I. Module imports, data input and cleaning

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples,silhouette_score
import matplotlib.cm as cm

%matplotlib inline
```

```
%pwd
```

```
'C:\\Users\\ASUS\\OneDrive\\NEU\\ADS\\L3'
```

```
!ls
```

```
'ls' is not recognized as an internal or external command,
operable program or batch file.
```

```
'''To find out more about this online retail data, please visit
https://archive.ics.uci.edu/ml/datasets/Online+Retail'''

df = pd.read_excel("Online Retail.xlsx")
print(df.shape) #shows rows and columns
df.head(3)
```

```
(541909, 8)
```

	InvoiceNo	StockCode	Description	Quantity	Invoic
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
```

```
 0   InvoiceNo    541909 non-null  object
 1   StockCode    541909 non-null  object
 2   Description  540455 non-null  object
 3   Quantity     541909 non-null  int64
 4   InvoiceDate  541909 non-null  datetime64[ns]
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1+ MB
```

```
'''Calculate percentage null values for each column or feature'''

null_vals = df.isnull().sum()/len(df)*100
null_vals = pd.DataFrame(null_vals)
null_vals.reset_index(inplace = True)
null_vals.columns = ["Feature","Percent missing"]
plt.figure(figsize = (8,10))
plt.xticks(rotation=45)
sns.barplot(x = "Percent missing",y ="Feature",data = null_vals,orient = "h")
```

```
<AxesSubplot: xlabel='Percent missing', ylabel='Feature'>
```


```
'''Drop rows with any null values'''

df1 = df.dropna(subset = ["CustomerID","Description"])
print(df.shape,"diff", df1.shape)
```

```
(541909, 8) diff (406829, 8)
```

```
'''Drop duplicated rows'''

df2 = df1.drop_duplicates()
print(df2.shape)
df2.head(2)
```

```
(401604, 8)
```

	InvoiceNo	StockCode	Description	Quantity	Invoic
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08

```
'''Select columns you need'''

df3 = df2 [['CustomerID','InvoiceDate','InvoiceNo','Quantity','UnitPrice']]
print(df3.shape)
df3.head(2)
```

```
(401604, 5)
```

	CustomerID	InvoiceDate	InvoiceNo	Quantity	UnitPrice
0	17850.0	2010-12-01 08:26:00	536365	6	2.55
1	17850.0	2010-12-01 08:26:00	536365	6	3.39

```
'''Create a total price column by multiplying quantity with unit price'''

df3['TotalPrice'] = df3['Quantity'] * df3['UnitPrice']
print(df3.shape)
df3.head(2)

# feature engineering - added more columns in the available data
```

```
(401604, 6)
C:\Users\ASUS\AppData\Local\Temp\ipykernel_20328\3978668798.py:3: SettingWithCop
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
  df3['TotalPrice'] = df3['Quantity'] * df3['UnitPrice']
```

	CustomerID	InvoiceDate	InvoiceNo	Quantity	UnitPrice	TotalPrice
0	17850.0	2010-12-01 08:26:00	536365	6	2.55	15.30
1	17850.0	2010-12-01 08:26:00	536365	6	3.39	20.34

```
'''Print out earliest and latest dates in the data'''

print('Min:{}; Max:{}'.format(df3["InvoiceDate"].min(), df3["InvoiceDate"].max()))
```

```
Min:2010-12-01 08:26:00; Max:2011-12-09 12:50:00
```

```
'''Create a reference point for the analysis'''

current_date = dt.datetime(2011,12,10)
current_date
```

```
datetime.datetime(2011, 12, 10, 0, 0)
```

```
'''Calculate the aggregates" recency, frequency and, monetary. Recency tells you how r
last transaction for each customer, frequency tells you how frequently does a customer
monetary tells you the total shopping spending for each customer'''

df4 = df3.groupby(['CustomerID']).agg({ 'InvoiceDate': lambda x: (current_date - x.max
'TotalPrice': 'sum'})
df4.rename(columns = {'InvoiceDate': 'Recency', 'InvoiceNo': 'Frequency','TotalPrice':
print(df4.shape)
df4.head(3)
```

```
(4372, 3)
```

	Recency	Frequency	Monetary
CustomerID			

```
'''Remove rows with any zero values. This is to facilitate downstream pre-processing a

df5 = df4[(df4 > 0).all(1)]
print(df5.shape)
```

```
(4284, 3)
```

II. Data Pre-processing

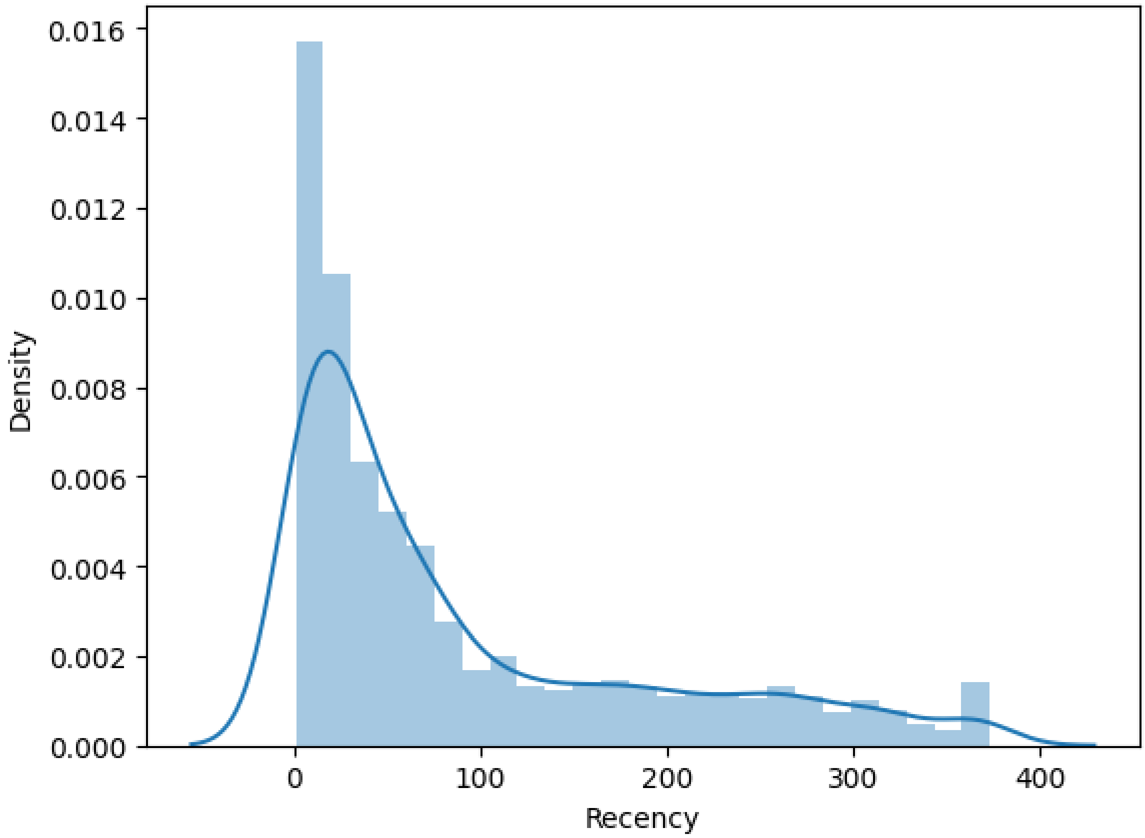
```
'''The K-means clustering algorithm has a few key assumptions about the data: (1) data
(2) features have the same mean and, (3) features have the same variance'''

df5.describe()
```

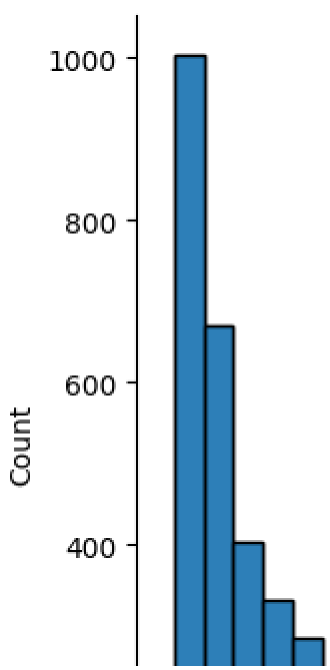
	Recency	Frequency	Monetary
count	4284.000000	4284.000000	4.284000e+03
mean	90.673436	90.187675	1.802891e+03
std	99.212825	217.749044	7.226246e+03
min	1.000000	1.000000	1.776357e-15
25%	17.000000	18.000000	2.988725e+02
50%	50.000000	42.000000	6.467200e+02
75%	140.000000	99.000000	1.596963e+03
max	373.000000	7812.000000	2.794890e+05

Looks like the means and standard deviations are so different. So, we need to transform the data to meet the requirements

```
'''Are the data dimensions skewed?'''

sns.distplot(df5['Recency'])  #deprecated
sns.displot(df5['Recency'])
sns.histplot(df5['Recency'])
```

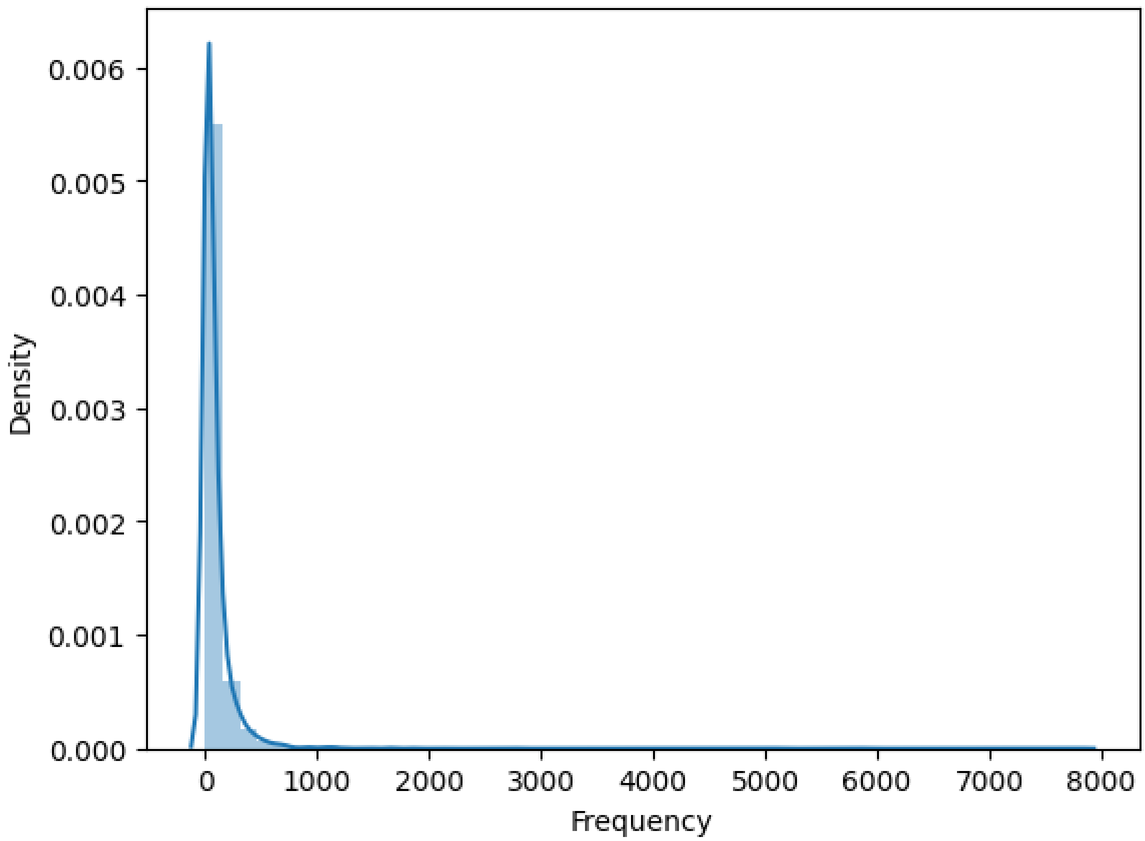
```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_20328\1653390121.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df5['Recency'])  #deprecated
<AxesSubplot: xlabel='Recency', ylabel='Count'>
```





```
sns.distplot(df5['Frequency'])
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_20328\590665190.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df5['Frequency'])
<AxesSubplot: xlabel='Frequency', ylabel='Density'>
```




```
sns.distplot(df5['Monetary'])
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_20328\1026065201.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df5['Monetary'])
<AxesSubplot: xlabel='Monetary', ylabel='Density'>
```

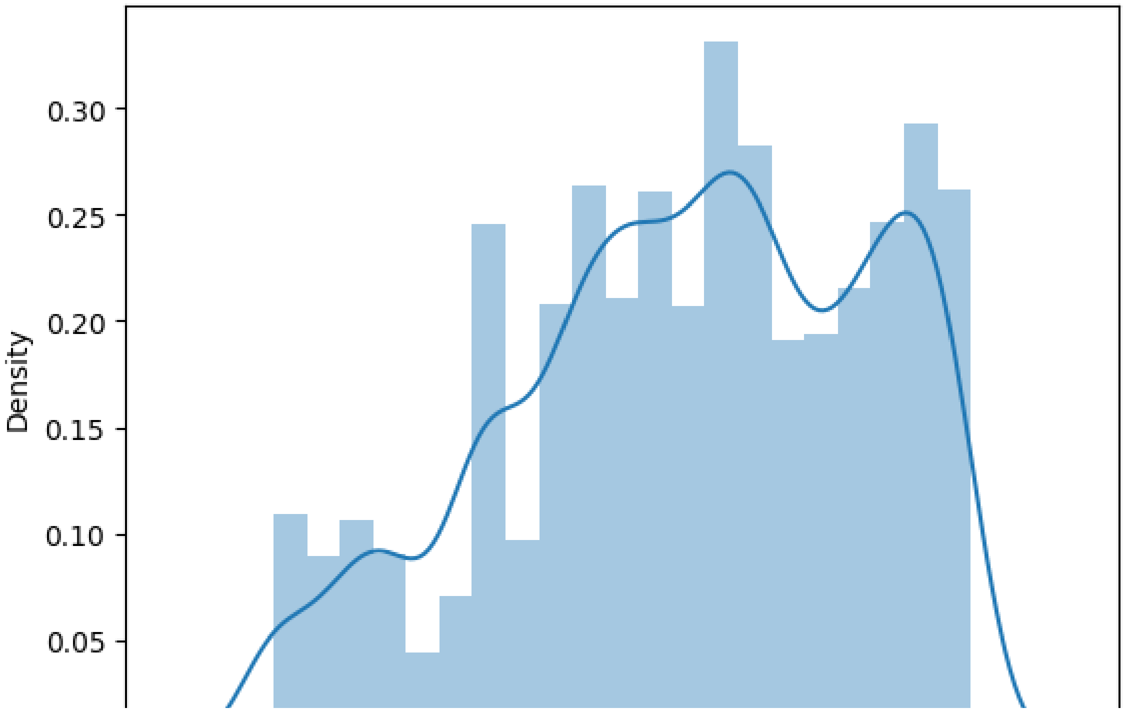

```
'''Looks like the data is skewed. Maybe monetary is not, but the other two definitely
transform the data to remove the skew. Add a constant to offset any negative values. '

df6 = (np.log(df5 + 1))
print(df6.shape)
df6.head(3)
```

```
(4284, 3)
```

CustomerID	Recency	Frequency	Monetary
12347.0	1.098612	5.209486	8.368925
12348.0	4.330733	3.465736	7.494564
12349.0	2.944439	4.304065	7.472245

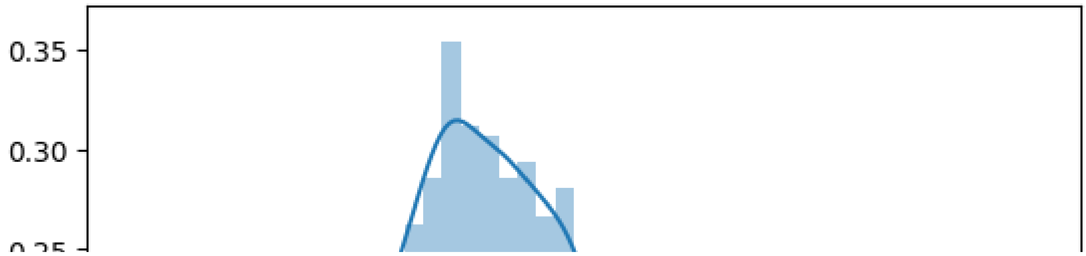
```
'''Has log transfors made any difference?'''

sns.distplot(df6['Recency'])
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_20328\4109149228.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df6['Recency'])
<AxesSubplot: xlabel='Recency', ylabel='Density'>
```




```
'''Has log transfors made any difference?'''

sns.distplot(df6['Frequency'])
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_20328\2722544356.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df6['Frequency'])
<AxesSubplot: xlabel='Frequency', ylabel='Density'>
```




```
'''Has log transfors made any difference?'''

sns.distplot(df6['Monetary'])
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_20328\2255834918.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
```

It has made the data look more normal !

```
'''Do scaling to make sure all dimensions have equal mean and variance'''

scaler = StandardScaler()
scaler.fit(df6)
df7 = pd.DataFrame(scaler.transform(df6))
df7.columns = df6.columns
df7.describe()
```

	Recency	Frequency	Monetary
count	4.284000e+03	4.284000e+03	4.284000e+03
mean	2.107462e-16	-2.522622e-16	-9.511014e-17
std	1.000117e+00	1.000117e+00	1.000117e+00
min	-2.257018e+00	-2.483190e+00	-5.244693e+00
25%	-6.530189e-01	-6.529482e-01	-6.740816e-01
50%	1.072545e-01	1.105753e-02	-5.693016e-02
75%	8.496283e-01	6.971834e-01	6.667453e-01
max	1.561752e+00	4.240429e+00	4.805312e+00

Monetary

III. K-means clustering

```
k_means = KMeans(n_clusters=2, random_state=1)
```

```
'''Let's see how this works:
Apply k-means on the preprocessed data and get cluster labels for each row'''

k_means.fit(df7)
clus_labels = k_means.labels_
```

```
'''Get cluster characteristics. Since we are interested in the original values,
we use the non-log transformed, non-standardized dataframe'''
```

```
df5_clus2 = df5.assign(Cluster = clus_labels)
print(df5_clus2.shape)
df5_clus2.head(2)
```

```
(4284, 4)
```

	Recency	Frequency	Monetary	Cluster
CustomerID				
12347.0	2	182	4310.00	0
12348.0	75	31	1797.24	1

```
df5_clus2.groupby(['Cluster']).agg({ 'Recency': 'mean',
'Frequency': 'mean',
'Monetary': ['mean', 'count'],
}).round(0)
```

	Recency	Frequency	Monetary	
	mean	mean	mean	count
Cluster				
0	30.0	171.0	3521.0	1905
1	139.0	25.0	427.0	2379

'''That sounds cool, but how do we determine the optimal value of K? Who said 2 clusters are optimal? Think hyperparameters from supervised learning. There are at least two ways to find the optimal number of clusters: (1) Elbow plot and, (2) Silhoutte plot'''

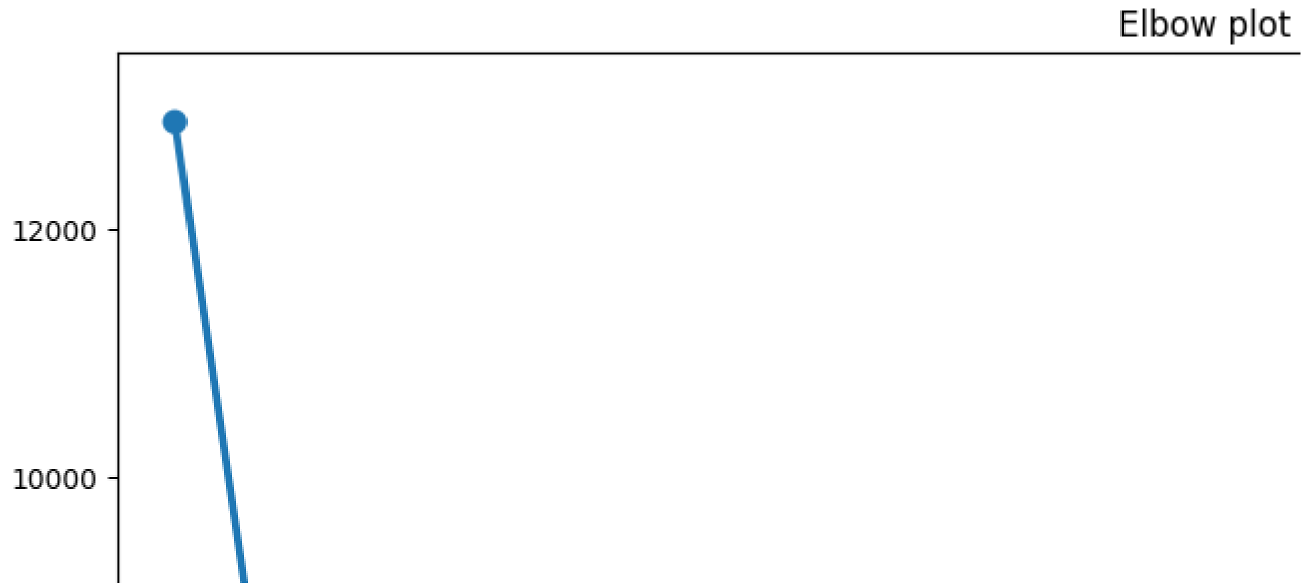
```
'''1. Elbow method'''

print(df7[1:21])

# Fit KMeans and calculate SSE for each *k*
ss_error = {}
for k in range(1, 20): #1-19
    k_means = KMeans(n_clusters=k, random_state=1)
    k_means.fit(df7)
    ss_error[k] = k_means.inertia_
```

```
    Recency  Frequency  Monetary
1  0.398462  -0.229147  0.761373
2 -0.613549   0.452393  0.743487
3  1.427092  -0.696903 -0.584353
4 -0.127012   0.663996  0.640466
5  1.122835  -1.738270 -1.638592
6  1.216297   0.268232  0.353083
```

```
7   1.157604  -0.901216 -0.330495
8  -0.474077   0.281895  1.119787
9  -0.188740   0.922891  1.754406
10 -2.257018  -0.611248  0.416288
11 -1.245007   1.448582  1.751213
12  0.135335   0.910479  1.076053
13  1.371003  -1.097274 -1.035995
14 -1.751013   1.519589  1.605454
15  0.668381  -0.463025 -0.183630
16 -1.245007   0.574568  0.510012
17  1.381072  -0.463025 -0.617799
18 -1.588115  -1.026536 -1.129389
19  0.121430   1.114096  1.304834
20  0.015884   0.334364  0.800816
```

```
# Make elbow plot
plt.figure(figsize = (14,10))
plt.title('Elbow plot')
plt.xlabel('Value of k')
plt.ylabel('Sum of squared error')
sns.pointplot(x=list(ss_error.keys()), y=list(ss_error.values()))
```

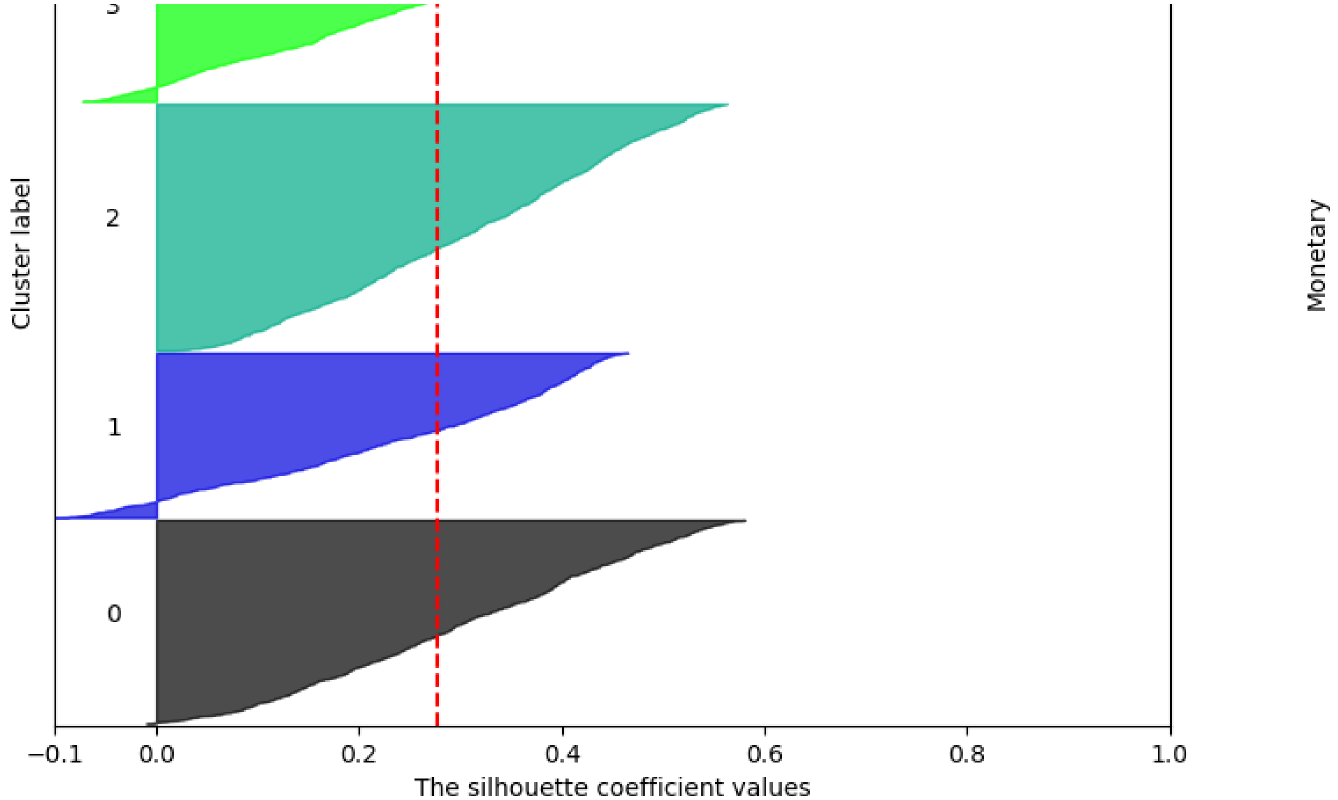
```
<AxesSubplot: title={'center': 'Elbow plot'}, xlabel='Value of k', ylabel='Sum o:
```



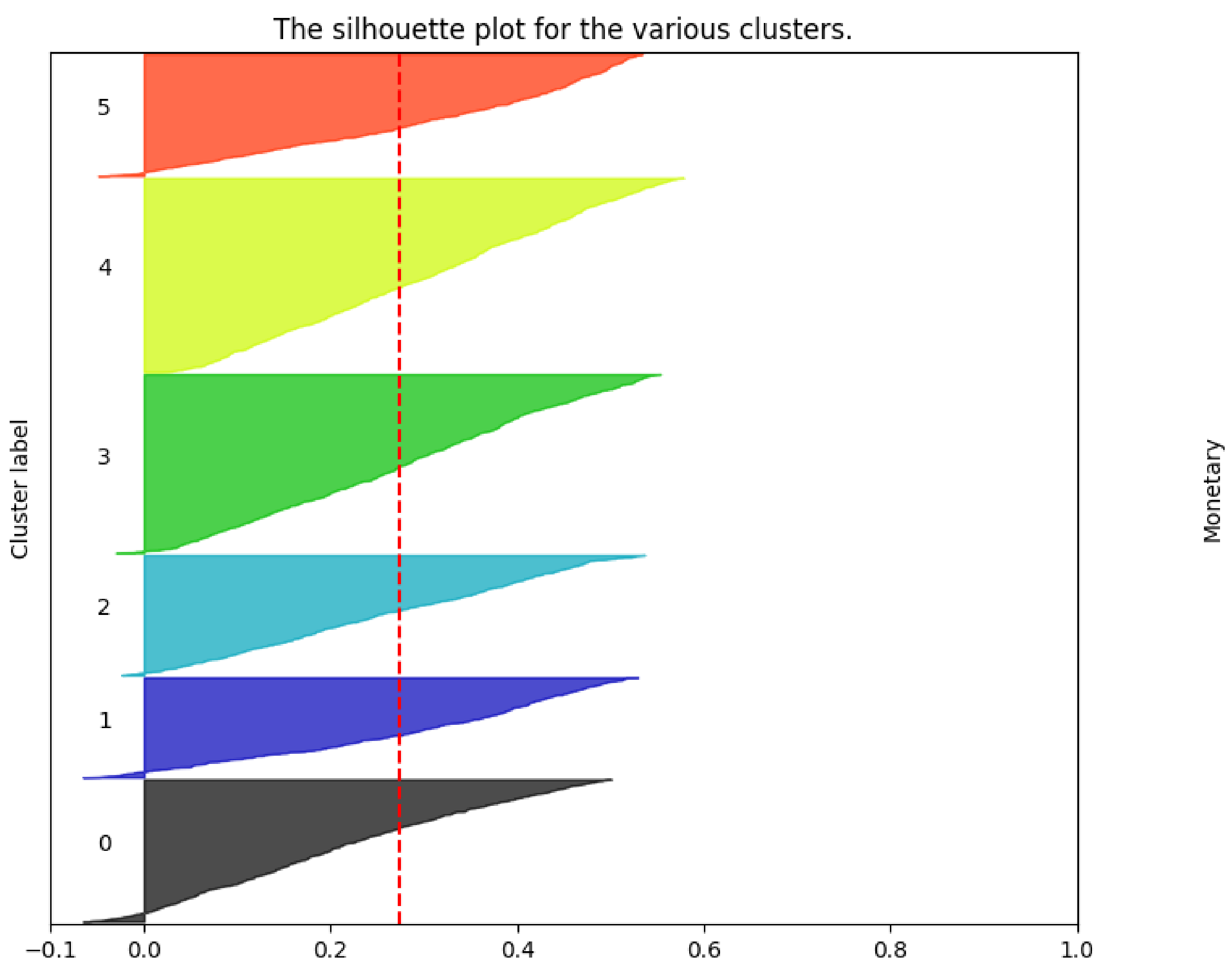

```
'''2. Silhoutte method.
Looks like k = 2 is a good solution. But always, explore other values of K around the
Finally disucss several solutions with stakeholders to see which makes most sense !
Here, we also use Silhoutte plots and scores'''

# Number of clusters confirmation by silhoutte scores
X = df7
range_n_clusters = [2, 3, 4, 5, 6,7,8,10,12,14]
for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=10,)
    cluster_labels = clusterer.fit_predict(X)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(X, cluster_labels)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)

    # Compute the silhouette scores for each sample
```

```
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

    # 2nd Plot showing the actual clusters formed
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(X["Frequency"], X["Monetary"], marker='.', s=30, lw=0, alpha=0.7,
                c=colors, edgecolor='k')

    # Labeling the clusters
    centers = clusterer.cluster_centers_
    # Draw white circles at cluster centers
    ax2.scatter(centers[:, 0], centers[:, 1], marker='o',
                c="white", alpha=1, s=200, edgecolor='k')

    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,
                    s=50, edgecolor='k')

    ax2.set_title("The visualization of the clustered data")
```

```
    ax2.set_xlabel("Frequency")
    ax2.set_ylabel("Monetary")

    plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
                  "with n_clusters = %d" % n_clusters),
                 fontsize=14, fontweight='bold')
```



Silhouette analysis for KMeans clustering o



The silhouette coefficient values

```
'''Looks like k = 2 has the best Silhoutte score. So let's pick k = 2 and do some inte
Add cluster column to the pre-processed data'''

df8 = df7.assign(Cluster = clus_labels)
print(df8.shape)
df8.head(3)
```

```
(4284, 4)
```

	Recency	Frequency	Monetary	Cluster
0	-1.961024	1.188477	1.462077	0
1	0.398462	-0.229147	0.761373	1
2	-0.613549	0.452393	0.743487	0

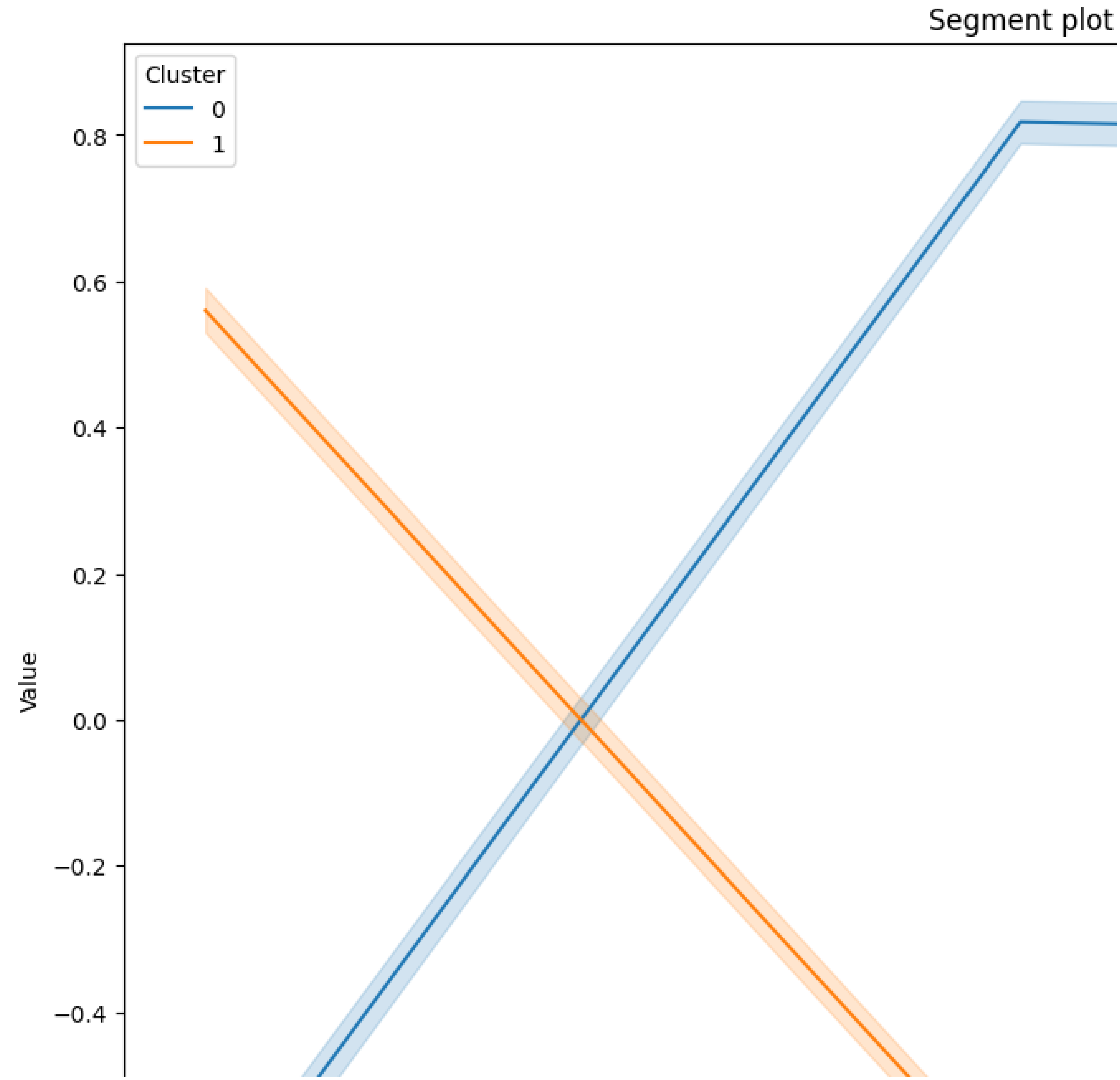
```
'''Use melt to transform the dataframe (not the data itself)'''

df8_melt = pd.melt(df8.reset_index(), id_vars=['Cluster'],
value_vars=['Recency', 'Frequency', 'Monetary'], var_name='Attribute',
value_name='Value')
```

```
df8_melt.head(3)
```

	Cluster	Attribute	Value
0	0	Recency	-1.961024
1	1	Recency	0.398462
2	0	Recency	-0.613549

```
'''Visualize segment characteristics to understand the clusters better'''

plt.figure(figsize = (14,10))
plt.title('Segment plot')
sns.lineplot(x="Attribute", y="Value", hue='Cluster', data=df8_melt)
```

```
<AxesSubplot: title={'center': 'Segment plot'}, xlabel='Attribute', ylabel='Valu
```



IV. Relative feature importances w.r.t clusters

```
cluster_avg = df5_clus2.groupby(['Cluster']).mean()
cluster_avg
```

Cluster	Recency	Frequency	Monetary
0	29.826772	171.479265	3521.034437
1	139.396805	25.092896	427.075520

```
population_avg = df5.mean()
population_avg
```

```
Recency        90.673436
Frequency      90.187675
Monetary     1802.890585
dtype: float64
```

```
relative_imp = cluster_avg / population_avg - 1
```

```
relative_imp.round(2)
```

Cluster	Recency	Frequency	Monetary
0	-0.67	0.90	0.95
1	0.54	-0.72	-0.76

```
plt.figure(figsize=(10, 6))
plt.title('Relative importance of attributes')
sns.heatmap(data=relative_imp, annot=True, fmt='.2f', cmap='Spectral')
```