In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [3]:
```
import warnings

warnings.filterwarnings('ignore')
```

In [4]:
```
data = 'A:/Live.csv'

df = pd.read_csv(data)
```

Check shape of the dataset

In [5]:
```
df.shape
```

Out[5]:
```
(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

In [6]:
```
df.head()
```

Out[6]:

	status_id	status_type	status_published	num_reactions	num_c
0	246675545449582_1649696485147474	video	4/22/2018 6:00	529	
1	246675545449582_1649426988507757	photo	4/21/2018 22:45	150	
2	246675545449582_1648730588577397	video	4/21/2018 6:17	227	
3	246675545449582_1648576705259452	photo	4/21/2018 2:29	111	
4	246675545449582_1645700502213739	photo	4/18/2018 3:22	213	

View summary of dataset

In [7]:
```
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

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
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

In [9]:
```
df.drop(['Column1', 'Column2', 'Column3', 'Column4'], axis=1, inplace=True)
```

Again view summary of dataset

In [10]:
```
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

In [11]:
```
df.describe()
```

Out[11]:

	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows
count	7050.000000	7050.000000	7050.000000	7050.000000	7050.000000	7050.000000
mean	230.117163	224.356028	40.022553	215.043121	12.728652	1.289362
std	462.625309	889.636820	131.599965	449.472357	39.972930	8.719650
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	17.000000	0.000000	0.000000	17.000000	0.000000	0.000000
50%	59.500000	4.000000	0.000000	58.000000	0.000000	0.000000
75%	219.000000	23.000000	4.000000	184.750000	3.000000	0.000000
max	4710.000000	20990.000000	3424.000000	4710.000000	657.000000	278.000000

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

Explore status_id variable

In [12]:
```
df['status_id'].unique()
```

Out[12]:
```
array(['246675545449582_1649696485147474',
       '246675545449582_1649426988507757',
       '246675545449582_1648730588577397', ...,
       '1050855161656896_1060126464063099',
       '1050855161656896_1058663487542730',
       '1050855161656896_1050858841656528'], dtype=object)
```

In [13]:
```
len(df['status_id'].unique())
```

Out[13]:
```
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

In [14]:
```
df['status_published'].unique()
```

Out[14]:
```
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)
```

In [15]:
```
len(df['status_published'].unique())
```

Out[15]:
```
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

In [16]:
```
df['status_type'].unique()
```

Out[16]:
```
array(['video', 'photo', 'link', 'status'], dtype=object)
```

In [17]:
```
len(df['status_type'].unique())
```

Out[17]:
```
4
```

Drop status_id and status_published variable from the dataset

In [18]:
```
df.drop(['status_id', 'status_published'], axis=1, inplace=True)
```

View the summary of dataset again

In [19]:
```
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

In [20]:
```
df.head()
```

Out[20]:

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_w
0	video	529	512	262	432	92	
1	photo	150	0	0	150	0	
2	video	227	236	57	204	21	
3	photo	111	0	0	111	0	
4	photo	213	0	0	204	9	

In [21]:

```
X = df

y = df['status_type']
```

In [22]:

```
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

In [23]:

```
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   int32
 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: int32(1), int64(9)
memory usage: 523.4 KB
```

Preview the dataset X

In [24]:

```
X.head()
```

Out[24]:

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_w
0	3	529	512	262	432	92	
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	

In [25]:
```
cols = X.columns
```

In [26]:
```
from sklearn.preprocessing import MinMaxScaler

ms = MinMaxScaler()

X = ms.fit_transform(X)
```

In [27]:
```
X = pd.DataFrame(X, columns=[cols])
```

In [28]:
```
X.head()
```

Out[28]:

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_w
0	1.000000	0.112314	0.024393	0.076519	0.091720	0.140030	0.01(
1	0.333333	0.031847	0.000000	0.000000	0.031847	0.000000	0.00(
2	1.000000	0.048195	0.011243	0.016647	0.043312	0.031963	0.00:
3	0.333333	0.023567	0.000000	0.000000	0.023567	0.000000	0.00(
4	0.333333	0.045223	0.000000	0.000000	0.043312	0.013699	0.00(

In [29]:
```
from sklearn.cluster import KMeans

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

kmeans.fit(X)
```

Out[29]:

▾ KMeans

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

In [30]:
```
kmeans.cluster_centers_
```

Out[30]:
```
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:-

In [31]:
```
kmeans.inertia_
```

Out[31]:
```
237.7572640441956
```

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.

In [32]:
```
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.s
```

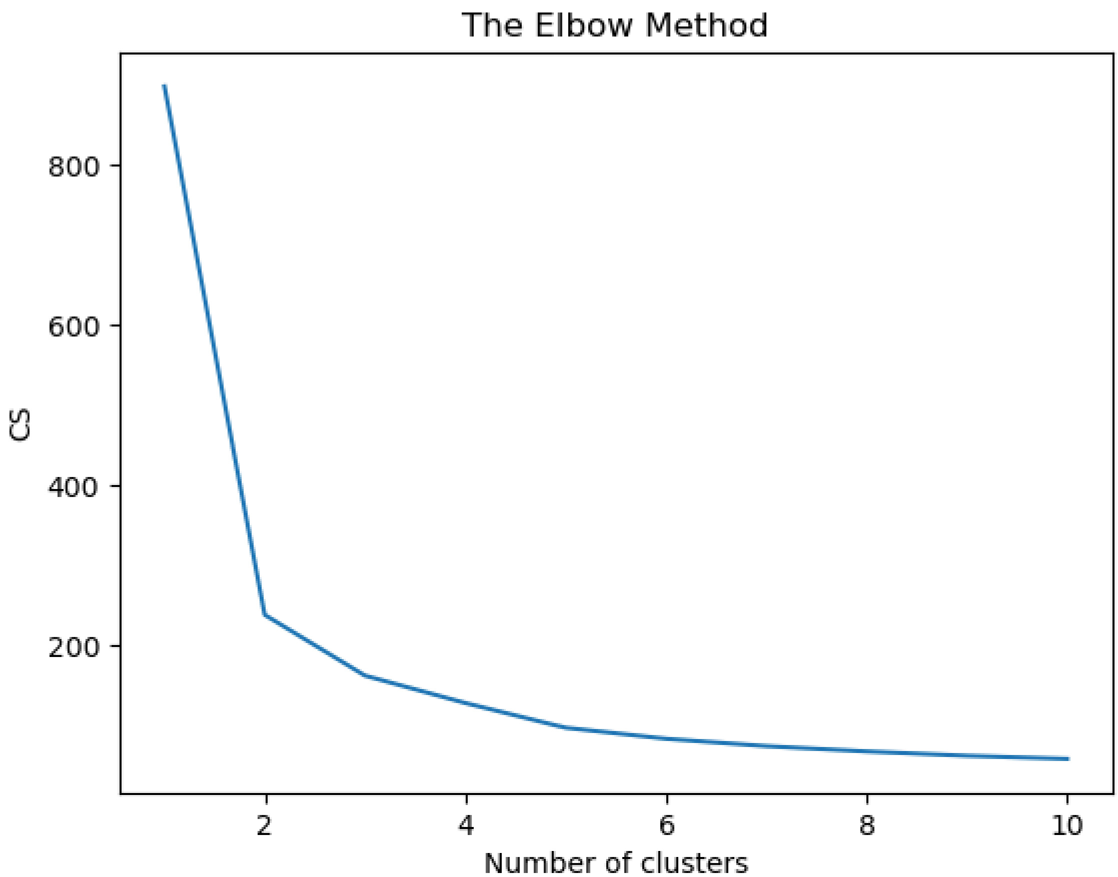
```
Result: 63 out of 7050 samples were correctly labeled.
```

In [33]:
```
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.

In [34]:
```
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 = 10
    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.

In [35]:
```
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.s

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.

K-Means model with 3 clusters

In [36]:
```
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.s
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

In [37]:
```
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.s
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
```

We have achieved a relatively high accuracy of 62% with k=4.

I have implemented the most popular unsupervised clustering technique called K-Means Clustering.

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.

I have find that the model has very high inertia of 237.7572. So, this is not a good model fit to the data.

I have achieved a weak classification accuracy of 1% with k=2 by our unsupervised model.

So, I have changed the value of k and find relatively higher classification accuracy of 62% with k=4.

Hence, we can conclude that k=4 being the optimal number of clusters.

In [ ]:

```

```