Data Mining 1 part assignment

Teammates : Batyrkhan , Sultan , Ayazhan , Alua.

BD-2004 group

In [1]:

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

In [2]:

```
df = pd.read_csv("transactions.csv",sep =';')
train = pd.read_csv('train_set.csv',sep=';')
test = pd.read_csv("test_set.csv",sep=';')
codes = pd.read_csv('codes.csv',sep=';')
types = pd.read_csv('types.csv',sep=';')
new_df = df.copy()
```

Read the transactions, train_set, test_set, codes and type datasets

Descriptive Statistics

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 130039 entries, 0 to 130038
Data columns (total 5 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   client_id  130039 non-null  int64  
 1   datetime   130039 non-null  object 
 2   code       130039 non-null  int64  
 3   type       130039 non-null  int64  
 4   sum        130039 non-null  float64
dtypes: float64(1), int64(3), object(1)
memory usage: 5.0+ MB
```

We used Pandas info() function to show us column names and their corresponding data types. As we can see, int64, float64 and object are the data types of our features, 3 features are of type object, and 4 features are numeric. With this same method, we can easily see if there are any missing values

In [4]:

```
Q1 = df['sum'].quantile(0.05)
Q3 = df['sum'].quantile(0.95)
IQR = Q3 - Q1
df = df[(df['sum'] >= Q1 - 1.5*IQR) & (df['sum'] <= Q3 + 1.5*IQR)]
```

We have calculated the lower limit and upper limit of 'sum' to calculate the thresholds. We used 0.05 and 0.95 quantiles to trim down the amount of data that is seen as outliers. Then we calculated First and Third quartiles. Using those quartiles calculated IQR. Then using IQR calculated limits for our values to lie in between

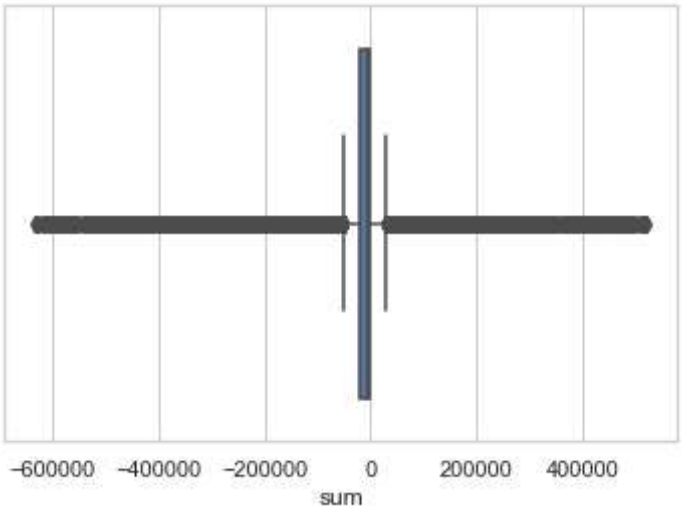
Outliers

In [5]:

```
sns.set_theme(style="whitegrid")
sns.boxplot(x=df["sum"])
```

Out[5]:

```
<AxesSubplot:xlabel='sum'>
```



We pass the "sum" field of the dataset in the x parameter which then generates the horizontal box plot. The box in the middle shows the spread of data

In [6]:

```
df
```

Out[6]:

	client_id	datetime	code	type	sum
0	96372458	421 06:33:15	6011	2010	-561478.94
1	24567813	377 17:20:40	6011	7010	67377.47
2	21717441	55 13:38:47	6011	2010	-44918.32
4	85302434	151 10:34:12	4814	1030	-3368.87
5	31421357	398 00:00:00	5411	1110	-1572.14
...	...	...	...	...	...
130034	15836839	147 11:50:53	5411	1010	-26344.59
130035	28369355	305 11:59:34	4829	2330	-24705.07
130036	40949707	398 21:13:58	5411	1110	-40353.72
130037	7174462	409 13:58:14	5411	1010	-25536.06
130038	92197764	319 00:00:00	5533	1110	-12127.95

126476 rows × 5 columns

In our df we have 126476 rows and 5 columns

In [7]:

```
print("Mean: "+str(df['sum'].mean()))
print("Median: "+str(df['sum'].median()))
print("Mode: "+str(df['sum'].mode()))
print("Standard deviation: "+ str(df['sum'].std()))
```

```
Mean: -17086.196120530276
Median: -5435.12
Mode: 0   -2245.92
dtype: float64
Standard deviation: 87024.42407699286
```

We calculated Mean, Median, Mode, and Standard Deviation

Visualization

In [8]:

```
codes_df = pd.merge(df,codes)
```

In [9]:

```
a = codes_df.groupby('code_description').code.count().sort_values(ascending = False)
```

In [10]:

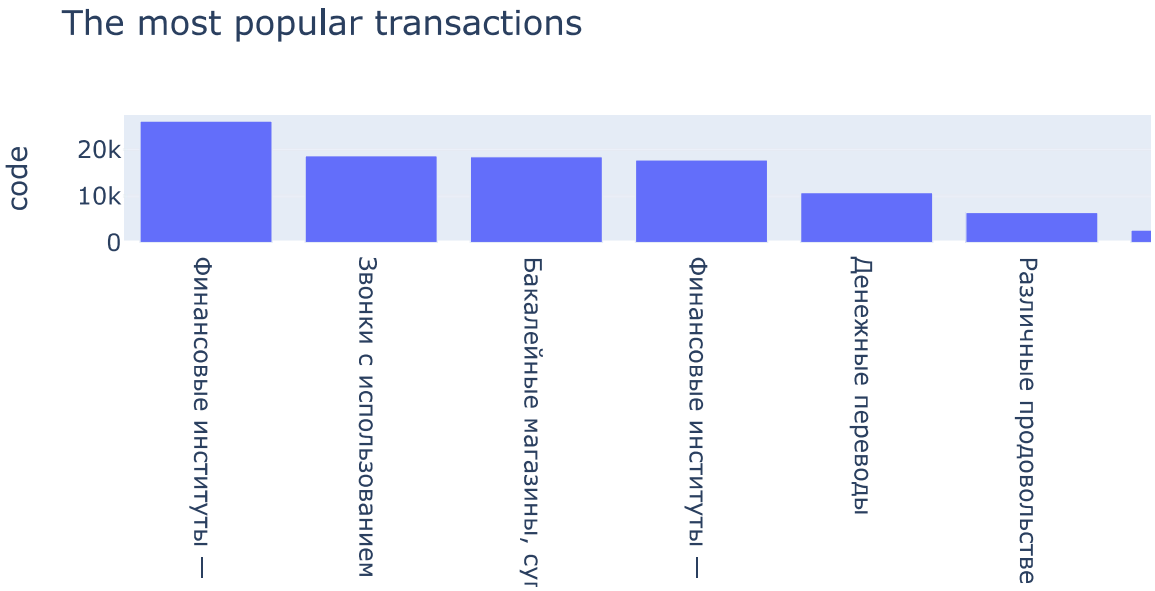
```
a = a.to_frame().reset_index()
```

At first, we merged dataframe which consists of transactions data and codes data. Then we grouped by code description and count code columns. We use to_frame to convert a series to a Data Frame

The most popular transactions

In [11]:

```
import plotly.express as px
fig = px.bar(data_frame=a.head(10), x='code_description', y='code',title = "The most popula
fig.show()
plt.tight_layout()
```




```
<Figure size 432x288 with 0 Axes>
```

We imported Plotly Express built-in part of the plotly library, for creating most common figures. In the x and y we set column names. In the graph we see the most popular transaction is Financial Institute.

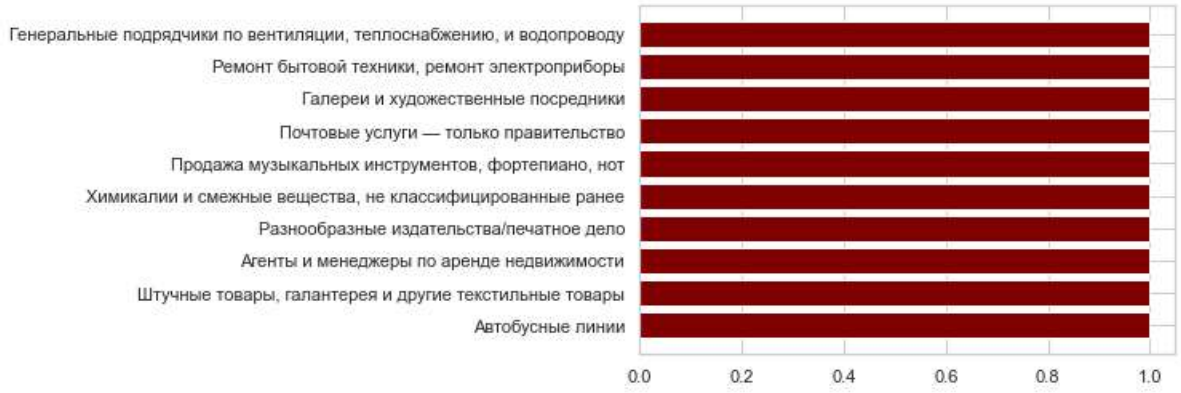
Not popular transactions

In [12]:

```
plt.barh(a['code_description'].tail(10),a['code'].tail(10) , color='maroon')
```

Out[12]:

```
<BarContainer object of 10 artists>
```



As we can see, all of transactions in tail in unpopular transactions list

Their types

In [13]:

```
types_df = pd.merge(df,types)
```

In [14]:

```
b = types_df.groupby('type_description').type.count().sort_values(ascending = False)
```

In [ ]:

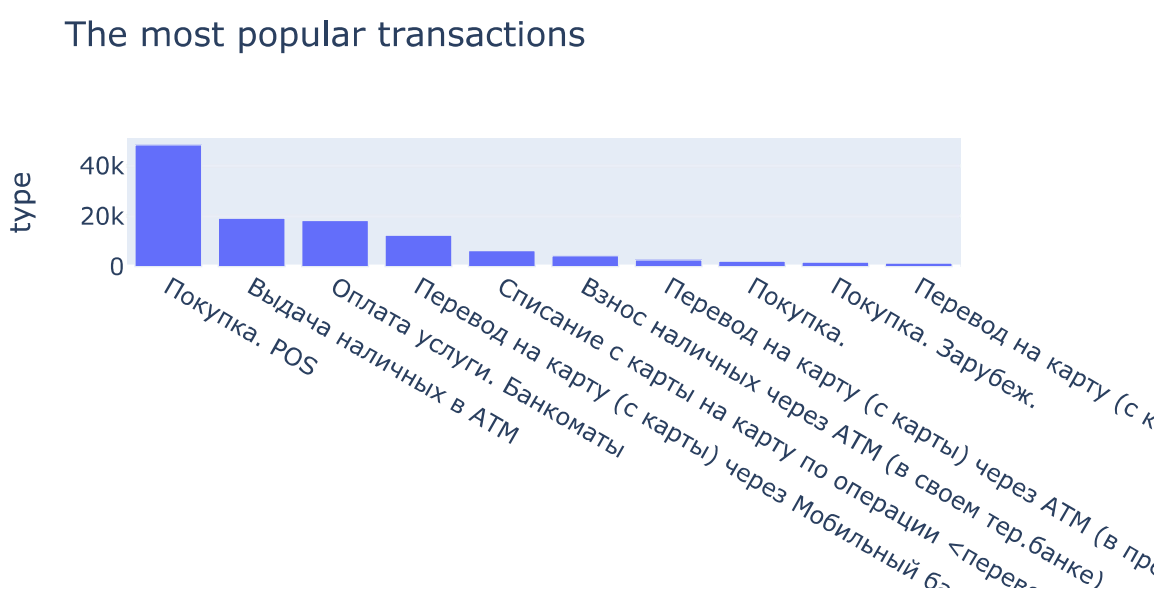
```
b = b.to_frame().reset_index()
b
```

First of all we merge two datasets(dataframe which consists of transactions data and another dataframe with transaction types).

Then we count each type of transactions and distribute them so that it is clear which of them is most common and which is less common. In order to show this more clearly, we use Plotly Express (the built-in part of the library).

In [16]:

```
import plotly.express as px
plt.figure(figsize=(20,15))
fig = px.bar(data_frame=b.head(10), x='type_description', y='type',title = "The most popula
fig.show()
plt.tight_layout()
```



<Figure size 1440x1080 with 0 Axes>

It is clearly shown here that the Purchase. POS is in a priority place. And the transfer to the card from the card via a mobile application is rarely performed.

Not popular types

As mentioned earlier, the transfer of money from card to card takes the first place here, crediting, payment, debit from the card occupy a last place in the list of transactions.

In [17]:

```
plt.barh(b['type_description'].tail(10),b['type'].tail(10) , color='maroon')
```

Out[17]:

```
<BarContainer object of 10 artists>
```

In [18]:

```
ordered_date = new_df['datetime'].str.split(" ",n=1,expand=True)
new_df["Ordered day number"] = ordered_date[0]
new_df['Ordered day number'] = pd.to_numeric(new_df['Ordered day number'])
new_df.describe()
```

Out[18]:

	client_id	code	type	sum	Ordered day number
count	1.300390e+05	130039.000000	130039.000000	1.300390e+05	130039.000000
mean	5.086859e+07	5594.629996	2489.372135	-1.812909e+04	243.258246
std	2.872854e+07	606.087084	2253.296578	5.584445e+05	130.355972
min	2.289900e+04	742.000000	1000.000000	-4.150030e+07	0.000000
25%	2.577174e+07	5211.000000	1030.000000	-2.244916e+04	132.000000
50%	5.235837e+07	5641.000000	1110.000000	-5.502490e+03	250.000000
75%	7.506302e+07	6010.000000	2370.000000	-1.122960e+03	356.000000
max	9.999968e+07	9402.000000	8145.000000	6.737747e+07	456.000000

Here we describe each column of dataframe with transaction data.

New column "Ordered day number" consists of day number, which was splitted from datetime column for clarity.

In [19]:

```
ord_day_num = new_df.groupby(['Ordered day number']).agg({'client_id' : 'count','sum' : 'su
ord_day_num.rename(columns = {'sum' : 'Total sum of all transactions'}, inplace=True)
print('We have in total 456 days of record \n If 456/30 = 15.2, we can give assumption that
print('\nSo that means we have total one cycle of all year and in addition to this 3 MONTHS
ord_day_num = ord_day_num.sort_values('Ordered day number',ascending=True)
ord_day_num
```

```
We have in total 456 days of record
 If 456/30 = 15.2, we can give assumption that approx 15 months

So that means we have total one cycle of all year and in addition to this 3
MONTHS
```

Out[19]:

Ordered day number	client_id	Total sum of all transactions
0	225	-2552528.95
1	223	-9243211.80
2	172	-3597756.37
3	232	-11484276.81
4	216	21534070.53
...	...	...
452	336	3595090.10
453	361	-7223600.99
454	352	-32630756.77
455	373	1125548.27
456	303	-1064634.47

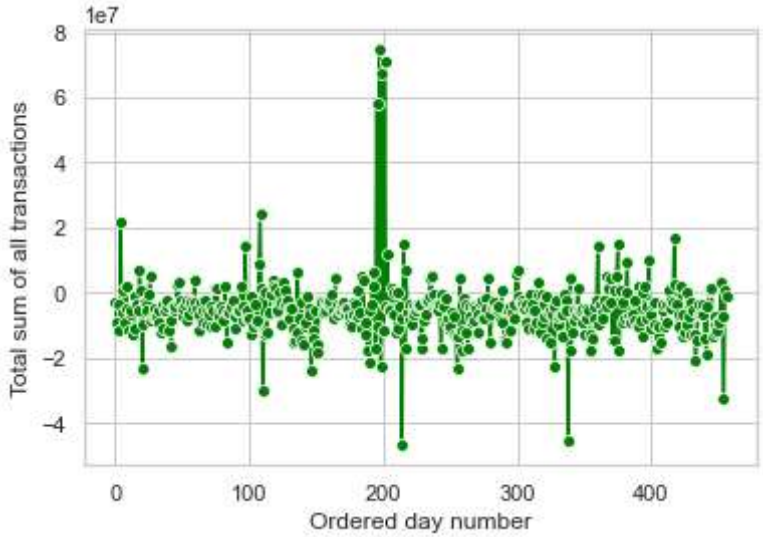
457 rows × 2 columns

In [20]:

```
sns.lineplot(data=ord_day_num, x='Ordered day number', y='Total sum of all transactions', l
```

Out[20]:

```
<AxesSubplot:xlabel='Ordered day number', ylabel='Total sum of all transacti
ons'>
```



Then lets analyze the "most" popular transactions between women and men.

Themost popular transactions between women and men

For this we do next steps:

1)First we merge all needed dataframes with codes. type of transactions.

2)Then we choose women (as we can see here "target = 0" was assigned as woman)

3)Then we group it by descriptions of code in order to count each of them.

4)At the end we show the popularity of each code in ascending order and visualize it for more clarity.

In [21]:

```
z = pd.merge(df,codes)
```

In [22]:

```
f = pd.merge(z,types)
```

In [23]:

```
a = pd.merge(f,train)
```

In [24]:

```
women = a[a['target']==0]
```

In [25]:

```
women = women.groupby('code_description').code.count().sort_values(ascending = False)
```

In [26]:

```
women = women.to_frame()
```

In [71]:

```
women = women.reset_index()
women
```

Out[71]:

	index	code_description	code
0	0	Финансовые институты — снятие наличности автом...	9529
1	1	Звонки с использованием телефонов, считывающих...	7429
2	2	Бакалейные магазины, супермаркеты	6800
3	3	Финансовые институты — снятие наличности вручную	6501
4	4	Денежные переводы	4247
...	...	...	...
144	144	Товары длительного пользования — нигде более н...	1
145	145	Драгоценные камни и металлы, часы и ювелирные ...	1
146	146	Жилье — отели, мотели, курорты	1
147	147	Колледжи, университеты, профессиональные школы...	1
148	148	Штучные товары, галантерея и другие текстильны...	1

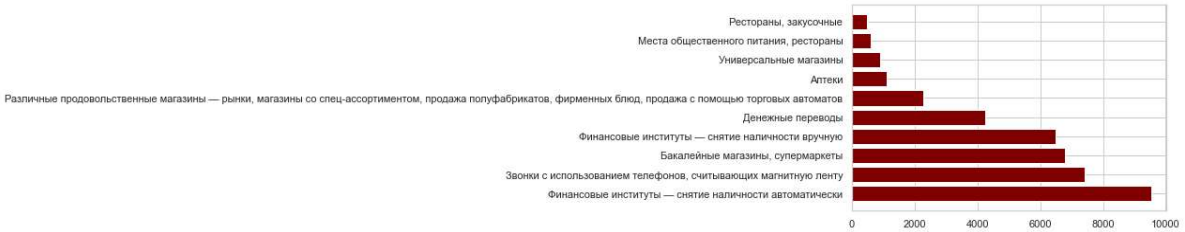
149 rows × 3 columns

In [28]:

```
plt.barh(women['code_description'].head(10),women['code'].head(10) , color='maroon')
```

Out[28]:

```
<BarContainer object of 10 artists>
```



In [29]:

```
women.head(10)
```

Out[29]:

	code_description	code
0	Финансовые институты — снятие наличности автом...	9529
1	Звонки с использованием телефонов, считывающих...	7429
2	Бакалейные магазины, супермаркеты	6800
3	Финансовые институты — снятие наличности вручную	6501
4	Денежные переводы	4247
5	Различные продовольственные магазины — рынки, ...	2277
6	Аптеки	1112
7	Универсальные магазины	901
8	Места общественного питания, рестораны	595
9	Рестораны, закусочные	493

The next is men's transactions:

Here we do same steps as previous:

1) First we merge all needed dataframes with codes. type of transactions.

2)Then we choose men (as we can see here "target = 1" was assigned as men)

3)Then we group it by descriptions of code in order to count each of them.

4)At the end we show the popularity of each code in ascending order and visualize it for more clarity.

In [30]:

```
men = a[a['target']==1]
```

In [31]:

```
men = men.groupby('code_description').code.count().sort_values(ascending = False)
```

In [32]:

```
men= men.to_frame()
```

In [33]:

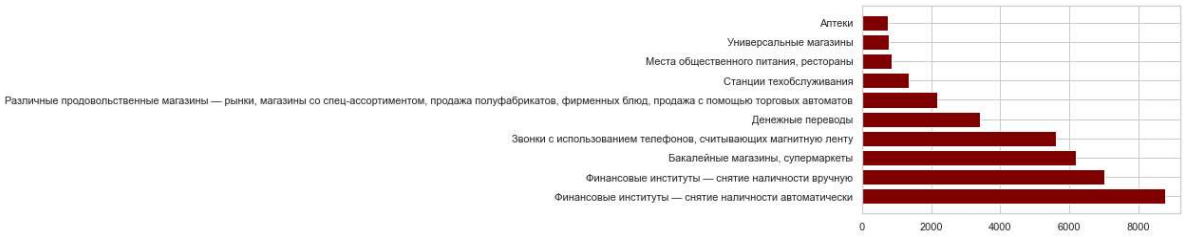
```
men = men.reset_index()
```

In [34]:

```
plt.barh(men['code_description'].head(10),men['code'].head(10) , color='maroon')
```

Out[34]:

```
<BarContainer object of 10 artists>
```



In [35]:

```
men.head(10)
```

Out[35]:

	code_description	code
0	Финансовые институты — снятие наличности автом...	8773
1	Финансовые институты — снятие наличности вручную	7017
2	Бакалейные магазины, супермаркеты	6202
3	Звонки с использованием телефонов, считывающих...	5615
4	Денежные переводы	3415
5	Различные продовольственные магазины — рынки, ...	2179
6	Станции техобслуживания	1357
7	Места общественного питания, рестораны	850
8	Универсальные магазины	780
9	Аптеки	750

Finally, we see that women spend more than men.

Feauture Engineering

In [36]:

```
dates = df['datetime'].str.split(" ",n=1,expand=True)
```

Here we split the date from datetime column.

In [37]:

```
df['datetime'] = dates[0]
```

In [38]:

```
date_list = []

for i in range(0, len(df['datetime'])) :

    day_number = int(df['datetime'].iloc[i])
    year = '2021'

    if day_number > 365:
        day_number = day_number - 365
        if day_number == 0:
            day_number = 30

    month_interval = day_number/30.4

    if month_interval >= 0 and month_interval < 1:
        month_index = '07'
        month_name = 'July'
        day = day_number
        if day == 0:
            day = 1
    elif month_interval > 1 and month_interval < 0:
        month_index = '08'
        month_name = 'August'
        day = day_number - 31
        if day == 0:
            day = 1
    elif month_interval > 2 and month_interval < 3:
        month_index = '09'
        month_name = 'September'
        day = day_number - 61
        if day == 0:
            day = 1
    elif month_interval > 3 and month_interval < 4:
        month_index = '10'
        month_name = 'October'
        day = day_number - 92
        if day == 0:
            day = 1
    elif month_interval > 4 and month_interval < 5:
        month_index = '11'
        month_name = 'November'
        day = day_number - 122
        if day == 0:
            day = 1
    elif month_interval > 5 and month_interval < 6:
        month_index = '12'
        month_name = 'December'
        day = day_number - 153
        if day == 0:
            day = 1
    elif month_interval > 6 and month_interval < 7:
        month_index = '01'
        month_name = 'January'
        day = day_number - 184
        if day == 0:
            day = 1
    elif month_interval > 7 and month_interval < 7.98:
        month_index = '02'
        month_name = 'February'
```

```
        day = day_number - 212
        if day == 0:
            day = 1
    elif month_interval > 7.98 and month_interval < 9:
        month_index = '03'
        month_name = 'March'
        day = day_number - 243
        if day == 0:
            day = 1
    elif month_interval > 9 and month_interval < 10:
        month_index = '04'
        month_name = 'April'
        day = day_number - 273
        if day == 0:
            day = 1
    elif month_interval > 10 and month_interval < 11:
        month_index = '05'
        month_name = 'May'
        day = day_number - 304
        if day == 0:
            day = 1
    elif month_interval > 11 and month_interval < 12:
        month_index = '06'
        month_name = 'June'
        day = day_number - 334
        if day == 0:
            day = 1

    strr = year + "-" + month_index + "-" + str(day)
    date_list.append(strr)


print(len(date_list))

# Date_Col = pd.DataFrame (date_list, columns = ['Date'])
# Date_Col

# date_df = pd.concat([transactions, Date_Col], axis=1, join="inner")
```

```
126476
```

Here we bring time into a normal and understandable form (generate them). For example, if a day is more than 365 days, then we accordingly minus 365 from it to get a new day.

In [39]:

```
df["datetime"] = date_list
```

In [40]:

```
df['datetime'] = np.where(df['datetime'] == '2021-02-29','2021-03-01',df['datetime'])
```

In [41]:

```
df['datetime'] = np.where(df['datetime'] == '2021-02-30','2021-03-02',df['datetime'])
```

In [42]:

```
df['datetime'] = np.where(df['datetime'] == '2021-01--1','2021-01-01',df['datetime'])
```

By "where" function, our datetime change the day of February to March, since we don't have 29th and 30th in this month.

In [43]:

```
df["datetime"] = pd.to_datetime(df['datetime'])
```

In [44]:

```
df.dtypes
```

Out[44]:

```
client_id             int64
datetime     datetime64[ns]
code                  int64
type                  int64
sum                 float64
dtype: object
```

finally convert this column data type to datetime type

In [45]:

```
rfm_r = df.groupby('client_id')["datetime"].max()
rfm_r = rfm_r.reset_index()
rfm_r.columns = ['client_id','LastPurshaceDate']
rfm_r.head()
```

Out[45]:

	client_id	LastPurshaceDate
0	22899	2021-12-14
1	27914	2021-12-14
2	28753	2021-10-20
3	31385	2021-12-20
4	38084	2021-12-19

rfm_r dataframe holds data about recently purchases and frequency of each clients purchasing.

Top 5 clients who have recently made a purchase

In [46]:

```
import datetime;

# ct stores current time
ct = datetime.datetime.now()
```

In [47]:

```
rfm_r["LastPurshaceDate"] = pd.to_datetime(rfm_r['LastPurshaceDate'])
```

In [48]:

```
rfm_r['Recency'] = rfm_r['LastPurshaceDate'].apply(lambda x: (ct - x).days)
```

In [49]:

```
rfm_r.drop('LastPurshaceDate',axis=1,inplace=True)
```

In [50]:

```
rfm = df.groupby('client_id')['sum'].sum()
```

In [51]:

```
rfm_f = df.groupby('client_id')['code'].count()
rfm_f = rfm_f.reset_index()
rfm_f.columns = ['client_id', 'Frequency']
rfm_f.head()
#here we calculate the transaction usage of each client by its ID
```

Out[51]:

	client_id	Frequency
0	22899	9
1	27914	4
2	28753	10
3	31385	13
4	38084	25

Result of counting each transactions of clients. For instance client with 22899 id make 9 purchases and so on.

In [52]:

```
rfm = pd.merge(rfm, rfm_f, on='client_id', how='inner')
```

In [53]:

```
rfm = pd.merge(rfm,rfm_r, on = 'client_id', how = 'inner')
```

In [54]:

```
Q1 = rfm['sum'].quantile(0.05)
Q3 = rfm['sum'].quantile(0.95)
IQR = Q3 - Q1
rfm = rfm[(rfm['sum'] >= Q1 - 1.5*IQR) & (rfm['sum'] <= Q3 + 1.5*IQR)]
```

Using IQR we delete outliers

Create a RFM table

In [55]:

```
from scipy import stats
rfm["Recency"] = stats.boxcox(rfm['Recency'])[0]
rfm["Frequency"] = stats.boxcox(rfm['Frequency'])[0]
rfm["sum"] = pd.Series(np.cbrt(rfm['sum'])).values
```

We used boxcox to bring to a normalized view

We used the scheme RFM. Recency - is simply the amount of time since the customer's most recent transaction. Frequency-is the total number of transactions. Monetary - sum, total amount has spent

In [56]:

```
rfm
```

	client_id	sum	Frequency	Recency
0	22899	37.047307	2.643215	1.538362
1	27914	42.005141	1.555790	1.538362
2	28753	-79.841317	2.795368	1.583935
3	31385	-43.712551	3.185828	1.529324
4	38084	-66.547277	4.235063	1.530935
...	...	...	...	...
8633	99967537	-69.581680	0.000000	1.621778
8634	99984336	42.837145	1.555790	1.572220
8635	99985917	-60.785196	0.000000	1.602594
8636	99991245	20.107933	1.203350	1.614855
8637	99999680	-86.694715	3.405954	1.529324

We calculated RFM metrics for each customer

client_id 22899 has frequency: 2.643215, sum(monetary value) : 37.047307 and recency: 1.538362

In [75]:

```
from sklearn.preprocessing import StandardScaler
rfm_df = rfm[['sum', 'Frequency','Recency']]

scaler = StandardScaler()


rfm_df_scaled = scaler.fit_transform(rfm_df)
rfm_df_scaled.shape
```

Out[75]:

```
(8594, 3)
```

In [ ]:

```
```

In [58]:

```
rfm_df_scaled = pd.DataFrame(rfm_df_scaled)
rfm_df_scaled.columns = ['sum', 'Frequency','Recency']
rfm_df_scaled.head()
```

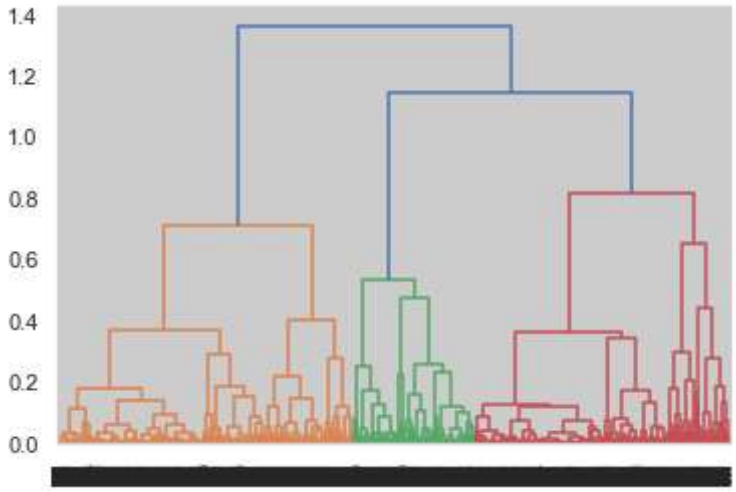
Out[58]:

	sum	Frequency	Recency
0	1.619690	-0.088575	-0.766176
1	1.721394	-0.890240	-0.766176
2	-0.778146	0.023594	0.616437
3	-0.037006	0.311447	-1.040405
4	-0.505434	1.084958	-0.991519

Using the dendrogram to find the optimal numbers of clusters. First thing we're going to do is to import scipy library. scipy is #an open source Python library that contains tools to do #hierarchical clustering and building dendrograms. Only import the #needed tool

In [59]:

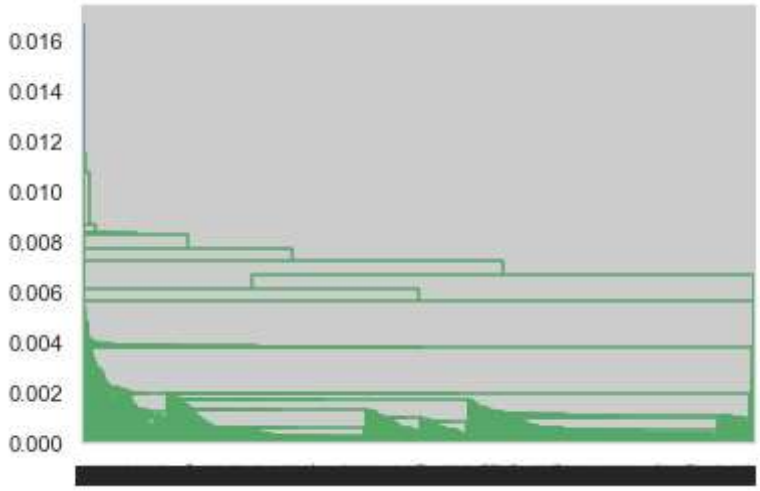
```
import scipy.cluster.hierarchy as model
dend_max = model.dendrogram(model.linkage(rfm_df_scaled, method='average',metric='cosine'))
```



Lets create a dendrogram variable linkage is actually the algorithm itself of hierarchical clustering and then in linkage we have to #specify on which data we apply and engage. This is rfm_df_scaled dataset

In [76]:

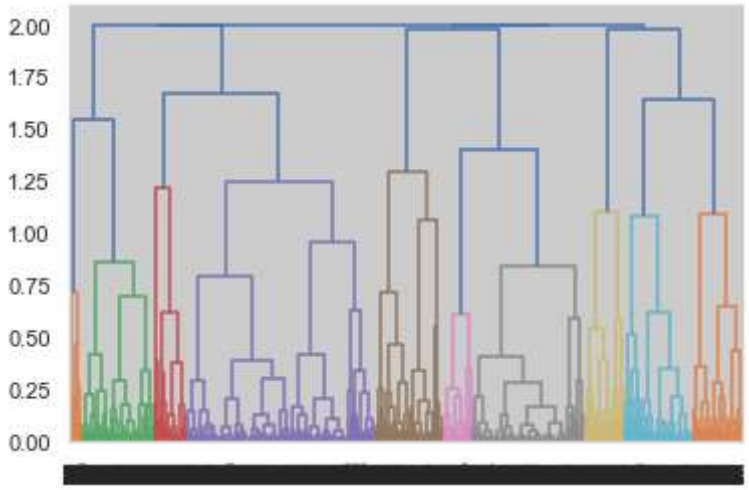
```
import scipy.cluster.hierarchy as model
dend_single = model.dendrogram(model.linkage(rfm_df_scaled, method='single',metric='cosine'
```



Plot the hierarchical clustering using Single Linkage

In [82]:

```
dend_complete = model.dendrogram(model.linkage(rfm_df_scaled, method='complete',metric='cos
```



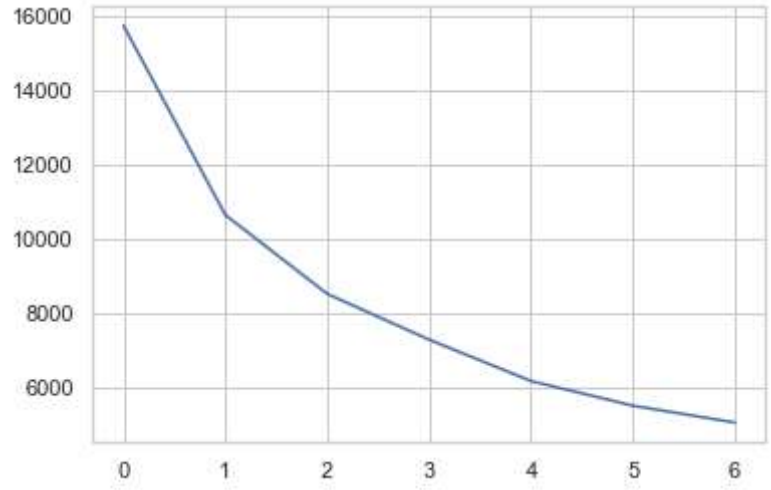
Hierarchical clustering using Complete Linkage

In [81]:

```
#### from sklearn.cluster import KMeans
from sklearn.cluster import KMeans
ssd = []
range_n_clusters = [2, 3, 4, 5, 6, 7, 8]
for num_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=num_clusters, max_iter=50)
    kmeans.fit(rfm_df_scaled)

    ssd.append(kmeans.inertia_)

# plot the SSDs for each n_clusters
plt.plot(ssd)
```

Out[81]:

```
[<matplotlib.lines.Line2D at 0x20729ca3880>]
```



In [80]:

```
rfm_df_scaled = np.array(rfm_df_scaled)
```

KMeans Clustering

In [ ]:

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, max_iter=100)
kmeans.fit(rfm_df_scaled)
```

Compute clustering

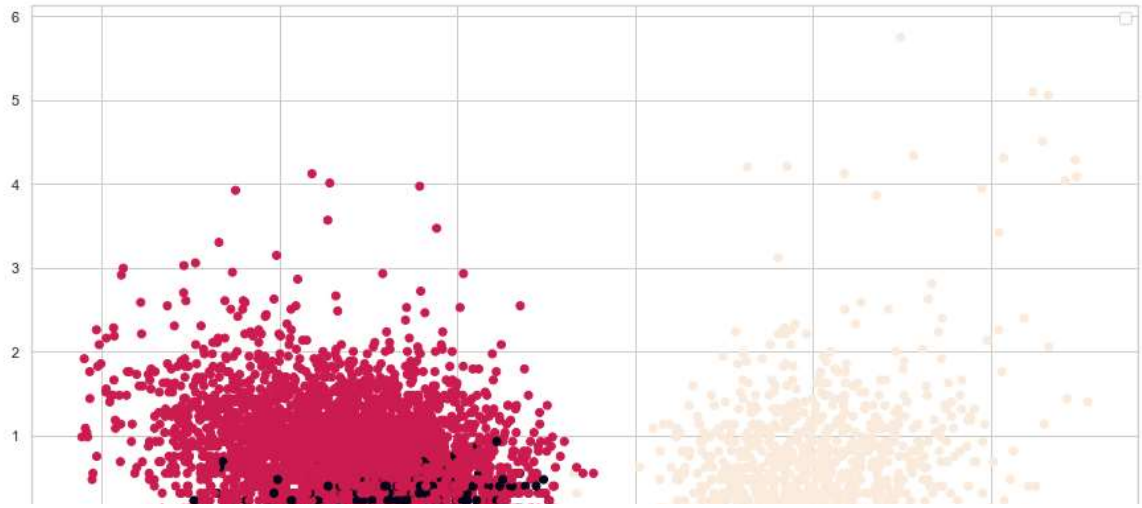
In [73]:

```
fig = plt.figure(figsize=(15,10))
plt.scatter(rfm_df_scaled[:, 0], rfm_df_scaled[:, 1], c=kmeans.labels_)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], c=[0, 1,2,], mark
plt.legend()
```

```
No handles with labels found to put in legend.
```

Out[73]:

```
<matplotlib.legend.Legend at 0x2072bf33f10>
```



We plot the cluster centers as determined by the k-means estimator.

In [ ]:

```
k = kmeans.labels_
```

In [78]:

```
rfm['cluster'] = k
```

In [79]:

```
rfm
```

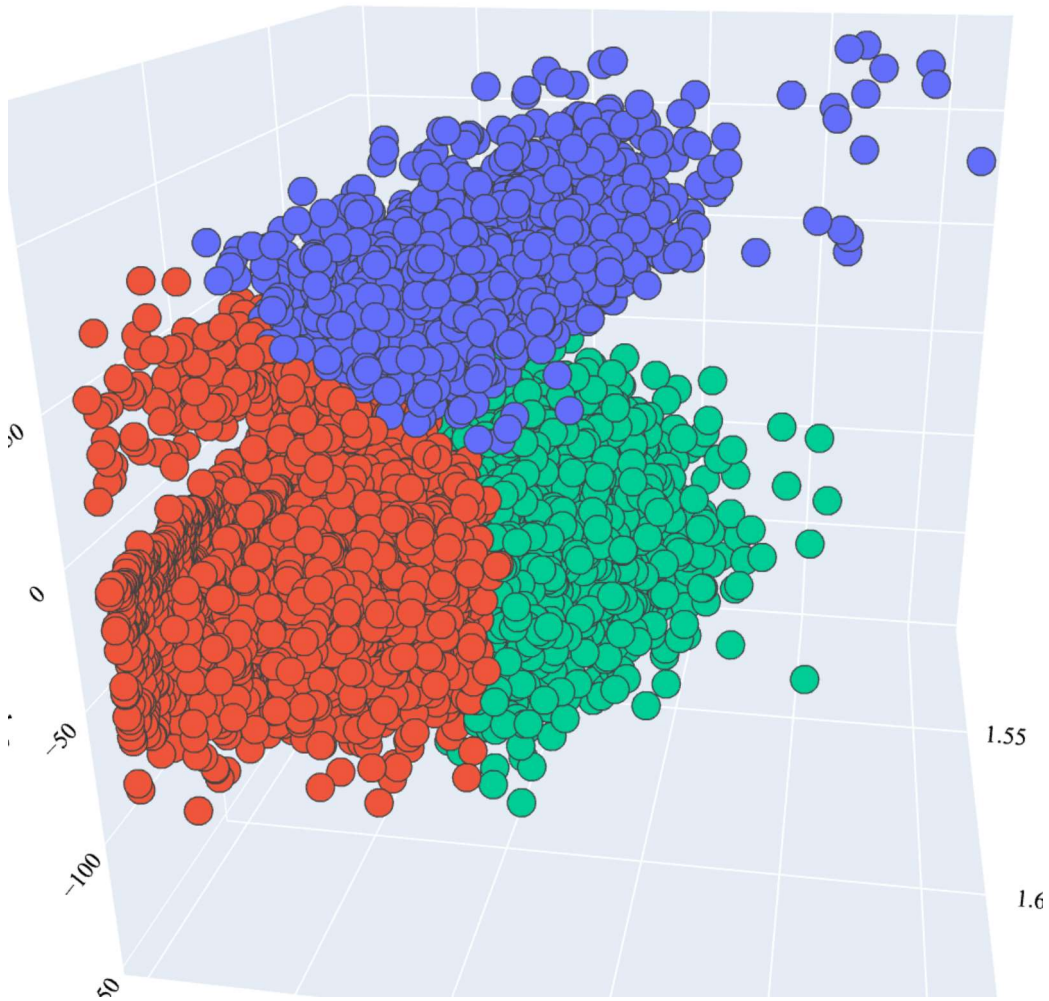
Out[79]:

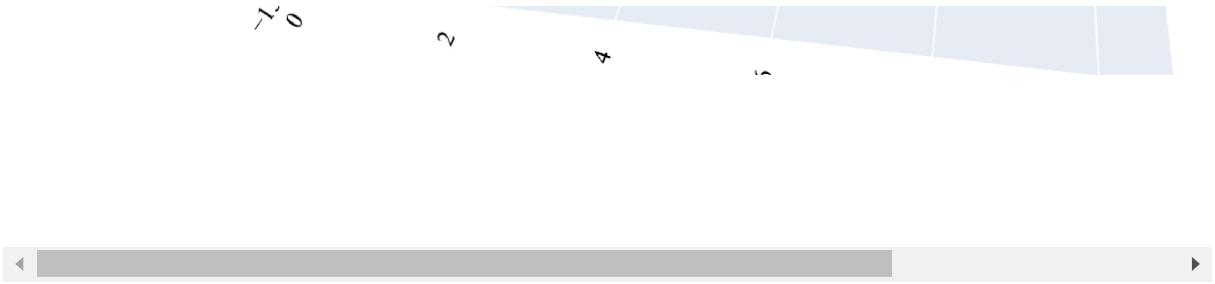
	client_id	sum	Frequency	Recency	cluster
0	22899	37.047307	2.643215	1.538362	2
1	27914	42.005141	1.555790	1.538362	2
2	28753	-79.841317	2.795368	1.583935	0
3	31385	-43.712551	3.185828	1.529324	1
4	38084	-66.547277	4.235063	1.530935	1
...	...	...	...	...	...
8633	99967537	-69.581680	0.000000	1.621778	0
8634	99984336	42.837145	1.555790	1.572220	2
8635	99985917	-60.785196	0.000000	1.602594	0
8636	99991245	20.107933	1.203350	1.614855	0
8637	99999680	-86.694715	3.405954	1.529324	1

8594 rows × 5 columns

In [72]:

```
import plotly.graph_objects as go

PLOT = go.Figure()

for C in list(rfm.cluster.unique()):

    PLOT.add_trace(go.Scatter3d(x = rfm[rfm.cluster == C]['Recency'],
                                y = rfm[rfm.cluster == C]['Frequency'],
                                z = rfm[rfm.cluster == C]['sum'],
                                mode = 'markers', marker_size = 8, marker_line_width = 1,
                                name = 'Cluster ' + str(C)))


PLOT.update_layout(width = 800, height = 800, autosize = True, showlegend = True,
                   scene = dict(xaxis=dict(title = 'Recency', titlefont_color = 'black'),
                                yaxis=dict(title = 'Frequency', titlefont_color = 'black'),
                                zaxis=dict(title = 'Monetary value', titlefont_color = 'bla
                   font = dict(family = "Gilroy", color  = 'black', size = 12))
```





Here in the figure, can be represented that we segmented our data into 3 clusters, cluster number had been identified by the elbow plot, and we used unsupervised Kmeans algorithm to make such type of decisions.

So, here by red color identify clients that used transactions a bit early, comapring with green ones, because by Frequency axis, which shows how frequent client did transactions

What about Blue dots?

Mailnly, they are stable clients of the organization, which make high (Monetary Value), frequent (Frequency) transactions every time (Recency)

Green dots that type of clients which makes small transactions and mainly makes expenditure comparing with blue ones