```
# Import the necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')

# Import dataset
df=pd.read_csv('Live_20210128.csv')

# Display the first few rows of the DataFrame
df.head()
```

```
   status_id status_type status_published  num_reactions  num_comments
\
0          1       video   4/22/2018 6:00            529           512

1          2       photo  4/21/2018 22:45            150             0

2          3       video   4/21/2018 6:17            227           236

3          4       photo   4/21/2018 2:29            111             0

4          5       photo   4/18/2018 3:22            213             0


   num_shares  num_likes  num_loves  num_wows  num_hahas  num_sads
num_angrys
0         262        432         92         3          1         1
0
1           0        150          0         0          0         0
0
2          57        204         21         1          1         0
0
3           0        111          0         0          0         0
0
4           0        204          9         0          0         0
0
```

Exploratory Data Analysis

```
# Check the shape of the dataset
df.shape
```

```
(7050, 12)
```

Our data has a shape of (7050, 12).

- 7050: This represents the number of rows (observations) in the dataset. Each likely represents a Facebook Live seller in Thailand.
- 12: This represents the number of columns (features) in the dataset. These are used to describe each seller, which could include information like the number of videos posted, average post length, follower count, etc.

```
# Check for missing values in the 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
dtype: int64
```

Data Summary:

- Rows: 7050 (number of sellers)
- Columns: 12 (features describing sellers)
- Numerical: 10 (e.g., number of reactions, comments, shares)
- Categorical: 2 (e.g., status type, publishing status)

```
# View the Summary of the 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   int64
 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(10), object(2)
memory usage: 661.1+ KB
```

We can see that there are no missing values in the dataset

```
# Generate summary statistics for the DataFrame
df.describe()
```

```
         status_id  num_reactions  num_comments   num_shares  num_likes  \
count  7050.000000    7050.000000   7050.000000  7050.000000  7050.000000
mean   3525.500000     230.117163    224.356028    40.022553   215.043121
std    2035.304031     462.625309    889.636820   131.599965   449.472357
min       1.000000       0.000000      0.000000     0.000000     0.000000
25%    1763.250000      17.000000      0.000000     0.000000    17.000000
50%    3525.500000      59.500000      4.000000     0.000000    58.000000
75%    5287.750000     219.000000     23.000000     4.000000   184.750000
max    7050.000000    4710.000000  20990.000000  3424.000000  4710.000000

         num_loves     num_wows    num_hahas     num_sads    num_angrys
count  7050.000000  7050.000000  7050.000000  7050.000000  7050.000000
mean     12.728652     1.289362     0.696454     0.243688     0.113191
std      39.972930     8.719650     3.957183     1.597156     0.726812
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000     0.000000     0.000000
75%       3.000000     0.000000     0.000000     0.000000     0.000000
max     657.000000   278.000000   157.000000    51.000000    31.000000


# Explore 'status_id' variable
len(df['status_id'].unique())

7050
```

Likely unique identifier, not useful for analysis.

```
# Explore 'status_published' 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', '09-10-2016 10:30'],
      dtype=object)
```

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

```
6913
```

Likely unique identifier, not useful for analysis.

```
# Dropping column 'status_id', 'status_published' (not relevant for
analysis)
df.drop(['status_id','status_published'],axis=1,inplace=True)

# Explore 'status_type' variable
# view the labels in the variable

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

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

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

```
4
```

There are 4 status_type:

1. video
2. photo
3. link
4. status

```
# View the summary of the 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
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

   num_wows  num_hahas  num_sads  num_angrys
0         3          1         1           0
1         0          0         0           0
2         1          1         0           0
3         0          0         0           0
4         0          0         0           0
```

```
# Declare feature vector and target variable rephrase
X = df # assigning dataframe

y = df['status_type'] # Selecting target variable

X
```

```
     status_type  num_reactions  num_comments  num_shares
num_likes  \
0          video            529           512         262        432

1          photo            150             0           0        150

2          video            227           236          57        204

3          photo            111             0           0        111

4          photo            213             0           0        204

...          ...            ...           ...         ...        ...
```

```
7045       photo              89              0            0          89

7046       photo              16              0            0          14

7047       photo               2              0            0           1

7048       photo             351             12           22         349

7049       photo              17              0            0          17


      num_loves  num_wows  num_hahas  num_sads  num_angrys
0            92         3          1         1           0
1             0         0          0         0           0
2            21         1          1         0           0
3             0         0          0         0           0
4             9         0          0         0           0
...         ...       ...        ...       ...         ...
7045          0         0          0         0           0
7046          1         0          1         0           0
7047          1         0          0         0           0
7048          2         0          0         0           0
7049          0         0          0         0           0

[7050 rows x 10 columns]

y

0       video
1       photo
2       video
3       photo
4       photo
        ...
7045    photo
7046    photo
7047    photo
7048    photo
7049    photo
Name: status_type, Length: 7050, dtype: object
```

```
# As my 'status_type' variable is categorical, we will convert it into
integers
from sklearn.preprocessing import LabelEncoder

# create an instance of labelEncoder
le=LabelEncoder()

# Transforming the 'status_type' Variable
X['status_type']=le.fit_transform(X['status_type'])
```

```
y = le.transform(y)

# View the summary
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
```

```
# Lets do Feature scaling now
cols=X.columns

# Import the MinMaxScaler class from sklearn.preprocessing for scaling
data.
from sklearn.preprocessing import MinMaxScaler

# Create an instance of the MinMaxScaler class
ms=MinMaxScaler()

# Scale the data in X using the MinMaxScaler instance, fitting the
scaler to the data and transforming it.
X = ms.fit_transform(X)

# Convert the scaled data into a pandas DataFrame with the original
column names.
X = pd.DataFrame(X, columns=[cols])

X
```

```
     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
...          ...           ...          ...        ...       ...
...
7045    0.333333      0.018896     0.000000   0.000000  0.018896
0.000000
7046    0.333333      0.003397     0.000000   0.000000  0.002972
0.001522
7047    0.333333      0.000425     0.000000   0.000000  0.000212
0.001522
7048    0.333333      0.074522     0.000572   0.006425  0.074098
0.003044
7049    0.333333      0.003609     0.000000   0.000000  0.003609
0.000000

      num_wows num_hahas  num_sads num_angrys
0     0.010791  0.006369  0.019608        0.0
1     0.000000  0.000000  0.000000        0.0
2     0.003597  0.006369  0.000000        0.0
3     0.000000  0.000000  0.000000        0.0
4     0.000000  0.000000  0.000000        0.0
...        ...       ...       ...        ...
7045  0.000000  0.000000  0.000000        0.0
7046  0.000000  0.006369  0.000000        0.0
7047  0.000000  0.000000  0.000000        0.0
7048  0.000000  0.000000  0.000000        0.0
7049  0.000000  0.000000  0.000000        0.0

[7050 rows x 10 columns]
```

K-Means Clustering using various clusters

K-Means model parameters study

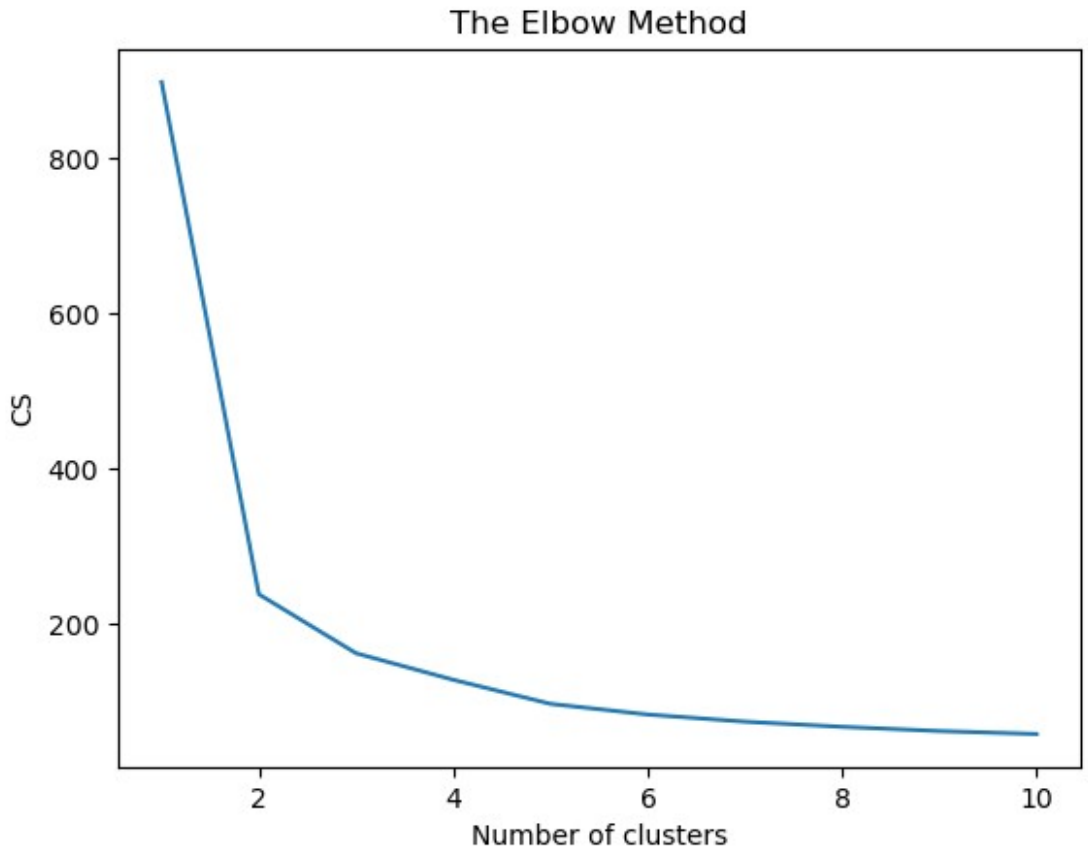
```
kmeans.cluster_centers_

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]])
```

- kmeans.cluster_centers_ is a key output from the K-Means clustering algorithm. It represents the centroids of the clusters.
- In simple terms, it is the mean or average of all the points in each cluster.
- This can help you visualize the clusters and understand the distribution of the data.

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 = 10, random_state = 0)
    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()
```



- kmeans.inertia_ is a property of the KMeans class in scikit-learn that represents the sum of squared distances of samples to their closest cluster center.
- It is a measure of the total variance within the clusters.

- The lower the inertia value, the better the clustering performance.

K-Means model with 2 clusters

```
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.size))

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

```
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.size))
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.size))
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.

- kmeans is an instance of the KMeans class from the scikit-learn library, which has been used to perform the K-Means clustering on the data.
- The labels_ attribute of the kmeans object contains a 1-dimensional NumPy array, where each element represents the cluster assignment for the corresponding data point in the dataset.
- The labels = kmeans.labels_ line assigns this array of cluster assignments to the variable labels.
- This allows you to access the cluster assignments for further analysis or visualization.