  5,  7, 10,  5,  5, 13,  5,  2,
        7,  8])
```

```python
# Original Classes
y
```

```
array([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, 1, 1, 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, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0,
       1], dtype=int64)
```

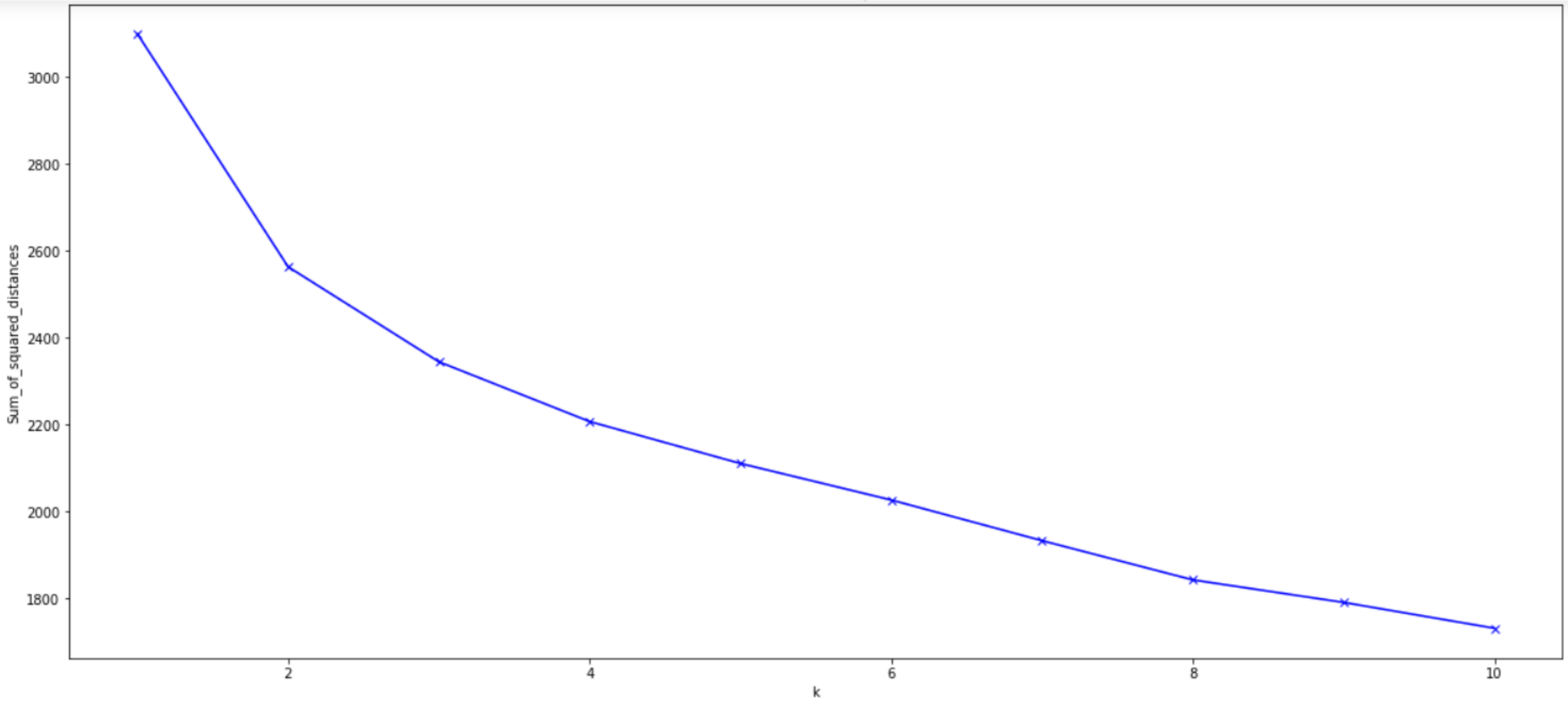
```python
# Accuracy
from sklearn.metrics import accuracy_score
acc = accuracy_score(y, kmeans.labels_)
error = (1-acc)*100
print("Error: ", error)
```

```
Error:  96.12903225806451
```

Analysis

Error yang dihasilkan sangat tinggi, yaitu 96.12 % dengan menggunakan Elbow Method dengan library “kneed” untuk mendapatkan nilai elbow yaitu di k=17.

Menurut saya, memang untuk pengaplikasian Elbow Method tidak bisa digunakan diberbagai macam kasus. Jika penggunaan Elbow Method ini digunakan looping yang sedikit, hasil mendapatkan nilai k yang optimum juga berbeda. Seperti contoh dibawah ini, saya looping hanya sebanyak 10x




```
Optimal k is:  3
Error:  87.74193548387098
```

Nilai K yang didapatkan dengan looping hanya sebanyak 10x, akan mendapatkan k yang optimum di k=3, dengan error masih tinggi 87.74%

Cara alternatif lain adalah dengan menggunakan Silhouette Method atau bisa juga dengan mencari Variance.