Analysis of Corporate Clients Based on Sales Documents

Prepared by: Anna Fischer and Boglárka Póra

Project Objectives:

- Exploration and transformation of available data into a model-ready format
- Execution of RFM analysis
- Creation of customer clusters
- Proposal of strategies based on the results

Data Description:

- DocumentId: The sales document number, a unique identifier - text
- PartnerId: The unique identifier of the commercial partner (client) - text
- DocumentDate: The date of the sales document - date
- City: The headquarters location of the commercial partner (client) - text
- Sales: The value of the sales document (invoice) - numeric

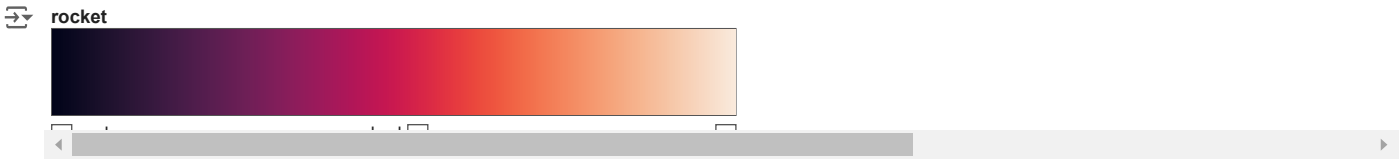
Methodology:

- Exploratory Data Analysis (EDA)
- RFM analysis
- Clustering algorithms
- Dimensionality reduction algorithms
- Interactive visualization techniques

Importing Libraries and Selecting Color Palette

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


sns.set_style("whitegrid")
sns.set_palette("rocket")


sns.color_palette()
```

```
sns.color_palette("rocket", as_cmap = True)
```




```
palette = sns.color_palette()
c_palette = sns.color_palette("rocket", as_cmap = True)
```

Data Import

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
data = pd.read_excel('/content/drive/MyDrive/Big Data Adatelemzes/data.xlsx')
data.head()
```

	DocumentId	PartnerId	DocumentDate	City	Sales
0	822217	331	2022-09-27	BRASOV	-79375.42
1	821727	12508	2022-08-01	NaN	-4345.24
2	840424	8939	2022-12-20	TIMISOARA	-2952.60
3	758393	1098	2022-01-11	BUCURESTI SECTORUL 6	-2428.48
4	830027	7569	2022-10-28	TIMISOARA	-2205.90

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
# Or
#data = pd.read_excel('data.xlsx')
#data.head()
```

Data Exploration and Cleaning

```
data.columns
```

```
Index(['DocumentId', 'PartnerId', 'DocumentDate', 'City', 'Sales'], dtype='object')
```

```
data.shape
```

```
(9818, 5)
```

As we can see, the dataset intended for analysis consists of 9,818 rows and 5 columns.

```
data.dtypes
```

	0
DocumentId	int64
PartnerId	int64
DocumentDate	datetime64[ns]
City	object
Sales	float64

We can observe the data types of the dimensions. The presence of int, float, and datetime types indicates that there are no erroneous values in the dataset that would convert a column to object type. However, there might still be missing values.

```
data.describe()
```

	DocumentId	PartnerId	DocumentDate	Sales
count	9818.000000	9818.000000	9818	9818.000000
mean	799375.430739	6961.042575	2022-06-29 09:40:39.845182464	829.039757
min	756724.000000	4.000000	2022-01-03 00:00:00	-79375.420000
25%	776993.250000	1141.000000	2022-04-05 00:00:00	270.475000
50%	800005.000000	8191.000000	2022-06-30 00:00:00	588.165000
75%	821269.750000	10961.000000	2022-09-22 00:00:00	1027.402500
max	843594.000000	12812.000000	2022-12-30 00:00:00	24313.420000
std	25175.856289	4416.929374	NaN	1446.307460

The DocumentId and PartnerId appear as numeric values, but these metrics do not hold meaningful significance for these variables. However, from this, we can easily observe that:

- The sales data covers the year 2022.
- There are also negative invoice values, as the minimum value across the entire dataset is -79,375.42.

```
data.duplicated().sum()
```

```
0
```

This indicates that there are no duplicate rows in the dataframe.

The next step is to systematically check each dimension for inappropriate values and clean them, or to fill in any missing values.

DocumentId

```
data['DocumentId'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 9818 entries, 0 to 9817
Series name: DocumentId
Non-Null Count  Dtype
--------------  -----
9818 non-null   int64
dtypes: int64(1)
memory usage: 76.8 KB
```

```
data['DocumentId'].nunique()
```

```
9818
```

There are as many unique values in this column as there are rows in the dataset, meaning there are no duplicate document IDs.

```
data['DocumentId'].isna().any()
```

```
False
```

None of the variable values are missing.

Since this is a unique identifier, we will not use it in the feature engineering process, as it does not provide additional information about the invoice record beyond identification.

PartnerId

```
data['PartnerId'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 9818 entries, 0 to 9817
Series name: PartnerId
Non-Null Count  Dtype
--------------  -----
9818 non-null   int64
dtypes: int64(1)
memory usage: 76.8 KB
```

```
data['PartnerId'].nunique()
```

```
814
```

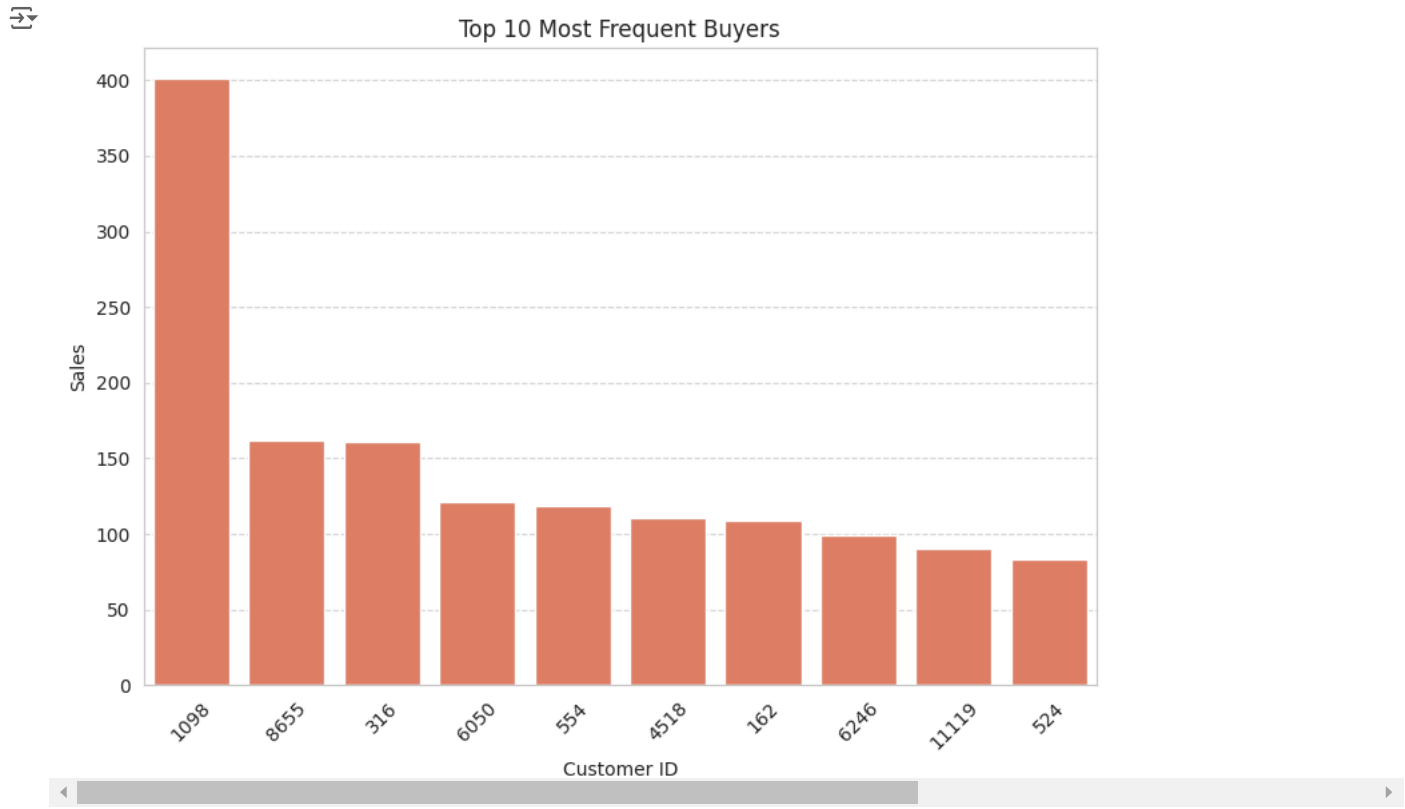
We know that the company invoiced 814 different clients in 2022.

```
data['PartnerId'].isna().any()
```

```
False
```

No invoice was issued without specifying the client.

```
partner_counts = data['PartnerId'].value_counts()
top_10_partners = partner_counts.head(10).sort_values(ascending=False)

plt.figure(figsize=(8, 6))
sns.barplot(x=top_10_partners.index, y=top_10_partners.values, color=palette[4], order=top_10_partners.index)

plt.title('Top 10 Most Frequent Buyers')
plt.xlabel('Customer ID')
plt.ylabel('Sales')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```



From this chart, we can easily identify which clients are the most active purchasers of our company and how many invoices have been issued to each of them.

DocumentDate

```
data['DocumentDate'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 9818 entries, 0 to 9817
Series name: DocumentDate
Non-Null Count  Dtype
--------------  -----
9818 non-null   datetime64[ns]
dtypes: datetime64[ns](1)
memory usage: 76.8 KB
```

```
data['DocumentDate'].nunique()
```

```
250
```

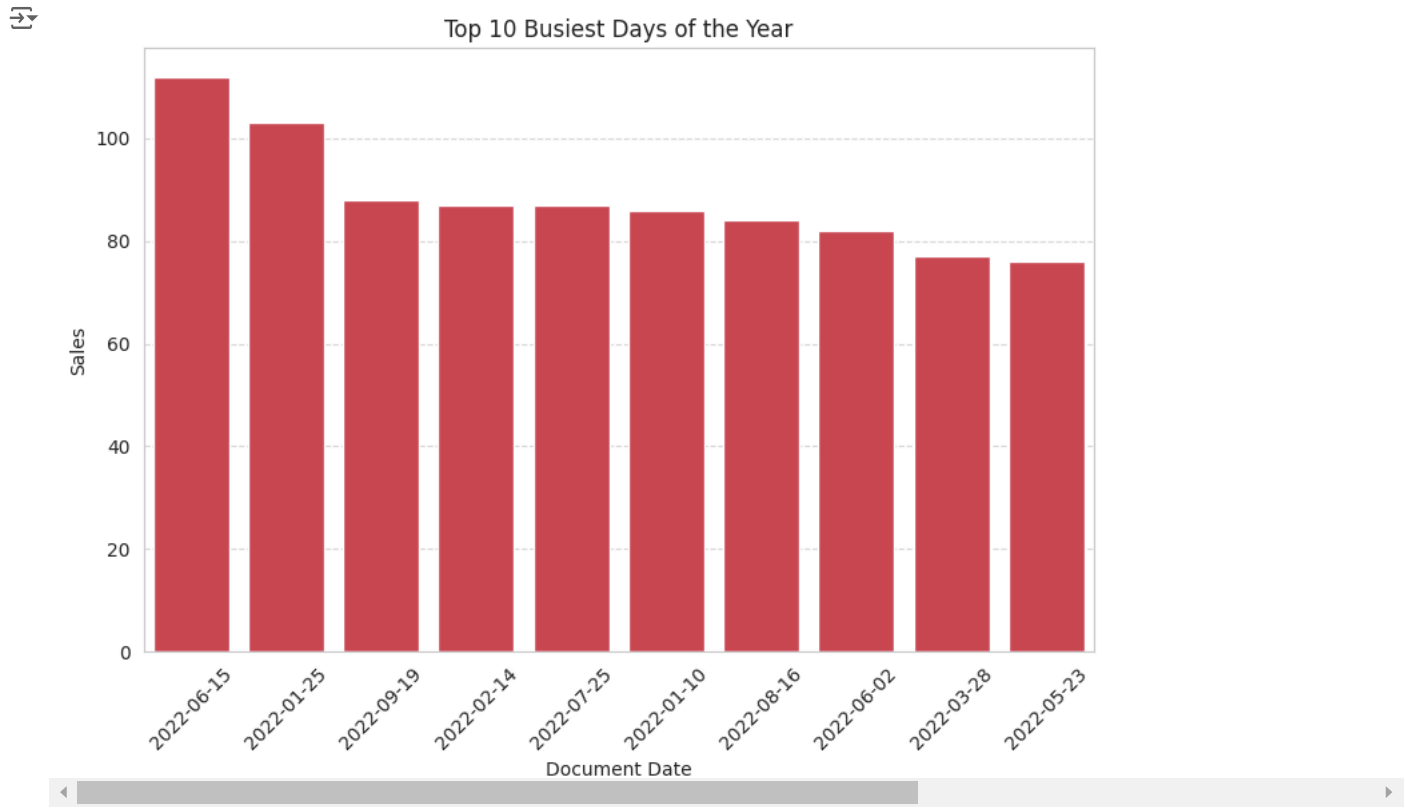
Sales occurred on 250 days of the year at the company.

```
data['DocumentDate'].isna().any()
```

```
False
```

No dates are missing from any of the invoices.

```
day_counts = data['DocumentDate'].value_counts()
top_10_days = day_counts.head(10).sort_values(ascending=False)

plt.figure(figsize=(8, 6))
sns.barplot(x=top_10_days.index, y=top_10_days.values, color=palette[3], order=top_10_days.index)

plt.title('Top 10 Busiest Days of the Year')
plt.xlabel('Document Date')
plt.ylabel('Sales')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```



The period of the year with the most profitable days varies significantly.

City

```
data['City'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 9818 entries, 0 to 9817
Series name: City
Non-Null Count  Dtype
--------------  -----
9807 non-null   object
dtypes: object(1)
memory usage: 76.8+ KB
```

```
data['City'].nunique()
```

```
256
```

The company's clients are headquartered in 256 different cities.

```
data['City'].isna().any()
```

```
True
```

```
data['City'].isna().sum()
```

```
11
```

In the headquarters dimension, there are already missing values (specifically 11). We need to examine which ones are missing.

```
data[data['City'].isna()]
```

	DocumentId	PartnerId	DocumentDate	City	Sales
1	821727	12508	2022-08-01	NaN	-4345.24
795	801975	7863	2022-07-08	NaN	81.59
1524	814134	7863	2022-07-27	NaN	165.59
4032	784554	7863	2022-05-05	NaN	457.11
6686	822062	12508	2022-09-01	NaN	879.46
6871	820064	7863	2022-09-16	NaN	912.08
7569	797966	7863	2022-05-04	NaN	1080.40
9401	761887	9179	2022-01-26	NaN	2486.83
9602	797970	7863	2022-05-05	NaN	3104.35
9697	814595	12508	2022-07-18	NaN	4345.24
9785	796390	12414	2022-06-02	NaN	11072.32

First, we can check if the headquarters information for the missing values appears on any other invoices belonging to the same partner, and if the omission was just accidental.

```
data[data['PartnerId'] == 12508]
```

	DocumentId	PartnerId	DocumentDate	City	Sales
1	821727	12508	2022-08-01	NaN	-4345.24
6686	822062	12508	2022-09-01	NaN	879.46
9697	814595	12508	2022-07-18	NaN	4345.24

```
data[data['PartnerId'] == 7863]
```

	DocumentId	PartnerId	DocumentDate	City	Sales
795	801975	7863	2022-07-08	NaN	81.59
1524	814134	7863	2022-07-27	NaN	165.59
4032	784554	7863	2022-05-05	NaN	457.11
6871	820064	7863	2022-09-16	NaN	912.08
7569	797966	7863	2022-05-04	NaN	1080.40
9602	797970	7863	2022-05-05	NaN	3104.35

```
data[data['PartnerId'] == 9179]
```

	DocumentId	PartnerId	DocumentDate	City	Sales
9401	761887	9179	2022-01-26	NaN	2486.83

```
data[data['PartnerId'] == 12414]
```

	DocumentId	PartnerId	DocumentDate	City	Sales
9785	796390	12414	2022-06-02	NaN	11072.32

Unfortunately, the headquarters information does not appear on any other invoices for the missing cases. Therefore, we will set these values to "Unknown."

```
data['City'].fillna('Unknown', inplace=True)
```

```
data[data['City'] == 'Unknown']
```

	DocumentId	PartnerId	DocumentDate	City	Sales
1	821727	12508	2022-08-01	Unknown	-4345.24
795	801975	7863	2022-07-08	Unknown	81.59
1524	814134	7863	2022-07-27	Unknown	165.59
4032	784554	7863	2022-05-05	Unknown	457.11
6686	822062	12508	2022-09-01	Unknown	879.46
6871	820064	7863	2022-09-16	Unknown	912.08
7569	797966	7863	2022-05-04	Unknown	1080.40
9401	761887	9179	2022-01-26	Unknown	2486.83
9602	797970	7863	2022-05-05	Unknown	3104.35
9697	814595	12508	2022-07-18	Unknown	4345.24
9785	796390	12414	2022-06-02	Unknown	11072.32

```
data['City'].isna().any()
```

```
False
```

The missing values have now been resolved.

Next, we need to check for city names that might appear as separate cities due to minor typographical errors but actually refer to the same location. These should be standardized to a consistent format.

```
unique_cities = sorted(data['City'].unique())
for city in unique_cities:
    print(city)
```

```
VLAHITA
VOLUNTARI
VULCAN
ZABALA
ZALAU
ZAM
ZARNESTI
ZETEA
ZOLTAN
```

We will remove hyphens from the city names:

```
data['City'] = data['City'].str.replace('-', ' ')
```

We will generalize headquarters located in different sectors of Bucharest to simply "Bucharest" to reflect the total number of companies based in Bucharest:

```
data['City'] = data['City'].str.replace(r'(.*)BUCURESTI SECTORUL(.*)', r'BUCURESTI', regex=True)
```

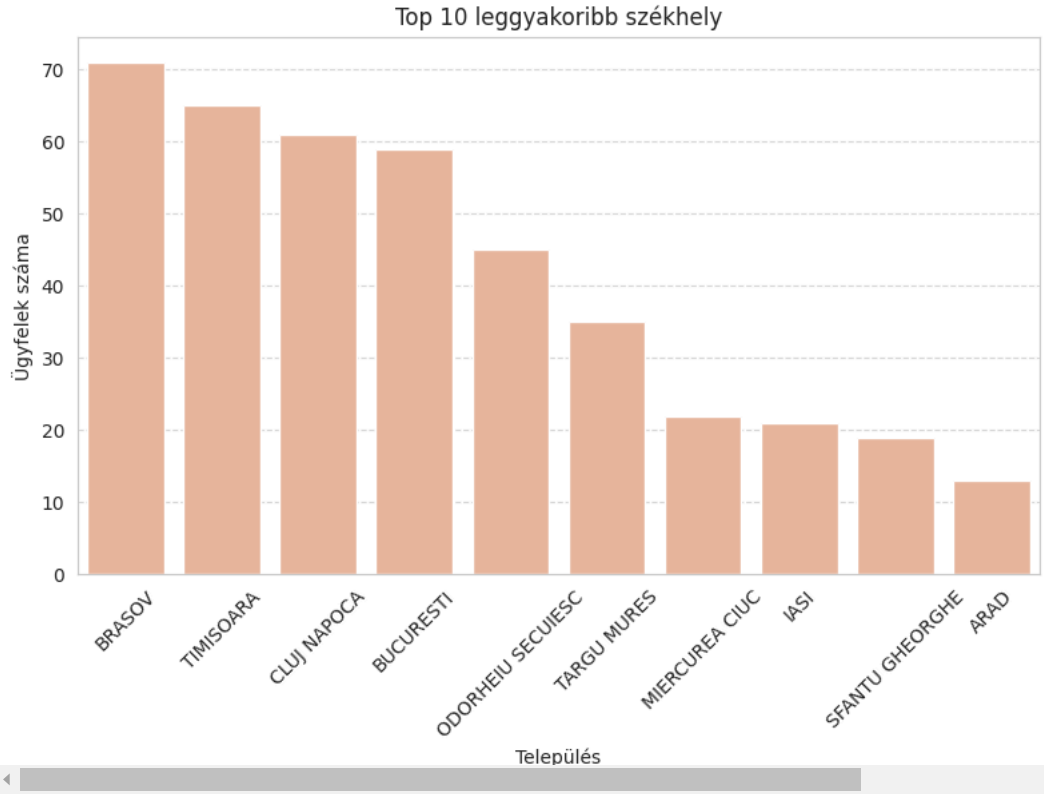
The following changes correct various city names or replace county names with "Unknown" for the city where only the county name is provided:

```
mask = data['City'] == 'CLUJ'
data.loc[mask, 'City'] = 'CLUJ Unknown City'

mask = data['City'] == 'HARGHITA'
data.loc[mask, 'City'] = 'HARGHITA Unknown City'

data['City'] = data['City'].str.replace('OD SECUIESC', 'ODORHEIU SECUIESC')
data['City'] = data['City'].str.replace('ODORHEIUL SECUIESC', 'ODORHEIU SECUIESC')

data['City'] = data['City'].str.replace('TG MURES', 'TARGU MURES')

data.loc[data['City'].str.contains('^TIMIS$', regex=True), 'City'] = 'TIMIS Unknown City'

data['City'] = data['City'].str.replace('TG SECUIESC', 'TARGU SECUIESC')

data['City'] = data['City'].str.replace(r'\s+', ' ', regex=True)

data['City'] = data['City'].str.replace(r'(.*)(GHEORGHE)(.*)', r'SFANTU GHEORGHE', regex=True)


unique_cities = sorted(data['City'].unique())
for city in unique_cities:
    print(city)
```

```
TURENI
TUSNAD
TUSNADU NOU
UNGURENI
URSENI
Unknown
VALEA LUI MIHAI
VALIUG
VARSAG
VISEU DE SUS
VLADIMIRESCU
VLAHA
VLAHITA
VOLUNTARI
VULCAN
ZABALA
ZALAU
ZAM
ZARNESTI
ZETEA
ZOLTAN
```

We can see that only unique city names remain in the dataset.

```
cities = data.drop_duplicates(subset=['PartnerId', 'City'])

city_counts = cities['City'].value_counts()

top_10_city = city_counts.head(10).sort_values(ascending=False)

plt.figure(figsize=(8, 6))
sns.barplot(x=top_10_city.index, y=top_10_city.values, color=palette[5], order=top_10_city.index)

plt.title('Top 10 leggyakoribb székhely')
plt.xlabel('Település')
plt.ylabel('Ügyfelek száma')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```



The majority of our company's partners are headquartered in Brașov.

Sales

```
data['Sales'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 9818 entries, 0 to 9817
Series name: Sales
Non-Null Count  Dtype
--------------  -----
9818 non-null   float64
dtypes: float64(1)
memory usage: 76.8 KB
```

Since the data type is float, it ensures that only numbers are present in the variable values, and no typographical errors have been mixed in.

```
data['Sales'].isna().any()
```

```
False
```

```
data.dtypes
```

	0
DocumentId	int64
PartnerId	int64
DocumentDate	datetime64[ns]
City	object
Sales	float64

dtype: object

There are no missing values among the invoice amounts.

```
data[data['Sales'] <= 0]
```

	DocumentId	PartnerId	DocumentDate	City	Sales
0	822217	331	2022-09-27	BRASOV	-79375.42
1	821727	12508	2022-08-01	Unknown	-4345.24
2	840424	8939	2022-12-20	TIMISOARA	-2952.60
3	758393	1098	2022-01-11	BUCURESTI	-2428.48
4	830027	7569	2022-10-28	TIMISOARA	-2205.90
...	...	...	...	...	...
338	833782	1720	2022-11-15	SACALAZ	0.00
339	840442	7330	2022-12-20	MIERCUREA CIUC	0.00
340	840444	8813	2022-12-20	MIERCUREA CIUC	0.00
341	840817	1804	2022-12-23	TIMISOARA	0.00
342	841065	11246	2022-12-27	IZVOARE	0.00

343 rows × 5 columns

Here, we can see that invoices may have negative values.

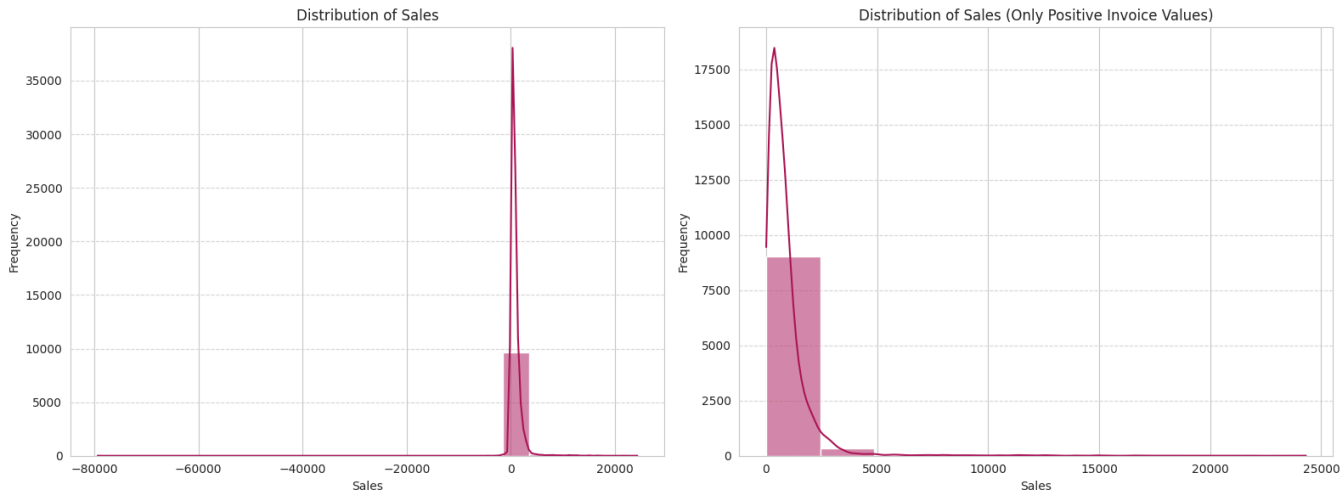
These could be due to:

- Credit invoices: The client returned the product.
- Advance payments: The client paid the full amount based on a proforma invoice.
- Invoice adjustments: Billing for a product with a discount of equal value.
- Post-purchase discounts: Discounts applied after multiple purchases.
- Other reasons.

Due to these reasons, these invoices cannot be considered separate purchases and will not be included in our further analysis.

```
data_pos = data[data['Sales'] > 0]
```

```
fig, axs = plt.subplots(1, 2, figsize=(16, 6))

sns.histplot(data['Sales'], kde=True, bins=20, color=palette[2], ax=axs[0])
axs[0].set_title('Distribution of Sales')
axs[0].set_xlabel('Sales')
axs[0].set_ylabel('Frequency')
axs[0].grid(axis='y', linestyle='--', alpha=0.7)

sns.histplot(data_pos['Sales'], kde=True, bins=10, color=palette[2], ax=axs[1])
axs[1].set_title('Distribution of Sales (Only Positive Invoice Values)')
axs[1].set_xlabel('Sales')
axs[1].set_ylabel('Frequency')
axs[1].grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```



We can see that the distribution of values changes significantly after removing negative and zero invoice amounts; however, this was a necessary step. This chart also hints at the presence of outlier values, which will need to be addressed before modeling.

```
data = data_pos

data.shape
```

```
(9475, 5)
```

We have 9,475 invoices remaining for analysis.

```
data.head()
```

	DocumentId	PartnerId	DocumentDate	City	Sales
343	779058	10592	2022-04-13	BRASOV	6.42
344	829740	12054	2022-10-27	GIROC	6.83
345	829748	12593	2022-10-27	CORUNCA	6.83
346	781716	11567	2022-04-26	BUCURESTI	10.90
347	803442	162	2022-07-14	SOVATA	11.76

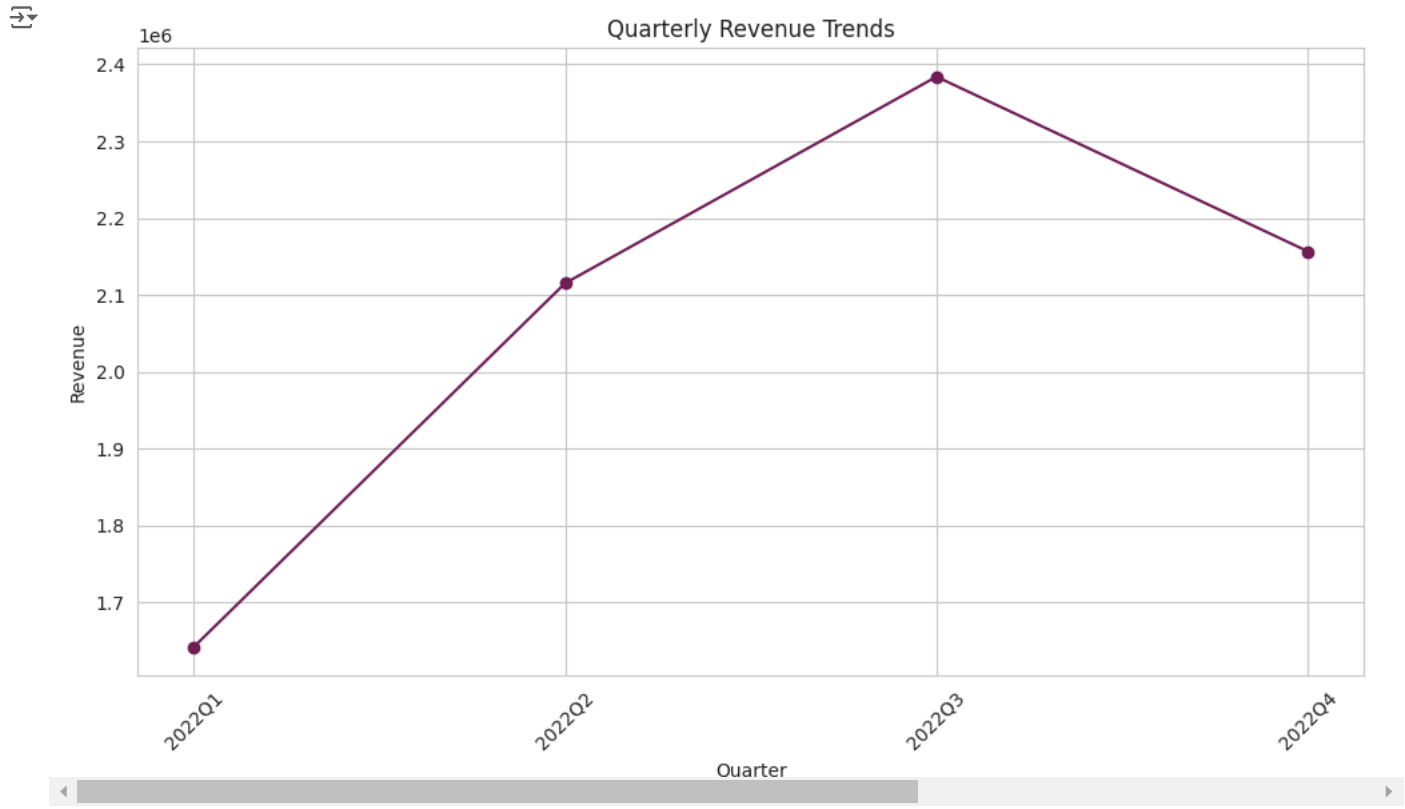
Next steps: Generate code with data | View recommended plots | New interactive sheet

Exploratory Data Analysis (EDA)

In this section, we will delve deeper into uncovering potential patterns within the data.

Quarterly Sales Trends

```
data['Quarter'] = data['DocumentDate'].dt.to_period('Q')
quarterly_sales = data.groupby('Quarter')['Sales'].sum().reset_index()

plt.figure(figsize=(10, 6))
plt.plot(quarterly_sales['Quarter'].astype(str), quarterly_sales['Sales'], marker='o', color=palette[1])
plt.title('Quarterly Revenue Trends')
plt.xlabel('Quarter')
plt.ylabel('Revenue')
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



We can see that the highest revenue for the company occurred during the summer months, in the third quarter.

The following charts display sales values that exceed the average of subsequent sales. In the chart titled "Sales by City," the total sales values for each city are shown. The chart titled "Number of Sales by City" illustrates the number of sales conducted in each city.

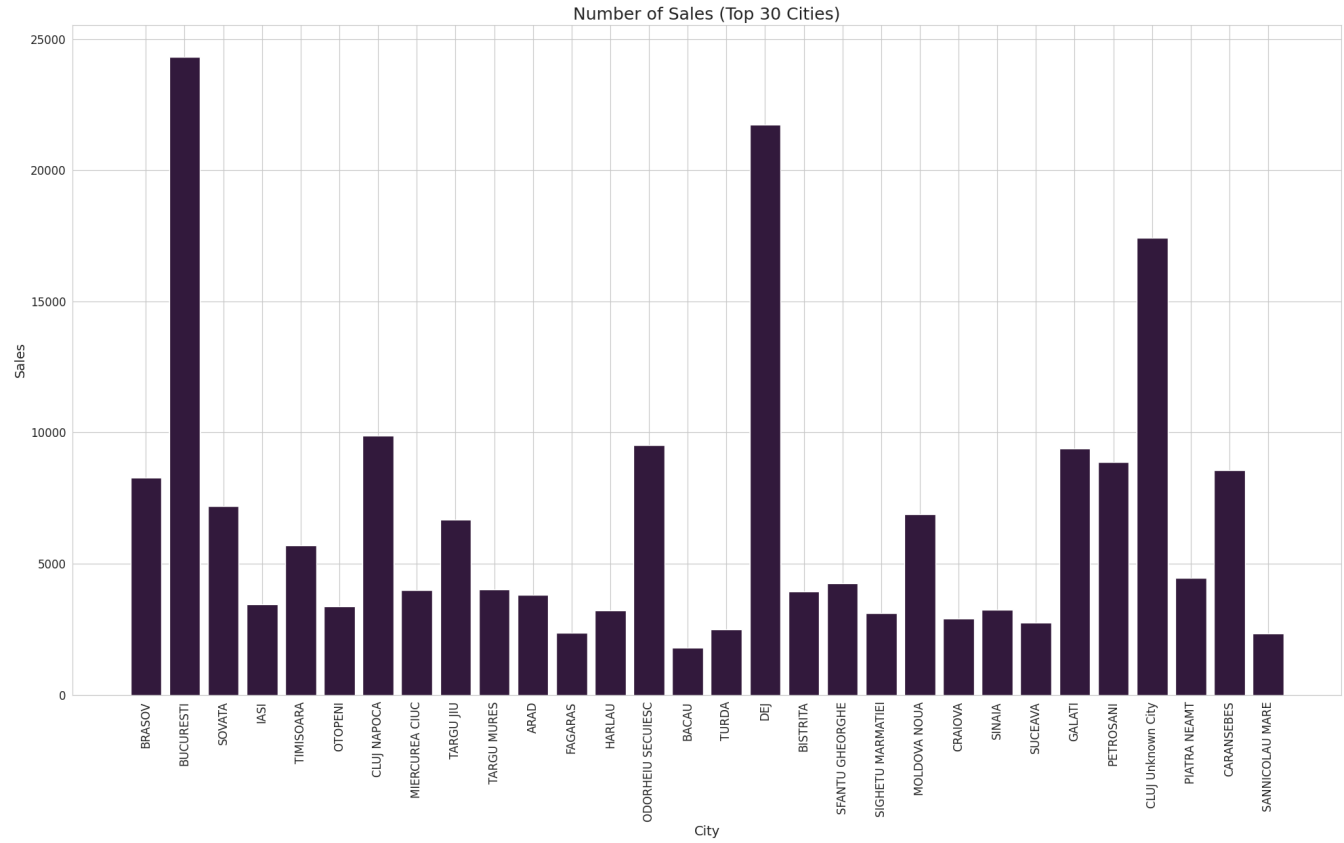
```
sales_mean = data['Sales'].mean()
sales_mean
```

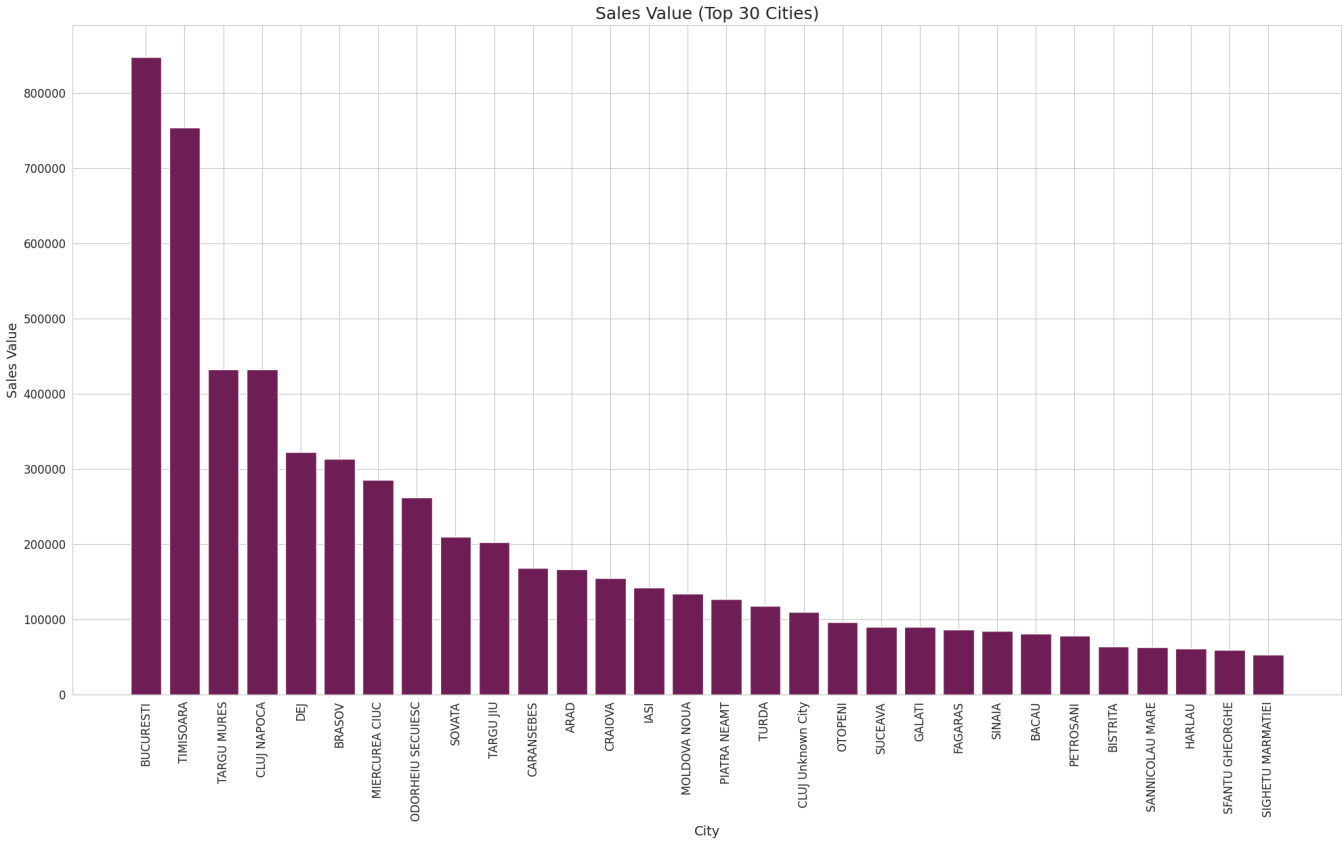
```
875.7196622691291
```

```
selected_cities = data[data['Sales'] >= sales_mean]['City']
filtered_data = data[data['City'].isin(selected_cities)]
```

Number and Value of Sales by City

```
city_sales_sum_filtered = filtered_data.groupby('City')['Sales'].sum().reset_index()
city_sales_sum_filtered.columns = ['City', 'Sales Sum']

top_30_cities = city_sales_sum_filtered.sort_values(by='Sales Sum', ascending=False).head(30)

fig, axs = plt.subplots(2, 1, figsize=(20, 25))

top_30_sales_data = data[data['City'].isin(top_30_cities['City'])]
axs[0].bar(top_30_sales_data['City'], top_30_sales_data['Sales'], color=palette[0], width=0.8)
axs[0].set_title('Number of Sales (Top 30 Cities)', fontsize=18)
axs[0].set_xlabel('City', fontsize=14)
axs[0].set_ylabel('Sales', fontsize=14)
axs[0].tick_params(axis='x', rotation=90, labelsize=12)
axs[0].tick_params(axis='y', labelsize=12)

axs[1].bar(top_30_cities['City'], top_30_cities['Sales Sum'], color=palette[1], width=0.8)
axs[1].set_title('Sales Value (Top 30 Cities)', fontsize=18)
axs[1].set_xlabel('City', fontsize=14)
axs[1].set_ylabel('Sales Value', fontsize=14)
axs[1].tick_params(axis='x', rotation=90, labelsize=12)
axs[1].tick_params(axis='y', labelsize=12)

plt.tight_layout()
plt.show()
```


Number of Sales (Top 30 Cities)
Sales
City
0
5000
10000
15000
20000
25000
BRASOV
BUCURESTI
SOVATA
IASI
TIMISOARA
OTOPENI
CLUJ NAPOCA
MIERCUREA CIUC
TARGU JIU
TARGU MURES
ARAD
FAGARAS
HARLAU
ODORHEIU SECUIESC
BACAU
TURDA
DEJ
BISTRITA
SFANTU GHEORGHE
SIGHETU MARMATIEI
MOLDOVA NOUA
CRAIOVA
SINAIA
SUCEAVA
GALATI
PETROSANI
CLUJ Unknown City
PIATRA NEAMT
CARANSEBES
SANNICOLAU MARE


Sales Value (Top 30 Cities)
Sales Value
City
0
100000
200000
300000
400000
500000
600000
700000
800000
BUCURESTI
TIMISOARA
TARGU MURES
CLUJ NAPOCA
DEJ
BRASOV
MIERCUREA CIUC
ODORHEIU SECUIESC
SOVATA
TARGU JIU
CARANSEBES
ARAD
CRAIOVA
IASI
MOLDOVA NOUA
PIATRA NEAMT
TURDA
CLUJ Unknown City
OTOPENI
SUCEAVA
GALATI
FAGARAS
SINAIA
BACAU
PETROSANI
BISTRITA
SANNICOLAU MARE
HARLAU
SFANTU GHEORGHE
SIGHETU MARMATIEI

Here are some key statistics related to the above charts.

Most Frequent Buyer

```
most_frequent_partner_count = data['PartnerId'].value_counts().max()
most_frequent_partner = data['PartnerId'].value_counts().idxmax()
city_of_most_frequent_partner = data[data['PartnerId'] == most_frequent_partner]['City'].iloc[0]

print("The ID of the most frequent buyer:", most_frequent_partner)
print("City associated with this customer:", city_of_most_frequent_partner)
print("Sales:", most_frequent_partner_count)
```

```
The ID of the most frequent buyer: 1098
City associated with this customer: BUCURESTI
Sales: 396
```

Customer with the Highest Spending

```
partner_with_highest_sales = data.groupby('PartnerId')['Sales'].sum().idxmax()
sales_of_highest_sales_partner = data.groupby('PartnerId')['Sales'].sum().max()
city_of_highest_sales_partner = data[data['PartnerId'] == partner_with_highest_sales]['City'].iloc[0]

print("The customer with the highest spending:", partner_with_highest_sales)
print("City associated with this customer:", city_of_highest_sales_partner)
print("Purchased Amount:", sales_of_highest_sales_partner)
```

```
The customer with the highest spending: 8191
City associated with this customer: TARGU MURES
Purchased Amount: 328839.63
```

Highest Sales Value:

```
max_positive_sales = data['Sales'].max()
max_positive_sales_data = data[data['Sales'] == max_positive_sales][['DocumentId', 'PartnerId', 'City']]

print("Highest Sales Value:", max_positive_sales)
print("\nThe corresponding data:")
print(max_positive_sales_data)
```

```
Highest Sales Value: 24313.420000000002

The corresponding data:
      DocumentId  PartnerId       City
9817      823019       8191  BUCURESTI
```

Lowest Sales Value:

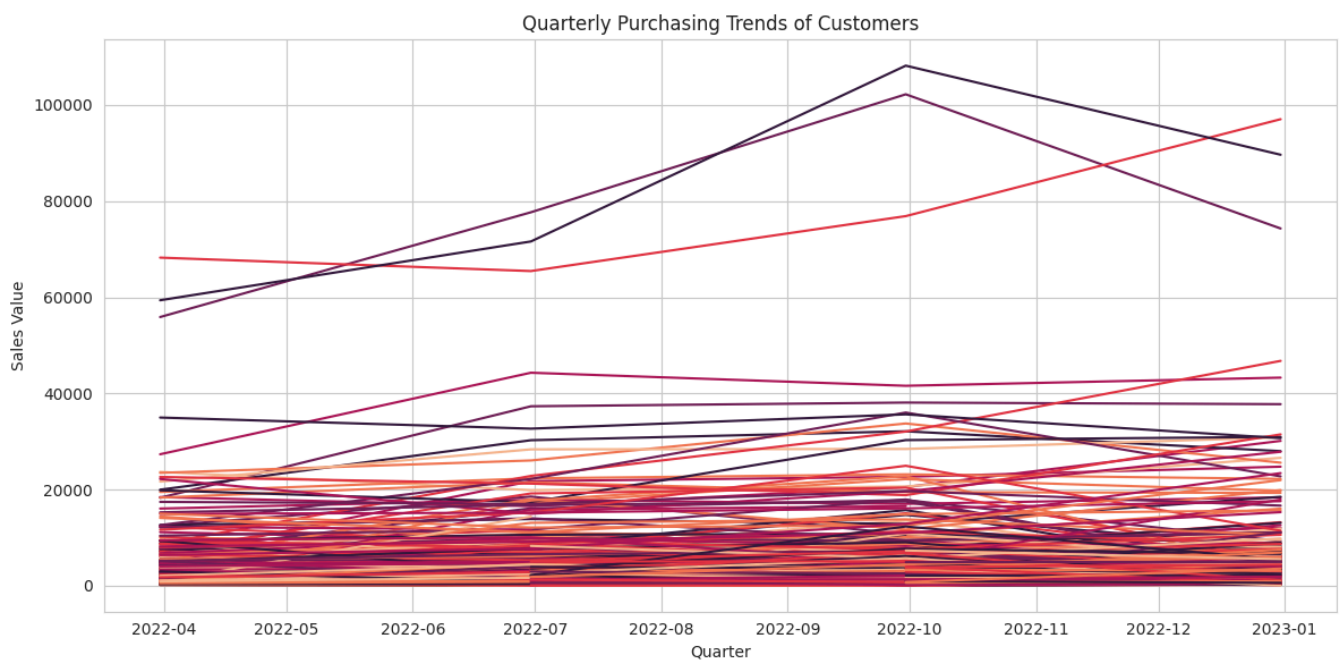
```
min_positive_sales = data['Sales'].min()
min_positive_sales_data = data[data['Sales'] == min_positive_sales][['DocumentId', 'PartnerId', 'City']]

print("Smallest Positive Sales Value:", min_positive_sales)
print("\nThe corresponding data:")
print(min_positive_sales_data)
```

```
Smallest Positive Sales Value: 6.42

The corresponding data:
     DocumentId  PartnerId    City
343      779058      10592  BRASOV
```

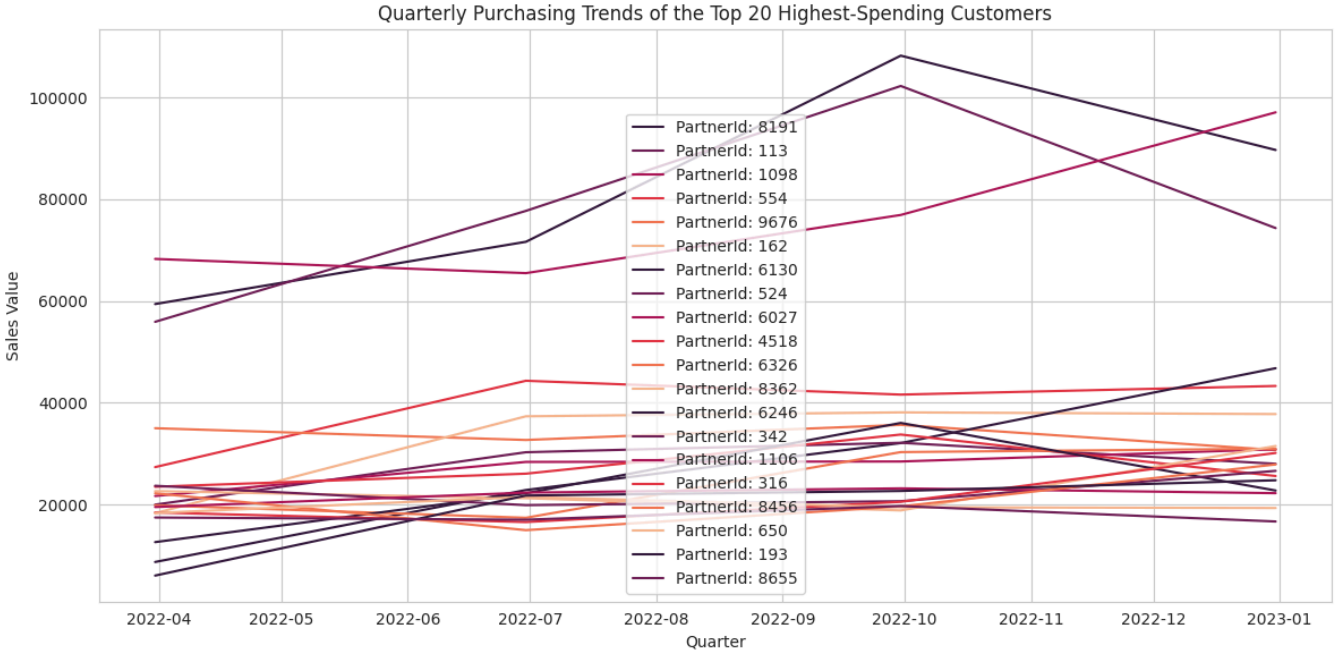
Monthly Purchasing Trends of Customers

```
data2 = data.copy()
```

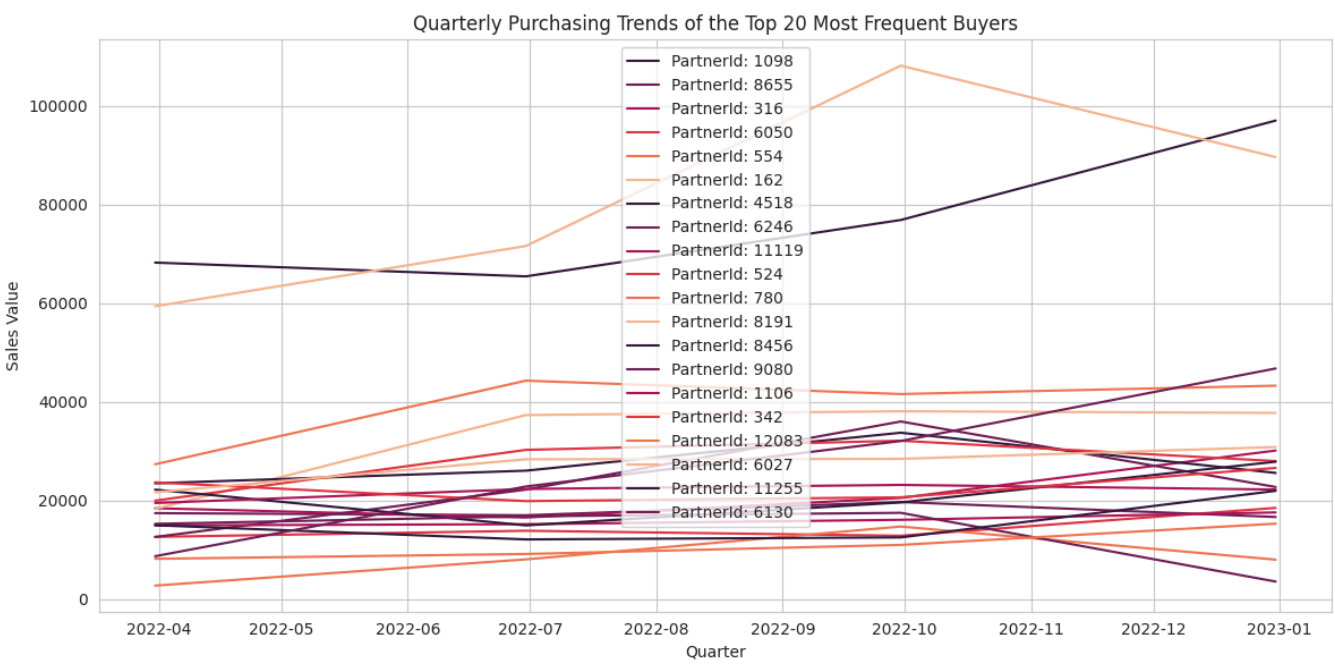
```
average_sales = data2['Sales'].mean()

data2['DocumentDate'] = pd.to_datetime(data2['DocumentDate'])
data2.set_index('DocumentDate', inplace=True)
quarterly_sales = data2.resample('Q')['Sales'].sum()

trending_customers = data2.groupby('PartnerId')['Sales'].sum()
trending_customers = trending_customers[trending_customers > average_sales]

plt.figure(figsize=(12, 6))
for partner_id in trending_customers.index:
    partner_sales = data2[data2['PartnerId'] == partner_id]['Sales'].resample('Q').sum()
    plt.plot(partner_sales.index, partner_sales.values)

plt.title('Quarterly Purchasing Trends of Customers')
plt.xlabel('Quarter')
plt.ylabel('Sales Value')
plt.grid(True)
plt.tight_layout()
plt.show()
```




```
top_20_customers = trending_customers.nlargest(20)

plt.figure(figsize=(12, 6))
for partner_id in top_20_customers.index:
    partner_sales = data2[data2['PartnerId'] == partner_id]['Sales'].resample('Q').sum()
    plt.plot(partner_sales.index, partner_sales.values, label=f'PartnerId: {partner_id}')

plt.title('Quarterly Purchasing Trends of the Top 20 Highest-Spending Customers')
plt.xlabel('Quarter')
plt.ylabel('Sales Value')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```




```
partner_sales_count = data2.groupby('PartnerId').size()
top_20_customers = partner_sales_count.nlargest(20)

plt.figure(figsize=(12, 6))
for partner_id in top_20_customers.index:
    partner_sales = data2[data2['PartnerId'] == partner_id]['Sales'].resample('Q').sum()
    plt.plot(partner_sales.index, partner_sales.values, label=f'PartnerId: {partner_id}')

plt.title('Quarterly Purchasing Trends of the Top 20 Most Frequent Buyers')
plt.xlabel('Quarter')
plt.ylabel('Sales Value')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Sales Statistics:

```
sales_std = data['Sales'].std()
sales_var = data['Sales'].var()

sales_median = data['Sales'].median()
sales_quartiles = np.percentile(data['Sales'], [25, 50, 75])

print("Sales Standard Deviation:", sales_std)
print("Variance of Sales:", sales_var)
print("Median of Sales:", sales_median)
print("First Quartile (Q1) of Sales:", sales_quartiles[0])
print("Median of Sales (Q2):", sales_quartiles[1])
print("Third Quartile (Q3) of Sales:", sales_quartiles[2])
```

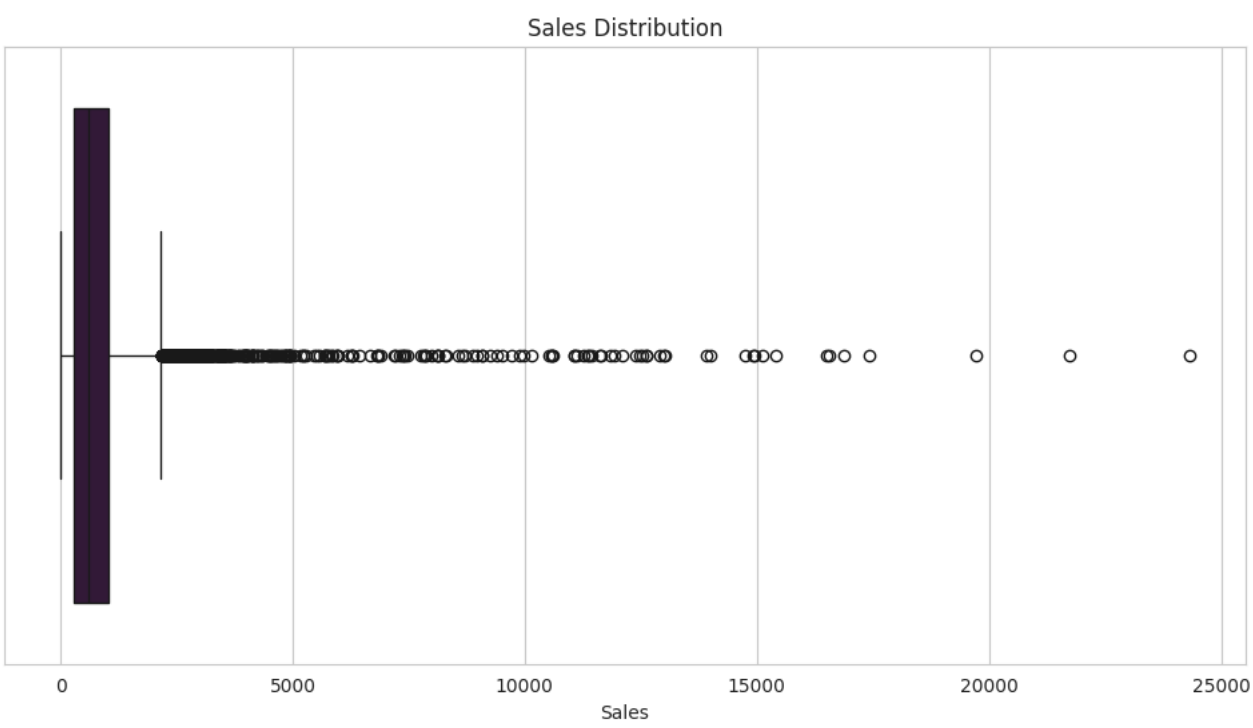
```
Sales Standard Deviation: 1198.6518553069932
Variance of Sales: 1436766.270230897
Median of Sales: 612.48
First Quartile (Q1) of Sales: 295.975
Median of Sales (Q2): 612.48
Third Quartile (Q3) of Sales: 1044.9499999999998
```

Handling Outliers

The next very important step in the analysis is handling outliers, which in some cases can distort the performance of the model.

The best way to visualize outliers is with a box plot.

```
plt.figure(figsize=(12, 6))
sns.boxplot(x=data['Sales'])
plt.title('Sales Distribution')
plt.xlabel('Sales')
plt.show()
```



We can also examine outliers using statistical methods; in this case, we will use the IQR (Interquartile Range) method.

```
Q1 = data['Sales'].quantile(0.25)
Q3 = data['Sales'].quantile(0.75)
IQR = Q3 - Q1

outliers = data[(data['Sales'] < (Q1 - 1.5 * IQR)) | (data['Sales'] > (Q3 + 1.5 * IQR))]

print(f'Number of outliers: {outliers.shape[0]}')
```

```
Number of outliers: 562
```

A significant number of outliers can be identified; however, from an analytical perspective, they are important and should not be considered erroneous data. They may contribute to cluster formation as important group characteristics.

RFM Analysis

To implement RFM (Frequency, Recency, and Monetary) analysis, it is necessary to calculate the following metrics for each partner:

- Frequency: The total number of transactions associated with each partner.
- Recency: The date of the most recent invoice for each partner.
- Monetary Value: The total monetary value of purchases made by each partner.

Recency Calculation

```
data_recency = data.groupby(by='PartnerId',
                      as_index=False)['DocumentDate'].max()
data_recency.columns = ['PartnerId', 'LastPurchaseDate']
recent_date = data_recency['LastPurchaseDate'].max()
data_recency['Recency'] = data_recency['LastPurchaseDate'].apply(
    lambda x: (recent_date - x).days)
data_recency.head()
```

	PartnerId	LastPurchaseDate	Recency
0	4	2022-12-14	16
1	113	2022-12-20	10
2	124	2022-11-07	53
3	125	2022-03-21	284
4	142	2022-10-17	74

Next steps: Generate code with `data_recency` View recommended plots New interactive sheet

We created a separate table that includes the `PartnerId`, the date of their most recent purchase, and the number of days elapsed from that purchase to the company's most recent transaction.

Frequency Calculation

```
frequency_data = data.drop_duplicates().groupby(
    by=['PartnerId'], as_index=False)['DocumentDate'].count()
frequency_data.columns = ['PartnerId', 'Frequency']
frequency_data.head()
```

	PartnerId	Frequency
0	4	2
1	113	45
2	124	7
3	125	1
4	142	3

Next steps: Generate code with `frequency_data` View recommended plots New interactive sheet

Here, we calculated the total number of transactions associated with each partner.

Monetary Value Calculation

```
monetary_data = data.groupby(by='PartnerId', as_index=False)['Sales'].sum()
monetary_data.columns = ['PartnerId', 'Monetary']
monetary_data.head()
```

	PartnerId	Monetary
0	4	1404.80
1	113	310114.50
2	124	2106.01
3	125	445.00
4	142	1193.42

Next steps: Generate code with monetary_data | View recommended plots | New interactive sheet

For this table, we aggregate the total sales value for each partner.

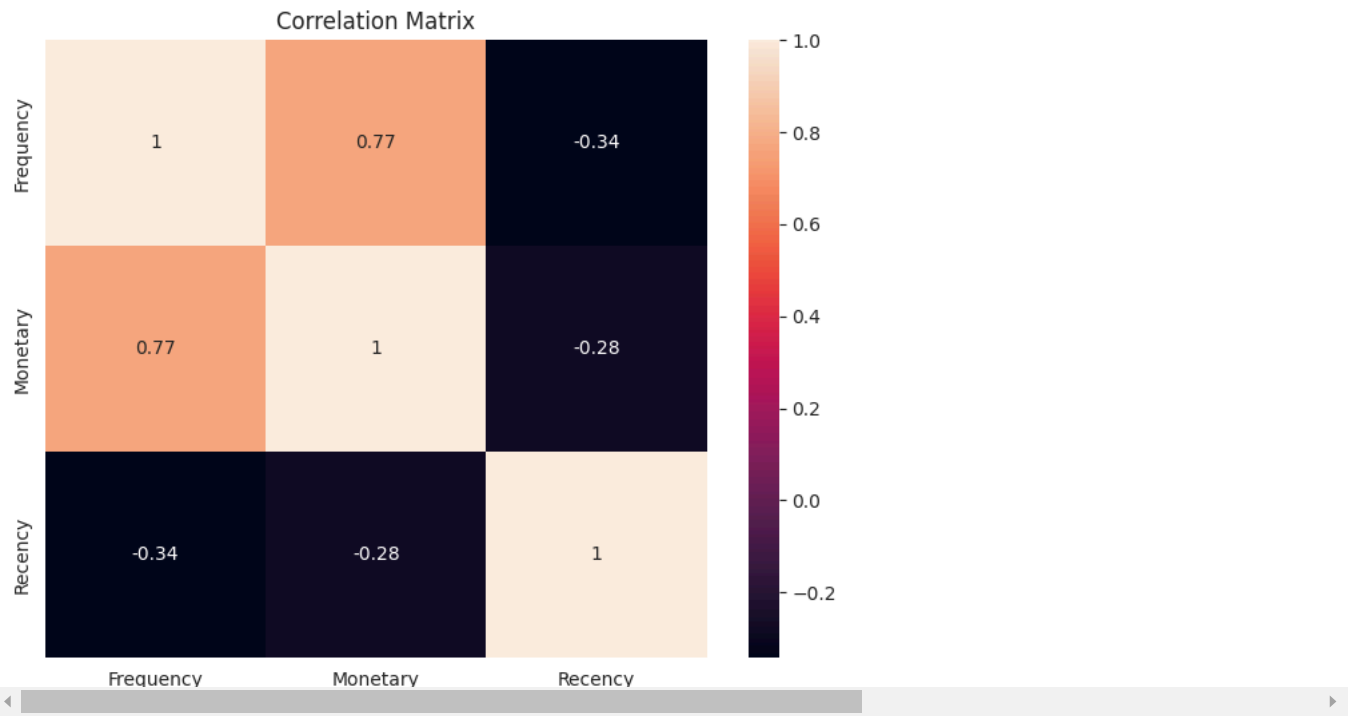
Summarization

```
rf_data = data_recency.merge(frequency_data, on='PartnerId')
rfm_data = rf_data.merge(monetary_data, on='PartnerId').drop(
    columns='LastPurchaseDate')
rfm_data.head()
```

	PartnerId	Recency	Frequency	Monetary
0	4	16	2	1404.80
1	113	10	45	310114.50
2	124	53	7	2106.01
3	125	284	1	445.00
4	142	74	3	1193.42

Next steps: Generate code with rfm_data | View recommended plots | New interactive sheet

```
numerical_cols = ['Frequency', 'Monetary', 'Recency']
correlation_matrix = rfm_data[numerical_cols].corr()

plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap=c_palette)

plt.title('Correlation Matrix')
plt.show()
```



This correlation matrix reveals the following:

- The strongest correlation exists between frequency and monetary value, and since the correlation is positive, an increase in one value is likely to result in an increase in the other.

- There is a weaker negative correlation between recency and both frequency, as well as recency and monetary value; as one increases, the other may decrease.

There is no causal relationship indicated among these features.

Using the following code snippet, we rank clients based on all three features and then calculate a scaled, normalized version of these ranking values.

```
rfm_data['R_rank'] = rfm_data['Recency'].rank(ascending=False)
rfm_data['F_rank'] = rfm_data['Frequency'].rank(ascending=True)
rfm_data['M_rank'] = rfm_data['Monetary'].rank(ascending=True)

rfm_data['R_rank_norm'] = (rfm_data['R_rank']/rfm_data['R_rank'].max())*100
rfm_data['F_rank_norm'] = (rfm_data['F_rank']/rfm_data['F_rank'].max())*100
rfm_data['M_rank_norm'] = (rfm_data['F_rank']/rfm_data['M_rank'].max())*100

rfm_data.drop(columns=['R_rank', 'F_rank', 'M_rank'], inplace=True)

rfm_data.head()
```

	PartnerId	Recency	Frequency	Monetary	R_rank_norm	F_rank_norm	M_rank_norm
0	4	16	2	1404.80	75.436409	29.900744	29.900744
1	113	10	45	310114.50	83.977556	95.471464	95.471464
2	124	53	7	2106.01	49.563591	59.429280	59.429280
3	125	284	1	445.00	9.538653	11.910670	11.910670
4	142	74	3	1193.42	44.139651	39.267990	39.267990

Next steps: Generate code with rfm_data View recommended plots New interactive sheet

RFM Value Calculation

The RFM value is calculated based on the normalized values of recency, frequency, and monetary value. Customers are segmented based on this score, which is rated on a 5-point scale.

The formula used to compute the FRM value is:

RFM Value=0.15×Recency Score+0.28×Frequency Score+0.57×Monetary Score

```
rfm_data['RFM_Score'] = 0.15 * rfm_data['R_rank_norm'] + 0.28 * rfm_data['F_rank_norm'] + 0.57 * rfm_data['M_rank_norm']
rfm_data['RFM_Score'] = rfm_data['RFM_Score'] * 0.05
rfm_data = rfm_data.round(2)
rfm_data[['PartnerId', 'RFM_Score']].head()
```

	PartnerId	RFM_Score
0	4	1.84
1	113	4.69
2	124	2.90
3	125	0.58
4	142	2.00

RFM Value Categories:

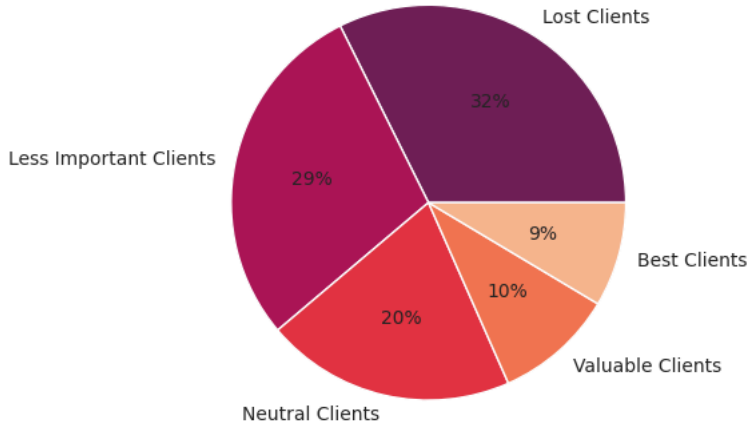
- RFM Value > 4.5: Best Clients
- 4.5 > RFM Value > 4: Valuable Clients
- 4 > RFM Value > 3: Neutral Clients
- 3 > RFM Value > 1.6: Less Important Clients
- 1.6 > RFM Value: Lost Clients

```
rfm_data["Customer_segment"] = np.where(rfm_data['RFM_Score'] >
                                        4.5, "Best Clients",
                                        (np.where(
                                          rfm_data['RFM_Score'] > 4,
                                          "Valuable Clients",
                                          (np.where(
    rfm_data['RFM_Score'] > 3,
                            "Neutral Clients",
                            np.where(rfm_data['RFM_Score'] > 1.6,
```

```
                        'Less Important Clients', 'Lost Clients'))))))
rfm_data[['PartnerId', 'RFM_Score', 'Customer_segment']].head()
```

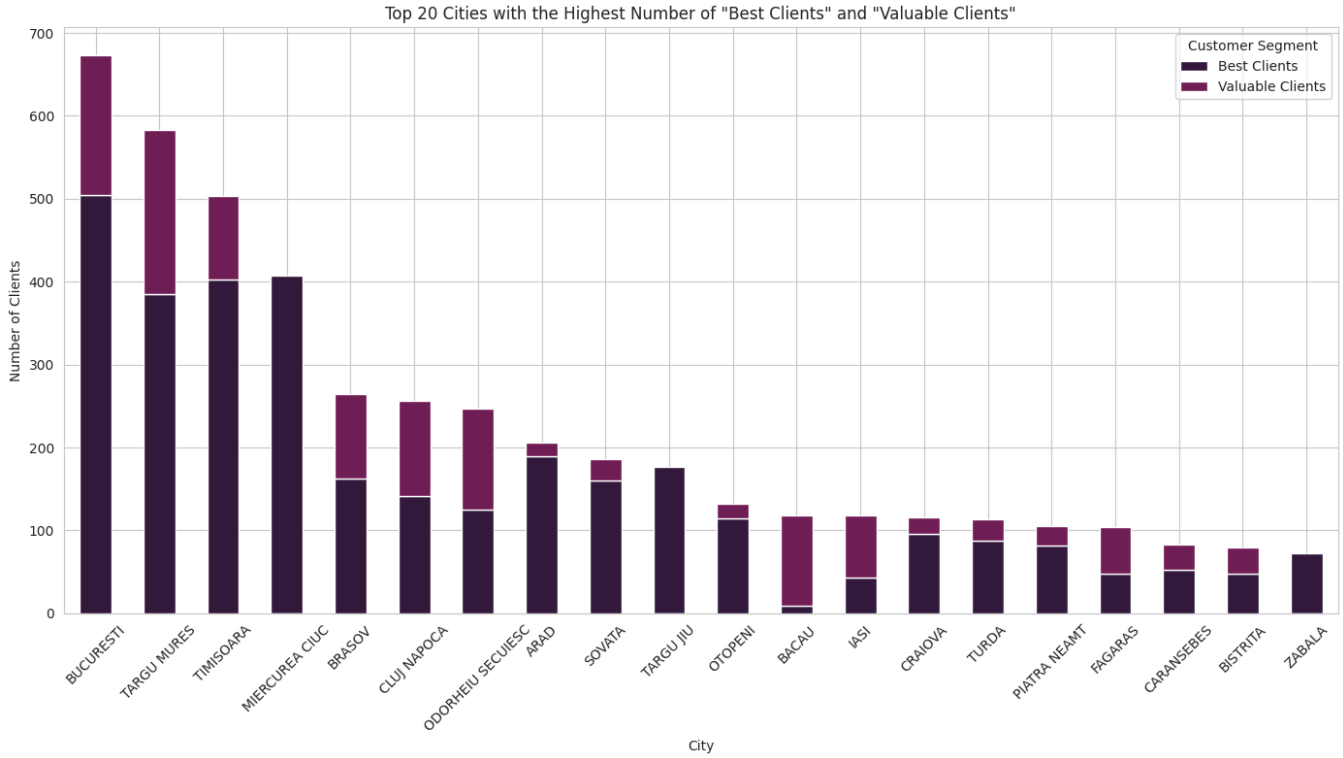
	PartnerId	RFM_Score	Customer_segment
0	4	1.84	Less Important Clients
1	113	4.69	Best Clients
2	124	2.90	Less Important Clients
3	125	0.58	Lost Clients
4	142	2.00	Less Important Clients

```
plt.pie(rfm_data.Customer_segment.value_counts(),
        labels=rfm_data.Customer_segment.value_counts().index,
        autopct='%.0f%%',
        colors=palette[1:6])
plt.show()
```

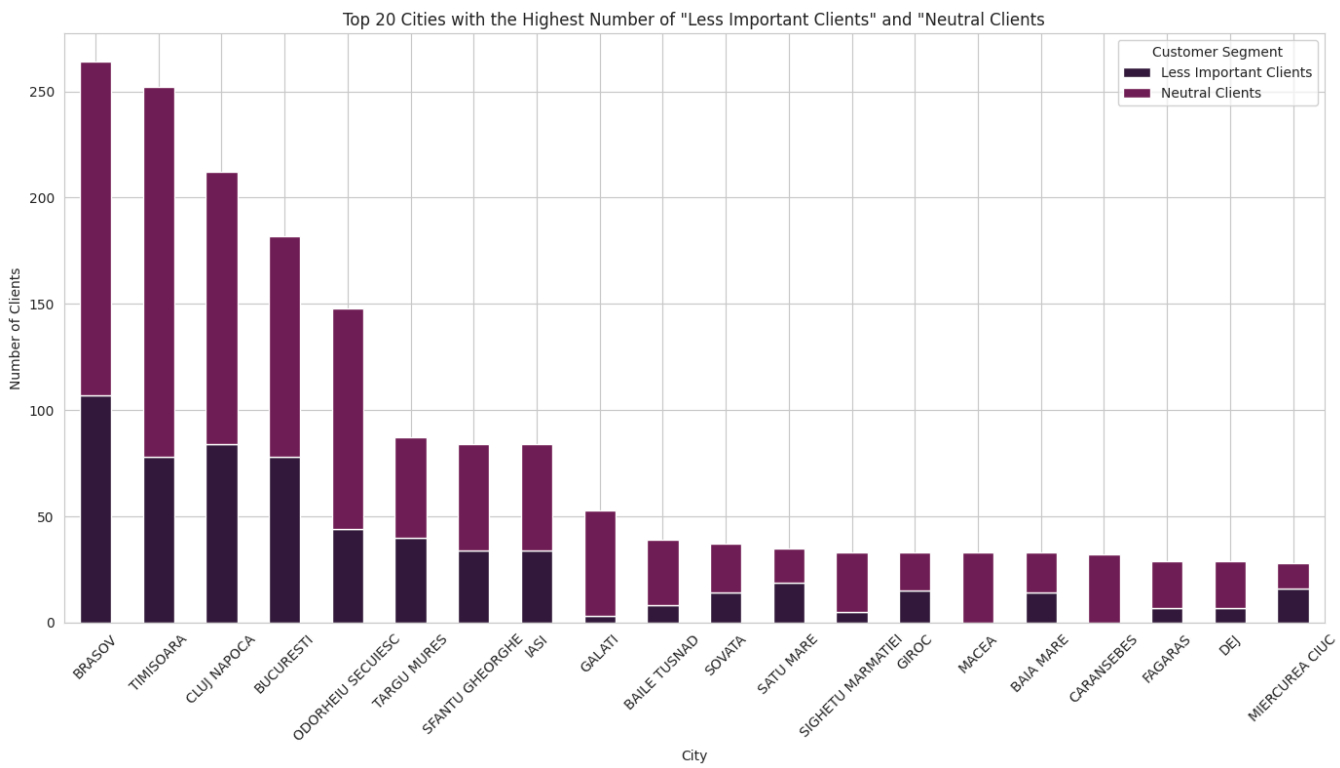


As we can see, based on the score, we have identified a significant number of lost clients.

```
label = data[['PartnerId', 'City']]
label = pd.merge(label, rfm_data, on = 'PartnerId')


filtered_data = label[label['Customer_segment'].isin(['Best Clients', 'Valuable Clients'])]
city_segment_counts = filtered_data.groupby(['City', 'Customer_segment']).size().unstack().fillna(0)
top_cities = city_segment_counts.sum(axis=1).sort_values(ascending=False).head(20).index
top_city_segment_counts = city_segment_counts.loc[top_cities]
top_city_segment_counts.plot(kind='bar', stacked=True, figsize=(14, 8))
plt.title('Top 20 Cities with the Highest Number of "Best Clients" and "Valuable Clients"')
plt.xlabel('City')
plt.ylabel('Number of Clients')
plt.legend(title='Customer Segment')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




```
filtered_data = label[label['Customer_segment'].isin(['Less Important Clients', 'Neutral Clients'])]
city_segment_counts = filtered_data.groupby(['City', 'Customer_segment']).size().unstack().fillna(0)
top_cities = city_segment_counts.sum(axis=1).sort_values(ascending=False).head(20).index
top_city_segment_counts = city_segment_counts.loc[top_cities]
top_city_segment_counts.plot(kind='bar', stacked=True, figsize=(14, 8))
plt.title('Top 20 Cities with the Highest Number of "Less Important Clients" and "Neutral Clients')
plt.xlabel('City')
plt.ylabel('Number of Clients')
plt.legend(title='Customer Segment')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



The next phase is to convert the data dimensions into a modelable format.

Feature Engineering

Transforming rfm_data

```
rfm_data.head()
```

	PartnerId	Recency	Frequency	Monetary	R_rank_norm	F_rank_norm	M_rank_norm	RFM_Score	Customer_segment
0	4	16	2	1404.80	75.44	29.90	29.90	1.84	Less Important Clients
1	113	10	45	310114.50	83.98	95.47	95.47	4.69	Best Clients
2	124	53	7	2106.01	49.56	59.43	59.43	2.90	Less Important Clients
3	125	284	1	445.00	9.54	11.91	11.91	0.58	Lost Clients
4	142	74	3	1193.42	44.14	39.27	39.27	2.00	Less Important Clients

Next steps: Generate code with rfm_data | View recommended plots | New interactive sheet

We will also create a DataFrame named labels to collect variables that we want to visualize interactively in the clustering visualizations.

```
labels = rfm_data[['PartnerId', 'RFM_Score', 'Customer_segment']]
labels.head()
```

	PartnerId	RFM_Score	Customer_segment
0	4	1.84	Less Important Clients
1	113	4.69	Best Clients
2	124	2.90	Less Important Clients
3	125	0.58	Lost Clients
4	142	2.00	Less Important Clients

Next steps: Generate code with labels | View recommended plots | New interactive sheet

```
rfm_data.dtypes
```

	0
PartnerId	int64
Recency	int64
Frequency	int64
Monetary	float64
R_rank_norm	float64
F_rank_norm	float64
M_rank_norm	float64
RFM_Score	float64
Customer_segment	object

dtype: object

Here, all values are numeric except for the customer_classification column, which needs to be transformed into dummy variables.

Customer_segment

```
encoded_data = pd.get_dummies(rfm_data['Customer_segment'])
rfm_data = pd.concat([rfm_data, encoded_data], axis=1)
rfm_data.drop('Customer_segment', axis=1, inplace=True)
rfm_data.head()
```

	PartnerId	Recency	Frequency	Monetary	R_rank_norm	F_rank_norm	M_rank_norm	RFM_Score	Best Clients	Less Important Clients	Lost Clients	Neutral Clients
0	4	16	2	1404.80	75.44	29.90	29.90	1.84	False	True	False	False
1	113	10	45	310114.50	83.98	95.47	95.47	4.69	True	False	False	False
2	124	53	7	2106.01	49.56	59.43	59.43	2.90	False	True	False	False
3	125	284	1	445.00	9.54	11.91	11.91	0.58	False	False	True	False

Next steps: Generate code with rfm_data | View recommended plots | New interactive sheet

```
rfm_data.dtypes
```

	0
PartnerId	int64
Recency	int64
Frequency	int64
Monetary	float64
R_rank_norm	float64
F_rank_norm	float64
M_rank_norm	float64
RFM_Score	float64
Best Clients	bool
Less Important Clients	bool
Lost Clients	bool
Neutral Clients	bool
Valuable Clients	bool

dtype: object

```
rfm_data.shape
```

```
(806, 13)
```

Data Transformation

```
data.head()
```

	DocumentId	PartnerId	DocumentDate	City	Sales	Quarter
343	779058	10592	2022-04-13	BRASOV	6.42	2022Q2
344	829740	12054	2022-10-27	GIROC	6.83	2022Q4
345	829748	12593	2022-10-27	CORUNCA	6.83	2022Q4
346	781716	11567	2022-04-26	BUCURESTI	10.90	2022Q2
347	803442	162	2022-07-14	SOVATA	11.76	2022Q3

Next steps: Generate code with `data` View recommended plots New interactive sheet

To merge the two DataFrames, we need to group the data DataFrame by PartnerId and aggregate the other features, ensuring that these dimensions are numeric.

DocumentId

We can remove the DocumentId column.

```
data = data[['PartnerId', 'DocumentDate', 'City', 'Sales', 'Quarter']]
data.head()
```

	PartnerId	DocumentDate	City	Sales	Quarter
343	10592	2022-04-13	BRASOV	6.42	2022Q2
344	12054	2022-10-27	GIROC	6.83	2022Q4
345	12593	2022-10-27	CORUNCA	6.83	2022Q4
346	11567	2022-04-26	BUCURESTI	10.90	2022Q2
347	162	2022-07-14	SOVATA	11.76	2022Q3

Next steps: Generate code with `data` View recommended plots New interactive sheet

For different columns, we will use different aggregation methods, so we will handle them separately and then consolidate them.

City

For the cities, we will take one (the first) value:

```
cities = data[['PartnerId', 'City']]
cities = cities.groupby('PartnerId')['City'].first().reset_index()
cities.head()
```

	PartnerId	City
0	4	BRASOV
1	113	DEJ
2	124	BRASOV
3	125	BRASOV
4	142	SIGHISOARA

Next steps: Generate code with cities | View recommended plots | New interactive sheet

```
labels = pd.merge(labels, cities, on='PartnerId')
labels.head()
```

	PartnerId	RFM_Score	Customer_segment	City
0	4	1.84	Less Important Clients	BRASOV
1	113	4.69	Best Clients	DEJ
2	124	2.90	Less Important Clients	BRASOV
3	125	0.58	Lost Clients	BRASOV
4	142	2.00	Less Important Clients	SIGHISOARA

Next steps: Generate code with labels | View recommended plots | New interactive sheet

We can also transform the cities using dummy variables.

```
encoded_data = pd.get_dummies(cities['City'])
cities = pd.concat([cities, encoded_data], axis=1)
cities.drop('City', axis=1, inplace=True)
cities.head()
```

	PartnerId	ADEA	AGHIRESU FABRICI	AIUD	ALBA IULIA	ALBESTI	AMARA	ANDRID	ARAD	ARMENIS	...	VLADIMIRESCU	VLAHA	VOLUNTARI	VULCAN	ZABAL
0	4	False	False	False	False	False	False	False	False	False	...	False	False	False	False	Fals
1	113	False	False	False	False	False	False	False	False	False	...	False	False	False	False	Fals
2	124	False	False	False	False	False	False	False	False	False	...	False	False	False	False	Fals
3	125	False	False	False	False	False	False	False	False	False	...	False	False	False	False	Fals
4	142	False	False	False	False	False	False	False	False	False	...	False	False	False	False	Fals

5 rows × 232 columns

```
cities.dtypes
```

	0
PartnerId	int64
ADEA	bool
AGHIRESU FABRICI	bool
AIUD	bool
ALBA IULIA	bool
...	...
ZALAU	bool
ZAM	bool
ZARNESTI	bool
ZETEA	bool
ZOLTAN	bool

232 rows × 1 columns

dtype: object

```
cities.shape
```

```
(806, 232)
```

DocumentDate

For the document dates, we will create three new variables: the date of the customer's first purchase, the date of the last purchase, and the number of days between these two dates.

```
docs = data[['PartnerId', 'DocumentDate']]
docs = docs.groupby('PartnerId').agg(
    EarliestDocument=('DocumentDate', 'min'),
    LatestDocument=('DocumentDate', 'max'),
).reset_index()
docs['DaysBetween'] = (docs['LatestDocument'] - docs['EarliestDocument']).dt.days
docs.head()
```

	PartnerId	EarliestDocument	LatestDocument	DaysBetween
0	4	2022-08-24	2022-12-14	112
1	113	2022-01-06	2022-12-20	348
2	124	2022-01-14	2022-11-07	297
3	125	2022-03-21	2022-03-21	0
4	142	2022-04-26	2022-10-17	174

Next steps: Generate code with docs View recommended plots New interactive sheet

```
labels = pd.merge(labels, docs, on='PartnerId')
labels.head()
```

	PartnerId	RFM_Score	Customer_segment	City	EarliestDocument	LatestDocument	DaysBetween
0	4	1.84	Less Important Clients	BRASOV	2022-08-24	2022-12-14	112
1	113	4.69	Best Clients	DEJ	2022-01-06	2022-12-20	348
2	124	2.90	Less Important Clients	BRASOV	2022-01-14	2022-11-07	297
3	125	0.58	Lost Clients	BRASOV	2022-03-21	2022-03-21	0
4	142	2.00	Less Important Clients	SIGHISOARA	2022-04-26	2022-10-17	174

Next steps: Generate code with labels View recommended plots New interactive sheet

```
docs.dtypes
```

	0
PartnerId	int64
EarliestDocument	datetime64[ns]
LatestDocument	datetime64[ns]
DaysBetween	int64

dtype: object

Here, we need to convert these dates to a numeric format by calculating the number of days that have passed in the year up to each given date.

```
reference_date = pd.Timestamp('2022-01-01')
docs['EarliestDocument'] = (docs['EarliestDocument'] - reference_date).dt.days
docs['LatestDocument'] = (docs['LatestDocument'] - reference_date).dt.days
docs.head()
```

	PartnerId	EarliestDocument	LatestDocument	DaysBetween
0	4	235	347	112
1	113	5	353	348
2	124	13	310	297
3	125	79	79	0
4	142	115	289	174

Next steps: Generate code with docs View recommended plots New interactive sheet

```
docs.dtypes
```

	0
PartnerId	int64
EarliestDocument	int64
LatestDocument	int64
DaysBetween	int64

dtype: object

```
docs.shape
```

```
(806, 4)
```

Next, we will handle the remaining dimensions.

Sales

For the purchase values, we will create two new variables: the average and the total sum of the values.

```
sales = data[['PartnerId', 'Sales']]
sales = sales.groupby('PartnerId').agg(
    SalesMean=('Sales', 'mean'),
    SalesSum=('Sales', 'sum'),
).reset_index()
sales.head()
```

	PartnerId	SalesMean	SalesSum
0	4	702.400000	1404.80
1	113	6891.433333	310114.50
2	124	300.858571	2106.01
3	125	445.000000	445.00
4	142	397.806667	1193.42

Next steps: Generate code with sales View recommended plots New interactive sheet

```
labels = pd.merge(labels, sales, on='PartnerId')
labels.head()
```

	PartnerId	RFM_Score	Customer_segment	City	EarliestDocument	LatestDocument	DaysBetween	SalesMean	SalesSum
0	4	1.84	Less Important Clients	BRASOV	2022-08-24	2022-12-14	112	702.400000	1404.80
1	113	4.69	Best Clients	DEJ	2022-01-06	2022-12-20	348	6891.433333	310114.50
2	124	2.90	Less Important Clients	BRASOV	2022-01-14	2022-11-07	297	300.858571	2106.01
3	125	0.58	Lost Clients	BRASOV	2022-03-21	2022-03-21	0	445.000000	445.00

Next steps: Generate code with `labels` View recommended plots New interactive sheet

```
sales.dtypes
```

	0
PartnerId	int64
SalesMean	float64
SalesSum	float64

dtype: object

```
sales.shape
```

```
(806, 3)
```

Quarter

Among the quarters in which the customer made purchases, we will take the most frequent value.

```
quarters = data[['PartnerId', 'Quarter']]
quarters = quarters.groupby('PartnerId')['Quarter'].agg(lambda x: x.mode()[0]).reset_index()
quarters.head()
```

	PartnerId	Quarter
0	4	2022Q3
1	113	2022Q2
2	124	2022Q1
3	125	2022Q1
4	142	2022Q2

Next steps: Generate code with `quarters` View recommended plots New interactive sheet

```
labels = pd.merge(labels, quarters, on='PartnerId')
labels.head()
```

	PartnerId	RFM_Score	Customer_segment	City	EarliestDocument	LatestDocument	DaysBetween	SalesMean	SalesSum	Quarte
0	4	1.84	Less Important Clients	BRASOV	2022-08-24	2022-12-14	112	702.400000	1404.80	2022Q
1	113	4.69	Best Clients	DEJ	2022-01-06	2022-12-20	348	6891.433333	310114.50	2022Q
2	124	2.90	Less Important Clients	BRASOV	2022-01-14	2022-11-07	297	300.858571	2106.01	2022Q
3	125	0.58	Lost Clients	BRASOV	2022-03-21	2022-03-21	0	445.000000	445.00	2022Q

Next steps: Generate code with `labels` View recommended plots New interactive sheet

```
quarters.dtypes
```

	0
PartnerId	int64
Quarter	period[Q-DEC]

dtype: object

```
quarters['Quarter'] = quarters['Quarter'].dt.quarter
quarters.head()
```

	PartnerId	Quarter
0	4	3
1	113	2
2	124	1
3	125	1
4	142	2

Next steps: Generate code with `quarters` View recommended plots New interactive sheet

```
quarters.dtypes
```

	0
PartnerId	int64
Quarter	int64

dtype: object

```
quarters.shape
```

```
(806, 2)
```

Now, our task is to merge all the tables into a single comprehensive table.

```
rfm_data.set_index('PartnerId', inplace=True)
docs.set_index('PartnerId', inplace=True)
cities.set_index('PartnerId', inplace=True)
sales.set_index('PartnerId', inplace=True)
quarters.set_index('PartnerId', inplace=True)

dfs = [rfm_data, docs, cities, sales, quarters]

merged_data = pd.concat(dfs, axis=1)
merged_data.reset_index(inplace=True)
merged_data.head()
```

	PartnerId	Recency	Frequency	Monetary	R_rank_norm	F_rank_norm	M_rank_norm	RFM_Score	Best Clients	Less Important Clients	...	VULCAN	ZABALA
0	4	16	2	1404.80	75.44	29.90	29.90	1.84	False	True	...	False	False
1	113	10	45	310114.50	83.98	95.47	95.47	4.69	True	False	...	False	False
2	124	53	7	2106.01	49.56	59.43	59.43	2.90	False	True	...	False	False
3	125	284	1	445.00	9.54	11.91	11.91	0.58	False	False	...	False	False
4	142	74	3	1193.42	44.14	39.27	39.27	2.00	False	True	...	False	False

5 rows × 250 columns

```
merged_data.shape
```

```
(806, 250)
```

The dataset required for modeling is now ready, consisting of 806 customer records and 250 features.

Modeling

We will test several clustering algorithms for the modeling, including:

- K-Means Clustering
- Hierarchical Clustering
- DBSCAN Clustering
- Spectral Clustering
- Gaussian Mixture Model

To determine the optimal number of clusters, we will use two methods:

- Silhouette Score
- Elbow Method

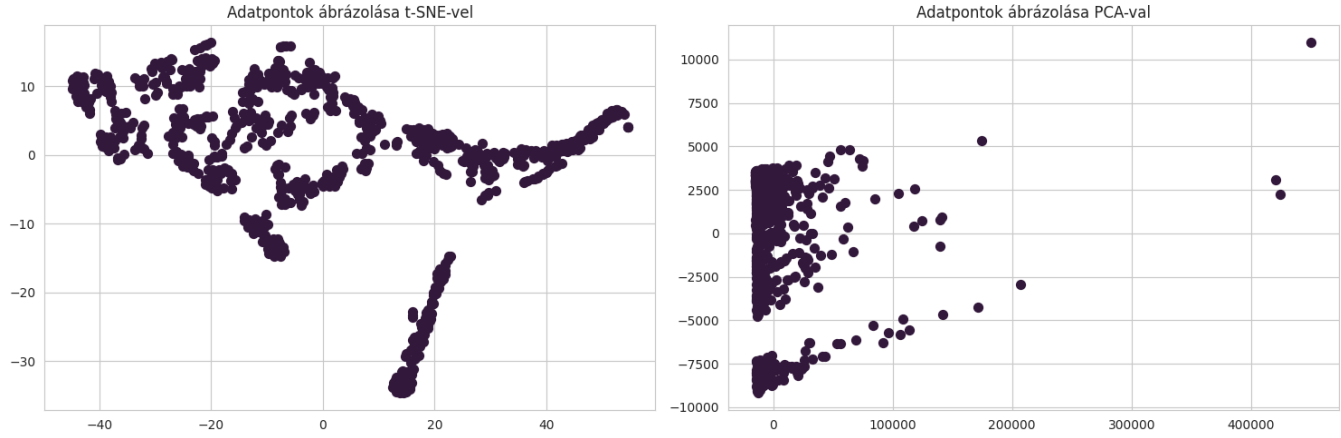
Additionally, we will employ two dimensionality reduction algorithms to visualize the results by projecting the data points into two dimensions:

- t-SNE
- PCA

Plotting Data Points in a Two-Dimensional Space

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
pca.fit(merged_data)
data_pca = pca.transform(merged_data)


from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, random_state=42)
data_tsne = tsne.fit_transform(merged_data)


fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))

axes[0].scatter(data_tsne[:, 0], data_tsne[:, 1], s=50, cmap=c_palette)
axes[0].set_title('Adatpontok ábrázolása t-SNE-vel')

axes[1].scatter(data_pca[:, 0], data_pca[:, 1], s=50, cmap=c_palette)
axes[1].set_title('Adatpontok ábrázolása PCA-val')

plt.tight_layout()
plt.show()
```

```
<ipython-input-116-a71d089aafd5>:3: UserWarning: No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  axes[0].scatter(data_tsne[:, 0], data_tsne[:, 1], s=50, cmap=c_palette)
<ipython-input-116-a71d089aafd5>:6: UserWarning: No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  axes[1].scatter(data_pca[:, 0], data_pca[:, 1], s=50, cmap=c_palette)
```



After creating the clusters, we will determine which clustering method is easier to interpret and visualize.

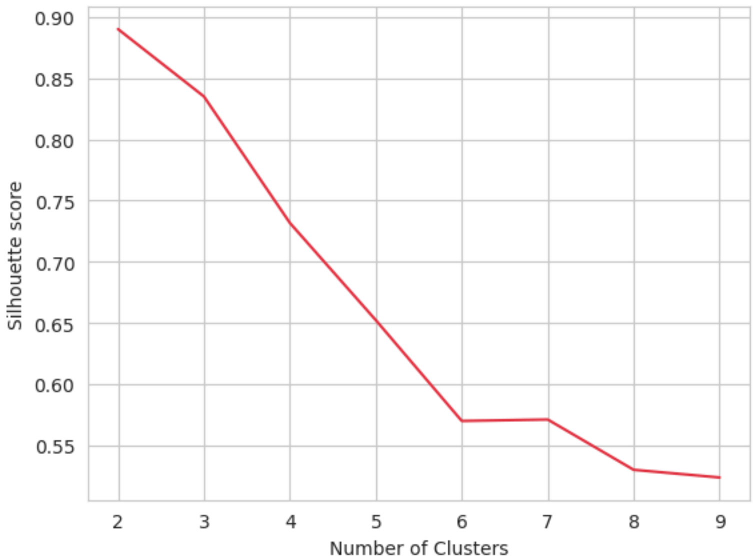
K-Means Clustering

First, we need to determine the optimal number of clusters. We will establish that there should be at least 3 customer groups, but no more than 10, as managing more than this would be challenging from a marketing strategy perspective, especially since only two people in the marketing department are handling this. Therefore, we will choose a number of clusters between 3 and 10.

Silhouette score

```
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans

silhouette_scores = []
for n_clusters in range(2, 10):
    kmeans = KMeans(n_clusters=n_clusters, init = 'k-means++', random_state=42)
    cluster_labels = kmeans.fit_predict(merged_data)
    silhouette_scores.append(silhouette_score(merged_data, cluster_labels))
plt.plot(range(2, 10), silhouette_scores, color=palette[3])
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette score')
plt.show()
```

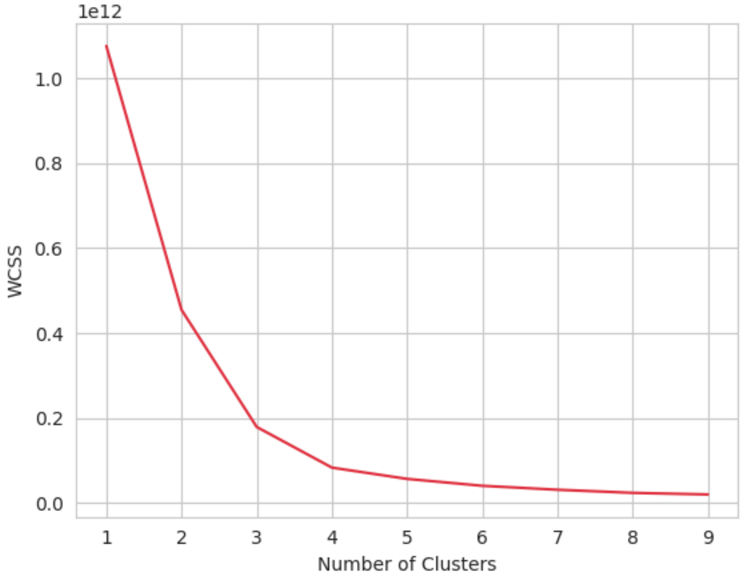
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
```



Elbow method

```
wcss = []
for i in range(1, 10):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(merged_data)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 10), wcss, color=palette[3])
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
```



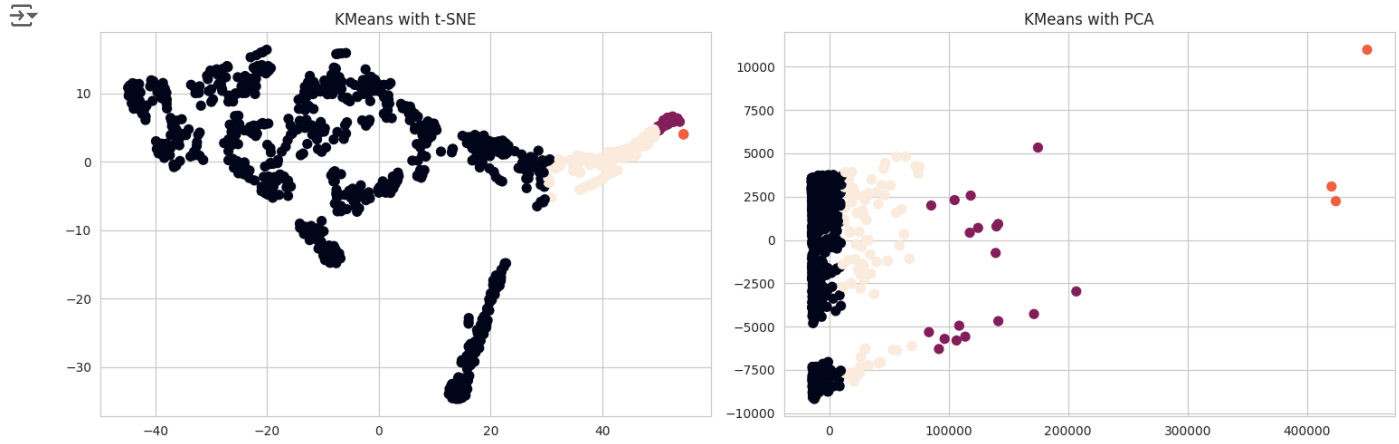
Based on the two charts, we can observe that overall, dividing into 4 clusters would provide the most accurate customer segments for the company.

```
kmeans = KMeans(n_clusters = 4, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(merged_data)
```

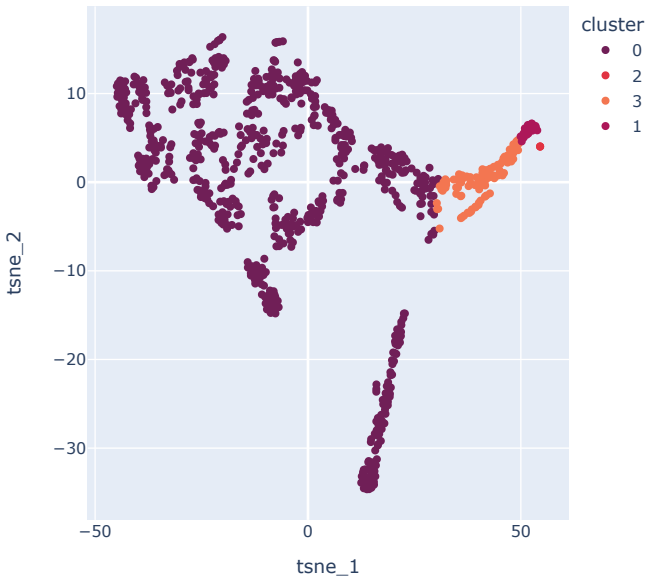
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
```

We will visualize the clusters using the Matplotlib library, and also create interactive plots with Plotly to enhance the interactivity.

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))

axes[0].scatter(data_tsne[:, 0], data_tsne[:, 1], c=y_kmeans, s=50, cmap=c_palette)
axes[0].set_title('KMeans with t-SNE')

axes[1].scatter(data_pca[:, 0], data_pca[:, 1], c=y_kmeans, s=50, cmap=c_palette)
axes[1].set_title('KMeans with PCA')

plt.tight_layout()
plt.show()
```




```
hover_labels = labels.apply(lambda row: ', '.join([f'{col}: {val}' for col, val in row.items()]), axis=1).tolist()


import plotly.express as px
from matplotlib.colors import rgb2hex

temp = pd.DataFrame({
    'tsne_1': data_tsne[:, 0],
    'tsne_2': data_tsne[:, 1],
    'cluster': y_kmeans,
    'hover_label': hover_labels
})

temp['cluster'] = temp['cluster'].astype(str)

hex_palette = [rgb2hex(color) for color in palette]

custom_cmap = {
    '0': hex_palette[1],
    '1': hex_palette[2],
    '2': hex_palette[3],
    '3': hex_palette[4]
}


fig = px.scatter(
    temp,
    x='tsne_1',
    y='tsne_2',
    color='cluster',
    hover_name='hover_label',
    color_discrete_map=custom_cmap
)

fig.update_traces(hovertemplate='%{hovertext}')
fig.show()
```




```
temp2 = pd.DataFrame({
    'pca_1': data_pca[:, 0],
    'pca_2': data_pca[:, 1],
    'cluster': y_kmeans,
    'hover_label': hover_labels
})

temp2['cluster'] = temp2['cluster'].astype(str)

hex_palette = [rgb2hex(color) for color in palette]

custom_cmap = {
    '0': hex_palette[1],
    '1': hex_palette[2],
    '2': hex_palette[3],
    '3': hex_palette[4]
}


fig = px.scatter(
    temp2,
    x='pca_1',
    y='pca_2',
    color='cluster',
    hover_name='hover_label',
    color_discrete_map=custom_cmap
)

fig.update_traces(hovertemplate='%{hovertext}')
fig.show()
```


```
y_kmeansd = pd.DataFrame(y_kmeans)
y_kmeansd.columns = ['Cluster']
groups = pd.concat([labels, y_kmeansd], axis=1)
```

Using the table and the charts below, we can browse through the members of the different customer groups.

```
groups[groups['Cluster'] == 0]
```

	PartnerId	RFM_Score	Customer_segment	City	EarliestDocument	LatestDocument	DaysBetween	SalesMean	SalesSum	Quar
0	4	1.84	Less Important Clients	BRASOV	2022-08-24	2022-12-14	112	702.400000	1404.80	2022
2	124	2.90	Less Important Clients	BRASOV	2022-01-14	2022-11-07	297	300.858571	2106.01	2022
3	125	0.58	Lost Clients	BRASOV	2022-03-21	2022-03-21	0	445.000000	445.00	2022
4	142	2.00	Less Important Clients	SIGHISOARA	2022-04-26	2022-10-17	174	397.806667	1193.42	2022
5	143	3.97	Neutral Clients	BRASOV	2022-01-04	2022-12-29	359	665.690714	9319.67	2022
...	...	...	...	...	...	...	...	...	...	
801	12801	1.16	Lost Clients	MOSNITA NOUA	2022-12-21	2022-12-21	0	76.010000	76.01	2022
802	12802	1.16	Lost Clients	BECLEAN	2022-12-21	2022-12-21	0	810.510000	810.51	2022
803	12807	1.94	Less Important Clients	BAIA DE FIER	2022-12-22	2022-12-22	0	1932.650000	3865.30	2022

```
groups[groups['Cluster'] == 1]
```

	PartnerId	RFM_Score	Customer_segment	City	EarliestDocument	LatestDocument	DaysBetween	SalesMean	SalesSum	Qua
9	162	4.92	Best Clients	SOVATA	2022-01-04	2022-12-27	357	1216.920093	131427.37	20
10	193	4.58	Best Clients	TIMISOARA	2022-02-01	2022-12-19	321	1976.872105	75121.14	20
17	316	4.96	Best Clients	TIMISOARA	2022-01-03	2022-12-28	359	551.989097	85558.31	20
20	342	4.77	Best Clients	MIERCUREA CIUC	2022-01-03	2022-12-19	350	1296.181429	90732.70	20
37	524	4.90	Best Clients	PIATRA NEAMT	2022-01-07	2022-12-27	354	1345.041098	110293.37	20
40	554	4.98	Best Clients	CRAIOVA	2022-01-03	2022-12-30	361	1326.078220	156477.23	20
48	650	4.49	Valuable Clients	PETROSANI	2022-01-11	2022-12-22	345	2616.808333	78504.25	20
93	1106	4.90	Best Clients	SUCEAVA	2022-01-03	2022-12-28	359	1227.491972	87151.93	20
112	1387	4.80	Best Clients	CLUJ NAPOCA	2022-01-05	2022-12-22	351	1214.727719	69239.48	20
133	4518	4.82	Best Clients	TARGU JIU	2022-01-03	2022-12-19	350	1007.468704	108806.62	20
180	6027	4.84	Best Clients	TIMISOARA	2022-01-06	2022-12-23	351	1655.506515	109263.43	20
184	6130	4.85	Best Clients	CLUJ Unknown City	2022-01-25	2022-12-27	336	1838.718000	110323.08	20
188	6246	4.88	Best Clients	TARGU JIU	2022-01-05	2022-12-22	351	954.178265	93509.47	20
194	6326	4.52	Best Clients	TIMISOARA	2022-01-20	2022-12-27	341	3280.757667	98422.73	20
271	8362	4.83	Best Clients	CARANSEBES	2022-01-10	2022-12-28	352	1775.577547	94105.61	20

```
groups[groups['Cluster'] == 2]
```

	PartnerId	RFM_Score	Customer_segment	City	EarliestDocument	LatestDocument	DaysBetween	SalesMean	SalesSum	Quart
1	113	4.69	Best Clients	DEJ	2022-01-06	2022-12-20	348	6891.433333	310114.50	2022
92	1098	4.97	Best Clients	BUCURESTI	2022-01-03	2022-12-28	359	776.868359	307639.87	2022

```
groups[groups['Cluster'] == 3]
```

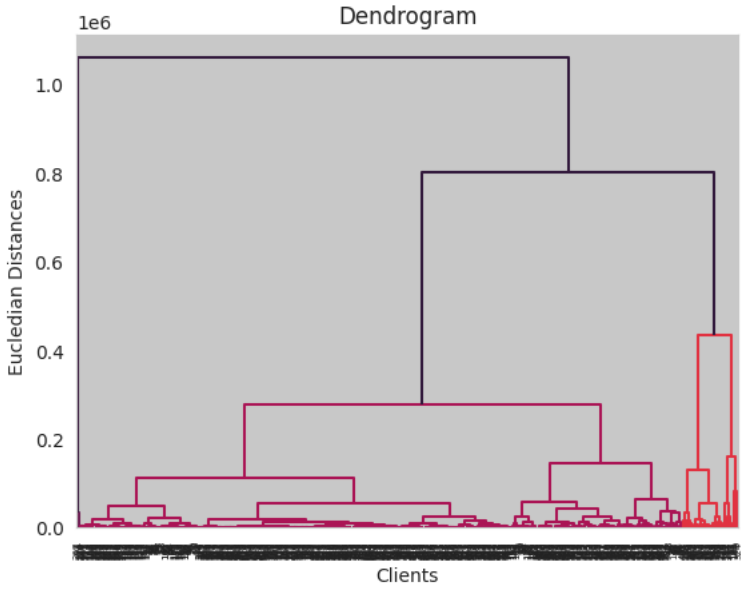
	PartnerId	RFM_Score	Customer_segment	City	EarliestDocument	LatestDocument	DaysBetween	SalesMean	SalesSum	Quart
8	161	4.48	Valuable Clients	BRASOV	2022-01-14	2022-12-08	328	675.157027	24980.81	2022
14	281	3.26	Neutral Clients	BUCURESTI	2022-03-14	2022-11-28	259	2717.640000	24458.76	2022
26	442	4.55	Best Clients	TURDA	2022-01-03	2022-12-19	350	775.891143	27156.19	2022
35	505	4.38	Valuable Clients	FOCSANI	2022-02-21	2022-12-19	301	934.858966	27110.91	2022
41	564	4.55	Best Clients	TARGU MURES	2022-01-10	2022-12-19	343	809.785714	28342.50	2022
...	...	...	...	...	...	...	...	...	...	
579	12083	4.86	Best Clients	MIERCUREA CIUC	2022-01-03	2022-12-27	358	640.333088	43542.65	2022
619	12215	4.67	Best Clients	BEIUS	2022-02-28	2022-12-27	302	1359.972973	50319.00	2022
630	12247	4.29	Valuable Clients	BRASOV	2022-03-21	2022-12-20	274	832.549167	19981.18	2022
644	12279	4.56	Best Clients	BRASOV	2022-04-04	2022-12-27	267	767.029677	23777.92	2022

Hierarchical Clustering

For this model, we also need to determine the optimal number of clusters, this time using a dendrogram. We still aim to keep the number of clusters between 3 and 10.

Dendogram

```
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(merged_data, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Clients')
plt.ylabel('Eucledian Distances')
plt.show()
```

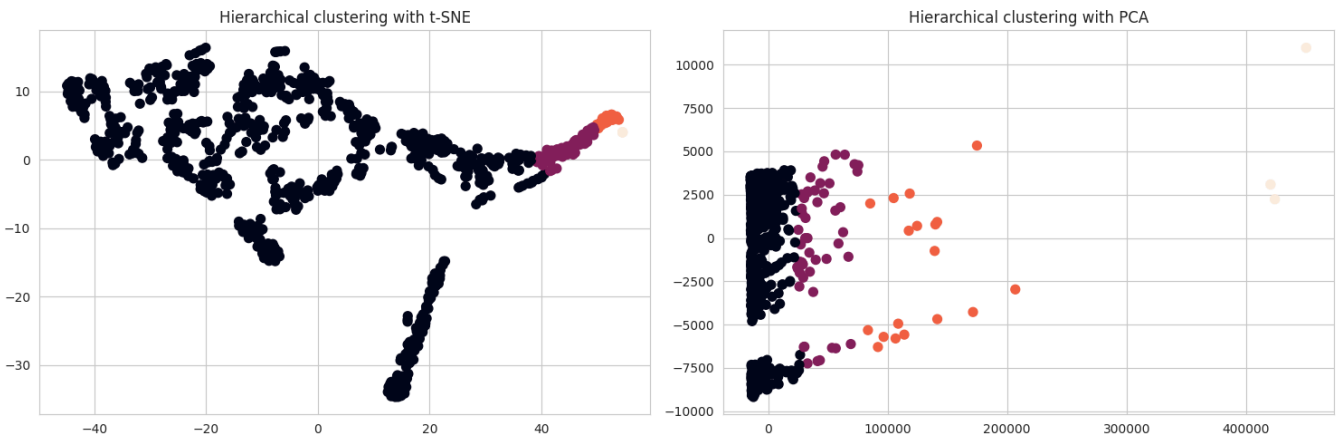


For this model, we can also choose 4 clusters.

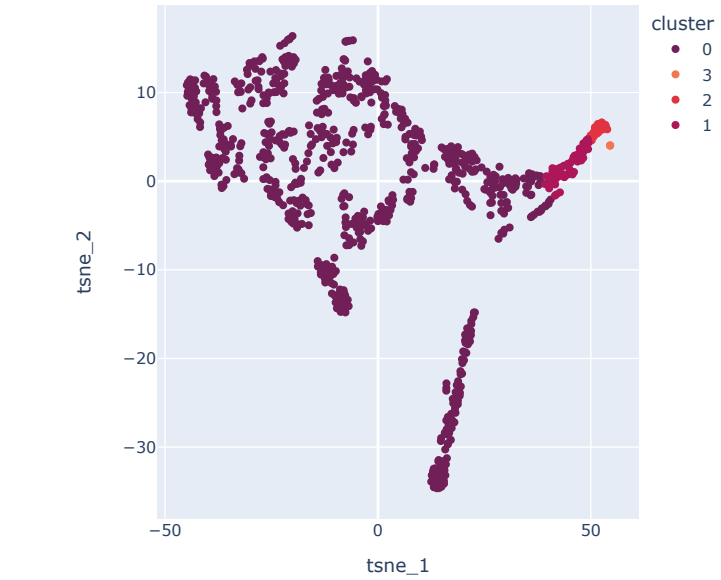
```
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 4, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(merged_data)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_agglomerative.py:1006: FutureWarning:

    Attribute `affinity` was deprecated in version 1.2 and will be removed in 1.4. Use `metric` instead
```

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))

axes[0].scatter(data_tsne[:, 0], data_tsne[:, 1], c=y_hc, s=50, cmap=c_palette)
axes[0].set_title('Hierarchical clustering with t-SNE')

axes[1].scatter(data_pca[:, 0], data_pca[:, 1], c=y_hc, s=50, cmap=c_palette)
axes[1].set_title('Hierarchical clustering with PCA')

plt.tight_layout()
plt.show()
```



```
temp = pd.DataFrame({
    'tsne_1': data_tsne[:, 0],
    'tsne_2': data_tsne[:, 1],
    'cluster': y_hc,
    'hover_label': hover_labels
})

temp['cluster'] = temp['cluster'].astype(str)

hex_palette = [rgb2hex(color) for color in palette]

custom_cmap = {
    '0': hex_palette[1],
    '1': hex_palette[2],
    '2': hex_palette[3],
    '3': hex_palette[4]
}


fig = px.scatter(
    temp,
    x='tsne_1',
    y='tsne_2',
    color='cluster',
    hover_name='hover_label',
    color_discrete_map=custom_cmap
)

fig.update_traces(hovertemplate='%{hovertext}')
fig.show()
```




```
temp2 = pd.DataFrame({
    'pca_1': data_pca[:, 0],
    'pca_2': data_pca[:, 1],
    'cluster': y_hc,
    'hover_label': hover_labels
})

temp2['cluster'] = temp2['cluster'].astype(str)

hex_palette = [rgb2hex(color) for color in palette]

custom_cmap = {
    '0': hex_palette[1],
    '1': hex_palette[2],
    '2': hex_palette[3],
    '3': hex_palette[4]
}
```

```
fig = px.scatter(
    temp2,
    x='pca_1',
    y='pca_2',
    color='cluster',
    hover_name='hover_label',
    color_discrete_map=custom_cmap
)

fig.update_traces(hovertemplate='%{hovertext}')
fig.show()
```


```
y_hcd = pd.DataFrame(y_hc)
y_hcd.columns = ['Cluster']
groups2 = pd.concat([labels, y_hcd], axis=1)
```

```
groups2[groups2['Cluster'] == 0]
```

	PartnerId	RFM_Score	Customer_segment	City	EarliestDocument	LatestDocument	DaysBetween	SalesMean	SalesSum	Quar
0	4	1.84	Less Important Clients	BRASOV	2022-08-24	2022-12-14	112	702.400000	1404.80	2022
2	124	2.90	Less Important Clients	BRASOV	2022-01-14	2022-11-07	297	300.858571	2106.01	2022
3	125	0.58	Lost Clients	BRASOV	2022-03-21	2022-03-21	0	445.000000	445.00	2022
4	142	2.00	Less Important Clients	SIGHISOARA	2022-04-26	2022-10-17	174	397.806667	1193.42	2022
5	143	3.97	Neutral Clients	BRASOV	2022-01-04	2022-12-29	359	665.690714	9319.67	2022
...	...	...	...	...	...	...	...	...	...	
801	12801	1.16	Lost Clients	MOSNITA NOUA	2022-12-21	2022-12-21	0	76.010000	76.01	2022
802	12802	1.16	Lost Clients	BECLEAN	2022-12-21	2022-12-21	0	810.510000	810.51	2022
803	12807	1.94	Less Important Clients	BAIA DE FIER	2022-12-22	2022-12-22	0	1932.650000	3865.30	2022

```
groups2[groups2['Cluster'] == 1]
```

	PartnerId	RFM_Score	Customer_segment	City	EarliestDocument	LatestDocument	DaysBetween	SalesMean	SalesSum
52	689	4.52	Best Clients	TIMISOARA	2022-01-06	2022-12-21	349	1269.998065	39369.94
53	690	4.80	Best Clients	SOVATA	2022-01-10	2022-12-27	351	783.558077	40745.02
63	780	4.74	Best Clients	OTOPENI	2022-01-05	2022-12-13	342	440.040263	33443.06
74	954	4.79	Best Clients	ARAD	2022-04-14	2022-12-28	258	1257.218936	59089.29
83	1035	4.73	Best Clients	OTOPENI	2022-01-25	2022-12-28	337	1286.601026	50177.44
96	1141	4.61	Best Clients	GHIRODA	2022-01-07	2022-12-14	341	1090.145455	47966.40
123	1804	4.50	Valuable Clients	TIMISOARA	2022-01-10	2022-12-23	347	1042.410000	31272.30
124	1809	4.81	Best Clients	BRASOV	2022-01-04	2022-12-23	353	555.354035	31655.18
138	4775	4.59	Best Clients	MOSNITA NOUA	2022-01-21	2022-12-14	327	922.476000	36899.04
152	5390	4.60	Best Clients	TARGU MURES	2022-01-10	2022-12-19	343	736.074615	28706.91
168	5820	4.46	Valuable Clients	BUCURESTI	2022-01-17	2022-12-19	336	999.203226	30975.30
179	6015	4.55	Best Clients	TARGU MURES	2022-01-10	2022-12-19	343	997.912857	34926.95
181	6050	4.83	Best Clients	ODORHEIU SECUIESC	2022-01-03	2022-12-19	350	477.348843	57759.21
186	6173	4.60	Best Clients	TIMISOARA	2022-01-10	2022-12-21	345	826.653143	28932.86
198	6387	3.95	Neutral Clients	BICAZ	2022-03-28	2022-11-04	221	2133.212857	44797.47
199	6437	3.91	Neutral Clients	TIMISOARA	2022-01-13	2022-11-18	309	1466.470000	27862.93
204	6557	4.74	Best Clients	DUMBRAVITA	2022-01-04	2022-12-19	349	465.508814	27465.02
206	6573	4.43	Valuable Clients	JIBOU	2022-01-21	2022-12-29	342	1535.943200	38398.58
207	6588	4.07	Valuable Clients	CRAIOVA	2022-01-06	2022-12-12	340	1540.934500	30818.69
211	6727	4.64	Best Clients	TARGU SECUIESC	2022-01-12	2022-12-22	344	816.590270	30213.84
216	6805	4.52	Best Clients	TARGU MURES	2022-01-10	2022-12-19	343	857.024412	29138.83
227	7021	4.52	Best Clients	BECLEAN	2022-01-04	2022-12-20	350	1621.486875	51887.58
231	7128	4.66	Best Clients	FOCSANI	2022-01-14	2022-12-22	342	911.935263	34653.54
246	7569	4.39	Valuable Clients	TIMISOARA	2022-02-28	2022-12-14	289	1821.754000	54652.62
257	7779	3.90	Neutral Clients	MARGINA	2022-01-27	2022-11-28	305	1643.777222	29587.99
262	7998	3.78	Neutral Clients	TARGU JIU	2022-03-29	2022-09-27	182	1766.401579	33561.63
265	8042	3.73	Neutral Clients	TIMISOARA	2022-01-17	2022-12-22	339	2684.937500	32219.25
285	8682	4.38	Valuable Clients	TIMISU DE JOS	2022-02-23	2022-12-21	301	1083.813462	28179.15
298	8992	4.71	Best Clients	BISTRITA	2022-01-10	2022-12-20	344	1042.935833	50060.92
302	9080	4.53	Best Clients	ZABALA	2022-01-04	2022-10-19	288	736.609306	53035.87
307	9214	4.73	Best Clients	ODORHEIU SECUIESC	2022-01-03	2022-12-27	358	772.030714	32425.29
314	9541	4.14	Valuable Clients	ODORHEIU SECUIESC	2022-01-26	2022-12-08	316	1321.505652	30394.63
336	9833	4.58	Best Clients	MIERCUREA CIUC	2022-01-10	2022-12-20	344	863.410000	30219.35
339	9866	4.72	Best Clients	MOSNITA NOUA	2022-01-04	2022-12-21	351	839.599362	39461.17
364	10236	4.48	Valuable Clients	CARANSEBES	2022-01-04	2022-12-21	351	1444.565667	43336.97
370	10387	4.28	Valuable Clients	CAMPULUNG	2022-01-10	2022-12-27	351	1505.980000	31625.58
392	10599	4.80	Best Clients	FAGARAS	2022-02-02	2022-12-28	329	791.306250	37982.70
394	10634	4.20	Valuable Clients	ZARNESTI	2022-01-25	2022-12-14	323	1458.706957	33550.26
396	10648	3.41	Neutral Clients	NEGRESTI OAS	2022-02-10	2022-11-23	286	2741.495455	30156.45
399	10691	4.60	Best Clients	TARGU MURES	2022-01-10	2022-12-19	343	1195.356154	46618.89
400	10696	4.27	Valuable Clients	SANNICOLAU MARE	2022-01-04	2022-12-13	343	1297.511538	33735.30

408	10771	4.83	Best Clients	TURDA	2022-01-12	2022-12-28	350	1191.656981	63157.82	

```
groups2[groups2['Cluster'] == 2]
```

	PartnerId	RFM_Score	Customer_segment	City	EarliestDocument	LatestDocument	DaysBetween	SalesMean	SalesSum	Qua
9	162	4.92	Best Clients	SOVATA	2022-01-04	2022-12-27	357	1216.920093	131427.37	20
10	193	4.58	Best Clients	TIMISOARA	2022-02-01	2022-12-19	321	1976.872105	75121.14	20
17	316	4.96	Best Clients	TIMISOARA	2022-01-03	2022-12-28	359	551.989097	85558.31	20
20	342	4.77	Best Clients	MIERCUREA CIUC	2022-01-03	2022-12-19	350	1296.181429	90732.70	20
37	524	4.90	Best Clients	PIATRA NEAMT	2022-01-07	2022-12-27	354	1345.041098	110293.37	20
40	554	4.98	Best Clients	CRAIOVA	2022-01-03	2022-12-30	361	1326.078220	156477.23	20
48	650	4.49	Valuable Clients	PETROSANI	2022-01-11	2022-12-22	345	2616.808333	78504.25	20
93	1106	4.90	Best Clients	SUCEAVA	2022-01-03	2022-12-28	359	1227.491972	87151.93	20
112	1387	4.80	Best Clients	CLUJ NAPOCA	2022-01-05	2022-12-22	351	1214.727719	69239.48	20
133	4518	4.82	Best Clients	TARGU JIU	2022-01-03	2022-12-19	350	1007.468704	108806.62	20
180	6027	4.84	Best Clients	TIMISOARA	2022-01-06	2022-12-23	351	1655.506515	109263.43	20
184	6130	4.85	Best Clients	CLUJ Unknown City	2022-01-25	2022-12-27	336	1838.718000	110323.08	20
188	6246	4.88	Best Clients	TARGU JIU	2022-01-05	2022-12-22	351	954.178265	93509.47	20
194	6326	4.52	Best Clients	TIMISOARA	2022-01-20	2022-12-27	341	3280.757667	98422.73	20
271	8362	4.83	Best Clients	CARANSEBES	2022-01-10	2022-12-28	352	1775.577547	94105.61	20

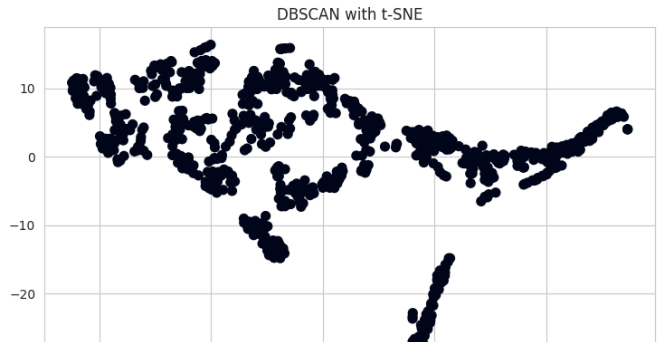
```
groups2[groups2['Cluster'] == 3]
```

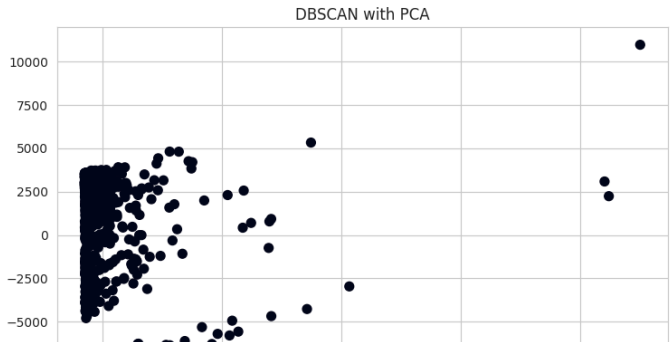
	PartnerId	RFM_Score	Customer_segment	City	EarliestDocument	LatestDocument	DaysBetween	SalesMean	SalesSum	Quart
1	113	4.69	Best Clients	DEJ	2022-01-06	2022-12-20	348	6891.433333	310114.50	2022
92	1098	4.97	Best Clients	BUCURESTI	2022-01-03	2022-12-28	359	776.868359	307639.87	2022
				TARGU						

DBSCAN Clustering

With the DBSCAN algorithm, you do not need to predefine the number of clusters; the model determines it automatically.

```
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=10, min_samples=5)
y_dbscan = dbscan.fit_predict(merged_data)


fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))

axes[0].scatter(data_tsne[:, 0], data_tsne[:, 1], c=y_dbscan, s=50, cmap=c_palette)
axes[0].set_title('DBSCAN with t-SNE')

axes[1].scatter(data_pca[:, 0], data_pca[:, 1], c=y_dbscan, s=50, cmap=c_palette)
axes[1].set_title('DBSCAN with PCA')

plt.tight_layout()
plt.show()
```





Despite various hyperparameter tuning adjustments, the model did not produce interpretable clusters, leading to the conclusion that DBSCAN does not work well for this dataset.

Spectral klaszterezés

Silhouette score

```
from sklearn.cluster import SpectralClustering
silhouette_scores = []
for n_clusters in range(2, 10):
    spectral = SpectralClustering(n_clusters=n_clusters, affinity='rbf')
    cluster_labels = spectral.fit_predict(merged_data)
    silhouette_scores.append(silhouette_score(merged_data, cluster_labels))
plt.plot(range(2, 10), silhouette_scores, color=palette[3])
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette score')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/manifold/_spectral_embedding.py:273: UserWarning:

Graph is not fully connected, spectral embedding may not work as expected.

/usr/local/lib/python3.10/dist-packages/sklearn/manifold/_spectral_embedding.py:273: UserWarning:

Graph is not fully connected, spectral embedding may not work as expected.

/usr/local/lib/python3.10/dist-packages/sklearn/manifold/_spectral_embedding.py:273: UserWarning:

Graph is not fully connected, spectral embedding may not work as expected.

/usr/local/lib/python3.10/dist-packages/sklearn/manifold/_spectral_embedding.py:273: UserWarning:

Graph is not fully connected, spectral embedding may not work as expected.
```