K-Means Clustering

Exercise 1: Simple Clustering on dummy data

Let us try the k-means clustering algorithm for clustering some random numbers between 0 and 1. The Python library and Scipy have some inbuilt methods to perform the algorithm and return a list defining which observation belongs to which cluster:

1. Define a set of observations consisting of random numbers ranging from 0 to 1. In this case, we have defined an observation set of 30x3:

```
Import numpy as np
obs=np.random.random(90).reshape(30,3)
obs
```

The output of the code looks as follows:

```
array([[ 0.57661327,  0.57228436,  0.09121444],
       [ 0.57243857,  0.50374038,  0.56191463],
       [ 0.6331554 ,  0.31386819,  0.18356078],
       [ 0.86898774,  0.0403591 ,  0.86725316],
       [ 0.2607788 ,  0.59915679,  0.90926909],
       [ 0.58096358,  0.5744143 ,  0.82951824],
       [ 0.84253324,  0.69768867,  0.04050713],
       [ 0.62160337,  0.97374147,  0.83282739],
       [ 0.74153731,  0.38723477,  0.11345058],
       [ 0.39491075,  0.56146279,  0.23015805],
```

A 30x3 array of random numbers between 0 and 1

- Decide that we want two clusters (no hard and fast rule, you can try with three clusters also). Select two observations at random to make them cluster centroids:

```
c1=np.random.choice(range(len(obs)))
c2=np.random.choice(range(len(obs)))
clust_cen=np.vstack([obs[c1],obs[c2]])
clust_cen
```

The output of the code looks as follows:

```
array([[ 0.47184647,  0.89249612,  0.73130458],
       [ 0.28342103,  0.64640709,  0.24428166]])
```

Selecting two rows (out of 30) at random to be initial cluster centers

The two rows in the `clust_cen` array correspond to the two cluster centroids.

- With the number of clusters and cluster centroids defined, one is ready to implement the k-means clustering. This can be done using the `cluster` method of Scipy:

```
from scipy.cluster.vq import vq
```

```
vq(obs,clust_cen)
```

```
(array([1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0,
        0, 1, 0, 0, 1, 0, 1], dtype=int64),
 array([ 0.33894755,  0.43582422,  0.48639827,  0.94991615,  0.40282613,
         0.35032629,  0.59729417,  0.19833007,  0.54236297,  0.14087217,
         0.19110092,  0.34641348,  0.        ,  0.25230902,  0.87490428,
         0.27554023,  0.5590402 ,  0.25151793,  0.28609226,  0.77169955,
         0.72043438,  0.        ,  0.44381884,  0.68654886,  0.80043611,
         0.33197907,  0.37141823,  0.5703176 ,  0.45580223,  0.77236007]))
```

Cluster label and distance from cluster centers for each observation

The first array gives us the information as to which cluster the observation belongs to. The first observation belongs to cluster c2, the second observation belongs to c1, the third belongs to c2, the fourth to c1, and so on.

The second array gives the distance of the observation from the final cluster centroid. Hence, the first observation is at a distance of 0.33 units from the centroid of the cluster c2, the second observation is at a distance of 0.43 from the centroid of the cluster c1, and so on.

- Find the cluster centroid for the two clusters. This is done using the `kmeans` method in Scipy:

```
from scipy.cluster.vq import kmeans
kmeans(obs,clust_cen)
```

The output of the code looks as follows:

```
(array([[ 0.52471403,  0.83763873,  0.67613882],
        [ 0.6133091 ,  0.34955296,  0.35004751]]), 0.35108732217332617)
```

The final cluster centers and the value of the squared error function or J-score

The two rows in the array correspond to the two final cluster centroids. The centroid of the first cluster is at (0.524, 0.837, 0.676). The number at the end is the value of the squared error function or J-score, which we seek to minimize. Its value comes out to be 0.35.

K-means also works if one provides just the number of required clusters and not the cluster centroids. If only the required number of clusters is provided, then the method will randomly select that many observations at random from the observation set to become a cluster centroid. Thus, we could have also written the following:

```
from scipy.cluster.vq import kmeans
kmeans(obs,2)
```

Exercise 2: Clustering on real data

Now, as we understand the mathematics behind the k-means clustering better, let us implement it on a dataset and see how to glean insights from the performed clustering.

The dataset we will be using for this is about wine. Each observation represents a separate sample of wine and has information about the chemical composition of that wine.

Some wine connoisseur painstakingly analyzed various samples of wine to create this dataset. Each column of the dataset has information about the composition of one chemical. There is one column called quality as well, which is based on the ratings given by the professional wine testers.

The prices of wines are generally decided by the ratings given by the professional testers. However, this can be very subjective and certainly there is a scope for a more logical process to wine prices. One approach is to cluster them based on their chemical compositions and quality and then price the similar clusters together based on the desirable components present in the wine clusters.

Importing and exploring the dataset

Let us import and have a look at this dataset:

```
import pandas as pd
df=pd.read_csv('C:/FILE_PATH/wine.csv',sep=';')
df.head()
```

The output looks as follows:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25	67	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15	54	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17	60	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5

The first few observations of the wine dataset

As one can observe, it has 12 columns as follows:

```
array(['fixed acidity', 'volatile acidity', 'citric acid',
       'residual sugar', 'chlorides', 'free sulfur dioxide',
       'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol',
       'quality'], dtype=object)
```

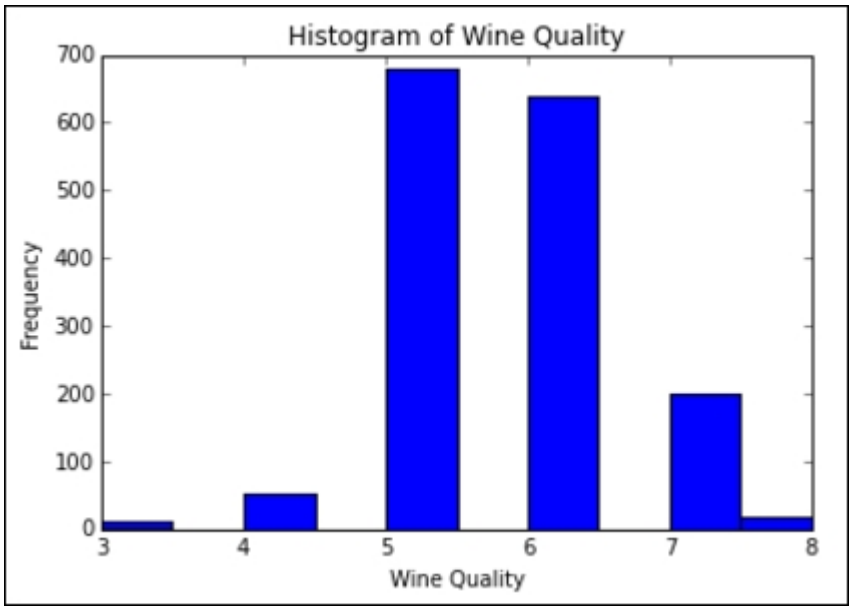
The column names of the wine dataset

There are 1599 observations in this dataset.

Let us focus on the quality variable for a while and plot a histogram to see the number of wine samples in each quality type:

```
import matplotlib.pyplot as plt
% matplotlib inline
plt.hist(df['quality'])
```

The code shows the following output:



The histogram of wine quality. The majority of samples have been rated 6 or 7 for quality

As it is evident from the plot, more than 75% of the samples were assigned the quality of **5** and **6**. Also, let's look at the mean of the various chemical compositions across samples for the different groups of the wine quality:

```
df.groupby('quality').mean()
```

The code shows the following output:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
quality											
3	8.360000	0.884500	0.171000	2.635000	0.122500	11.000000	24.900000	0.997464	3.398000	0.570000	9.955000
4	7.779245	0.693962	0.174151	2.694340	0.090679	12.264151	36.245283	0.996542	3.381509	0.596415	10.265094
5	8.167254	0.577041	0.243686	2.528855	0.092736	16.983847	56.513950	0.997104	3.304949	0.620969	9.899706
6	8.347179	0.497484	0.273824	2.477194	0.084956	15.711599	40.869906	0.996615	3.318072	0.675329	10.629519
7	8.872362	0.403920	0.375176	2.720603	0.076588	14.045226	35.020101	0.996104	3.290754	0.741256	11.465913
8	8.566667	0.423333	0.391111	2.577778	0.068444	13.277778	33.444444	0.995212	3.267222	0.767778	12.094444

The mean values of all the numerical columns for each value of quality

Some observations based on this table are as follows:

- The lesser the **volatile acidity** and **chlorides**, the higher the wine quality
- The more the **sulphates** and **citric acid** content, the higher the wine quality

- The **density** and **pH** don't vary much across the wine quality

Next, let's proceed with clustering these observations using k-means.

Normalizing the values in the dataset

As discussed, normalizing the values is important to get the clustering right. This can be achieved by applying the following formula to each value in the dataset:

$$Z_i = \left(X_i - X_{min} \right) / \left(X_{max} - X_{min} \right)$$

To normalize our dataset, we write the following code snippet:

```
df_norm = (df - df.min()) / (df.max() - df.min())
df_norm.head()
```

This results in a data frame with normalized values for entire data frame as follows:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	0.247788	0.397260	0.00	0.068493	0.106845	0.140845	0.098940	0.567548	0.606299	0.137725	0.153846	0.4
1	0.283186	0.520548	0.00	0.116438	0.143573	0.338028	0.215548	0.494126	0.362205	0.209581	0.215385	0.4
2	0.283186	0.438356	0.04	0.095890	0.133556	0.197183	0.169611	0.508811	0.409449	0.191617	0.215385	0.4
3	0.584071	0.109589	0.56	0.068493	0.105175	0.225352	0.190813	0.582232	0.330709	0.149701	0.215385	0.6
4	0.247788	0.397260	0.00	0.068493	0.106845	0.140845	0.098940	0.567548	0.606299	0.137725	0.153846	0.4

Normalized wine dataset

Hierarchical clustering using scikit-learn

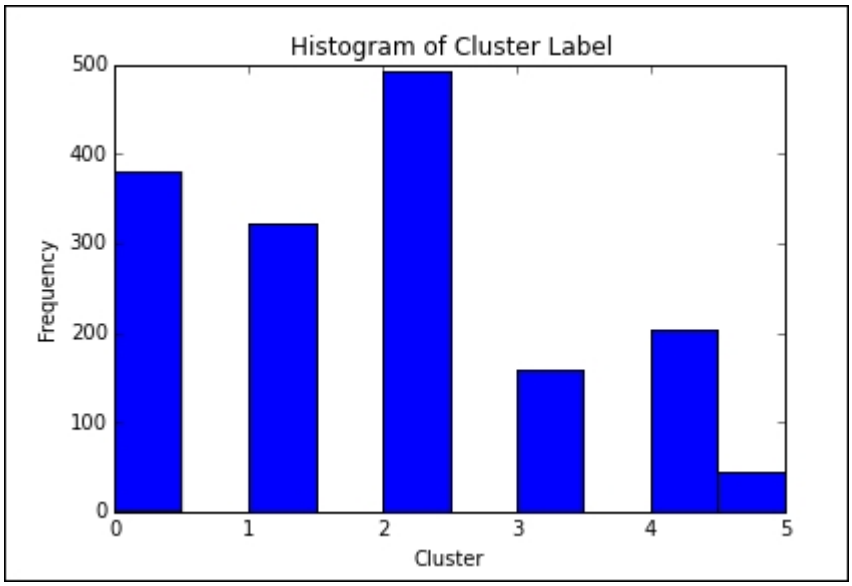
Hierarchical clustering or agglomerative clustering can be implemented using the `AgglomerativeClustering` method in scikit-learn's cluster library as shown in the following code. It returns a label for each row denoting which cluster that row belongs to. The number of clusters needs to be defined in advance. We have used the `ward` method of linkage:

```
from sklearn.cluster import AgglomerativeClustering
ward = AgglomerativeClustering(n_clusters=6, linkage='ward').fit(df_norm)
md=pd.Series(ward.labels_)
```

We can plot a histogram of cluster labels to get a sense of how many rows belong to a particular cluster:

```
import matplotlib.pyplot as plt
% matplotlib inline
plt.hist(md)
plt.title('Histogram of Cluster Label')
plt.xlabel('Cluster')
plt.ylabel('Frequency')
```

The plot looks as follows. The observations are more uniformly distributed across the cluster except Cluster 2 that has more observations than the others:



The histogram of Cluster Labels. Samples are more-or-less uniformly distributed across clusters

It also outputs the children for each non-leaf node. This would be an array with the shape (number of non-leaf nodes, 2) as there would be two immediate children for any non-leaf node:

```
ward.children_
```

The code shows the following output:

```
array([[1592, 1596],
       [1579, 1581],
       [1564, 1567],
       ...,
       [3179, 3191],
       [3192, 3193],
       [3194, 3195]], dtype=int64)
```

The child array containing two child elements for each non-leaf node

K-Means clustering using scikit-learn

Let us randomly choose `6` as the required number of clusters for now as there were that many groups of quality in the dataset. Then, to cluster the observations, one needs to write the following code snippet:

```
from sklearn.cluster import KMeans
from sklearn import datasets
model=KMeans(n_clusters=6)
model.fit(df_norm)
```

The preceding snippet fits the k-means clustering model to the wine dataset. To know which observation belongs to which of the clusters, one can call the `labels_` parameter of the `model`. It returns an array depicting the cluster the row belongs to:

```
model.labels_
```

The output of the code is as follows:

```
array([2, 2, 2, ..., 5, 5, 3])
```

Cluster labels for each row

For better observation, let us make this array part of the data frame so that we can look at the cluster each row belongs to, in the same data frame:

```
md=pd.Series(model.labels_)
df_norm['clust']=md
df_norm.head()
```

The output of the code shows the following datasheet:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	clust
0	0.247788	0.397260	0.00	0.068493	0.106845	0.140845	0.098940	0.567548	0.606299	0.137725	0.153846	0.4	2
1	0.283186	0.520548	0.00	0.116438	0.143573	0.338028	0.215548	0.494126	0.362205	0.209581	0.215385	0.4	2
2	0.283186	0.438356	0.04	0.095890	0.133556	0.197183	0.169611	0.508811	0.409449	0.191617	0.215385	0.4	2
3	0.584071	0.109589	0.56	0.068493	0.105175	0.225352	0.190813	0.582232	0.330709	0.149701	0.215385	0.6	1
4	0.247788	0.397260	0.00	0.068493	0.106845	0.140845	0.098940	0.567548	0.606299	0.137725	0.153846	0.4	2

The wine dataset with a clust column depicting the cluster the row belongs to

The last column **clust** of the data frame denotes the cluster to which that particular observation belongs. The 1st, 2nd, 3rd, and 5th observations belong to the 3rd cluster (counting starts from 0), while the 4th observation belongs to the 2nd cluster.

The final cluster's centroids for each cluster can be found out as follows:

```
model.cluster_centers_
```

Note that each cluster centroid would have 12 coordinates as there are 12 variables in the dataset.

The dataset is as follows:

```
array([[ 0.31657408,  0.27714878,  0.30457413,  0.15202455,  0.12698872,
         0.38048163,  0.29785645,  0.5390477 ,  0.43592737,  0.17890402,
         0.21400954,  0.45615142],
       [ 0.57666747,  0.20368051,  0.50645914,  0.131816  ,  0.1305873 ,
         0.13722804,  0.09139157,  0.6528223 ,  0.3351512 ,  0.22556444,
         0.29055173,  0.54941634],
       [ 0.26395373,  0.35904298,  0.12348425,  0.09132645,  0.12283596,
         0.14482367,  0.10768343,  0.49211153,  0.48648397,  0.1554293 ,
         0.22016455,  0.43976378],
       [ 0.36290046,  0.15975098,  0.42329457,  0.10547414,  0.10543412,
         0.16350038,  0.08386063,  0.40849071,  0.41863517,  0.23557536,
         0.49129398,  0.69689922],
       [ 0.32987489,  0.28412848,  0.48758621,  0.07416155,  0.54953658,
         0.21369597,  0.2114049 ,  0.51240569,  0.23486288,  0.59281437,
         0.16127321,  0.46896552],
       [ 0.17610619,  0.32432996,  0.08913043,  0.09416319,  0.09608042,
         0.24856093,  0.12292211,  0.32587627,  0.57733653,  0.18516011,
         0.48637681,  0.61043478]])
```

Cluster centroids for each of the six clusters

The *J*-score can be thought of as the sum of the squared distance between points and cluster centroid for each point and cluster. For an efficient cluster, the *J*-score should be as low as possible. The value of the *J*-score can be found as follows:

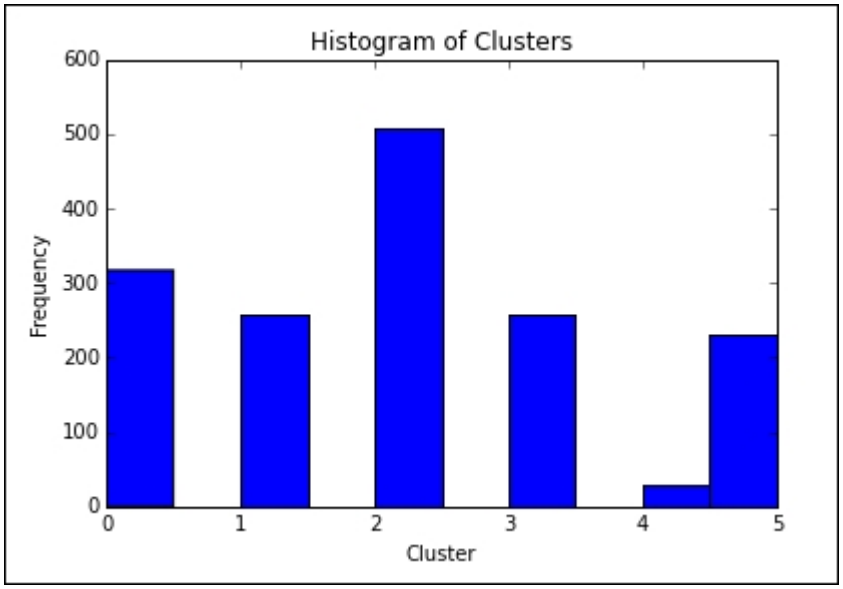
```
model.inertia_
```

The value comes out to be 186.56.

Let us plot a histogram for the `clust` variable to get an idea of the number of observations in each cluster:

```
import matplotlib.pyplot as plt
% matplotlib inline
plt.hist(df_norm['clust'])
plt.title('Histogram of Clusters')
plt.xlabel('Cluster')
plt.ylabel('Frequency')
```

The code shows the following output:



The histogram of cluster labels

As can be observed, the number of wine samples is more uniformly (or rather normally) distributed in this case when compared to the distribution based on the wine quality. This is an improvement from the classification based on the wine quality as it provides us with better segregated and identifiable clusters.

Interpreting the cluster

This clustering can be used to price the wine samples in the same cluster similarly and target the customers who prefer the particular ingredient of wine by marketing them as a different brand having that ingredient as its specialty.

Let us calculate the mean of the composition for each cluster and each component. If you observe the output table, it is exactly similar to the six cluster centroids observed above. This is because the cluster centroids are nothing but the mean of the coordinates of all the observations in a particular cluster:

```
df_norm.groupby('clust').mean()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
clust												
0	0.316574	0.277149	0.304574	0.152025	0.126989	0.380482	0.297856	0.539048	0.435927	0.178904	0.214010	0.456151
1	0.576667	0.203681	0.506459	0.131816	0.130587	0.137228	0.091392	0.652822	0.335151	0.225564	0.290552	0.549416
2	0.263954	0.359043	0.123484	0.091326	0.122836	0.144824	0.107683	0.492112	0.486484	0.155429	0.220165	0.439764
3	0.362900	0.159751	0.423295	0.105474	0.105434	0.163500	0.083861	0.408491	0.418635	0.235575	0.491294	0.696899
4	0.329875	0.284128	0.487586	0.074162	0.549537	0.213696	0.211405	0.512406	0.234863	0.592814	0.161273	0.468966
5	0.176106	0.324330	0.089130	0.094163	0.096080	0.248561	0.122922	0.325876	0.577337	0.185160	0.486377	0.610435

The mean of all the numerical columns for different clusters

The wine quality and taste mainly depends on the quantity of acid, alcohol, and sugar. A few examples of how the information on clustering can be used for efficient marketing and pricing are as follows:

- People from cooler regions prefer wines with higher volatile acid content. So, clusters 2 and 5 can be marketed in cooler (temperature-wise) markets.
- Some people might prefer wine with higher alcohol content, and the wine samples from clusters 3 and 5 can be marketed to them.
- Some connoisseurs trust others' judgment more and they might like to go with professional wine testers' judgments. These kinds of people should be sold the wine samples from clusters 3 and 5 as they have high mean quality.

More information from the wine industry can be combined with this result to form a better marketing and pricing strategy.