K-Means Clustering with Python

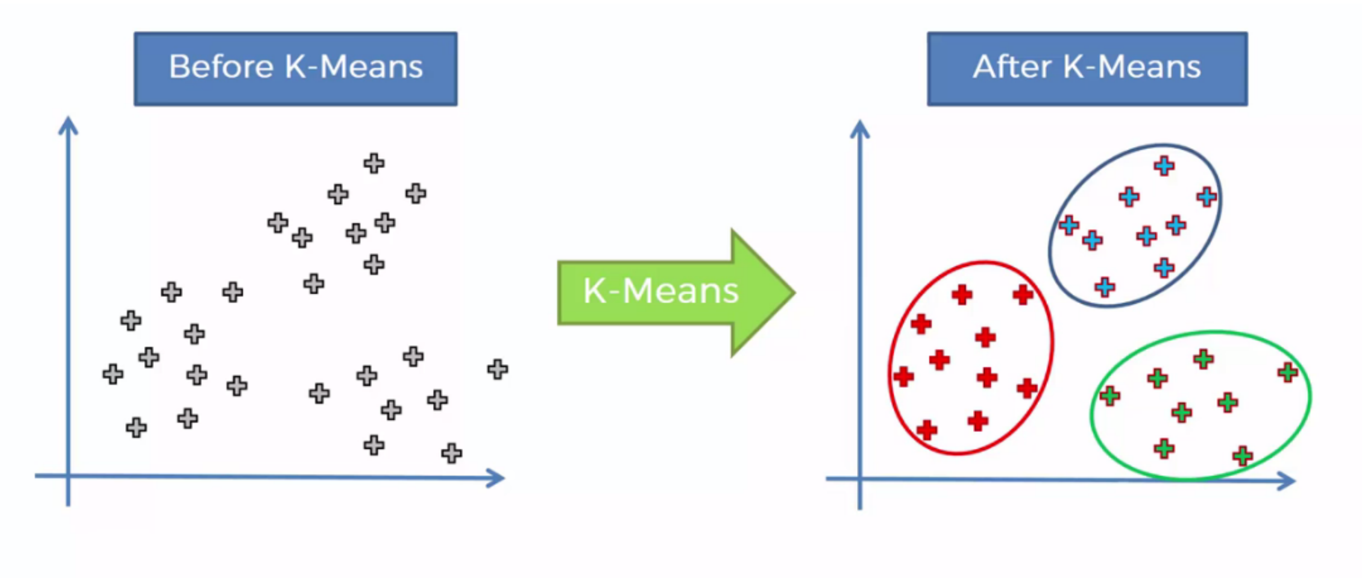
1. Introduction to K-Means Clustering

Machine learning algorithms can be broadly classified into two categories - supervised and unsupervised learning. There are other categories also like semi-supervised learning and reinforcement learning. But, most of the algorithms are classified as supervised or unsupervised learning. The difference between them happens because of presence of target variable. In unsupervised learning, there is no target variable. The dataset only has input variables which describe the data. This is called unsupervised learning.

K-Means clustering is the most popular unsupervised learning algorithm. It is used when we have unlabelled data which is data without defined categories or groups. The algorithm follows an easy or simple way to classify a given data set through a certain number of clusters, fixed apriori. K-Means algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity.

K-Means clustering can be represented diagrammatically as follows:-

K-Means



2. Applications of clustering

- K-Means clustering is the most common unsupervised machine learning algorithm. It is widely used for many applications which include-
    1. Image segmentation

2. Customer segmentation
3. Species clustering
4. Anomaly detection
5. Clustering languages

3. K-Means Clustering intuition

K-Means clustering is used to find intrinsic groups within the unlabelled dataset and draw inferences from them. It is based on centroid-based clustering.

Centroid - A centroid is a data point at the centre of a cluster. In centroid-based clustering, clusters are represented by a centroid. It is an iterative algorithm in which the notion of similarity is derived by how close a data point is to the centroid of the cluster. K-Means clustering works as follows:- The K-Means clustering algorithm uses an iterative procedure to deliver a final result. The algorithm requires number of clusters K and the data set as input. The data set is a collection of features for each data point. The algorithm starts with initial estimates for the K centroids. The algorithm then iterates between two steps:-

> ## 3.1 Data assignment step

Each centroid defines one of the clusters. In this step, each data point is assigned to its nearest centroid, which is based on the squared Euclidean distance. So, if ci is the collection of centroids in set C, then each data point is assigned to a cluster based on minimum Euclidean distance.

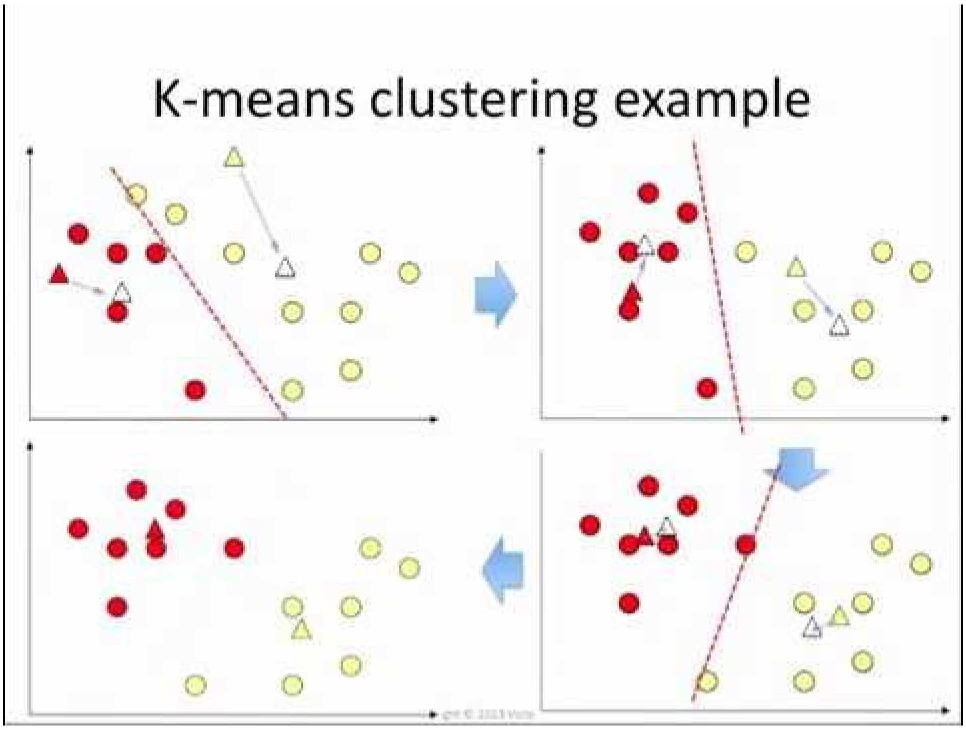
> ## 3.2 Centroid update step

In this step, the centroids are recomputed and updated. This is done by taking the mean of all data points assigned to that centroid's cluster.

The algorithm then iterates between step 1 and step 2 until a stopping criteria is met. Stopping criteria means no data points change the clusters, the sum of the distances is minimized or some maximum number of iterations is reached. This algorithm is guaranteed to converge to a result. The result may be a local optimum meaning that assessing more than one run of the algorithm with randomized starting centroids may give a better outcome.

The K-Means intuition can be represented with the help of following diagram:-

K-Means intuition



4. Choosing the value of K

The K-Means algorithm depends upon finding the number of clusters and data labels for a pre-defined value of K. To find the number of clusters in the data, we need to run the K-Means clustering algorithm for different values of K and compare the results. So, the performance of K-Means algorithm depends upon the value of K. We should choose the optimal value of K that gives us best performance. There are different techniques available to find the optimal value of K. The most common technique is the **elbow method** which is described below.

5. The elbow method

The elbow method is used to determine the optimal number of clusters in K-means clustering. The elbow method plots the value of the cost function produced by different values of K. The below diagram shows how the elbow method works:-

The elbow method

When K increases, average distortion will decrease. Then each cluster will have fewer constituent instances, and the instances will be closer to their respective centroids. However, the improvements in average distortion will decline as K increases. The value of K at which improvement in distortion declines the most is called the elbow, at which we should stop dividing the data into further clusters.

6. Import libraries

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # for data visualization
import seaborn as sns # for statistical data visualization
%matplotlib inline
```

Ignore warnings

```
import warnings

warnings.filterwarnings('ignore')
```

7. Import dataset

Mount the google drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

About the dataset

This dataset has been obtained from UCI ML Repository.
https://archive.ics.uci.edu/ml/datasets/Facebook+Live+Sellers+in+Thailand

Live selling is becoming increasingly popular in Asian countries. Small vendors can now reach a wider audience and connect with many customers.

Research Paper: Nassim Dehouche and Apiradee Wongkitrungrueng. Facebook Live as a Direct Selling Channel, 2018, Proceedings of ANZMAC 2018: The 20th Conference of the Australian and New Zealand Marketing Academy. Adelaide (Australia), 3-5 December 2018.

Load & read the dataset

```
data = '/content/drive/MyDrive/Live.csv'

df = pd.read_csv(data)
```

8. Exploratory data analysis (EDA)

Check shape of the dataset

```
df.shape
```

```
(7050, 16)
```

We can see that there are 7050 instances and 16 attributes in the dataset. In the dataset description, it is given that there are 7051 instances and 12 attributes in the dataset.

So, we can infer that the first instance is the row header and there are 4 extra attributes in the dataset. Next, we should take a look at the dataset to gain more insight about it.

Preview the dataset

```
df.head()
```

	status_id	status_type	status_published	num_reaction
0	246675545449582_1649696485147474	video	4/22/2018 6:00	52
1	246675545449582_1649426988507757	photo	4/21/2018 22:45	15
2	246675545449582_1648730588577397	video	4/21/2018 6:17	22
3	246675545449582_1648576705259452	photo	4/21/2018 2:29	11
4	246675545449582_1645700502213739	photo	4/18/2018 3:22	21

View summary of dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   status_id         7050 non-null   object
 1   status_type       7050 non-null   object
```

```
 2   status_published  7050 non-null   object 
 3   num_reactions     7050 non-null   int64  
 4   num_comments      7050 non-null   int64  
 5   num_shares        7050 non-null   int64  
 6   num_likes         7050 non-null   int64  
 7   num_loves         7050 non-null   int64  
 8   num_wows          7050 non-null   int64  
 9   num_hahas         7050 non-null   int64  
 10  num_sads          7050 non-null   int64  
 11  num_angrys        7050 non-null   int64  
 12  Column1           0 non-null      float64
 13  Column2           0 non-null      float64
 14  Column3           0 non-null      float64
 15  Column4           0 non-null      float64
dtypes: float64(4), int64(9), object(3)
memory usage: 881.4+ KB
```

Check for missing values in dataset

```
df.isnull().sum()
```

```
status_id              0
status_type            0
status_published       0
num_reactions          0
num_comments           0
num_shares             0
num_likes              0
num_loves              0
num_wows               0
num_hahas              0
num_sads               0
num_angrys             0
Column1             7050
Column2             7050
Column3             7050
Column4             7050
dtype: int64
```

We can see that there are 4 redundant columns in the dataset. We should drop them before proceeding further.

Drop redundant columns

```
df.drop(['Column1', 'Column2', 'Column3', 'Column4'], axis=1, inplace=True)
```

Again view summary of dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   status_id         7050 non-null   object
 1   status_type       7050 non-null   object
 2   status_published  7050 non-null   object
 3   num_reactions     7050 non-null   int64 
 4   num_comments      7050 non-null   int64 
 5   num_shares        7050 non-null   int64 
 6   num_likes         7050 non-null   int64 
 7   num_loves         7050 non-null   int64 
 8   num_wows          7050 non-null   int64 
 9   num_hahas         7050 non-null   int64 
 10  num_sads          7050 non-null   int64 
 11  num_angrys        7050 non-null   int64 
dtypes: int64(9), object(3)
memory usage: 661.1+ KB
```

Now, we can see that redundant columns have been removed from the dataset.

We can see that, there are 3 character variables (data type = object) and remaining 9 numerical variables (data type = int64).

View the statistical summary of numerical variables

```
df.describe()
```

	num_reactions	num_comments	num_shares	num_likes	num_loves	num_w
count	7050.000000	7050.000000	7050.000000	7050.000000	7050.000000	7050.000
mean	230.117163	224.356028	40.022553	215.043121	12.728652	1.289
std	462.625309	889.636820	131.599965	449.472357	39.972930	8.719
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	17.000000	0.000000	0.000000	17.000000	0.000000	0.000
50%	59.500000	4.000000	0.000000	58.000000	0.000000	0.000
75%	219.000000	23.000000	4.000000	184.750000	3.000000	0.000
max	4710.000000	20990.000000	3424.000000	4710.000000	657.000000	278.000

There are 3 categorical variables in the dataset. I will explore them one by one.

Explore `status_id` variable

```
# view the labels in the variable

df['status_id'].unique()
```

```
array(['246675545449582_1649696485147474',
       '246675545449582_1649426988507757',
       '246675545449582_1648730588577397', ...,
       '1050855161656896_1060126464063099',
       '1050855161656896_1058663487542730',
       '1050855161656896_1050858841656528'], dtype=object)
```

```
# view how many different types of variables are there

len(df['status_id'].unique())
```

```
6997
```

We can see that there are 6997 unique labels in the `status_id` variable. The total number of instances in the dataset is 7050. So, it is approximately a unique identifier for each of the instances. Thus this is not a variable that we can use. Hence, I will drop it.

Explore `status_published` variable

```
# view the labels in the variable

df['status_published'].unique()
```

```
array(['4/22/2018 6:00', '4/21/2018 22:45', '4/21/2018 6:17', ...,
       '9/21/2016 23:03', '9/20/2016 0:43', '9/10/2016 10:30'],
      dtype=object)
```

```
# view how many different types of variables are there

len(df['status_published'].unique())
```

```
6913
```

Again, we can see that there are 6913 unique labels in the `status_published` variable. The total number of instances in the dataset is 7050. So, it is also a approximately a unique identifier for each of the instances. Thus this is not a variable that we can use. Hence, I will drop it also.

Explore `status_type` variable

```
# view the labels in the variable

df['status_type'].unique()
```

```
array(['video', 'photo', 'link', 'status'], dtype=object)
```

```
# view how many different types of variables are there

len(df['status_type'].unique())
```

```
4
```

We can see that there are 4 categories of labels in the `status_type` variable.

Drop `status_id` and `status_published` variable from the dataset

```
df.drop(['status_id', 'status_published'], axis=1, inplace=True)
```

View the summary of dataset again

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   status_type    7050 non-null   object
 1   num_reactions  7050 non-null   int64 
 2   num_comments   7050 non-null   int64 
 3   num_shares     7050 non-null   int64 
 4   num_likes      7050 non-null   int64 
 5   num_loves      7050 non-null   int64 
 6   num_wows       7050 non-null   int64 
 7   num_hahas      7050 non-null   int64 
 8   num_sads       7050 non-null   int64 
 9   num_angrys     7050 non-null   int64 
dtypes: int64(9), object(1)
memory usage: 550.9+ KB
```

Preview the dataset again

```
df.head()
```

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves
0	video	529	512	262	432	92

We can see that there is 1 non-numeric column `status_type` in the dataset. I will convert i
integer equivalents.

3	photo	111	0	0	111	0

9. Declare feature vector and target variable

```
X = df
y = df['status_type']
```

10. Convert categorical variable into integers

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

X['status_type'] = le.fit_transform(X['status_type'])

y = le.transform(y)
```

View the summary of X

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   status_type    7050 non-null   int64
 1   num_reactions  7050 non-null   int64
 2   num_comments   7050 non-null   int64
 3   num_shares     7050 non-null   int64
 4   num_likes      7050 non-null   int64
 5   num_loves      7050 non-null   int64
 6   num_wows       7050 non-null   int64
 7   num_hahas      7050 non-null   int64
 8   num_sads       7050 non-null   int64
 9   num_angrys     7050 non-null   int64
dtypes: int64(10)
memory usage: 550.9 KB
```

Preview the dataset X

```
X.head()
```

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves
0	3	529	512	262	432	
1	1	150	0	0	150	0
2	3	227	236	57	204	21
3	1	111	0	0	111	0
4	1	213	0	0	204	9

11. Feature Scaling

```
cols = X.columns
```

```
from sklearn.preprocessing import MinMaxScaler

ms = MinMaxScaler()

X = ms.fit_transform(X)
```

```
X = pd.DataFrame(X, columns=[cols])
```

```
X.head()
```

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves
0	1.000000	0.112314	0.024393	0.076519	0.091720	0.140030
1	0.333333	0.031847	0.000000	0.000000	0.031847	0.000000
2	1.000000	0.048195	0.011243	0.016647	0.043312	0.031963
3	0.333333	0.023567	0.000000	0.000000	0.023567	0.000000
4	0.333333	0.045223	0.000000	0.000000	0.043312	0.013699

12. K-Means model with two clusters

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, random_state=0)
```

```
kmeans.fit(X)
```

```
▾ KMeans
KMeans(n_clusters=2, random_state=0)
```

13. K-Means model parameters study

```
kmeans.cluster_centers_
```

```
array([[3.28506857e-01, 3.90710874e-02, 7.54854864e-04, 7.53667113e-04,
        3.85438884e-02, 2.17448568e-03, 2.43721364e-03, 1.20039760e-03,
        2.75348016e-03, 1.45313276e-03],
       [9.54921576e-01, 6.46330441e-02, 2.67028654e-02, 2.93171709e-02,
        5.71231462e-02, 4.71007076e-02, 8.18581889e-03, 9.65207685e-03,
        8.04219428e-03, 7.19501847e-03]])
```

- The KMeans algorithm clusters data by trying to separate samples in n groups of equal variances, minimizing a criterion known as **inertia**, or within-cluster sum-of-squares Inertia, or the within-cluster sum of squares criterion, can be recognized as a measure of how internally coherent clusters are.
- The k-means algorithm divides a set of N samples X into K disjoint clusters C, each described by the mean j of the samples in the cluster. The means are commonly called the cluster **centroids**.
- The K-means algorithm aims to choose centroids that minimize the inertia, or within-cluster sum of squared criterion.

Inertia

- **Inertia** is not a normalized metric.
- The lower values of inertia are better and zero is optimal.
- But in very high-dimensional spaces, euclidean distances tend to become inflated (this is an instance of `curse of dimensionality`).
- Running a dimensionality reduction algorithm such as PCA prior to k-means clustering can alleviate this problem and speed up the computations.
- We can calculate model inertia as follows:-

```
kmeans.inertia_
```

```
237.75726404419646
```

- The lesser the model inertia, the better the model fit.
- We can see that the model has very high inertia. So, this is not a good model fit to the data.

14. Check quality of weak classification by the model

```
labels = kmeans.labels_

# check how many of the samples were correctly labeled
correct_labels = sum(y == labels)

print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.
```

```
Result: 63 out of 7050 samples were correctly labeled.
```

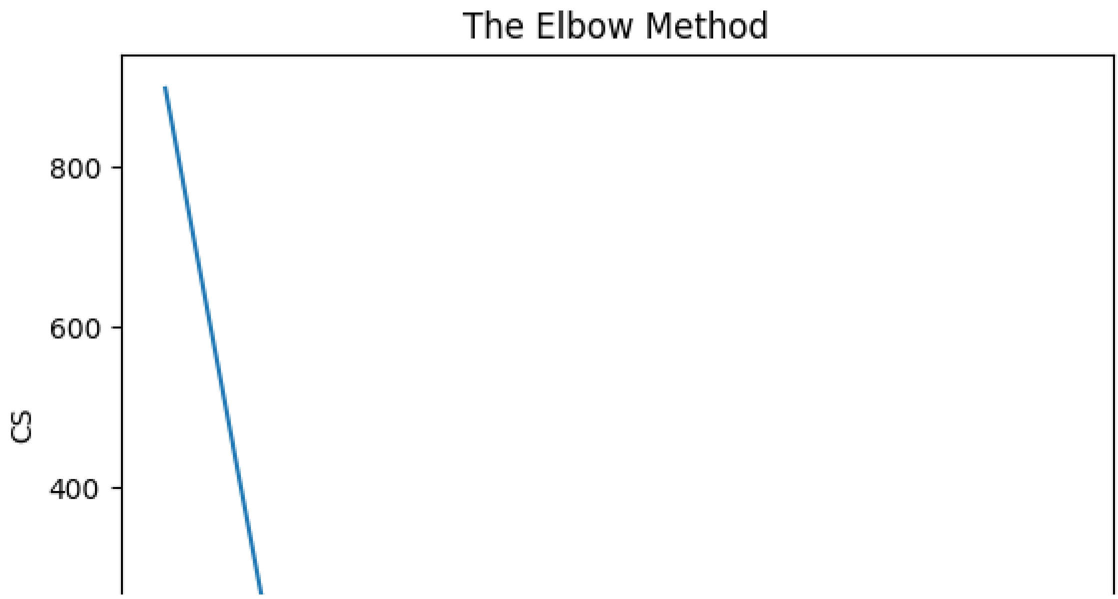
```
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Accuracy score: 0.01
```

We have achieved a weak classification accuracy of 1% by our unsupervised model.

15. Use elbow method to find optimal number of clusters

```
from sklearn.cluster import KMeans
cs = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 1
    kmeans.fit(X)
    cs.append(kmeans.inertia_)
plt.plot(range(1, 11), cs)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('CS')
plt.show()
```



- By the above plot, we can see that there is a kink at k=2.
- Hence k=2 can be considered a good number of the cluster to cluster this data.
- But, we have seen that I have achieved a weak classification accuracy of 1% with k=2.
- I will write the required code with k=2 again for convinience.

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2,random_state=0)

kmeans.fit(X)

labels = kmeans.labels_

# check how many of the samples were correctly labeled

correct_labels = sum(y == labels)

print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.

print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 63 out of 7050 samples were correctly labeled.
Accuracy score: 0.01
```

So, our weak unsupervised classification model achieved a very weak classification accuracy of 1%.

I will check the model accuracy with different number of clusters.

16. K-Means model with different clusters

K-Means model with 3 clusters

```
kmeans = KMeans(n_clusters=3, random_state=0)

kmeans.fit(X)

# check how many of the samples were correctly labeled
labels = kmeans.labels_

correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 138 out of 7050 samples were correctly labeled.
Accuracy score: 0.02
```

K-Means model with 4 clusters

```
kmeans = KMeans(n_clusters=4, random_state=0)

kmeans.fit(X)

# check how many of the samples were correctly labeled
labels = kmeans.labels_

correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 4340 out of 7050 samples were correctly labeled.
Accuracy score: 0.62
```

K-Means model with 5 clusters

```
kmeans = KMeans(n_clusters=5, random_state=0)

kmeans.fit(X)

# check how many of the samples were correctly labeled
labels = kmeans.labels_

correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 82 out of 7050 samples were correctly labeled.
Accuracy score: 0.01
```

The accuracy value is decreasing if we further increase the k value.

17. Results and conclusion

1. In this project, I have implemented the most popular unsupervised clustering technique called **K-Means Clustering**.
2. I have applied the elbow method and find that k=2 (k is number of clusters) can be considered a good number of cluster to cluster this data.
3. I have find that the model has very high inertia of 237.7572. So, this is not a good model fit to the data.
4. I have achieved a weak classification accuracy of 1% with k=2 by our unsupervised model.
5. So, I have changed the value of k and find relatively higher classification accuracy of 62% with k=4.
6. Hence, we can conclude that k=4 being the optimal number of clusters.