How to determine the optimal number of clusters for k-means clustering using Elbow Method

Introduction
K-means is a type of unsupervised learning and one of the popular methods of clustering unlabelled data into k clusters. One of the trickier tasks in clustering is identifying the appropriate number of clusters k. In this tutorial, we will provide an overview of how k-means works and discuss how to implement your own clusters.

We will also understand how to use the elbow method as a way to estimate the value k. Another popular method of estimating k is through silhouette analysis, a scikit learn example can be found here.

We will use the wholesale customer dataset which can be downloaded here.

K-means Overview
Before diving into the dataset, let us briefly discuss how k-means works:

1. The process begins with k centroids initialised at random.

2. These centroids are used to assign points to its nearest cluster.

3. The mean of all points within the cluster is then used to update the position of the centroids.

4. The above steps are repeated until the values of the centroids stabilise.

Getting Started
In this tutorial, we will be using the scikit-learn's implementation of k-means which can be found here

The dataset
The dataset we will study refers to clients of a wholesale distributor. It contains information such as clients annual spend on fresh product, milk products, grocery products etc. Below is some more information an each feature:

1. FRESH: annual spending (m.u.) on fresh products (Continuous)

2. MILK: annual spending (m.u.) on milk products (Continuous)
3. GROCERY: annual spending (m.u.) on grocery products (Continuous)
4. FROZEN: annual spending (m.u.) on frozen products (Continuous)
5. DETERGENTS_PAPER: annual spending (m.u.) on detergents and paper products (Continuous)
6. DELICATESSEN: annual spending (m.u.) on delicatessen products (Continuous)
7. CHANNEL: customer channels - Horeca (Hotel/Restaurant/Cafe) or Retail channel (Nominal)
8. REGION: customer regions - Lisnon, Oporto or Other (Nominal)

```
# Import required packages
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans

import matplotlib.pyplot as plt
```

Read in data and inspect the first 5 records.

```
data = pd.read_csv('Wholesale customers data.csv')
data.head()
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185

Below is a split of categorical and continuous features

```
categorical_features = ['Channel', 'Region']
continuous_features = ['Fresh', 'Milk', 'Grocery', 'Frozen',
'Detergents_Paper', 'Delicassen']
```

Descriptive statistics below shows on average clients spend the most on fresh groceries and the least on delicassen.

```
data[continuous_features].describe()
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
count	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000
mean	12000.297727	5796.265909	7951.277273	3071.931818	2881.493182	1524.870455
std	12647.328865	7380.377175	9503.162829	4854.673333	4767.854448	2820.105937
min	3.000000	55.000000	3.000000	25.000000	3.000000	3.000000
25%	3127.750000	1533.000000	2153.000000	742.250000	256.750000	408.250000
50%	8504.000000	3627.000000	4755.500000	1526.000000	816.500000	965.500000
75%	16933.750000	7190.250000	10655.750000	3554.250000	3922.000000	1820.250000
max	112151.000000	73498.000000	92780.000000	60869.000000	40827.000000	47943.000000

To use the categorical features, we need to convert the categorical features to binary using pandas get dummies.

```
for col in categorical_features:
    dummies = pd.get_dummies(data[col], prefix=col)
    data = pd.concat([data, dummies], axis=1)
    data.drop(col, axis=1, inplace=True)

data.head()
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen	Channel_1	Channel_2	Region_1	Region_2	Region_3
0	12669	9656	7561	214	2674	1338	0	1	0	0	1
1	7057	9810	9568	1762	3293	1776	0	1	0	0	1
2	6353	8808	7684	2405	3516	7844	0	1	0	0	1
3	13265	1196	4221	6404	507	1788	1	0	0	0	1
4	22615	5410	7198	3915	1777	5185	0	1	0	0	1

To give equal importance to all features, we need to scale the continuous features. We will be using scikit-learn's MinMaxScaler as the feature matrix is a mix of binary and continuous features . Other alternatives includes StandardScaler.

```
mms = MinMaxScaler()
mms.fit(data)
data_transformed = mms.transform(data)
```

For each k value, we will initialise k-means and use the inertia attribute to identify the sum of squared distances of samples to the nearest cluster centre.

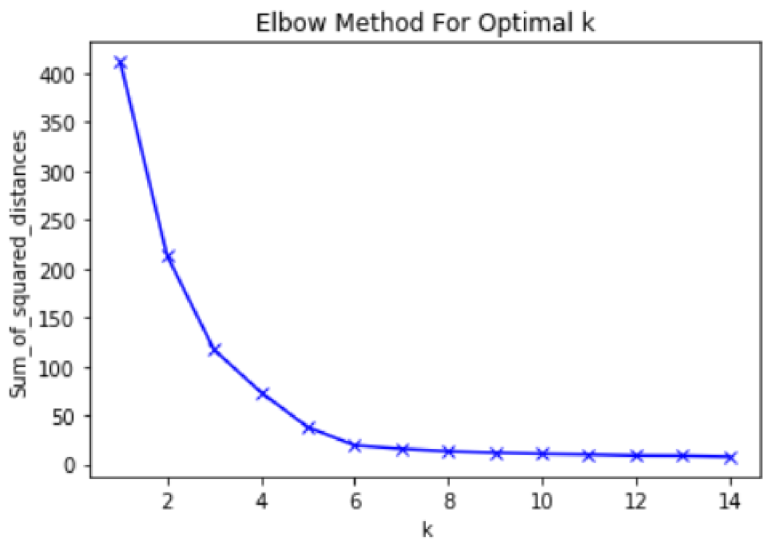
```
Sum_of_squared_distances = []
K = range(1,15)
for k in K:
```

```
    km = KMeans(n_clusters=k)
    km = km.fit(data_transformed)
    Sum_of_squared_distances.append(km.inertia_)
```

As k increases, the sum of squared distance tends to zero. Imagine we set k to its maximum value n (where n is number of samples) each sample will form its own cluster meaning sum of squared distances equals zero.

Below is a plot of sum of squared distances for k in the range specified above. If the plot looks like an arm, then the elbow on the arm is optimal k.

```
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()
```



In the plot above the elbow is at k=5 indicating the optimal k for this dataset is around 5. Therefore, you should review the results for 4, 5 and 6 while designing your k-means algorithm.

Source