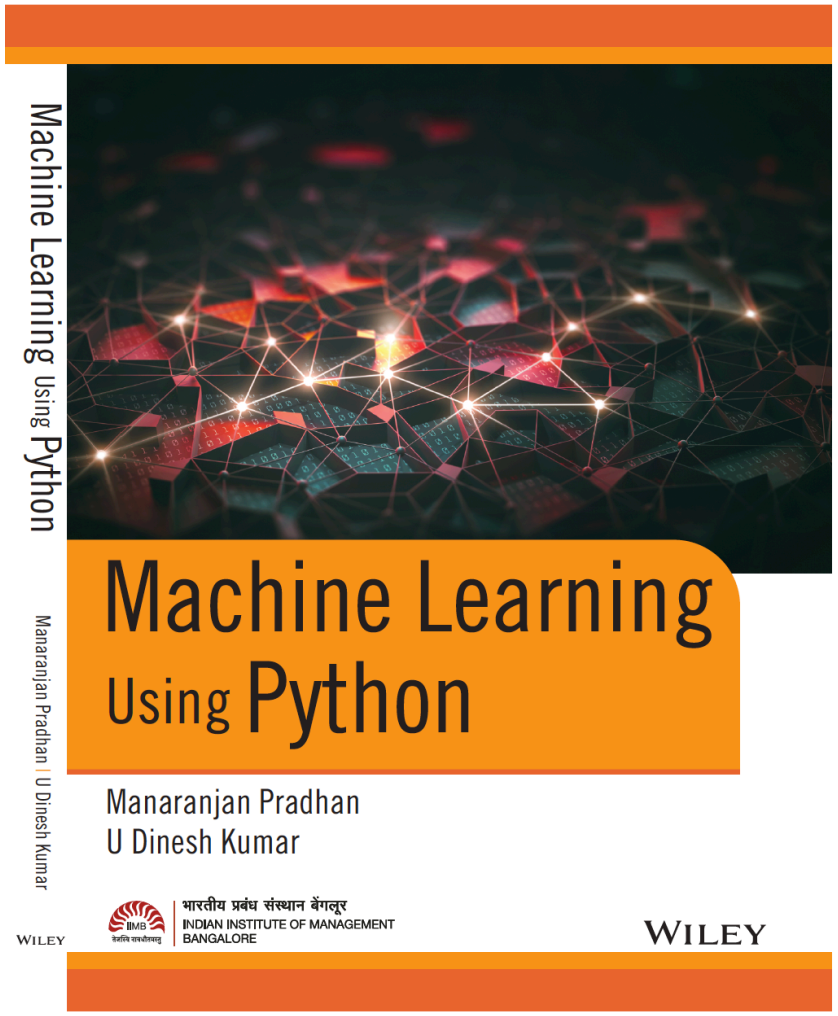


Chapter 07: Clustering

Learning Objectives

- Role and Importance of Clusters in Analytics
- Learn different types of clustering techniques
- Using distance measures such as Euclidean distance in clustering.
- Finding an optimal number of clusters in the data.
- Learn to build clusters using *sklearn* library in Python.

Introduction to Clustering

Clustering is a divide-and-conquer strategy which divides the dataset into homogenous groups which can be further used to prescribe the right strategy for different groups. In clustering, the objective is to ensure that the variation within a cluster is minimized while the variation between clusters is maximized.

How Does Clustering Work?

Clustering algorithms use different distance or similarity or dissimilarity measures to derive different clusters. The type of distance/similarity measure used plays a crucial role in the final cluster formation. Larger distance would imply that observations are far away from one another, whereas higher similarity would indicate that the observations are similar.

Loading Dataset

```
import pandas as pd
customers_df = pd.read_csv("customers.csv")
```

```
customers_df.head(5)
```

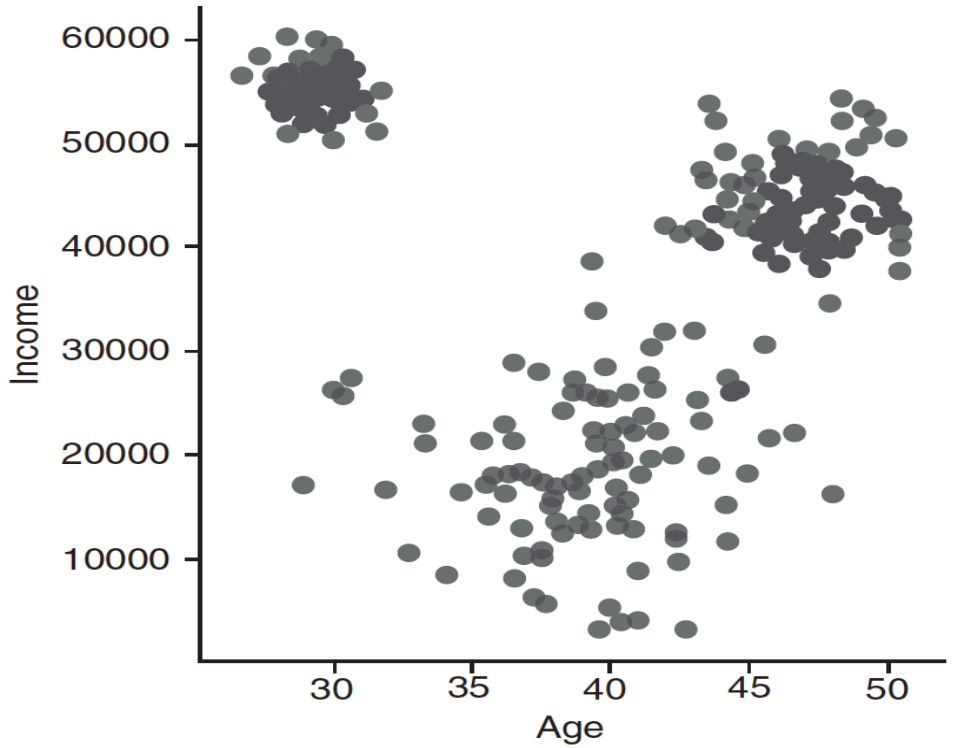
	Income	Age
0	41100.0	48.75
1	54100.0	28.10
2	47800.0	46.75
3	19100.0	40.25
4	18200.0	35.80

Visualizing the Relationship

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
%matplotlib inline
```

```
sn.lmplot("age", "income", data=customers_df, fit_reg = False,
          size = 4);
```

Scatter Plot



Finding Similarities Using Distances

Clustering techniques assume that there are subsets in the data that are similar or homogeneous. One approach for measuring similarity is through distances measured using different metrics. Few distance measures used in clustering are discussed in the following sections.

Euclidean Distance

Euclidean distance between two observations X_1 and X_2 with n features can be calculated as

$$D(X_1, X_2) = \sqrt{\sum (X_{i1} - X_{i2})^2}$$

where X_{i1} and X_{i2} are the values of the i^{th} feature for first observation and second observation, respectively.

Other Distance Measures

- Minkowski Distance
- Jaccard Similarity Coefficient
- Cosine Similarity
- Gower’s Similarity Coefficient

K-Means Clustering

The following steps are used in *K*-means clustering algorithm:

1. Decide the value of *K*.
2. Choose *K* observations from the data that are likely to be in different clusters. Choose observations that are farthest.
3. The *K* observations selected in step 2 are the centroids of those clusters.
4. For remaining observations, find the cluster closest to the centroid. Add the new observation (say observation *j*) to the cluster with the closest centroid. Adjust the centroid after adding a new observation to the cluster. The closest centroid is chosen based upon an appropriate distance measure.
5. Repeat step 4 until all observations are assigned to a cluster.

Creating Clusters

```
from sklearn.cluster import KMeans
clusters = KMeans(3)
clusters.fit(customers_df)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

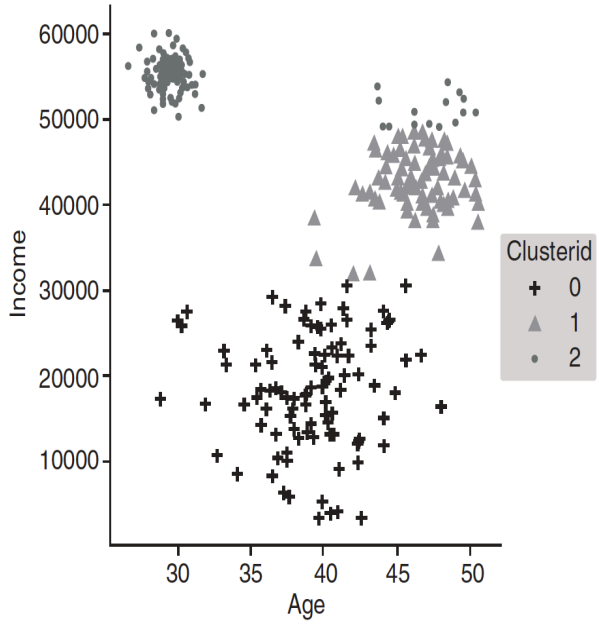
```
customers_df["clusterid"] = clusters.labels_
```

```
customers_df[0:5]
```

	Income	Age	Clusterid
0	41100.0	48.75	1
1	54100.0	28.10	2
2	47800.0	46.75	1
3	19100.0	40.25	0
4	18200.0	35.80	0

Plotting Customers with their Segments

```
markers = ['+','^','.']
sn.lmplot("age", "income",
          data = customers_df,
          hue = "clusterid",
          fit_reg = False,
          markers = markers,
          size = 4);
```



Normalizing Features

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
scaled_customers_df = scaler.fit_transform(customers_df[["age",
                      "income"]])
scaled_customers_df[0:5]
```

```
array([[ 1.3701637,    0.09718548],
       [-1.3791283,    0.90602749],
       [ 1.10388844,   0.51405021],
       [ 0.23849387,   -1.27162408],
       [-0.35396857,   -1.32762083]])
```

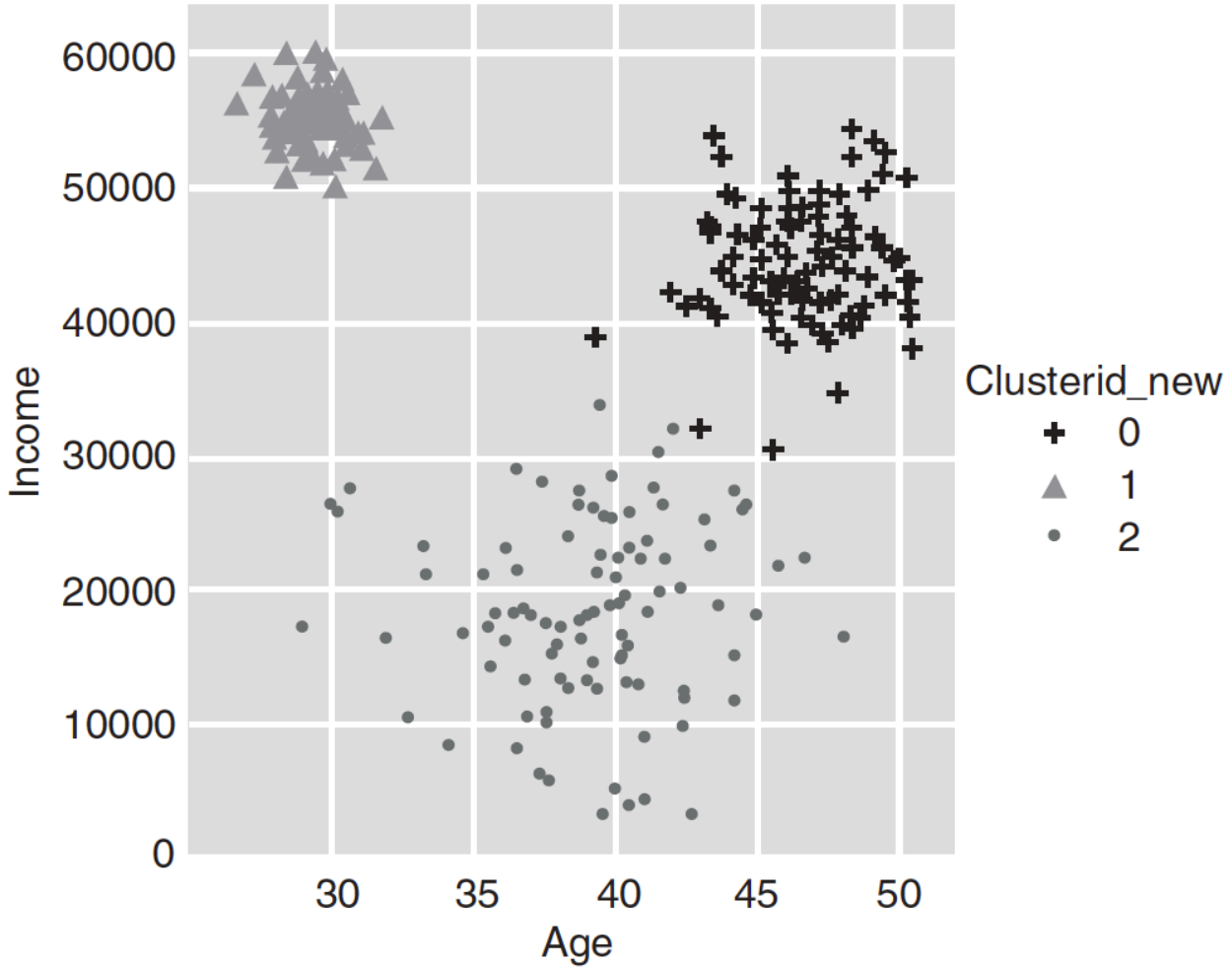
Create Clusters Using Normalized Feature Set

```
from sklearn.cluster import KMeans
clusters_new = KMeans(3, random_state=42)
clusters_new.fit(scaled_customers_df)
customers_df["clusterid_new"] = clusters_new.labels_
```

The new segments created can be plotted using the scatter plot


```
from sklearn.cluster import KMeans
clusters_new = KMeans(3, random_state=42)
clusters_new.fit(scaled_customers_df)
customers_df["clusterid_new"] = clusters_new.labels_
```

Scatter Plot



Cluster Centers

```
clusters.cluster_centers_
```

```
array([[ 0.5361335,  0.96264295,  0.51632566, -0.61618888],
       [-0.92065895, -0.6352664, -0.86290041, -0.33288365],
       [ 0.7690509,  -0.6547531,  0.69314951, 1.89814505]])
```

```
customers_df.groupby('clusterid')['age',
                                  'income'].agg(["mean",
                                                 'std']).reset_index()
```

Clusterid	Age		Income	
	Mean	Std	Mean	Std
0	31.700435	6.122122	54675.652174	2362.224320
1	39.174479	3.626068	18144.791667	6745.241906
2	46.419101	2.289620	43053.932584	3613.769632

Product Segmentation Using Clustering

- A company would like to enter the market with a new beer brand.
- Two things to understand before launching

1. Kinds of existed products in the market.
2. Kinds of segments the products address.

Loading Beer Dataset

```
beer_df = pd.read_csv('beer.csv')
beer_df
```

	Name	Calories	Sodium	Alcohol	Cost
0	Budweiser	144	15	4.7	0.43
1	Schlitz	151	19	4.9	0.43
2	Lowenbrau	157	15	0.9	0.48
3	Kronenbourg	170	7	5.2	0.73
4	Heineken	152	11	5.0	0.77
5	Old_Milwaukee	145	23	4.6	0.28
6	Augsberger	175	24	5.5	0.40
7	Srohs_Bohemian_Style	149	27	4.7	0.42
8	Miller_Lite	99	10	4.3	0.43

Continued

	Name	Calories	Sodium	Alcohol	Cost
9	Budweiser_Light	113	8	3.7	0.40
10	Coors	140	18	4.6	0.44
11	Coors_Light	102	15	4.1	0.46
12	Michelob_Light	135	11	4.2	0.50
13	Becks	150	19	4.7	0.76
14	Kirin	149	6	5.0	0.79
15	Pabst_Extra_Light	68	15	2.3	0.38
16	Hamms	139	19	4.4	0.43
17	Heilemans_Old_Style	144	24	4.9	0.43
18	Olympia_Goled_Light	72	6	2.9	0.46
19	Schlitz_Light	97	7	4.2	0.47

Normalizing the Features

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
scaled_beer_df = scaler.fit_transform(beer_df[['calories',
                                               'sodium',
                                               'alcohol',
                                               'cost']])
```

How Many Clusters Exist?

The following techniques can be used for discovering the possible number of clusters for high-dimensional data

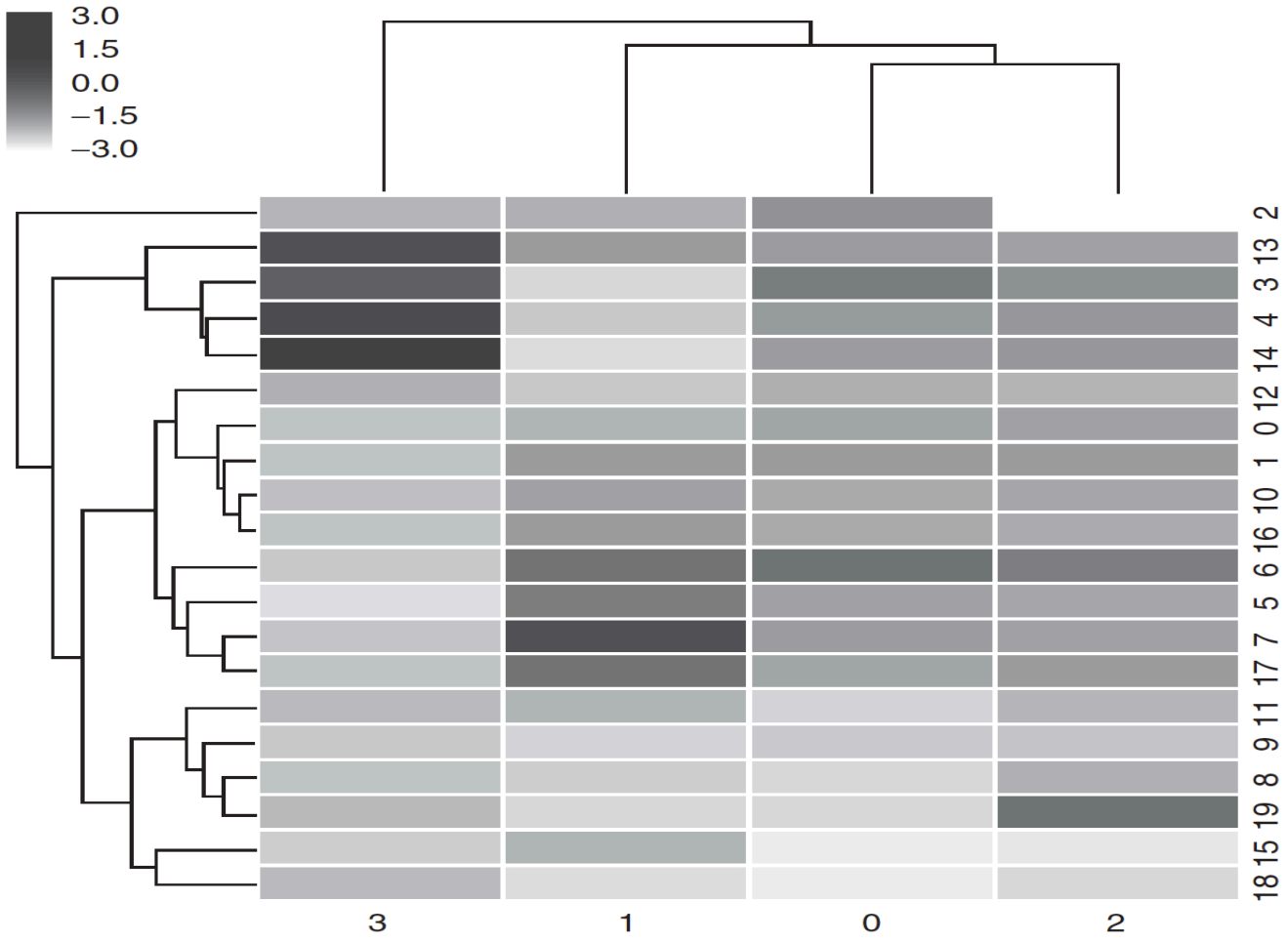
1. Dendogram
2. Elbow Method

Dendogram

A dendrogram is a cluster tree diagram which groups those entities together that are nearer to each other.

```
cmap = sn.cubehelix_palette(as_cmap=True, rot=-.3, light=1)
sn.clustermap(scaled_beer_df, cmap=cmap, linewidths=.2,
              figsize = (8,8));
```

Continued



Continued….

```
beer_df.ix[[10,16]]
```

	Name	Calories	Sodium	Alcohol	Cost
10	Coors	140	18	4.6	0.44
16	Hamms	139	19	4.4	0.43

```
beer_df.ix[[2,18]]
```

	Name	Calories	Sodium	Alcohol	Cost
2	Lowenbrau	157	15	0.9	0.48
18	Olympia_Goled_Light	72	6	2.9	0.46

It can be observed that both the beer brands *Coors* and *Hamms* are very similar across all features. Similarly, brands 2 and 18 seem to be most different as the distance is highest. They are represented on two extremes of the dendrogram.

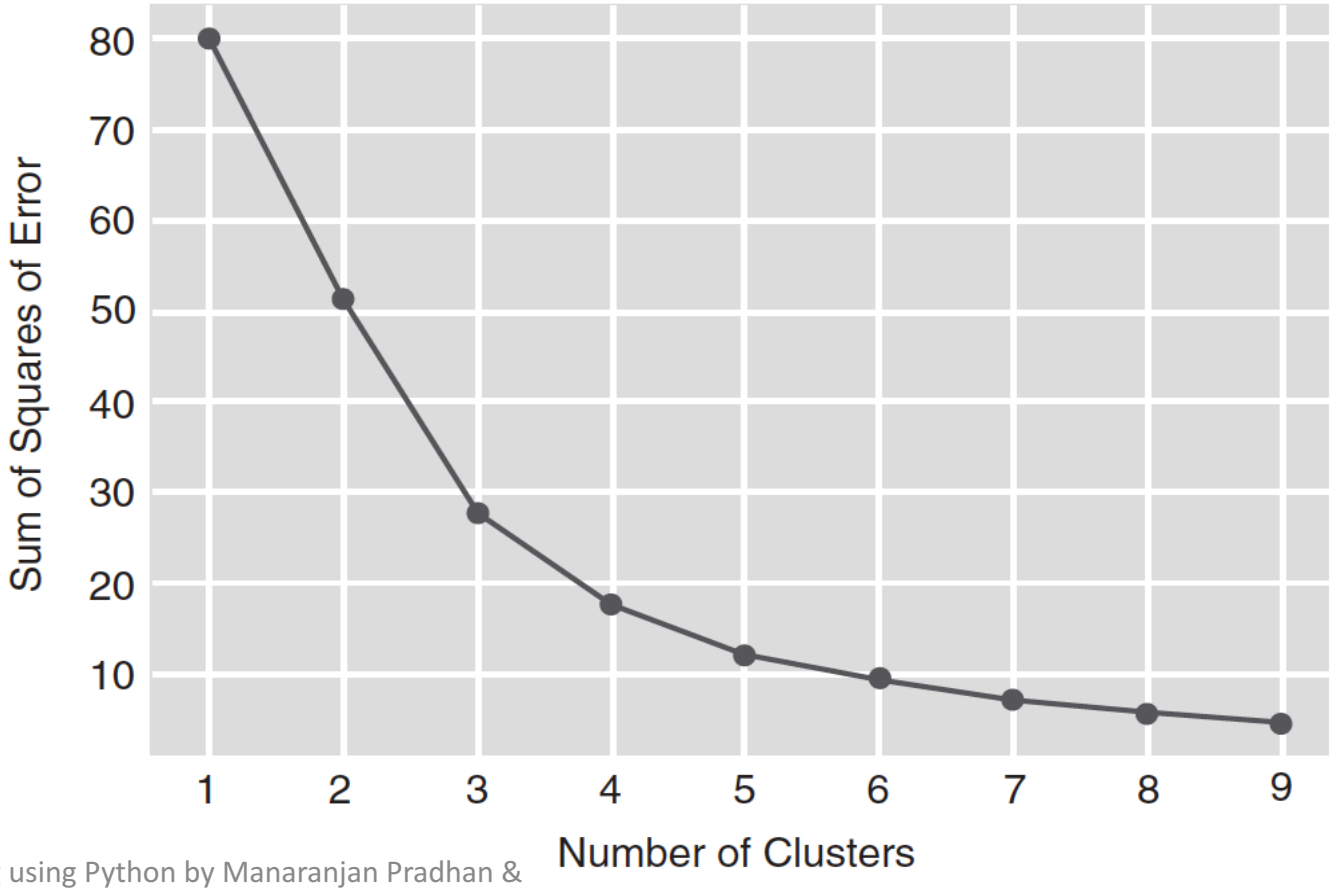
Elbow Curve Method

```
cluster_range = range(1, 10)
cluster_errors = []

for num_clusters in cluster_range:
    clusters = KMeans(num_clusters)
    clusters.fit(scaled_beer_df)
    cluster_errors.append(clusters.inertia_)

plt.figure(figsize=(6,4))
plt.plot(cluster_range, cluster_errors, marker = "o");
```

Elbow Diagram

Then the *cluster_errors* is plotted against the number of clusters

The plot indicates that the elbow point is at 3 indicating there there might be there clusters.



Normalizing the Features

```
scaler = StandardScaler()

scaled_beer_df = scaler.fit_transform (beer_df[['calories',
                                                'sodium',
                                                'alcohol',
                                                'cost']])
```

Creating Clusters

```
k = 3
clusters = KMeans(k, random_state = 42)
clusters.fit(scaled_beer_df)
beer_df[“clusterid”] = clusters.labels_
```

Interpreting the Clusters

Cluster 0:

```
beer_df[beer_df.clusterid == 0]
```

	Name	Calories	Sodium	Alcohol	Cost	Clusterid
0	Budweiser	144	15	4.7	0.43	0
1	Schlitz	151	19	4.9	0.43	0
5	Old_Milwaukee	145	23	4.6	0.28	0
6	Augsberger	175	24	5.5	0.40	0
7	Srohs_Bohemian_Style	149	27	4.7	0.42	0
10	Coors	140	18	4.6	0.44	0
16	Hamms	139	19	4.4	0.43	0
17	Heilemans_Old_Style	144	24	4.9	0.43	0

Continued….

Cluster 1:

```
beer_df[beer_df.clusterid == 1]
```

	Name	Calories	Sodium	Alcohol	Cost	Clusterid
2	Lowenbrau	157	15	0.9	0.48	1
8	Miller_Lite	99	10	4.3	0.43	1
9	Budweiser_Light	113	8	3.7	0.40	1
11	Coors_Light	102	15	4.1	0.46	1
12	Michelob_Light	135	11	4.2	0.50	1
15	Pabst_Extra_Light	68	15	2.3	0.38	1
18	Olympia_Goled_Light	72	6	2.9	0.46	1
19	Schlitz_Light	97	7	4.2	0.47	1

Continued….

Cluster 2:

```
beer_df[beer_df.clusterid == 2]
```

	Name	Calories	Sodium	Alcohol	Cost	Clusterid
3	Kronenbaurg	170	7	5.2	0.73	2
4	Heineken	152	11	5.0	0.77	2
13	Becks	150	19	4.7	0.76	2
14	Kirin	149	6	5.0	0.79	2

Hierarchical Clustering

Hierarchical Clustering is a clustering algorithm which uses the following steps to develop clusters:

1. Start with each data point in a cluster.
2. Find the data points with the shortest distance and merge them to form a cluster.
3. Repeat step 2 until all data points are merged together to form a single cluster.

```
from sklearn.cluster import AgglomerativeClustering
```

```
h_clusters = AgglomerativeClustering(3)
h_clusters.fit(scaled_beer_df)
beer_df["h_clusterid"] = h_clusters.labels_
```

Comparison of K-Means & Hierarchical

```
beer_df[beer_df.h_clusterid == 0]
```

	Name	Calories	Sodium	Alcohol	Cost	Clusterid	h_clusterid
2	Lowenbrau	157	15	0.9	0.48	1	0
8	Miller_Lite	99	10	4.3	0.43	1	0
9	Budweiser_Light	113	8	3.7	0.40	1	0
11	Coors_Light	102	15	4.1	0.46	1	0
12	Michelob_Light	135	11	4.2	0.50	1	0
15	Pabst_Extra_Light	68	15	2.3	0.38	1	0
18	Olympia_Goled_Light	72	6	2.9	0.46	1	0
19	Schlitz_Light	97	7	4.2	0.47	1	0

Continued….

```
beer_df[beer_df.h_clusterid == 1]
```

	Name	Calories	Sodium	Alcohol	Cost	Clusterid	h_clusterid
0	Budweiser	144	15	4.7	0.43	0	1
1	Schlitz	151	19	4.9	0.43	0	1
5	Old_Milwaukee	145	23	4.6	0.28	0	1
6	Augsberger	175	24	5.5	0.40	0	1
7	Srohs_Bohemian_Style	149	27	4.7	0.42	0	1
10	Coors	140	18	4.6	0.44	0	1
16	Hamms	139	19	4.4	0.43	0	1
17	Heilemans_Old_Style	144	24	4.9	0.43	0	1

Continued….

```
beer_df[beer_df.h_clusterid == 2]
```

	Name	Calories	Sodium	Alcohol	Cost	Clusterid	h_clusterid
3	Kronenbourg	170	7	5.2	0.73	2	2
4	Heineken	152	11	5.0	0.77	2	2
13	Becks	150	19	4.7	0.76	2	2
14	Kirin	149	6	5.0	0.79	2	2

Both the clustering algorithms have created similar clusters. Only cluster ids have changed.

Thank You!