Table of Contents

Introduction

The K-Means clustering beams at partitioning the ‘n’ number of observations into a mentioned number of ‘k’ clusters (produces sphere-like clusters). The K-Means is an unsupervised learning algorithm and one of the simplest algorithm used for clustering tasks. The K-Means divides the data into non-overlapping subsets without any cluster-internal structure. The values which are within a cluster are very similar to each other but, the values across different clusters vary enormously. K-Means clustering works really well with medium and large-sized data.

Despite the algorithm’s simplicity, K-Means is still powerful for clustering cases in data science. In this article, we are going to tackle a clustering problem which is customer segmentation (dividing customers into groups based on similar characteristics) using the K-means algorithm. Now let’s see a little bit about the case we are going to solve.

Importing the Packages

Every task must begin with importing the required packages into the respective environment (python in our case). Our primary packages include pandas for working on the data, NumPy for working with the arrays, matplotlib & seaborn for visualization, mplot3d for three-dimensional visualization, and finally scikit-learn for building the K-Means model. Let’s import all the primary

packages into our python environment.

```
import numpy as np # for linear algebra
import pandas as pd # for data processing, csv io
from matplotlib import pyplot as plt # data plots
import seaborn as sns # pretty data plots
sns.set()
import re
from sklearn.preprocessing import LabelEncoder # for label normalization
from sklearn.metrics import mean_squared_error, mean_squared_log_error
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
import seaborn as sb # visualization
from mpl_toolkits.mplot3d import Axes3D # 3d plot
from termcolor import colored as cl # text customization

from sklearn.preprocessing import StandardScaler # data normalization
from sklearn.cluster import KMeans # K-means algorithm

plt.rcParams['figure.figsize'] = (20, 10)
sb.set_style('whitegrid')
```

Now that we have imported all the required primary packages into our python environment. Let's proceed to import the transactions data.

Importing Data

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)

We will use the 'read_csv' method provided by the Pandas package to read and import the data into our python environment. We are using the 'read_csv' method because the data we are going to use is in the '.csv' format. If it is an excel sheet, it is recommended to use the 'read_excel' method to read it into the python environment. Now let's import our data in python. Then we joined all datasets into one.

```
df_trans = pd.read_csv('Downloads/Datasets/transactions.csv')
df_types = pd.read_csv('Downloads/Datasets/types.csv', sep='delimiter')
df_codes = pd.read_csv('Downloads/Datasets/codes.csv', sep='delimiter')
df_train = pd.read_csv('Downloads/Datasets/train_set.csv')
df_test = pd.read_csv('Downloads/Datasets/test_set.csv')
```

```
df_trans[['client_id', 'datetime', 'code', 'type', 'sum']] = df_trans['client_id;datetime;code;type;sum'].str.split(
del df_trans['client_id;datetime;code;type;sum']

df_types[['type', 'type_description']] = df_types['type;type_description'].str.split(';', expand=True)
del df_types['type;type_description']

df_codes[['code', 'code_description']] = df_codes['code;code_description'].str.split(';', expand=True)
del df_codes['code;code_description']

df_train[['client_id', 'target']] = df_train['client_id;target'].str.split(';', expand=True)
del df_train['client_id;target']
```

```
join_codes = df_trans.merge(df_codes, on='code', how='left')
```

```
join_types = join_codes.merge(df_types, on='type', how='left')
```

```
df_joined = join_types.merge(df_train, on='client_id', how='left')
```

Now that we have successfully imported our customer segmentation data into our python environment. Let'sexplore and gain some information about the data.

Exploratory Data Analysis

In the data preprocessing part, we will mainly look for

missings and fill them

changing the necessary data types

```
print('Number of missing data:')
print(df_joined.isnull().sum())
df_joined.describe(include='all')
```

```
Number of missing data:
client_id               0
datetime                0
code                    0
type                    0
sum                     0
code_description        0
type_description       41
target              38213
dtype: int64
```

In this part we deleted all null values and work with datatypes

Does gender have an influence on the total sum

```
gender_sum = df_joined.groupby('target', as_index=False).agg({'sum': 'sum'}).sort_values(by='sum', ascending=False)
gender_sum['percentage (%)'] = gender_sum['sum'] / sum(gender_sum['sum']) * 100
gender_sum
```

	target	sum	percentage (%)
0	0.0	-5.037782e+08	32.91084
1	1.0	-1.026958e+09	67.08916

As we can see, gender is affecting on the sum of the transactions

Which transaction amount type is most considerable

```
PaymentMethod_TotalCharges = df.groupby('type_description', as_index=False).agg({'sum': 'sum'}).sort_values(by='sum'
PaymentMethod_TotalCharges.head()
```

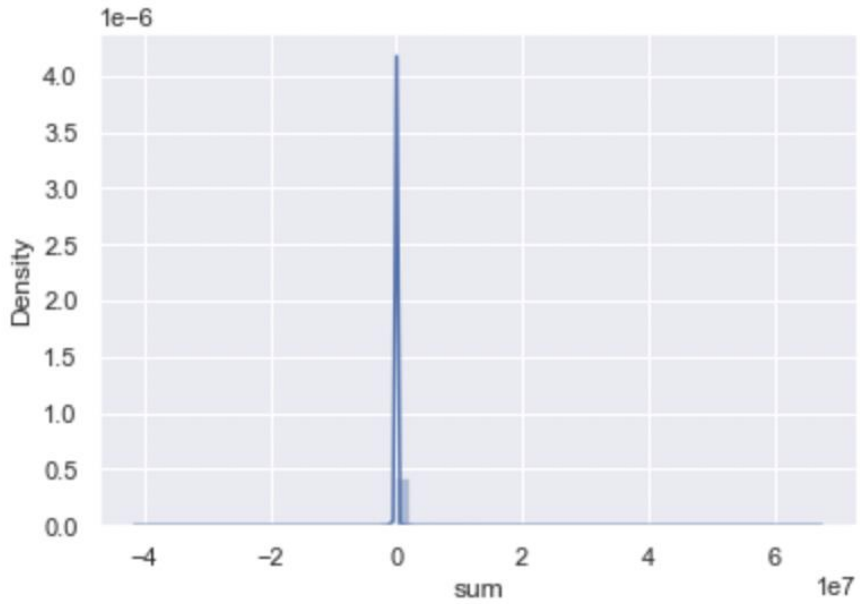
	type_description	sum
27	Перевод на карту (с карты) через Мобильный бан...	1.550763e+09
8	Взнос наличных через ATM (в своем тер.банке)	1.307821e+09
25	Перевод на карту (с карты) через ATM (в предел...	4.132172e+08
2	Взнос наличных через POS	3.197199e+08
28	Перевод на карту (с карты) через Мобильный бан...	2.378725e+08

Transactions using mobile application is most demanded

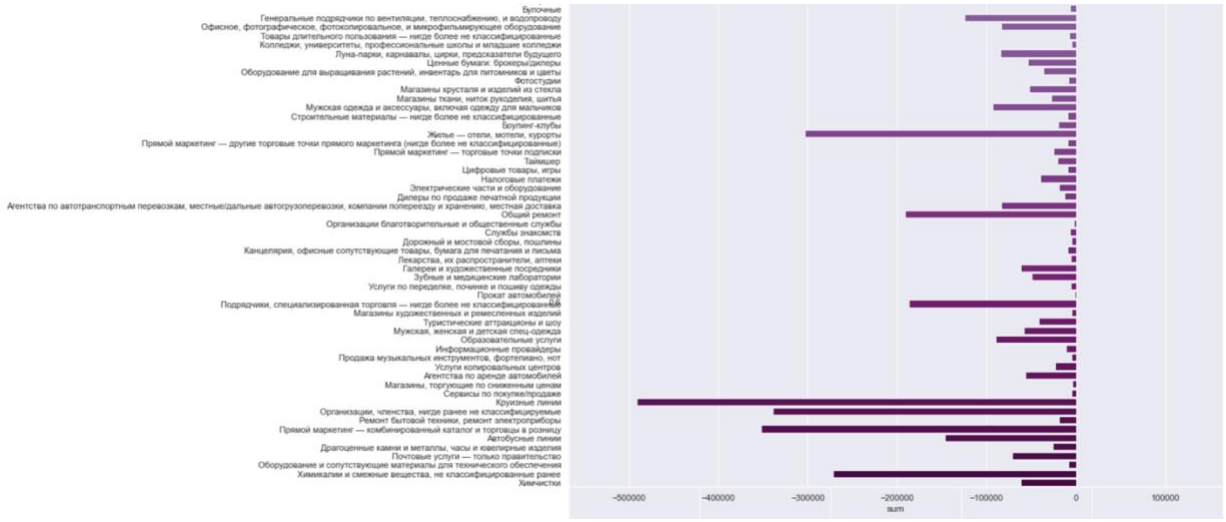
Through the EDA we found some information about features, for example gender is affecting on the sum of the transactions; Transactions using mobile application is most demanded; Most of the transactions were carried out using Financial Institutions - cash withdrawal manually

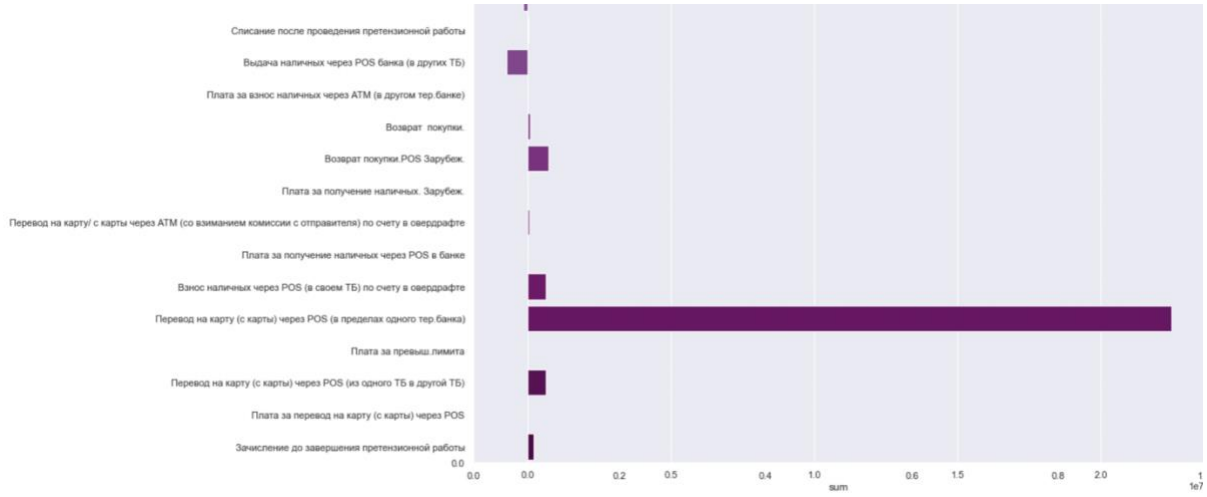
Visualization

Distribution of the sum of transations



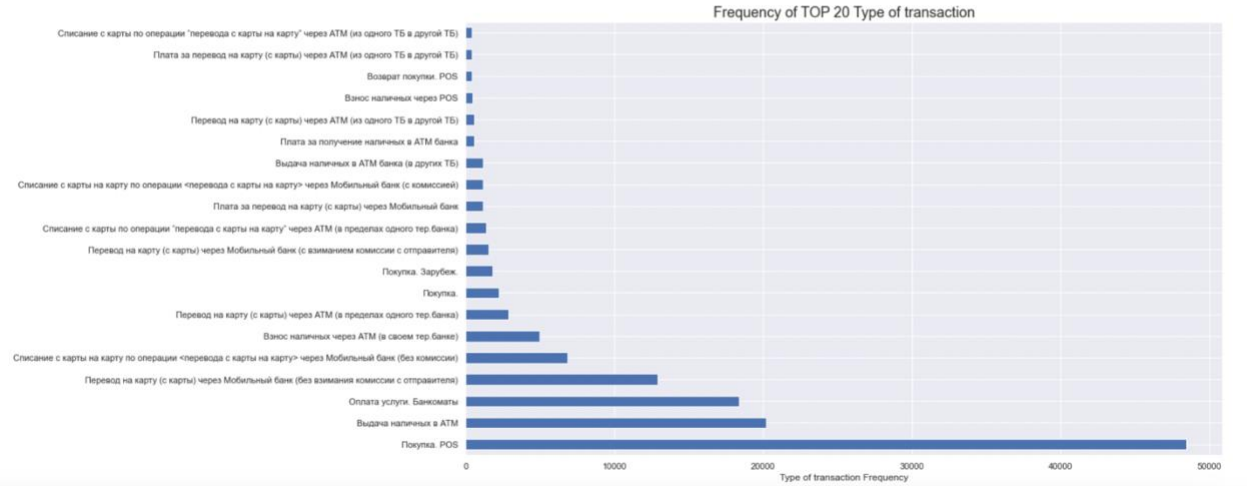
Amount of transactions by code amd type descriptions





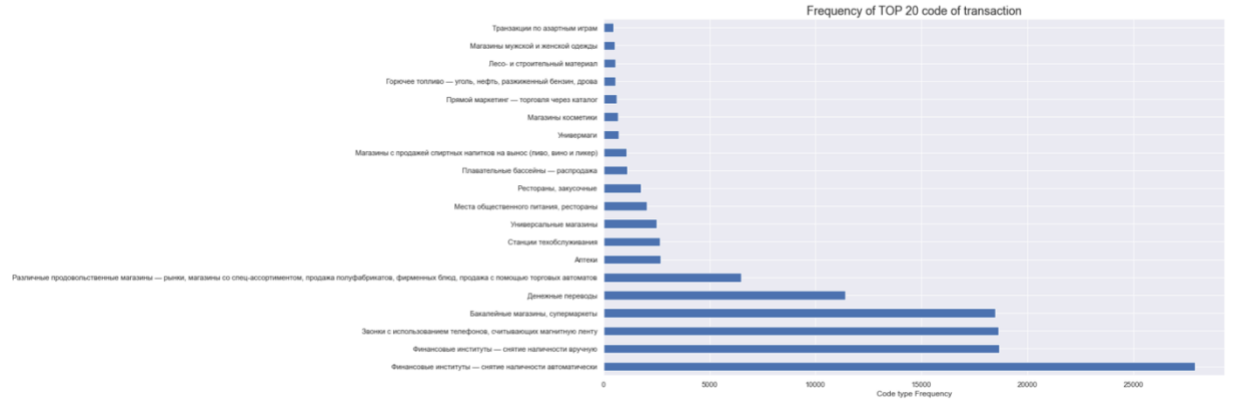
Graph of Frequency of TOP 20 Type of transactions. As we can see from the graph, Покупка POS and выдача наличных в ATM are most popular ones.

```
plt.figure(figsize=(17,10))
df.type_description.value_counts().nlargest(20).plot(kind='barh')
plt.xlabel('Type of transaction Frequency')
plt.title("Frequency of TOP 20 Type of transaction",fontsize=18)
plt.show()
```

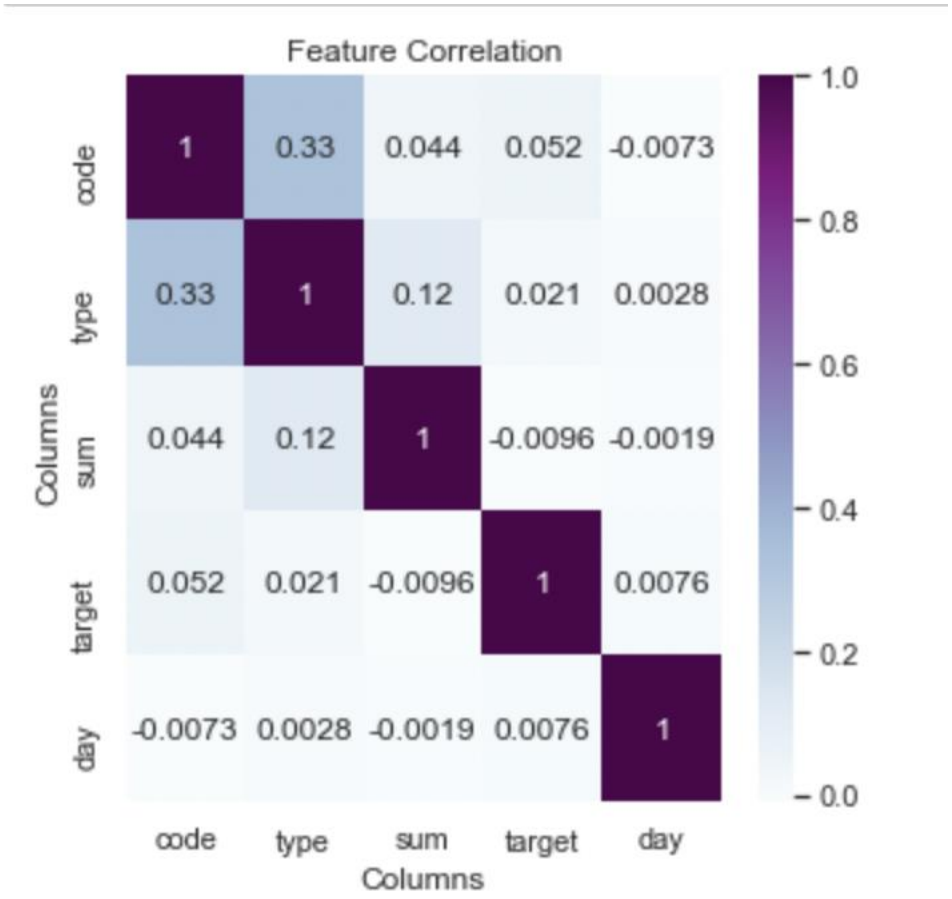


Graph of Frequency of TOP 20 Code of transactions. As we can see from the graph, Финансовые институты — снятие наличности автоматически and Финансовые институты — снятие наличности вручную are most popular ones.

```
plt.figure(figsize=(17,10))
df.code_description.value_counts().nlargest(20).plot(kind='barh')
plt.xlabel('Code type Frequency')
plt.title("Frequency of TOP 20 code of transaction",fontsize=18)
plt.show()
```



Correlation between features



Data Processing

In this step, we are going to normalize the dataset and it is very important to build our model. But what is normalization?

Normalization is a statistical method that helps mathematical-based algorithms to interpret features with different magnitudes and distributions equally

Using the 'StandardScaler' function provided by the scikit-learn package, we can feasibly perform normalization over the dataset in python.

```
X = df_joined.values
X = np.nan_to_num(X)

sc = StandardScaler()

cluster_data = sc.fit_transform(X)
print(cl('Cluster data samples : ', attrs = ['bold']), cluster_data[:5])
```

```
Cluster data samples :  [[ 1.57323554  0.6884737  -0.23501079 -0.87417515 -0.98268211  1.36698954]
 [-1.05552774  0.6884737  -0.23501079 -0.04532599 -0.98268211 -1.44433108]
 [ 1.18343625 -1.29943102 -0.66192027  0.02134232 -0.98268211 -0.70693551]
 [-0.71383204 -0.30796977 -0.62707051  0.02422527 -0.98268211  1.19032185]
 [ 1.16668999  0.68681296  1.96923609  0.03582159  1.01762308  1.38235195]]
```

Modeling

We can build the K-Means in python using the 'KMeans' algorithm provided by the scikit-learn package.

The KMeans class has many parameters that can be used, but we will be using these three:

- **init** - Initialization method of the centroids. The value will be: 'k-means++'. **k-means++** - Selects initial cluster centers for the k-means clustering in a smart way to speed up convergence.
- **n_clusters** - The number of clusters to form as well as the number of centroids to generate. The value will be 3
- **n_init** - Number of times the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia. The value will be 12

After building the model, we will be fitting and define a variable 'labels' to store the cluster labels of the built model.

```
clusters = 3
model = KMeans(init = 'k-means++',
               n_clusters = clusters,
               n_init = 12)
model.fit(X)

labels = model.labels_
print(cl(labels[:100], attrs = ['bold']))
```

```
[0 1 0 1 0 0 0 1 0 1 1 1 0 0 2 0 0 0 0 1 0 2 2 1 2 0 0 1 0 1 0 1 2 0 2 0 0
 0 0 1 1 0 0 2 1 0 0 0 0 2 1 0 2 2 0 0 1 0 2 1 0 0 2 0 1 1 2 2 2 1 0 0 2 2
 0 0 2 0 2 1 1 1 1 1 2 0 0 1 0 0 1 0 2 2 2 1 1 2 1 1]
```

```
df_joined['cluster_num'] = labels
df_joined.head()
```

	client_id	code	type	sum	target	day	cluster_num
0	96372458	6011	2010	-561478.94	0.0	421	0
1	21717441	6011	2010	-44918.32	0.0	55	1
2	85302434	4814	1030	-3368.87	0.0	151	0
3	31421357	5411	1110	-1572.14	0.0	398	1
4	84826852	6010	7070	5654.99	1.0	423	0

Now we have successfully built and fitted our K-Means model and stores the cluster labels into the ‘labels’ variable. Using the labels produced by the model we can find some useful insights about the model and come to a conclusion.

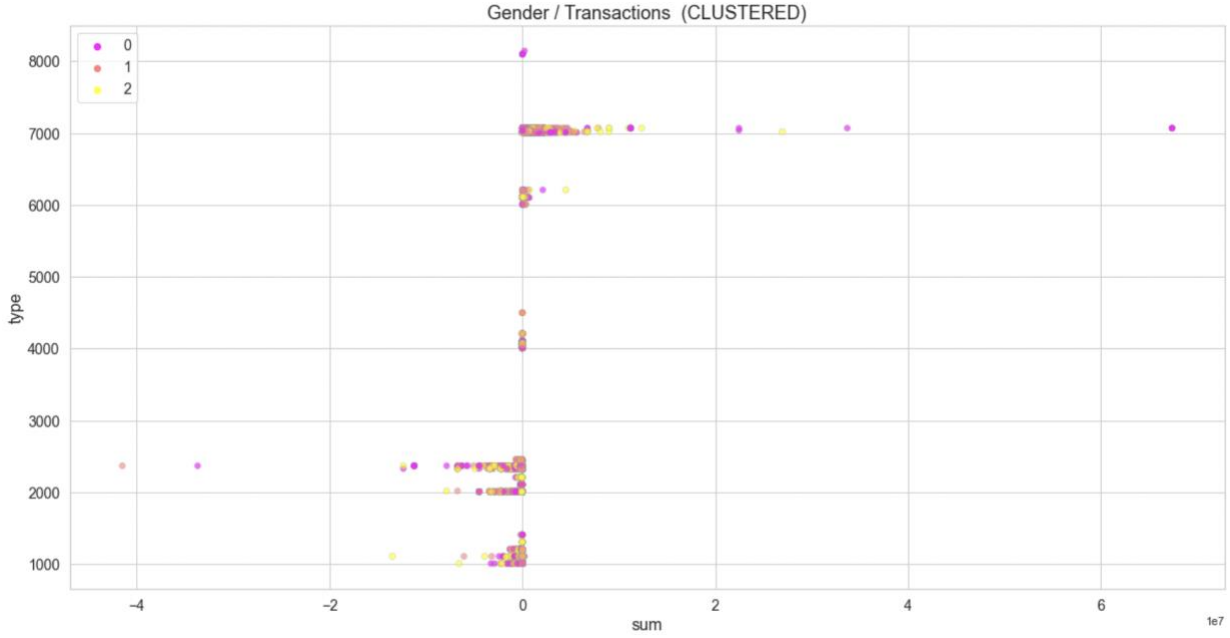
As you can see, we have created a new attribute called ‘cluster_num’ in the customer data that represents which cluster value does each of the rows belong to.

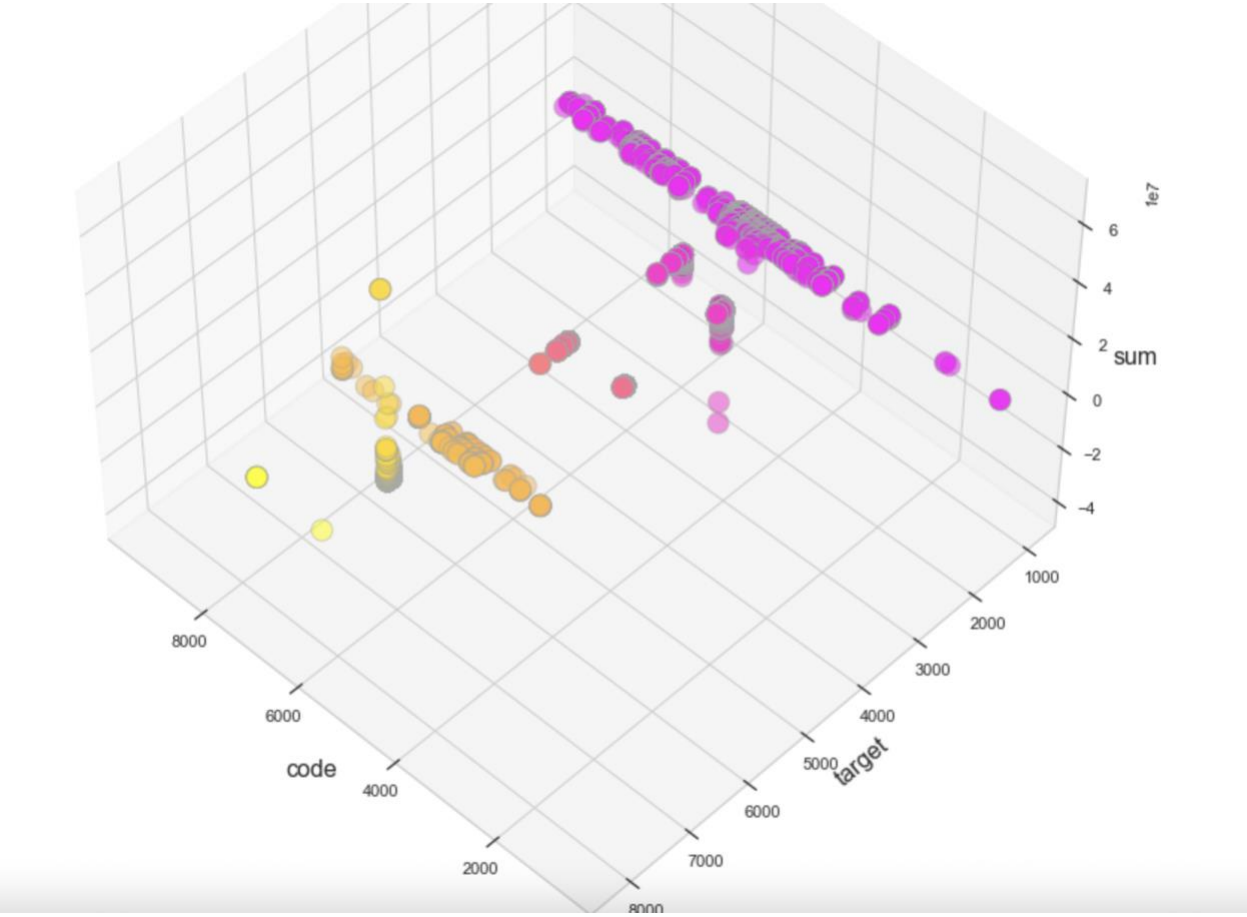
Now let’s use the ‘groupby’ method to group the cluster value and see the mean value of each of the attributes in the dataset using the ‘mean’ method.

```
df_joined.groupby('cluster_num').mean()
```

cluster_num	code	type	sum	target	day
0	5580.395041	2415.258440	-11317.747498	0.460354	243.609050
1	5584.866667	2351.862005	-21590.721714	0.492184	244.615948
2	5618.647037	2816.964900	-16956.090426	0.515220	241.293051

Let’s look at the distribution of customers based on their type and transactions amount using a bubble plot and the color represents the cluster value.





Our K-Means model has partitioned the customers into mutually exclusive groups, which are three clusters in our case.

The customers in each cluster are similar to each other demographically. Now we can create a profile for each group, considering the common characteristics of each cluster.

Conclusion

In conclusion, K-means clustering is one of the most popular and extensively used techniques for data cluster analysis. However, its performance is usually not as good as other sophisticated clustering techniques. But it can still provide us great insights and help us understand the data