Report

Assignment 4

Group: BD-2005
Performed by: Maryam Baizhigitova
Asanali Kazhymurat
Zhanibek Balabayev
Moldir Kumarbek
Aiym Yermakhan

Introduction

In this report, the main in-depth understanding and analysis of unsupervised learning-K-means algorithm.
Clustering is a process of organizing data sets into categories that are similar in some respects. Clustering is a technique for discovering this inherent structure. Clustering techniques are often referred to as unsupervised learning. k-means clustering is the most famous partition clustering algorithm, and its simplicity and efficiency make it the most widely used of all clustering algorithms. Given a set of data points and the required number of clusters k, k is specified by the user, and the k-means algorithm repeatedly divides the data into k clusters according to a certain distance function.
K-Means Life Example：

1) Item transfer optimization Use the combination of K-means algorithm to find the best launch position of UAV and genetic algorithm to solve the problem of traveling salesman's driving route and optimize the UAV item transfer process.
2) Identifying crime locations Using crime data related to specific areas in a city, analyzing crime types, crime locations, and the relationship between the two, high-quality surveys can be made on crime-prone areas in cities or regions.

I. Explore the dataset

Dataset Description:

- types.csv - reference of transaction types

- codes.csv - reference of transaction codes

- transactions.csv - transactional data on banking operations

- train_set.csv - training set with client gender marking (0/1 - client gender)

transactions.csv columns description:

- client_id - client is id

- datetime -transaction date (format - ordered day number hh:mm:ss - 421 06:33:15) - code - transaction code

- type - transaction type

- sum - sum of transaction

1. Importing libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline
plt.style.use('bmh')
```

2. Descriptive stat

```
train_set = pd.read_csv('train_set.csv', delimiter = ';')
train_set.head()
```

	client_id	target
0	75063019	0
1	86227647	1
2	6506523	0
3	50615998	0
4	95213230	0

```
# dataset is not disbalanced
train_set.describe()
```

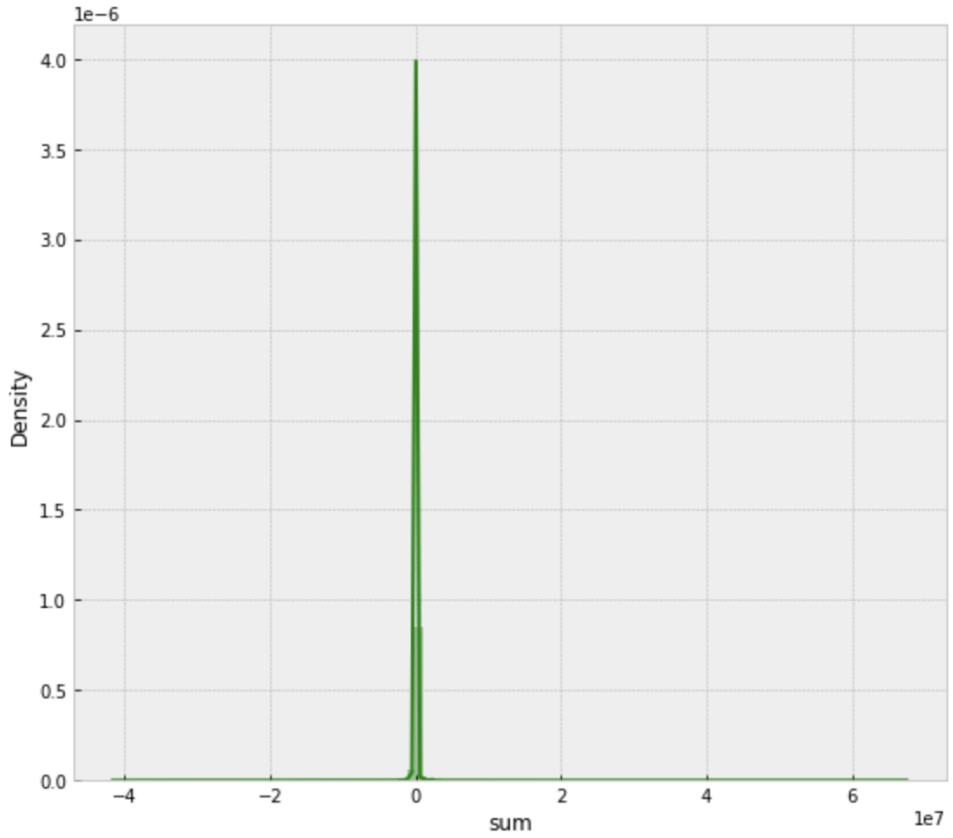
	client_id	target
count	6.000000e+03	6000.000000
mean	5.102984e+07	0.443333
std	2.881391e+07	0.496820
min	2.289900e+04	0.000000
25%	2.612906e+07	0.000000
50%	5.164080e+07	0.000000
75%	7.590927e+07	1.000000
max	9.999124e+07	1.000000

II. Exploratory data analysis

```
codes = pd.read_csv('codes.csv', delimiter = ';')
codes.head()
```

	code	code_description
0	5944	Магазины по продаже часов, ювелирных изделий и...
1	5621	Готовые сумочные изделия
2	5697	Услуги по переделке, починке и пошиву одежды
3	7995	Транзакции по азартным играм
4	5137	Мужская, женская и детская спец-одежда

```
plt.figure(figsize=(9, 8))
sns.distplot(transactions['sum'], color='g', bins=100, hist_kws={'alpha': 0.4});
```




```
popular_types = transactions['type'].value_counts()[:20]
popular_types
```

```
1010    30802
2010    20204
1030    18368
1110    17648
7070    12915
2370     6848
7010     4989
7030     2860
1100     2232
1200     1797
7071     1538
2330     1370
4071     1175
2371     1170
2011     1139
4010      583
7031      568
6110      409
4051      393
2331      390
Name: type, dtype: int64
```

```
popular_codes = transactions['code'].value_counts()[:20]
popular_codes
```

```
6011    27917
6010    18684
4814    18641
5411    18490
4829    11423
5499     6480
5912     2685
5541     2672
5331     2518
5812     2052
5814     1766
5999     1108
5921     1094
5311      708
5977      676
5964      630
5983      577
5211      564
5691      527
7995      469
Name: code, dtype: int64
```

III. Feature engineering. Encodings, generating the features from date-time, sum and from other columns.

IV.

Filling missing values

```
types.loc[types['type_description'].isin(['н/д(нет данных)','н/д']),['type_description']]=np.nan
types.head()
```

	type	type_description
0	8001	Установление расх. лимита по карте
1	2411	Перевод с карты на счет др.лица в одном тер. б...
2	4035	NaN
3	3001	Комиссия за обслуживание ссудного счета
4	2420	Перевод с карты на счет физ.лица в другом тер....

In our types data frame, we had ‘н/д(нет данных)’ and ‘н/д’ values in the description which we replaced with NaN. In order to that, we have chosen type_description column with the condition and made it equal to np.nan. Drop()

was used there in order to remove columns that are not necessary for our table and can cause some problems.

Transactions sum

```
transcactions_sum=transactions.groupby("client_id").sum().drop(columns = ['code','type'])
transcactions_count=aggregated_1 = transactions.groupby("client_id").count().drop(columns = ['code','type', 'datetime']).rename(c
transcactions_sum_count = pd.concat([transcactions_sum, transcactions_count], axis=1, join="outer")
transcactions_sum_count.head()
```

client_id	sum	count_transactions
22899	50847.54	9
27914	74115.21	4
28753	-2589800.29	13
31385	-83525.38	13
38084	693495.66	26

At this part of our task we used groupby(), sum(), counts() command in order to find total sum and number of each client id. It should be noted that he had found them as two different tables and with help of the concat() command, they were merged into one.

Features

```
time = transactions.drop(columns = ['code', 'type'])
time['day_number'] = transactions['datetime'].str[:3]
time['time'] = transactions['datetime'].str[-8:].str[:2]
time['time'] = time['time'].astype(int)
time.head()
```

	client_id	datetime	sum	day_number	time
0	96372458	421 06:33:15	-561478.94	421	6
1	24567813	377 17:20:40	67377.47	377	17
2	21717441	55 13:38:47	-44918.32	55	13
3	14331004	263 12:57:08	-3368873.66	263	12
4	85302434	151 10:34:12	-3368.87	151	10

First of all, we created time table which has day_numer, time additional columns. To create them we used the str[] command with index in order to choose needed parts from the time column.

```
buffer = transactions.groupby("client_id").median().drop(columns = 'sum')
predictors = pd.concat([transcactions_sum_count, buffer], axis=1, join="outer")
predictors = predictors.rename(columns={'sum': 'sum_of_transactions', 'code': 'Code_mode', 'type': 'Type_mode'})
predictors['Code_mode'] = predictors['Code_mode'].astype("category")
predictors['Type_mode'] = predictors['Type_mode'].astype("category")
buffer = time.drop(columns = ['sum', 'day_number', 'datetime']).groupby("client_id").mean()
buffer= buffer.rename(columns={'time': 'hour_mean'})
predictors = pd.merge(predictors, buffer, on = 'client_id', how ="left")
train_set = pd.merge(train_set, predictors, on = 'client_id', how ="left")
predictors.head()
```

client_id	sum_of_transactions	count_transactions	Code_mode	Type_mode	hour_mean
22899	50847.54	9	6010.0	4010.0	13.555556
27914	74115.21	4	5412.0	4020.0	12.250000
28753	-2589800.29	13	5661.0	1030.0	7.000000
31385	-83525.38	13	5411.0	1030.0	14.538462
38084	693495.66	26	5411.0	1210.0	13.000000

After creating time table we had to find the mode (most popular value), mean of values for each client id. Buffer variable was used as a temporary variable that stores medians and sum. Predictor variable stores result of merging the main table with client id and buffer. The main logic is merging buffer with predictor every time we change the values of the buffer.

```
train_set['Transactions_tendency']='0'
train_set.loc[train_set['sum_of_transactions']>0,'Transactions_tendency']='+'
train_set.loc[train_set['sum_of_transactions']<0,'Transactions_tendency']='-'
train_set['Transaction_mean']=train_set['sum_of_transactions']/train_set['count_transactions']
train_set.head()
```

	client_id	target	sum_of_transactions	count_transactions	Code_mode	Type_mode	hour_mean	Transactions_tendency	Transaction_mean
0	75063019	0	89032.60	29	5411.0	1110.0	15.068966	+	3070.089655
1	86227647	1	-606058.60	27	5411.0	1110.0	11.222222	-	-22446.614815
2	6506523	0	2635753.74	53	6010.0	7030.0	13.735849	+	49731.202642
3	50615998	0	-42672.40	7	4829.0	2370.0	14.142857	-	-6096.057143
4	95213230	0	214292.66	34	5812.0	1030.0	9.617647	+	6302.725294

There we added two new columns. First is ‘Transactions_tendency’ and values of it based on conditions if sum negative or positive. Second column is Transactions_mean which shows us mean of value of transactions of each client id

V. Unsupervised learning

Cluster analysis. Segment the customers.

Step 1

Unsupervised machine learning helps us find hidden and unknown patterns in data.The necessary libraries are imported.

4. Cluster Analysis. Segment the customers

```
: import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
```

Step 2

To analyze our dataset, we need to delete the outliers from the count_transactions column. Because, outliers will interfere with further analysis. First, we needed to define the highest and lowest allowed values. Here we used the IQR(Interquartile range) formula, to calculate them. Then, give them variables as upper_limit and lower_limit. In the second cell, we apply the capping on outliers. To check the result, we see the statistics using the describe() function.

```
#Capping on Outliers
upper_limit = train_set['count_transactions'].mean() + 3*train_set['count_transactions'].std
lower_limit = train_set['count_transactions'].mean() - 3*train_set['count_transactions'].std
```

```
#Now, apply the Capping
train_set['count_transactions'] = np.where(
    train_set['count_transactions']>upper_limit,
    upper_limit,
    np.where(
        train_set['count_transactions']<lower_limit,
        lower_limit,
        train_set['count_transactions']
    )
)
```

```
#Now see the statistics using "Describe" Function
train_set['count_transactions'].describe()
```

```
count    6000.000000
mean       14.325166
std        14.901623
min         1.000000
25%         5.000000
50%        11.000000
75%        19.000000
max       140.275776
Name: count_transactions, dtype: float64
```

Step 3

We have a look at the data. As we see, the data seems to be interesting. Let us look at the data distribution.

```
train_set.head()
```

	client_id	target	sum_of_transactions	count_transactions	Code_mode	Type_mode	hour_mean	transactions_calls
0	75063019	0	89032.60	29.0	5411.0	1110.0	15.068966	5
1	86227647	1	-606058.60	27.0	5411.0	1110.0	11.222222	7
2	6506523	0	2635753.74	53.0	6010.0	7030.0	13.735849	0
3	50615998	0	-42672.40	7.0	4829.0	2370.0	14.142857	3
4	95213230	0	214292.66	34.0	5812.0	1030.0	9.617647	12

```
train_set.corr()
```

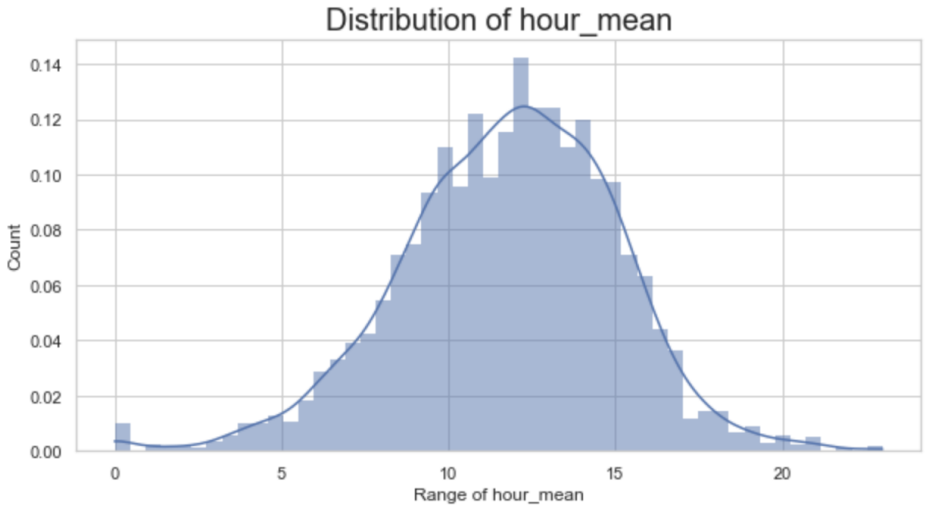
	client_id	target	sum_of_transactions	count_transactions	hour_mean	transactions_calls
client_id	1.000000	0.003311	0.007854	-0.004135	0.016593	-0.019806
target	0.003311	1.000000	-0.036427	0.069390	-0.013989	-0.020960
sum_of_transactions	0.007854	-0.036427	1.000000	-0.012241	0.013085	-0.014327
count_transactions	-0.004135	0.069390	-0.012241	1.000000	-0.033385	0.391650
hour_mean	0.016593	-0.013989	0.013085	-0.033385	1.000000	0.040596
transactions_calls	-0.019806	-0.020960	-0.014327	0.391650	0.040596	1.000000

Step 4

Below, from the histogram we see, that the distribution of hour_mean is between 10 and 15.

```
#Distribution of mean_hour
plt.figure(figsize=(10, 5))
sns.set(style = 'whitegrid')
sns.histplot(train_set['hour_mean'], kde=True, stat="density", linewidth=0)
plt.title('Distribution of hour_mean', fontsize = 20)
plt.xlabel('Range of hour_mean')
plt.ylabel('Count')
```

```
Text(0, 0.5, 'Count')
```

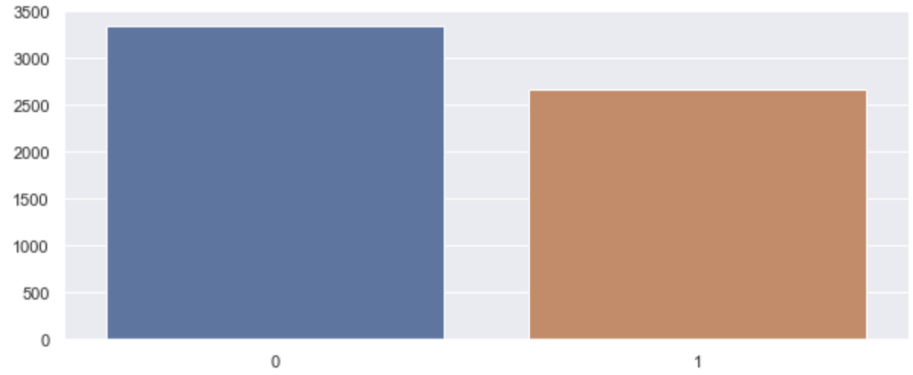


From the next histogram, we see that the count of transactions is between 10 and 60.

Step 5

And from the genders barplot, we can observe that the dataset has more females than males.

```
targets = train_set.target.value_counts()
sns.set_style("darkgrid")
plt.figure(figsize=(10,4))
sns.barplot(x=targets.index, y=targets.values)
plt.show()
```



More 0(female) customers than 1(male).

K-means clustering

Step 1

We create a new dataframe with necessary columns. First, we work with two features only, count_transactions and hour_mean. The reason why we choose it is because of the customer segmentation analysis. By using corr method, we find out that these 2 columns will properly used in the clustering.

```
#We take just the count_transactions and mean_hour
df1 = train_set[["client_id", "target", "sum_of_transactions", "count_transactions", "hour_m
X = df1[["count_transactions","hour_mean"]]
```

```
#The input data
X.head()
```

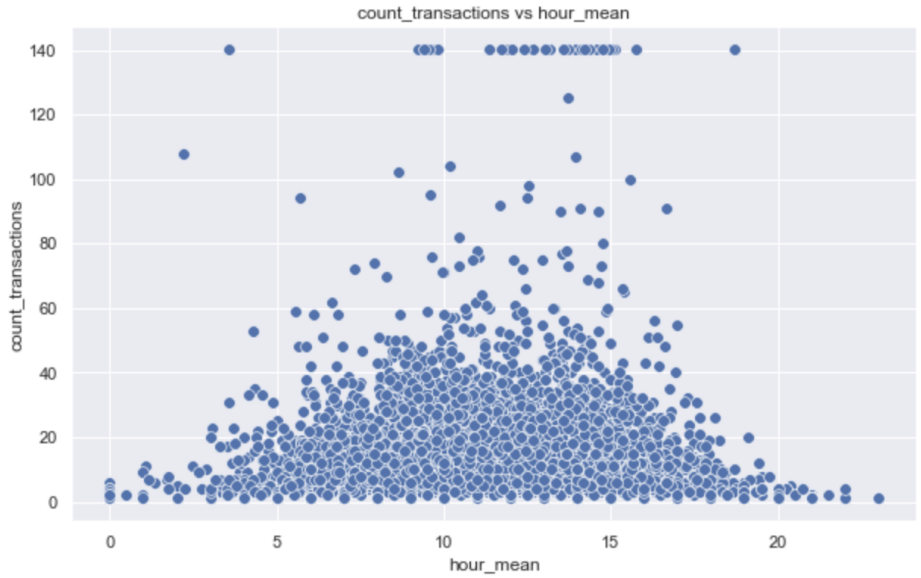
	count_transactions	hour_mean
0	29.0	15.068966
1	27.0	11.222222
2	53.0	13.735849
3	7.0	14.142857
4	34.0	9.617647

Step 2

The data does seem to hold some patterns

```
#Scatterplot of the input data
plt.figure(figsize=(10,6))
sns.scatterplot(x = 'hour_mean',y = 'count_transactions',  data = X  ,s = 60 )
plt.xlabel('hour_mean')
plt.ylabel('count_transactions')
plt.title('count_transactions vs hour_mean')
plt.show()
```

.



Step 3

Now we calculate the Within Cluster Sum of Squared Errors (WSS) for different values of k. Next, we choose the k for which WSS first starts to diminish. This value of K gives us the best number of clusters to make from the raw data.

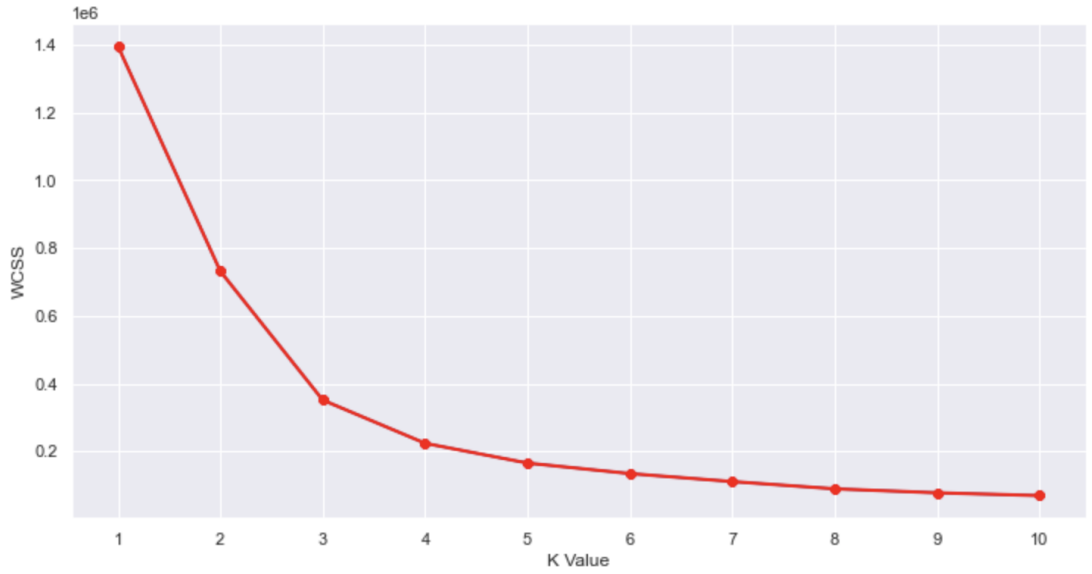
```
#Importing KMeans from sklearn
from sklearn.cluster import KMeans
```

```
wcss=[]
for i in range(1,11):
    km=KMeans(n_clusters=i)
    km.fit(X)
    wcss.append(km.inertia_)
```

```
#The elbow curve
plt.figure(figsize=(12,6))
plt.plot(range(1,11),wcss)
plt.plot(range(1,11),wcss, linewidth=2, color="red", marker ="8")
plt.xlabel("K Value")
plt.xticks(np.arange(1,11,1))
plt.ylabel("WCSS")
plt.show()
```

Step 4

The plot below shows us the elbow curve.This is known as the elbow graph, the x-axis being the number of clusters, the number of clusters is taken at the elbow joint point. This point is the point where making clusters is most relevant as here the value of WCSS suddenly stops decreasing. Here in the graph, after 5 the drop is minimal, so we take 5 to be the number of clusters.



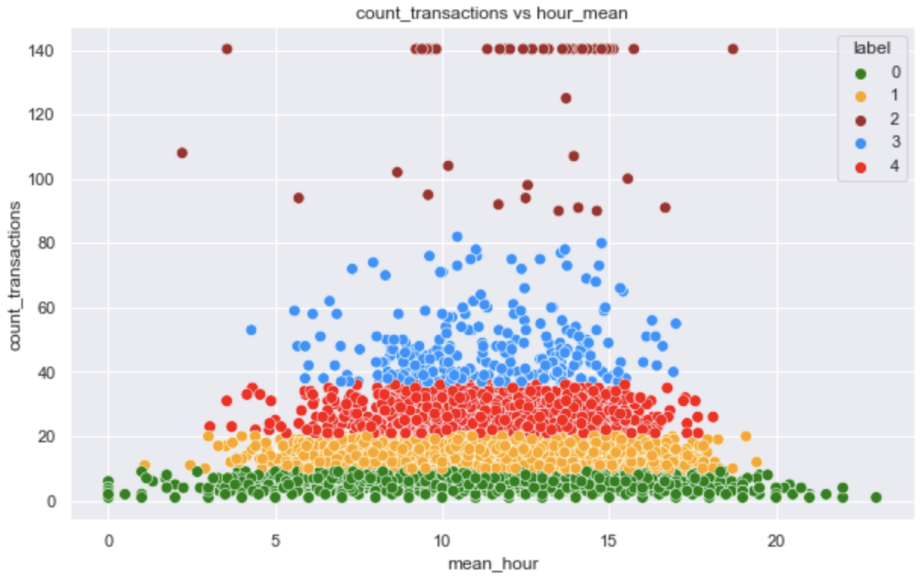

```
#Taking 5 clusters
km1=KMeans(n_clusters=5)
km1.fit(X)

y=km1.predict(X)

train_set["label"] = y
train_set.head()
```

	client_id	target	sum_of_transactions	count_transactions	Code_mode	Type_mode	hour_mean	transactions_calls	label
0	75063019	0	89032.60	29.0	5411.0	1110.0	15.068966	5	4
1	86227647	1	-606058.60	27.0	5411.0	1110.0	11.222222	7	4
2	6506523	0	2635753.74	53.0	6010.0	7030.0	13.735849	0	3
3	50615998	0	-42672.40	7.0	4829.0	2370.0	14.142857	3	0
4	95213230	0	214292.66	34.0	5812.0	1030.0	9.617647	12	4

Step 5

Creating a scatter plot, by using a new column “label”. From the scatter plot below, we can clearly see that 5 different clusters have been formed from the data. The red cluster is the customers with the least count transactions, similarly, the blue cluster is the customers with the most count transactions.

```
#Scatterplot of the clusters
plt.figure(figsize=(10,6))
sns.scatterplot(x = 'hour_mean',y = 'count_transactions',hue="label",
                palette=['green','orange','brown','dodgerblue','red'], legend='full',data =
plt.xlabel('mean_hour')
plt.ylabel('count_transactions')
plt.title('count_transactions vs hour_mean')
plt.show()
```



Hierarchical Clustering

With different metrics, linkages. Visualize the clusters etc.

Ввод [122]:
```
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram
from scipy.cluster.hierarchy import cut_tree
```

Ввод [123]:
```
df_scaled = train_set[['sum_of_transactions', 'count_transactions']]
df_scaled
```

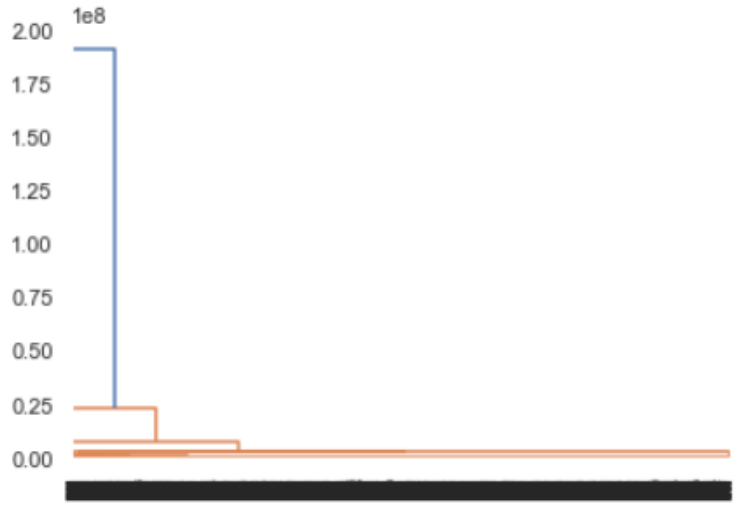
Out[123]:

	sum_of_transactions	count_transactions
0	89032.60	29.0
1	-606058.60	27.0
2	2635753.74	53.0
3	-42672.40	7.0
4	214292.66	34.0
...	...	...
5995	-114601.90	5.0
5996	-262708.36	7.0
5997	-42863.31	6.0
5998	-75992.84	19.0
5999	-16392.05	10.0

6000 rows × 2 columns

Importing necessary libraries and taking only sum of transactions and their count value.

```
mergings = linkage(df_scaled, method="single", metric='euclidean')
dendrogram(mergings)
plt.show()
```



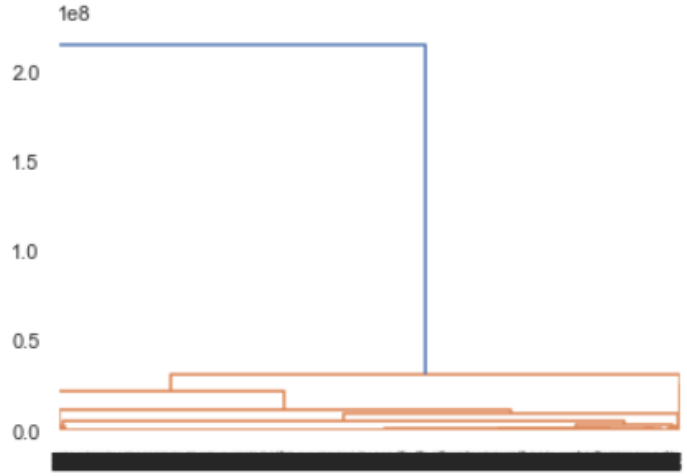
Method= ‘single’

After that, we input methods and metrics. In our situation we have a lot of data in the dataframe and dendrogram building takes about 20-60 minut to complete visualize. So we take by default the metric type ‘euclidean’.

```
mergings = linkage(df_scaled, method="complete", metric='euclidean')
dendrogram(mergings)
plt.show()
```

Method= ‘complete’

```
mergings = linkage(df_scaled, method="average", metric='euclidean')
dendrogram(mergings)
plt.show()
```



Method= ‘average’

```
cluster_labels1 = cut_tree(mergings, n_clusters=5).reshape(-1, )
cluster_labels1
```

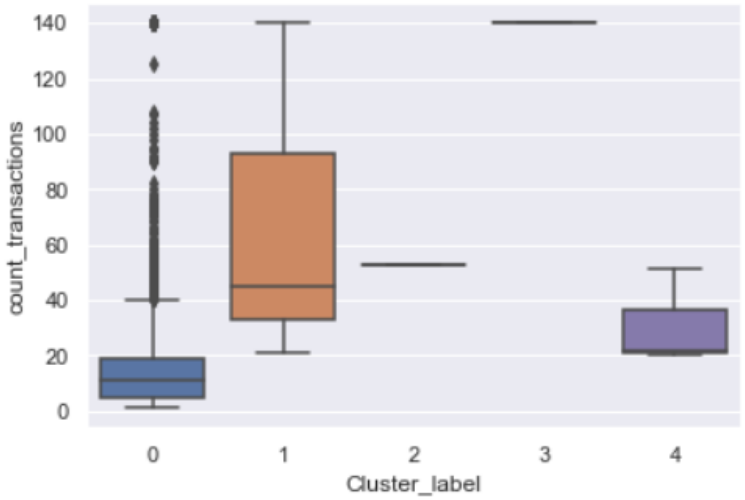
```
array([0, 0, 0, ..., 0, 0, 0])
```

With cut_tree we take cluster levels of customers according to linkage and store them in an array in 1D.

Now we plot our result to see differences of customer by values

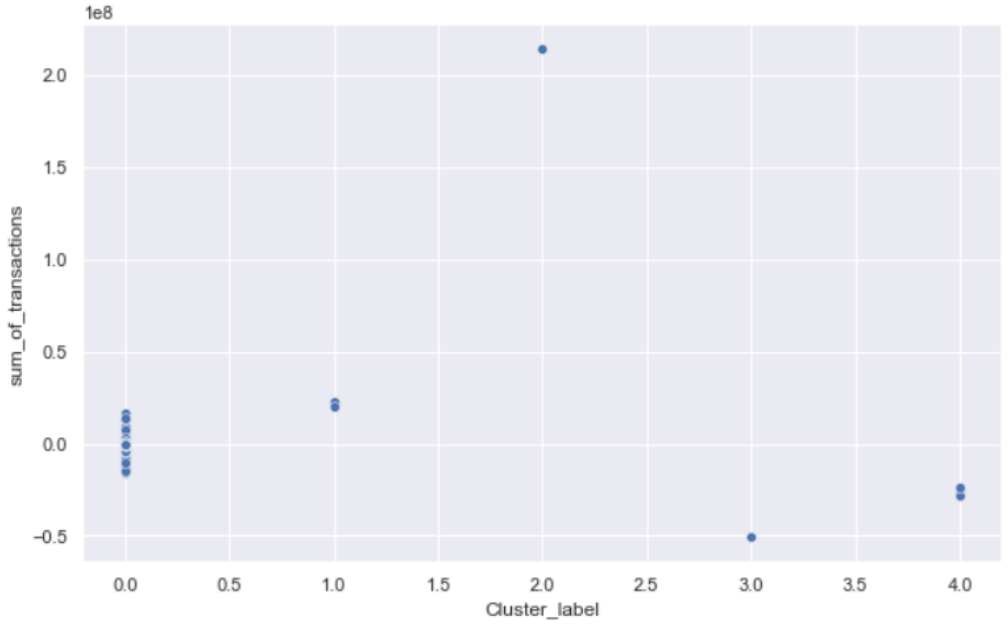
```
sns.boxplot(x='Cluster_label', y='count_transactions', data=df_f)
```

```
<AxesSubplot:xlabel='Cluster_label', ylabel='count_transactions'>
```



```
# Plot Cluster Id vs count_transactions
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Cluster_label', y='sum_of_transactions', data=df_f)
```

```
<AxesSubplot:xlabel='Cluster_label', ylabel='sum_of_transactions'>
```



Hierarchical Clustering with 5 Cluster Labels

Customers with Cluster_Labels 0 and 1 are the customers with high amount of transactions as compared to other customers.

Customers with Cluster_Labels 1 are frequent customer in bank and their sum of transactions is higher than other customers.

Also Customers with Cluster_labels 0 are active customer in bank

VI. Analyzing the results

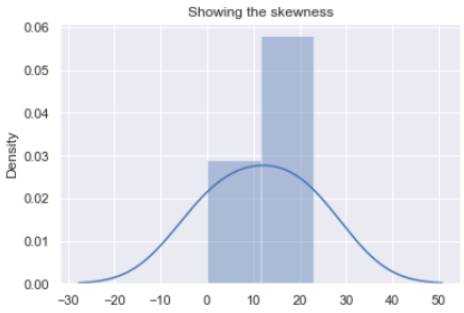
At first we found the minimum, maximum and average hour, then visualized it

```
tr_avg=train_set['hour_mean'].mean()
tr_min=train_set['hour_mean'].min()
tr_max=train_set['hour_mean'].max()
Diff=[tr_min,tr_avg,tr_max]
print('Average of hour:',tr_avg)
print('Minimum of hour:',tr_min)
print('Maximum of hour:',tr_max)
```

```
Average of hour: 11.792256940510223
Minimum of hour: 0.0
Maximum of hour: 23.0
```

```
sns.distplot(Diff)
plt.title("Showing the skewness");
```

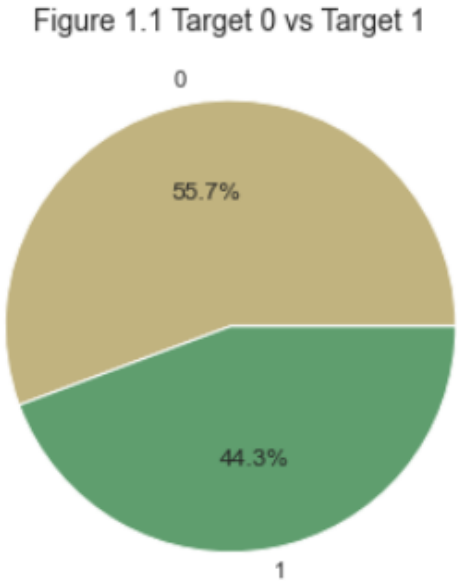
```
C:\Users\User\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function
and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with simila
r flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In this displot you can see the skewness between a minimum, maximum and average time spent.

In the next part you can see the pie chart.

```
#grouping dataframe into two groups: Target 0 and Target 1
tr_s = train_set.groupby(['target'], as_index = False).count()

#color for parts
clrs = ['#c1b37f','#5f9e6e']

plt.figure(figsize = (14, 5))
#Displaying the pie chart.
plt.pie(tr_s['client_id'],
        labels = tr_s['target'],
        colors = clrs,
        autopct='%1.1f%%',#displaying the percent
        shadow=False,
        startangle=0)

plt.title('Figure 1.1 Target 0 vs Target 1', size = 14)
plt.show()
```



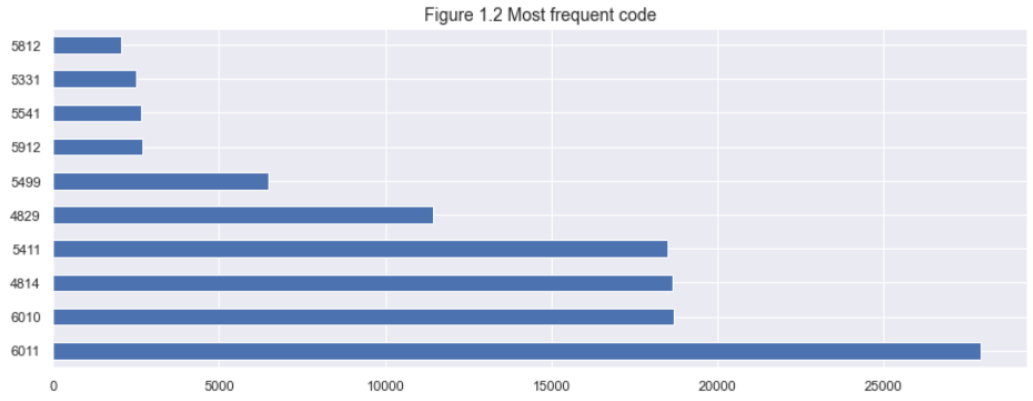
By this pie, you can find out that the most of half is 0 target.

First, we used groupby method to ‘target’ column then count the quantity. For figure, we choose figure size equally to (14, 5). Then display it by percentage.

Next figure is about the most frequent code.

In transactions data we did the value count method to ‘code’ column and plot it.

```
plt.figure(figsize = (14,5))
#Displaying the chart.
ax=transactions['code'].value_counts()[:10].plot(kind='barh')

ax.set_title('Figure 1.2 Most frequent code', size = 14)

ax.xaxis.grid(True)
```



This chart shows the most frequent code in transaction data. As you can see the most frequensed code is 6011, which is equal to approximately 28000.

VII. Conclusion

The first form of classification is the method called *k-means clustering* or the mobile center algorithm. As a reminder, this method aims at partitioning n observations into k clusters in which each observation belongs to the cluster with the closest average, serving as a prototype of the cluster.

For this task, we used the train_set database. Problems like segmenting customers are often deceptive and tricky because we are not working with any target variable in mind. We are officially in the land of unsupervised learning where we need to figure out patterns and structures without a set outcome in mind. It's both challenging and thrilling as a data scientist.