Identifying The Key Factors That Impact Engagement Rates During Facebook Live Sales.

The Dataset:

For the exploratory data analysis in this report, I have carefully chosen the data from the UCI Machine Learning Repository. The dataset consists of 7051 observations and 12 attributes, which are in a Comma Separated Values (CSV) format. These observations have been taken from Thai fashion online sellers who use various types of Facebook posts. Each record in the dataset contains information about the time at which live information of sales is posted on Facebook, along with the engagements received. The engagements are regular Facebook interactions such as shares, comments, and emoji reactions, which are described as the traditional “likes”, “love”, “wow”, “haha”, “sad”, and “angry”. For more information about the dataset, you can visit the website. https://archive.ics.uci.edu/dataset/488/facebook+live+sellers+in+thailand.

Explanation and preparation of dataset (Exploratory Data Analysis)

1. Open Jupyter Notebook, either from the Anaconda Prompt or the Anaconda Navigator. Open a new notebook by clicking the New tab button and select Python 3 (ipykernel) to create a new notebook workspace.
2. Run the below codes to import all the required libraries and packages:

```
In [2]: # Fitting in the neccessary packages and libraries

        import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
        from sklearn.decomposition import PCA
        from sklearn.cluster import KMeans
        import seaborn as sns
        from matplotlib import pyplot as pl
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.cm as cm
        from sklearn.preprocessing import StandardScaler

        from sklearn.cluster import KMeans, DBSCAN
        from sklearn.decomposition import PCA

        from sklearn import metrics
```

3. Before we proceed to import our dataset, we should set the filter on warnings to 'ignore'. This means that any warning messages that would normally appear while our code is running will no longer be printed to the workspace and will instead be muted. To do this, run the below code.

```
In [5]: import warnings

        warnings.filterwarnings('ignore')
```

4. Import data using the read_csv() function in the pandas library to read the **Live_20210128.csv** dataset into a pandas data frame.

```
In [6]: # Loading the dataset
        data = pd.read_csv('Live_20210128.csv')
```

5. To get a better understanding of the dataset and the columns in the data, Run the below code:

(a) Data.head()

This function is used to display the first few rows of the Dataset.

```
In [8]: data.head()
```

Out[8]:

	status_id	status_type	status_published	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_ha
0	246675545449582_1649696485147474	video	4/22/2018 6:00	529	512	262	432	92	3	
1	246675545449582_1649426988507757	photo	4/21/2018 22:45	150	0	0	150	0	0	
2	246675545449582_1648730588577397	video	4/21/2018 6:17	227	236	57	204	21	1	
3	246675545449582_1648576705259452	photo	4/21/2018 2:29	111	0	0	111	0	0	
4	246675545449582_1645700502213739	photo	4/18/2018 3:22	213	0	0	204	9	0	

(b) Data.shape()

This function is used to retrieve the shape of the dataset.

```
In [10]: data.shape
Out[10]: (7050, 16)
```

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

(c) Data.info()

This function returns a concise summary of the dataset such as data types, non-null values, and memory usage.

```
In [12]: data.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
```

(d) missing = data.isnull().sum()

This function checks for missing values in the dataset.

```
In [13]: missing = data.isnull().sum()

         print(missing)
```

```
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 did not encounter any missing values in the dataset as it was extracted using Facebook API and anonymised according to the Facebook Platform Policy for Developers. Additionally, there are four unnecessary columns in the dataset which need to be removed before further exploration. To do this, run the code below.

```
In [14]: data.drop(['Column1', 'Column2', 'Column3', 'Column4'], axis=1, inplace=True)
```

Again, run the .info() code to view the changes.

```
In [16]: data.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 unnecessary columns have been removed from the dataset.

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

(e) Data.describe()

This function generates a statistical summary of numerical variables in the dataset.

```
In [17]: data.describe()
```

Out[17]:

	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
count	7050.000000	7050.000000	7050.000000	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	0.696454	0.243688	0.113191
std	462.625309	889.636820	131.599965	449.472357	39.972930	8.719650	3.957183	1.597156	0.726812
min	0.000000	0.000000	0.000000	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	0.000000	0.000000	0.000000
50%	59.500000	4.000000	0.000000	58.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	219.000000	23.000000	4.000000	184.750000	3.000000	0.000000	0.000000	0.000000	0.000000
max	4710.000000	20990.000000	3424.000000	4710.000000	657.000000	278.000000	157.000000	51.000000	31.000000

There are 3 categorical variables in the dataset. Let's explore them one after the other by executing the following codes:

```
In [18]: # labels in the variable

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

```
Out[18]: array(['246675545449582_1649696485147474',
                 '246675545449582_1649426988507757',
                 '246675545449582_1648730588577397', ...,
                 '1050855161656896_1060126464063099',
                 '1050855161656896_1058663487542730',
                 '1050855161656896_1050858841656528'], dtype=object)
```

```
In [19]: # view how many different types of variables are there

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

```
Out[19]: 6997
```

There are 6997 different labels present in the 'status_id' column of the dataset. As the total number of instances in the dataset is 7050, it means that each instance has a unique identifier, which is not useful for analysis purposes. Therefore, we cannot use this variable and will remove it from the dataset.

```
In [20]: # labels in the variable

         data['status_published'].unique()

Out[20]: 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 [21]: # view how many different types of variables are there

         len(data['status_published'].unique())

Out[21]: 6913
```

Again, it's worth noting that the variable 'status_published' contains 6913 unique labels and there are a total of 7050 instances in the dataset. As such, it can be considered a unique identifier for each instance. However, it is not a variable that can be used for analysis, and therefore, we'll remove it from the dataset.

```
In [22]: # labels in the variable

         data['status_type'].unique()

Out[22]: array(['video', 'photo', 'link', 'status'], dtype=object)

In [23]: # view how many different types of variables are there

         len(data['status_type'].unique())

Out[23]: 4
```

From the variable 'status_type', we can observe that there are four distinct categories of labels. We can now proceed to Drop 'status_id and status_published' variable from the dataset Using this code below.

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

We should review the data once more to ensure that all the final changes have been made.

```
In [19]: data.head()
```

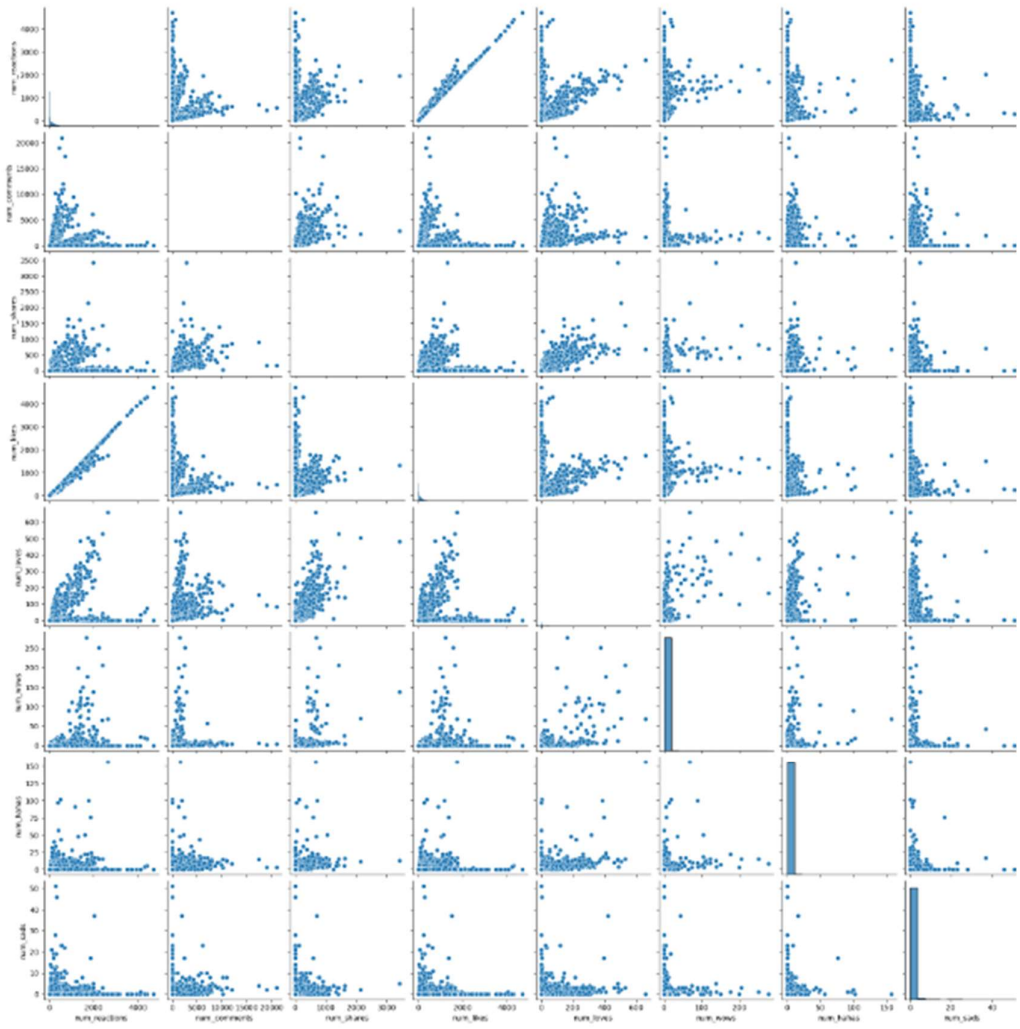
Out[19]:

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

To gain more insight into the dataset, we can use Seaborn's pairplot() function to explore numerical variables. This function generates scatterplots for each pair of variables and a histogram for each variable. By looking at the scatterplot, we can see how variables such as number_shares, number_likes, number_reactions, number_hahas, number_wows, number_sads, number_comments, and num_sads are correlated with each other. To generate this output, simply execute the code below.

```
In [22]: data = data.iloc[:,1:9]
         sns.pairplot(data)

Out[22]: <seaborn.axisgrid.PairGrid at 0x21dbd802490>
```

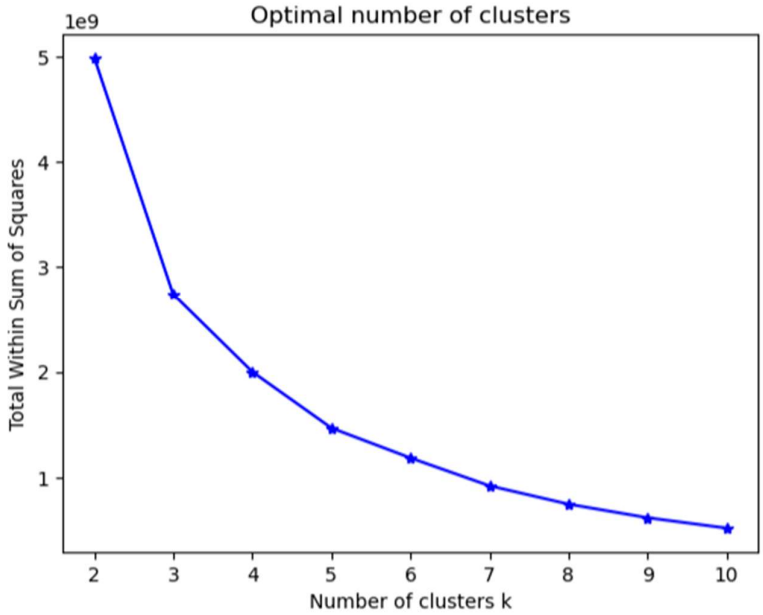


Next, we will determine the optimal number of clusters (k) in k-means clustering using the elbow method. In this method, we will perform k-means clustering on the dataset for a range of values of k from 2 to 10. For each value of k, we will calculate the total within-cluster sum of squares (WSS), which measures the internal coherence of the clusters. We will then create a plot of this WSS value against the number of clusters using Matplotlib. The 'elbow' point on the plot is the point where the rate of decrease in WSS slows down, indicating that increasing the number of clusters beyond that point doesn't significantly improve the model. To do this, execute the following code.

```
In [23]: # Using elbow method to find optimal number of clusters

wss = []

K = range(2,11)
for k in K:
    kmeans = KMeans(n_clusters=k, random_state=123)
    kmeans = kmeans.fit(data)
    wss.append(kmeans.inertia_)
plt.plot(K, wss, "b*-")
plt.xlabel("Number of clusters k")
plt.ylabel("Total Within Sum of Squares")
plt.title("Optimal number of clusters")
plt.show()
```



Having identified the optimal number of clusters, we can proceed to implement k-means clustering on the dataset to reduce the dimensionality of the data using Principal Component Analysis (PCA) and visualise the clustered data in a scatter plot using the code below.

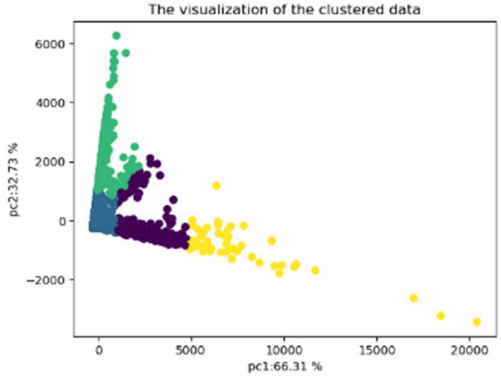
```
In [24]: # Visualizing K-means

pca = PCA()
pca_data = pca.fit_transform(data)
pca_data = pd.DataFrame(pca_data, columns=["pc"+str(i+1) for i in range(len(data.columns))])


pca_data1 = pca_data[["pc1","pc2"]].copy()
data1 = data.copy() # data1 is created, i do not want to change the original data as adding the cluster column.

kmeans = KMeans(n_clusters=4, random_state=2464863)
data1["clusters"] = kmeans.fit_predict(data1)

plt.scatter(pca_data1["pc1"], pca_data1["pc2"], c=data1.clusters)
plt.title("The visualization of the clustered data")
plt.xlabel("pc1:" + "{:.2f}".format(pca.explained_variance_ratio_[0] * 100) + " %")
plt.ylabel("pc2:" + "{:.2f}".format(pca.explained_variance_ratio_[1] * 100) + " %")
plt.show()
```

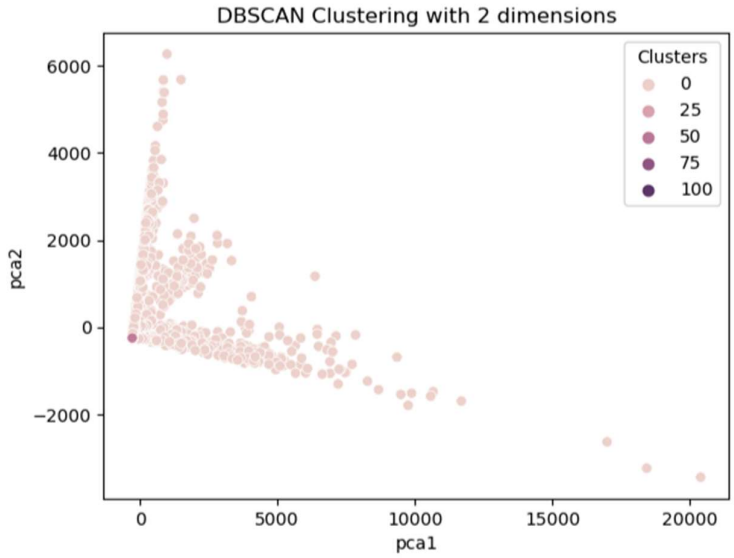


To determine the best clustering method for our dataset, we will be implementing DBSCAN, which is a density-based method. This method requires us to set two parameters- epsilon and the minimum number of points. If not specified, the algorithm will use default values. The best part about this method is that we don't have to predefine the number of clusters. However, we do need to identify a value for epsilon and a minimum number of points. We can instantiate a DBSCAN using the fit_predict() method as follows:

```
In [28]: # DBSCAN clustering
         clustering_kmeans = DBSCAN()
         data['Clusters'] = clustering_kmeans.fit_predict(data)

In [29]: reduced_data = PCA(n_components=2).fit_transform(data)
         results = pd.DataFrame(reduced_data,columns=['pca1','pca2'])

         sns.scatterplot(x="pca1", y="pca2", hue=data['Clusters'], data=results)
         plt.title('DBSCAN Clustering with 2 dimensions')
         plt.show()
```



Results analysis and discussion

After performing two clustering methods on our dataset, it became apparent that the k-means algorithm was more effective than the density-based method. This was due to its ability to analyse and visualise the correlation of numerical variables with the help of principal component analysis (PCA). Therefore, it is the most suitable algorithm for our dataset. Additionally, the dataset we used for this analysis is licensed under a Creative Commons Attribution 4.0 International (CC 4.0) license. which permits the sharing and adaptation of the datasets for any purpose, provided that appropriate credit is given.

Conclusions

We conclude our report with the following findings and suggestions. First, our analysis of the live-streaming sales process indicates that, except for follow-up, which some sellers may completely implement, most sellers adhere to the same multi-step procedure. In today's highly competitive market, follow-up plays a vital role in fostering customer confidence in new clients and preserving continuous communication with existing ones.

Some of the limitations of this report can be addressed in further studies. which are descriptive and exploratory. To identify combinations of sales tactics appropriate for various circumstances, future studies can investigate the relationship between sales approaches/strategies and other relationship outcomes.