МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ
по ИДЗ
по дисциплине «Машинное обучение»
ТЕМА: Задача кластеризации для RFM-анализа

Студент гр. 6307 _____ Мишанов А. А.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург
2020

Оглавление

Задание: Задача кластеризации для RFM-анализа (выбрать лучший набор клиентов, на который должна ориентироваться компания).

Датасет: Online Retail. Датасет, который содержит все транзакции, произошедшие с 01.12.2010 по 09.12.2011 для британской и зарегистрированной розничной торговли через Интернет. В основном компания продает уникальные подарки на все случаи жизни. Многие клиенты компании являются оптовиками.

Описание задачи

RFM-анализ — это метод анализа, позволяющий сегментировать клиентов по частоте и сумме покупок и выявлять тех клиентов, которые приносят больше денег.

Аббревиатура RFM расшифровывается:

- Recency — давность (как давно клиенты что-то у вас покупали);
- Frequency — частота (как часто они у вас покупают);
- Monetary — деньги (общая сумма покупок).

По этим признакам можно разделить всех клиентов на группы, понять, кто из клиентов покупает у вас часто и много, кто — часто, но мало, а кто вообще давно ничего не покупал.

Как правило, небольшой процент клиентов реагирует на общие рекламные предложения. RFM является отличным методом сегментации для прогнозирования реакции клиента и улучшения взаимодействия, а также повышение прибыли. RFM использует поведение покупателей, чтобы определить, как работать с каждой группой клиентов.

Ход работы

1. **Анализ первичных данных**

Датасет содержит 541909 записей. Всего 8 столбцов:

- InvoiceNo – номер счета оплаты.

- StockCode – код товара.
- Description – описание товара.
- Quantity – количество.
- InvoiceDate – дата выставления счета.
- UnitPrice – цена.
- CustomerID – ID покупателя.
- Country – страна покупки.

2. Подготовка данных

Для непосредственного RFM-анализа необходимо подготовить данные, а именно: давность, частоту и сумму покупок для каждого покупателя.

Перед формированием набора данных необходимо «очистить» датасет от пустых значений.

```
InvoiceNo       0.000000
StockCode       0.000000
Description     0.268311
Quantity        0.000000
InvoiceDate     0.000000
UnitPrice       0.000000
CustomerID     24.926694
Country         0.000000
dtype: float64
```

Рисунок 1. Процент недостающих значений

Далее можно заметить, что поле Quantity содержит отрицательные значения, хотя, руководствуясь здравым смыслом, такого быть не должно. Всего отрицательных значений – 8905, поэтому их можно спокойно отбросить. Количество записей после предварительной подготовки – 397924.

Следующий шаг: сформировать датасет для непосредственного обучения. Как уже было сказано, RFM-анализ содержит три признака: давность, частота, количество потраченных денег. Последовательно создадим эти атрибуты для каждого покупателя.

- **Recency** создается путём группировки по полю CustomerID количества оплат покупателя.

- **Frequency** вычисляется на основе даты последней транзакции. Далее находится разница между датами всех покупок и датой последней транзакции, группируется результат по полю CustomerID минимальной (т. е. сколько прошло времени с последней) даты покупки.
- **Monetary** формируется на основе умножения двух полей (Quantity – количество проданных товаров данного типа для отдельного покупателя, UnitPrice – цена товара) и их группировки по полю CustomerID.

Все таблицы сгруппированы в одну. Результат представлен на рисунке 3.

	CustomerID	Recency	Frequency	Monetary
0	12346.0	325	1	77183.60
1	12347.0	1	182	4310.00
2	12348.0	74	31	1797.24
3	12349.0	18	73	1757.55
4	12350.0	309	17	334.40
...	...	...	...	...
4334	18280.0	277	10	180.60
4335	18281.0	180	7	80.82
4336	18282.0	7	12	178.05
4337	18283.0	3	756	2094.88
4338	18287.0	42	70	1837.28

4339 rows × 4 columns

Рисунок 2. Итоговый датасет для построения модели

Исследование на выбросы

Построим диаграмму размаха и гистограмму по трем признакам, чтобы визуализировать возможные выбросы.

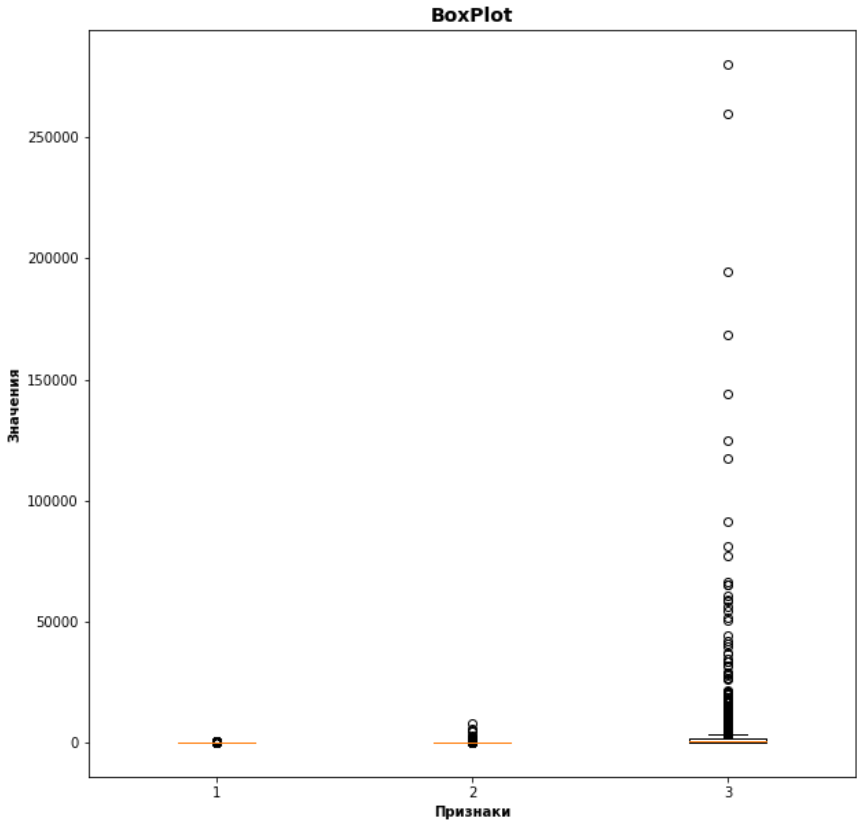

Рисунок 3. Диаграмма размаха

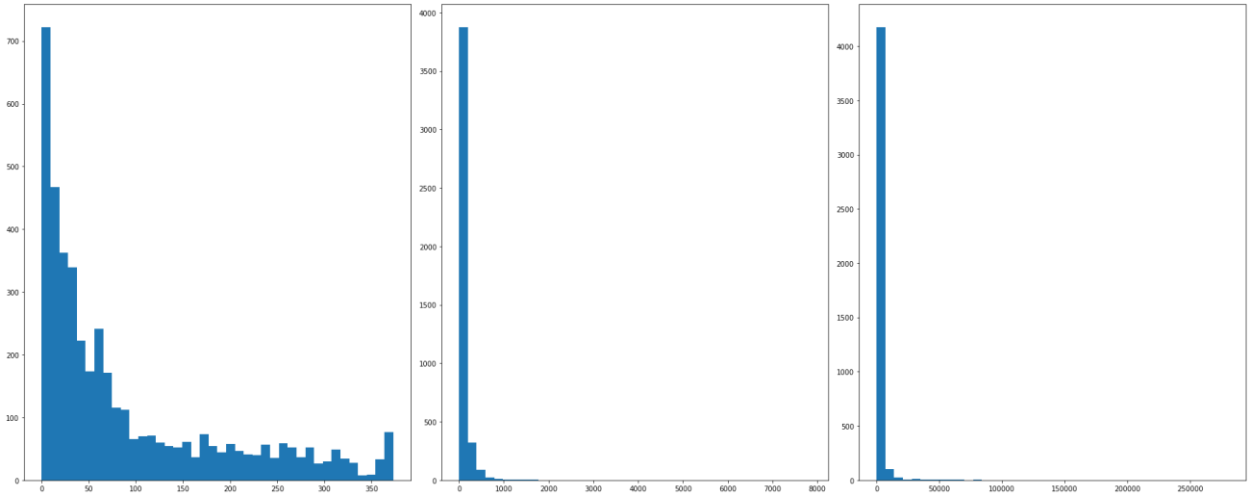

Рисунок 4. Гистограммы признаков

Как видно из графиков, наиболее подвержен выбросам признак Monetary, который обозначает общую сумму покупок. Учитывая, что в датасете содержаться данные как и о розничных покупках, так и о крупных заказов поставщиков, вовсе необязательно что эти данные неадекватны. Однако на всякий случай необходимо удалить крайние значения.

Воспользуемся формулой для определения выбросов, основанной на межквартильном расстоянии, но возьмем 0,95 и 0,05 квантили.

$$[(x_{25} - 1{,}5 \cdot (x_{75} - x_{25})),\ (x_{75} + 1{,}5 \cdot (x_{75} - x_{25}))],$$

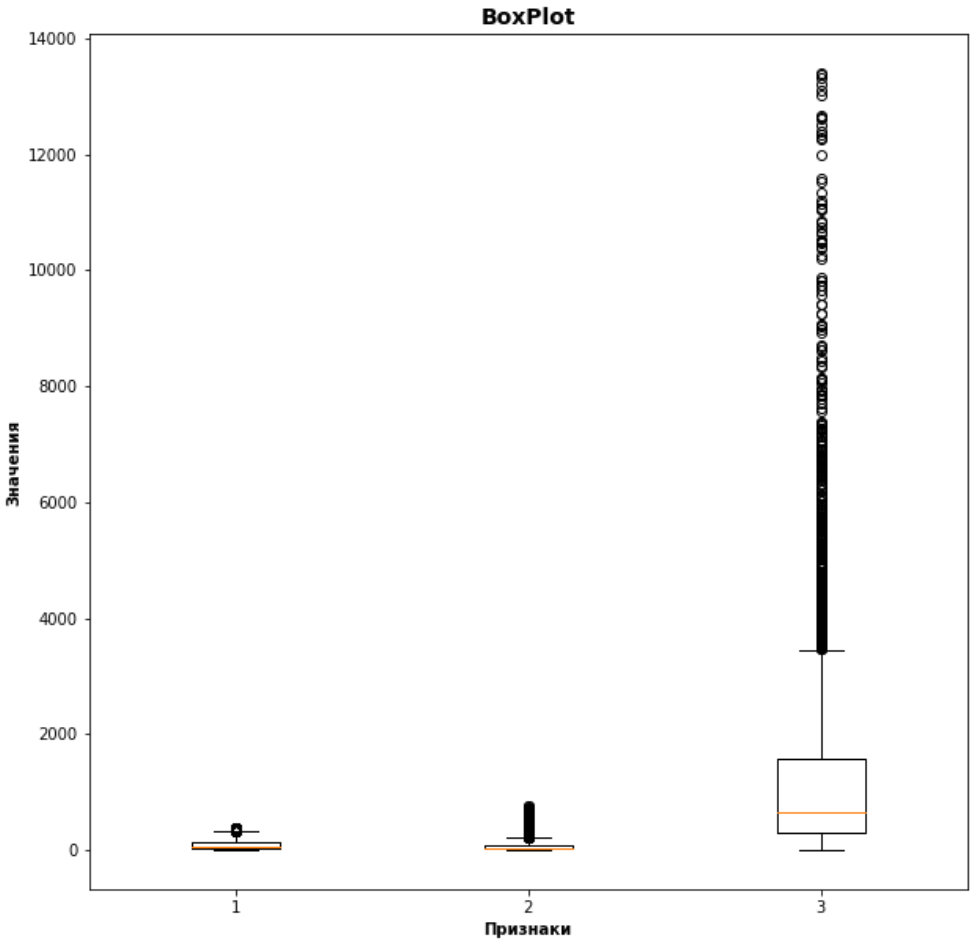

Рисунок 5. Диаграмма размаха

3. Стандартизация данных

Стандартизация набора данных является общим требованием для многих алгоритмов машинного обучения: они могут вести себя плохо, если отдельные функции более или менее не выглядят как стандартные нормально распределенные данные.

В работе использовался *StandardScaler*, который преобразует данные так, что их распределение будет иметь среднее значение 0 и стандартное отклонение 1. Учитывая распределение данных, каждое значение в наборе

данных будет вычитаться из среднего значения выборки, а затем делиться на стандартное отклонение всего набора данных.

$$Y = \frac{X - mean(X)}{std(X)}$$

4. Построение модели

После успешной подготовки данных, можно приступить непосредственно к процессу обучения модели. В данной работе для кластеризации будет использоваться алгоритм K-Means, однако будут рассмотрены и другие подходы.

4.1. K-Means кластеризация

Для кластеризации нам нужна некоторая функция оценки, которая оценивает её качество. Эта функция оценки суммы квадратов ошибок определяется как:

$$SSE(C) = \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} \left\| \mathbf{x}_j - \boldsymbol{\mu}_i \right\|^2$$

Цель состоит в том, чтобы найти кластеризацию, которая минимизирует оценку SSE:

$$C^* = \arg\min_{C} \{SSE(C)\}$$

K-средних инициализирует кластеры путем случайной генерации k точек в пространстве данных. Обычно это делается путем генерации равномерной случайной величины в пределах диапазона для каждого измерения. Каждая итерация K-средних состоит из двух шагов: назначение кластера и обновление центроида.

4.2. DBSCAN

Метод основан на плотности данных в пространстве признаков. Он выполняется следующим образом:

- Алгоритм группирует плотнорасположенные объекты вместе.
- Объекты, находящиеся дальше определенного расстояния, считает шумовыми точками.
- Алгоритма начинается с произвольной точки, которая не была рассмотрена ранее.
- Выбирается окрестность точки, определенного радиуса. Если она содержит достаточное количество точек, чтобы объединить их в один кластер, происходит объединение. Если найдена точка, принадлежащая какому-либо кластеру, то ее окружность тоже является частью кластера. Следовательно, все точки, находящиеся в ее области, тоже являются частью этого кластера. Этот процесс продолжается пока не останется свободных точек. Все точки будут либо принадлежать одному кластеру, либо являться шумовыми точками.

4.3. Иерархическая кластеризация

Для n точек в d-мерном пространстве, цель иерархической кластеризации – создать последовательность вложенных разделов, которую можно удобно визуализировать с помощью дерева или иерархии кластеров, также называемой дендрограммой кластеров. Кластеры в иерархии варьируются от слабо детализированных до сильно детализированных – нижний уровень дерева (листья) состоит из каждой точки в своем собственном кластере, тогда как самый высокий уровень (корень) состоит из всех точек в одном кластере. Обе эти группы можно рассматривать как тривиальные кластеры. На каком-то промежуточном уровне мы можем найти значимые кластеры. Если пользователь предоставляет k, желаемое количество

кластеров, мы можем выбрать уровень, на котором будет находитьсяk кластеров.

4.3.1. Агломеративная иерархическая кластеризация

В агломеративной иерархической кластеризации мы начинаем с каждой из n точек в отдельном кластере. Мы многократно объединяем два ближайших кластера, пока все точки не станут членами одного кластера. Формально, имея набор кластеров: $C = \{C_1, C_1, \dots, C_m\}$ мы находим ближайшую пару кластеров C_i и C_j и объединяем их в новый кластер $C_{ij} = C_i \cup C_j$. Затем мы обновляем набор кластеров, удаляя C_i и C_j и добавляя C_{ij}, таки образом $C = (C \setminus \{ C_i , C_j\}) \cup \{C_{ij}\}$.

Мы повторяем процесс до тех пор, пока C не будет содержать только один кластер. Поскольку количество кластеров уменьшается на один на каждом этапе, этот процесс приводит к последовательности n вложенных кластеров. Если надо, мы можем остановить процесс слияния, когда останется ровно k кластеров.

Основным этапом алгоритма является определение ближайшей пары кластеров.

Метод одиночной связи

$$\delta(C_i, C_j) = \min\{\delta(\boldsymbol{x}, \boldsymbol{y}) \mid \boldsymbol{x} \in C_i, \boldsymbol{y} \in C_j\}$$

Метод полной связи

$$\delta(C_i, C_j) = \max\{\delta(\boldsymbol{x}, \boldsymbol{y}) \mid \boldsymbol{x} \in C_i, \boldsymbol{y} \in C_j\}$$

Метод средней связи

$$\delta(C_i, C_j) = \frac{\sum_{\boldsymbol{x} \in C_i} \sum_{\boldsymbol{y} \in C_i} \delta(\boldsymbol{x}, \boldsymbol{y})}{n_i \cdot n_j}$$

Метод Уорда

$$SSE_i = \sum_{\boldsymbol{x} \in C_i} \|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2$$

$$\delta(C_i, C_j) = \Delta SSE_{ij} = SSE_{ij} - SSE_i - SSE_j$$

4.4. Обучение модели

4.4.1. Выбор оптимального количества кластеров

Фундаментальным шагом для любого алгоритма кластеризации является определение оптимального количества кластеров, в которые могут быть сгруппированы данные. Рассмотри два метода: elbow curve («локтевой» метод) и silhouette analysis (анализ силуэта).

Метод «локтя»

Это эвристический метод, подразумевающий многократное циклическое исполнение алгоритма с увеличением количества выбираемых кластеров, а также последующим откладыванием на графике балла кластеризации, вычисленного как функция от количества кластеров. Необходимо найти точку изгиба в качестве точки отсечения, в которой убывающая отдача больше не стоит дополнительных затрат. В данном случае, выбор стоит между 3, 4, 5 – количеством кластеров.

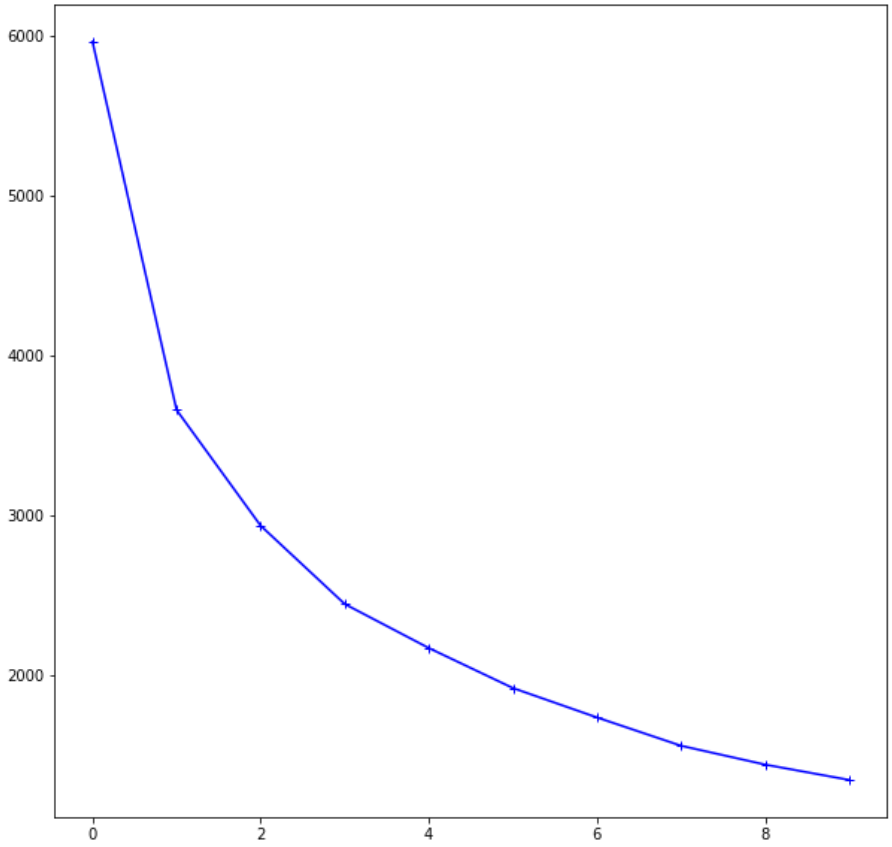

Рисунок 6. Метод «локтя»

Анализ силуэта

Коэффициент «силуэта» вычисляется с помощью среднего внутрикластерного расстояния a и среднего расстояния до ближайшего кластера b по каждому образцу. Силуэт вычисляется как:

$$\frac{p - q}{max(p, q)}$$

b – это расстояние между a и ближайшим кластером, в который a не входит. Можно вычислить среднее значение силуэта по всем образцам и использовать его как метрику для оценки количества кластеров.

- Значение диапазона оценки находится в диапазоне от -1 до 1.
- Оценка, близкая к 1, означает, что точка очень похожа на другой набор точек в кластере.
- Оценка, близкая к -1, указывает на то, что точка не похожа на набор точек в своем кластере.

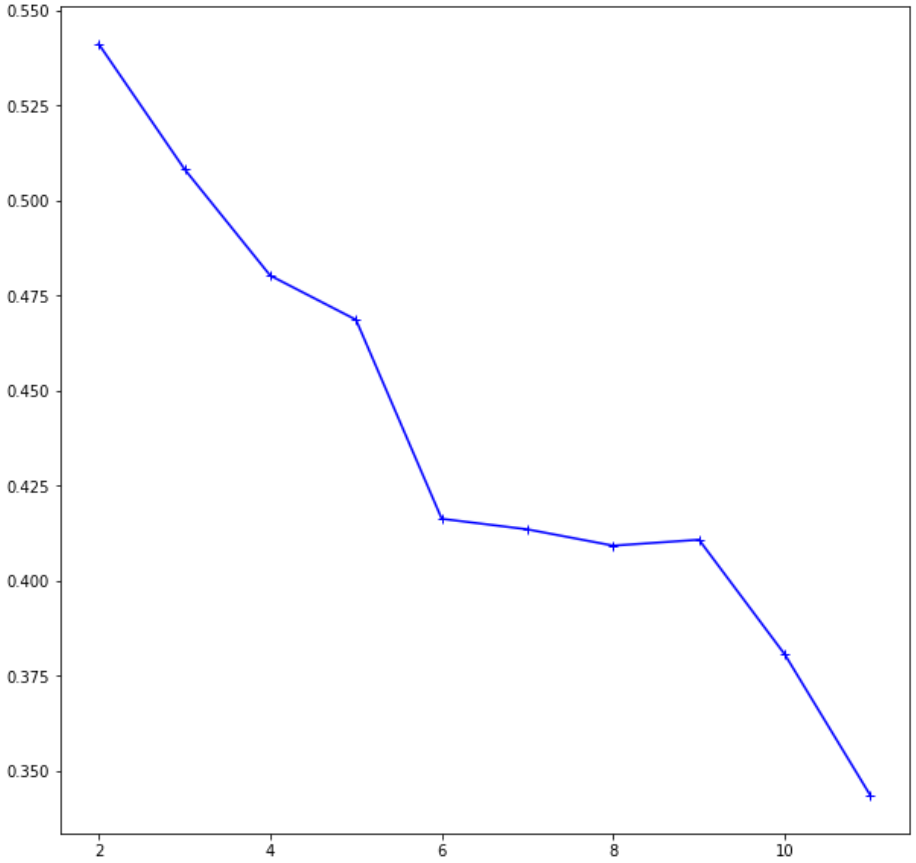

Рисунок 7. Метод силуэта

Как видно из графика, значение оценки только уменьшается от количества кластеров. Выберем наиболее оптимальное количество кластеров k=3 для адекватной интерпретации результатов обучения для сегментации покупателей.

5. Визуализация результатов

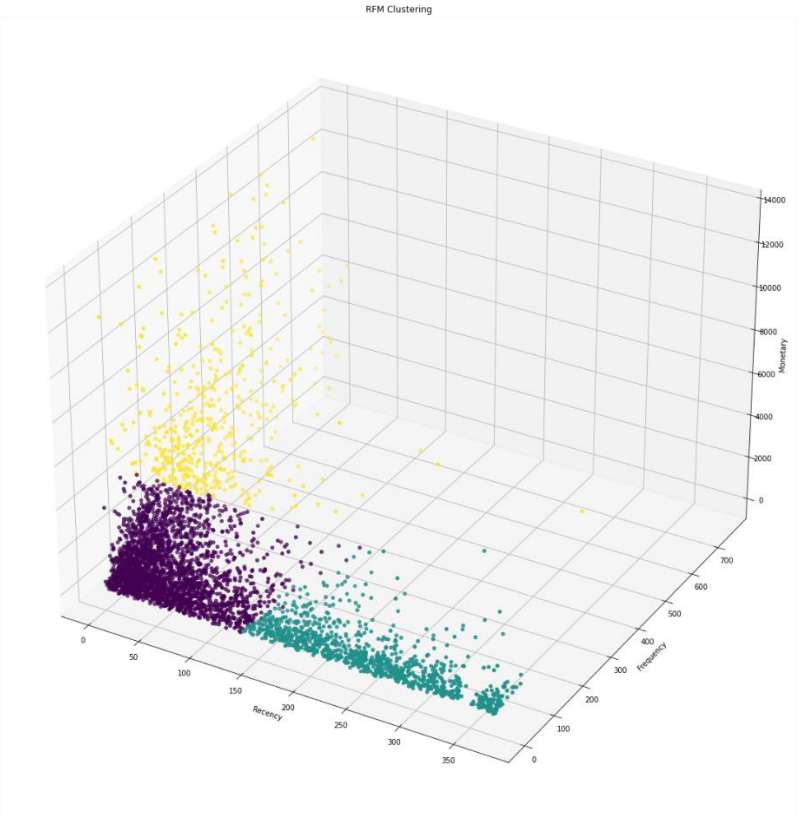

Рисунок 8. Трехмерный график данных, распределенных на 3 кластера

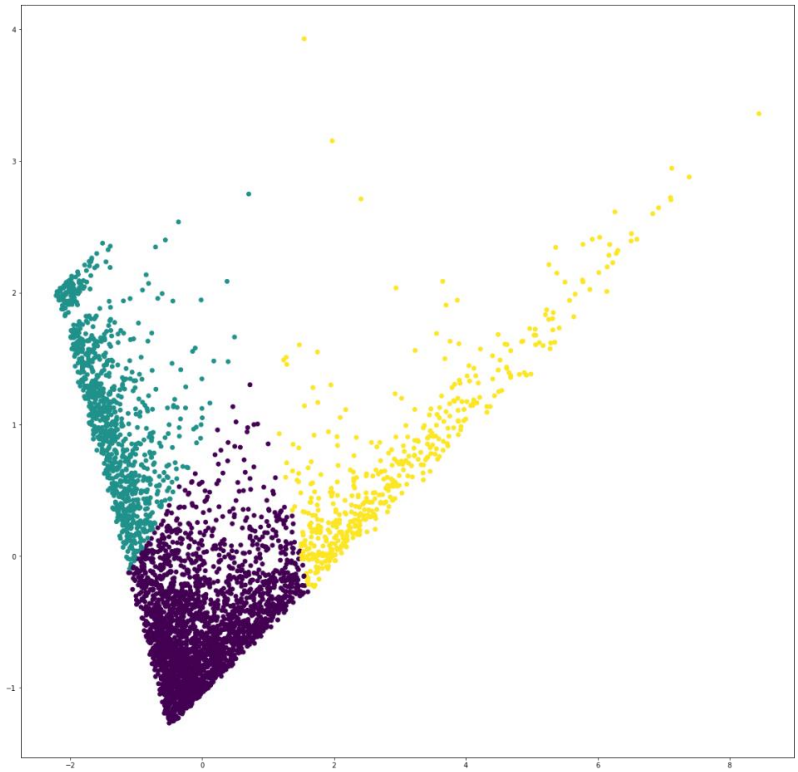

Рисунок 9. Понижение размерности до двухмерного пространства с помощью PCA

6. Другие модели

6.1. DBSCAN

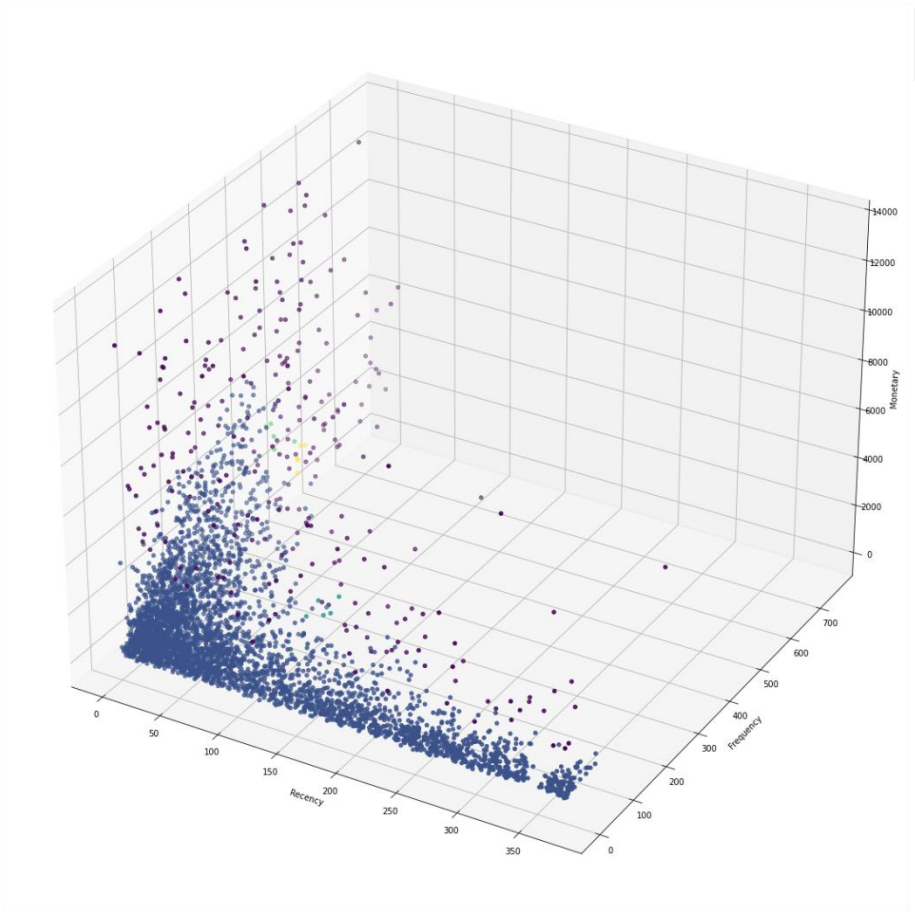

Рисунок 10. Трехмерный график данных после кластеризации DBSCAN

6.2. Иерархическая кластеризация

6.2.1. Метод одиночной связи

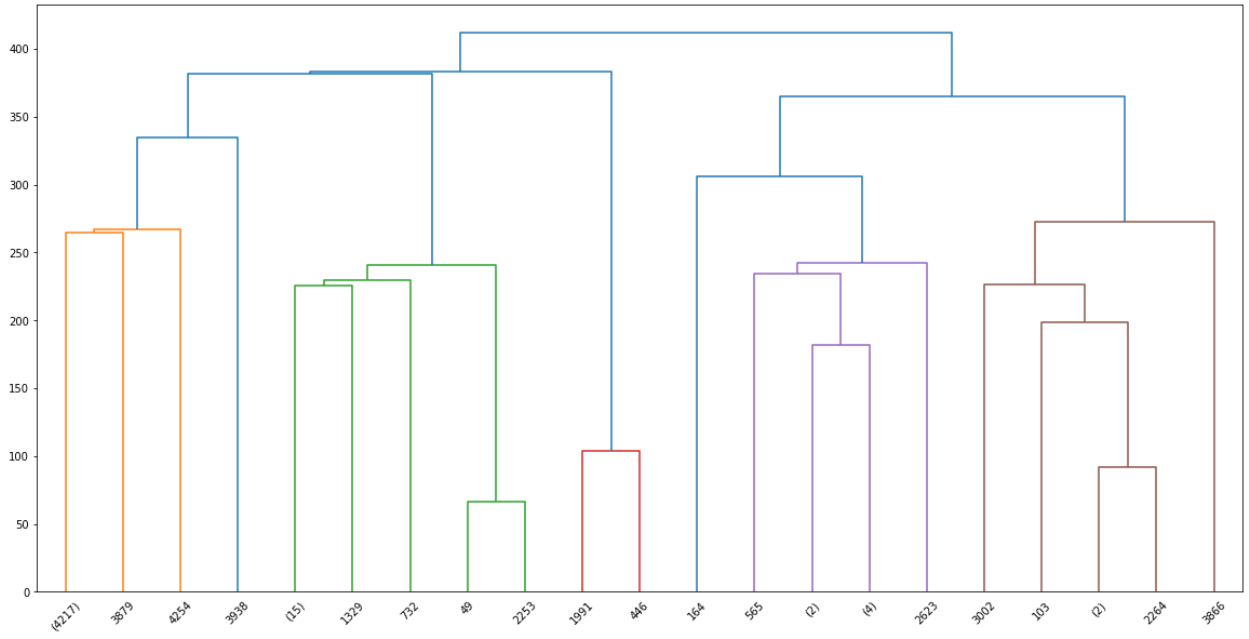

Рисунок 11. Дендрограмма уровня 5

6.2.2. Метод полной связи

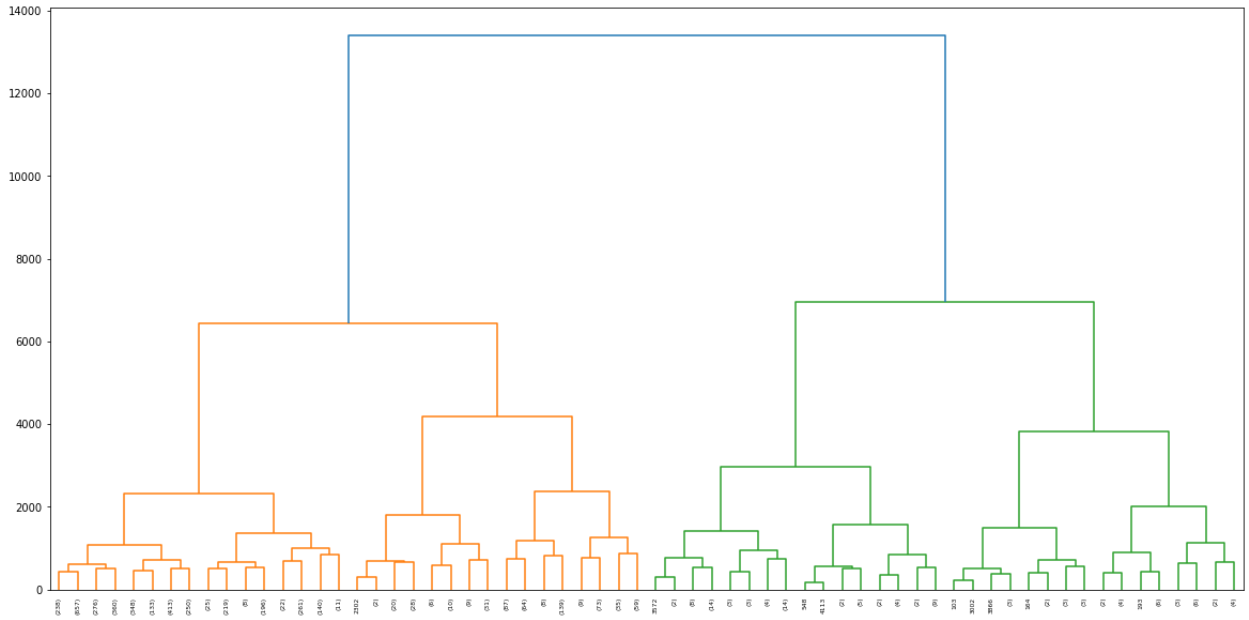

Рисунок 12. Дендрограмма уровня 5

6.2.3. Метод средней связи

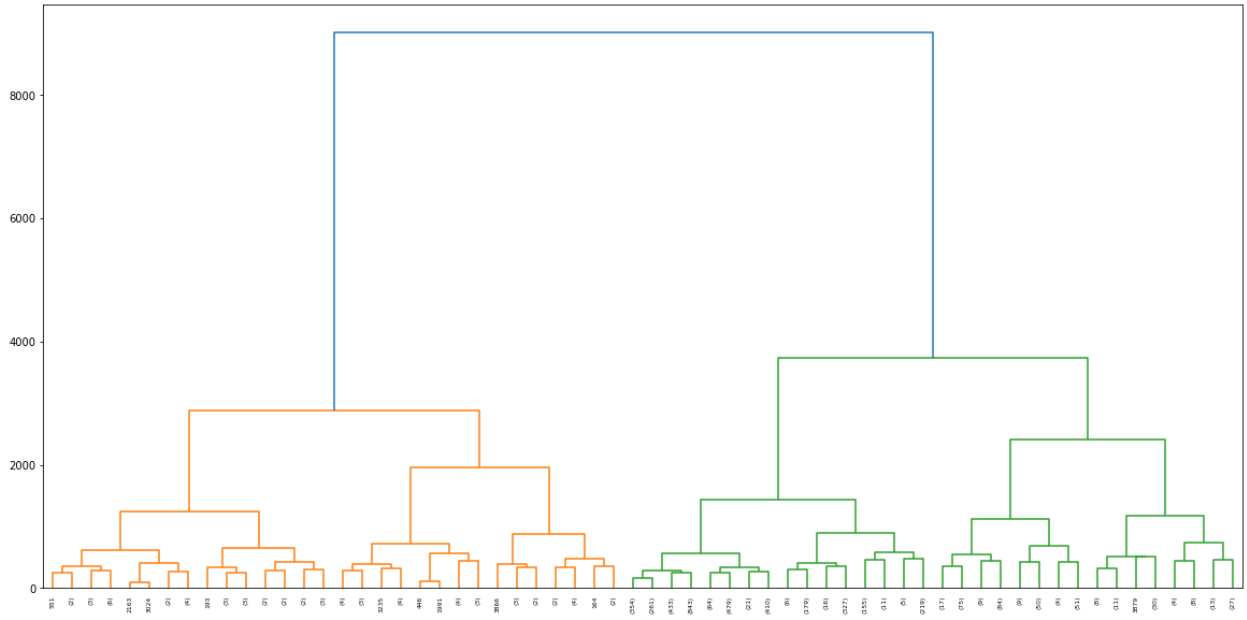

Рисунок 13. Дендрограмма уровня 5

6.2.4. Метод Уорда

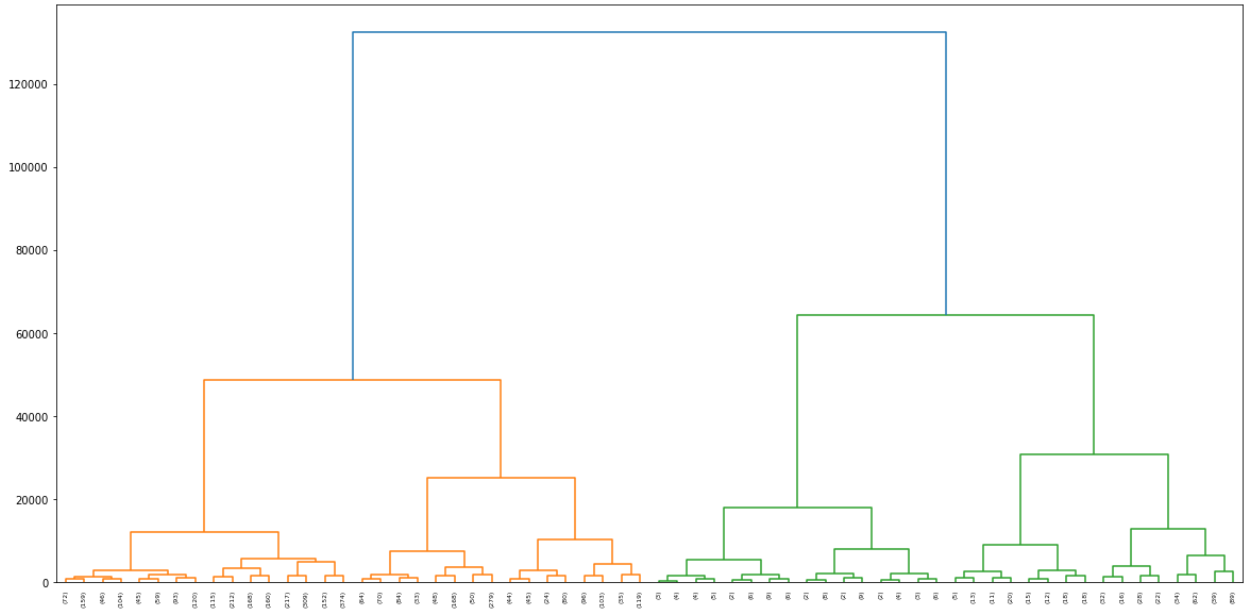

Рисунок 14. Дендрограмма уровня 5

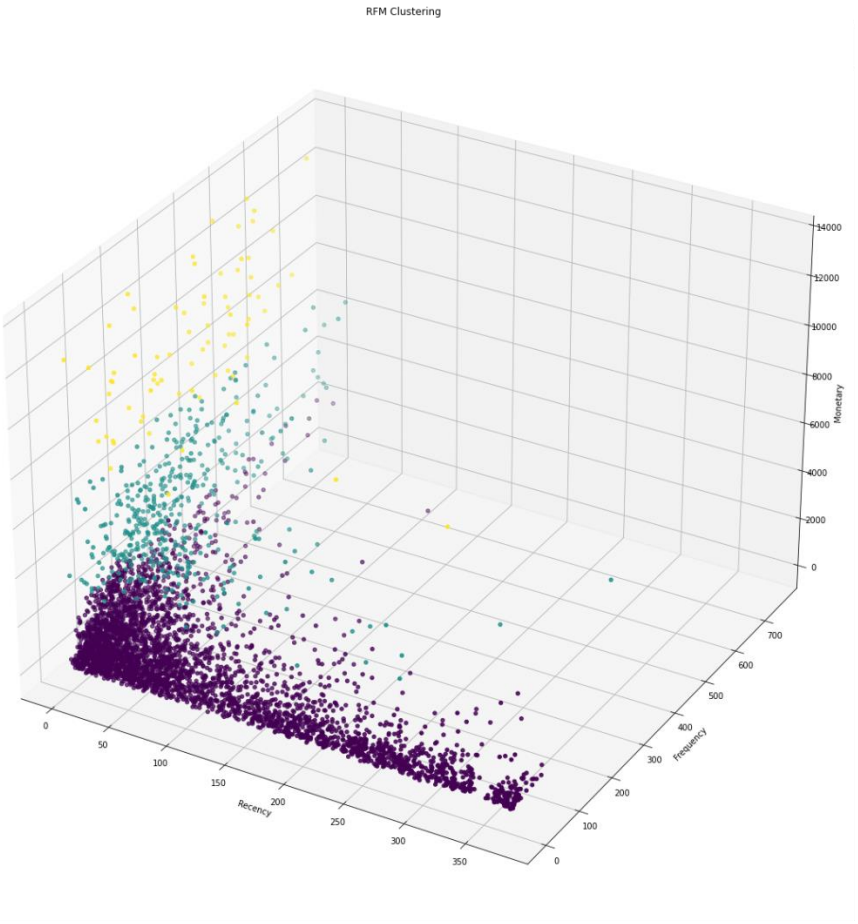

Рисунок 14. Трехмерный график данных, распределенных на 3 кластера (метод Уорда)

7. Итоги

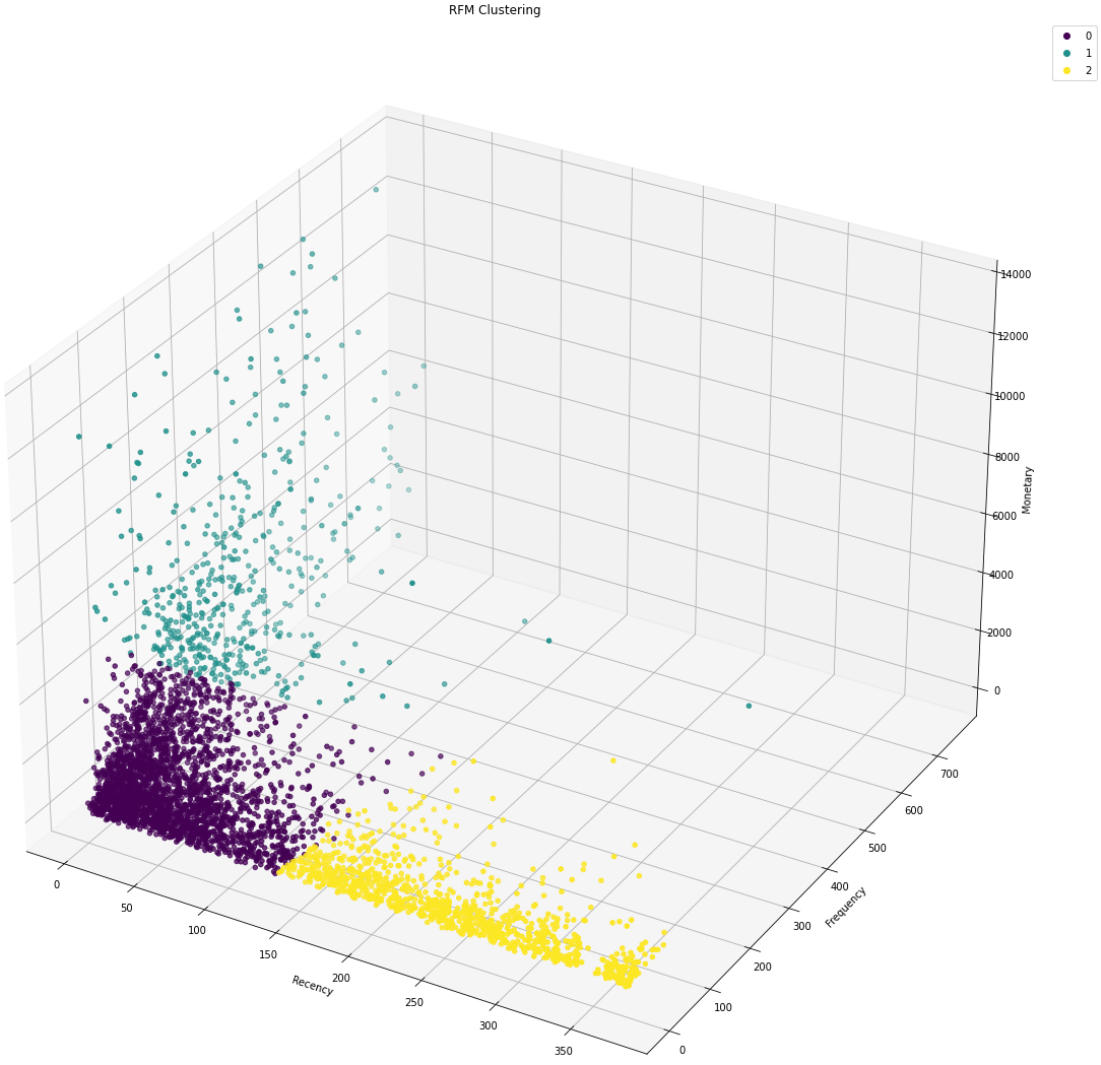

Рисунок 15. Трехмерный график данных, распределенных на 3 кластера (K-Means, итог)

Сведём все данные в итоговую таблицу и проанализируем её.

	ClusterID	CustomerCount	Customer%	MeanRecency	MeanFrequency	Revenue	Revenue%	TransactionCount	Transaction%	AvgPerTransaction	AvgPerCustomer
0	0	2710	63.674812	45.545018	58.797786	2658070.203	47.611961	159342	48.809641	16.681542	980.837713
1	1	494	11.607143	22.965587	283.712551	2471471.280	44.269558	140154	42.931972	17.633969	5002.978300
2	2	1052	24.718045	248.068441	25.627376	453236.771	8.118481	26960	8.258387	16.811453	430.833433

Рисунок 16. Итоговая таблица

K-Means Clustering (k = 3)

- Кластер 0 - обычные клиенты
    - Они составляют большую часть (около 63%) от всех клиентов и это главный по величине источник дохода (47%).

    - Средняя давность около 48 дней назад.
    - Они часто делают покупки.
- Кластер 1 - ценные клиенты / оптовики.
    - Есть только 494 клиента кластера 0, которые составляют всего 11% от общего числа клиентов и 42% от общего числа транзакций и более 44% общего дохода поступает от них.
    - Они также любят делать покупки и в целом тратят гораздо больше. Их средняя сумма покупки за транзакцию составляет около 17 долларов, что является самым высоким показателем среди всех трех групп клиентов. Их средняя общая сумма расходов составляет более 5002 долларов.
    - Они покупают довольно часто со средней давностью 22 дня.
- Кластер 2 - разовые, нечастые клиенты.
    - Они составляют 24% от всех, но приносят лишь немногим более 8% от общего дохода. Их сумма транзакции по счету - 16 долларов.
    - В среднем они последний раз совершали покупки в Интернете около 248 дней назад.
    - Они совершают покупки лишь изредка

Возникшие проблемы

1. Проблема выбора количества кластеров для K-Means алгоритма. Было принято решение использовать два различных способа для определения оптимального количества кластеров.
2. Не получилось успешно применить алгоритм DBSCAN. При его использовании с разными параметрами так и не удалось получить адекватных результатов, итоговое разбиение на кластеры получалось в виде одного большего и нескольких других незначительных выбросов.

ПРИЛОЖЕНИЕ

```
# # Задача кластеризации для RFM-анализа

# In[1]:


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
from mpl_toolkits.mplot3d import Axes3D

import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA


# ## 1. Анализ первичных данных

# In[2]:


# Загрузка датасета

retail = pd.read_csv('OnlineRetail.csv', sep=',', encoding='ISO-8859-1',
header=0)
retail.head()


# In[3]:


retail.shape


# In[4]:


retail.info()


# In[5]:


retail.describe()


# In[6]:


# Процент недостающих значений в датафрейме

df_null = 100 * (retail.isnull().sum()) / len(retail)
df_null
```

```
# In[7]:


# Удаление строк с пропущенными значениями

retail = retail.dropna()
retail.shape


# In[8]:


retail.min()


# In[9]:


# Количество отрицательных и нулевых значений для столбца Quantity

len(retail[retail['Quantity'] <= 0])


# In[10]:


# Удаление отрицательных значений и нулевых значений

retail = retail[retail['Quantity'] > 0]
retail.shape


# In[11]:


retail['CustomerID'] = retail['CustomerID'].astype(str)


# ## 2. Подготовка данных

#  Для RFM-анализа необходимо создать новый набор данных для последующей кластеризации.
# - R (Recency) - давность.
# - F (Frequency) - частота.
# - M (Monetary) - деньги.

# In[12]:


# Monetary

retail['Amount'] = retail['Quantity'] * retail['UnitPrice']
rfm_m = retail.groupby('CustomerID')['Amount'].sum()
rfm_m = rfm_m.reset_index()
rfm_m.columns = ['CustomerID', 'Monetary']
rfm_m.head()


# In[13]:
```

```
# Frequency

rfm_f = retail.groupby('CustomerID')['InvoiceNo'].count()
rfm_f = rfm_f.reset_index()
rfm_f.columns = ['CustomerID', 'Frequency']
rfm_f.head()


# In[14]:


# Recency

retail['InvoiceDate'] = pd.to_datetime(retail['InvoiceDate'], format='%d-%m-%Y
%H:%M')


# In[15]:


# Максимальная дата - дата последней транзакции

max_date = max(retail['InvoiceDate'])
max_date


# In[16]:


# Разница между максимальной датой и датой транзакции = как давно что-то
покупали

retail['Diff'] = max_date - retail['InvoiceDate']
retail.head()


# In[17]:


# Вычислить дату последней транзакции, чтобы узнать время последней покупки

rfm_r = retail.groupby('CustomerID')['Diff'].min()
rfm_r = rfm_r.reset_index()
rfm_r.head()


# In[18]:


# Только количество дней

rfm_r['Diff'] = rfm_r['Diff'].dt.days
rfm_r.head()


# In[19]:


# Объединение в итоговую таблицу

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

```
rfm = pd.merge(rfm, rfm_m, on='CustomerID', how='inner')
rfm.columns = ['CustomerID', 'Recency', 'Frequency', 'Monetary']
rfm


# In[20]:


rfm.describe()


# #### Исследование на выбросы

# In[21]:


attributes = ['Recency', 'Frequency', 'Monetary']
plt.figure(figsize=(10, 10))
plt.boxplot(rfm[attributes])
plt.title("BoxPlot", fontsize = 14, fontweight = 'bold')
plt.ylabel("Значения", fontweight = 'bold')
plt.xlabel("Признаки", fontweight = 'bold')


# In[22]:


fig, ax = plt.subplots(ncols=3, figsize=(25, 10))
ax[0].hist(rfm['Recency'], bins=40)
ax[1].hist(rfm['Frequency'], bins=40)
ax[2].hist(rfm['Monetary'], bins=40)
fig.tight_layout()


# In[23]:


# Удаление статистическиз выбросов для каждого признака
Q1 = rfm['Recency'].quantile(0.05)
Q3 = rfm['Recency'].quantile(0.95)
IQR = Q3 - Q1
rfm = rfm[(rfm['Recency'] >= Q1 - 1.5*IQR) & (rfm['Recency'] <= Q3 + 1.5*IQR)]

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

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


# In[24]:


attributes = ['Recency', 'Frequency', 'Monetary']
plt.figure(figsize=(10, 10))
```

```
plt.boxplot(rfm[attributes])
plt.title("BoxPlot", fontsize = 14, fontweight = 'bold')
plt.ylabel("Значения", fontweight = 'bold')
plt.xlabel("Признаки", fontweight = 'bold')


# ## 3. Стандартизация данных

# In[25]:


# Standard Scaler

rfm_df = rfm[['Recency', 'Frequency', 'Monetary']]
scaler = StandardScaler()
rfm_df_scaled = scaler.fit_transform(rfm_df)
rfm_df_scaled.shape


# In[26]:


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


# ## 4. Построение модели

# ### K-Means Clustering

# K-Means Clustering - один из самых простых и популярных алгоритмов кластеризации.
#
# Алгоритм работает следующим образом:
#
# - Сначала мы случайным образом инициализируем k точек.
# - Мы классифицируем каждый элемент по ближайшему среднему значению и обновляем координаты среднего значения, которые представляют собой средние значения элементов, отнесенных к этому среднему значению на данный момент.
# - Мы повторяем процесс для заданного количества итераций, и в конце получаем группы кластеров.

# In[27]:


# K-Means с 4 кластерами

kmeans = KMeans(n_clusters=4, max_iter=100)
kmeans.fit(rfm_df_scaled)


# In[28]:


kmeans.labels_


# ### Поиск оптимального количества кластеров
```

```
      # Фундаментальным шагом для любого алгоритма кластеризации является
определение оптимального количества кластеров, в которые могут быть сгруппированы
данные.

      # #### "Локтевой" метод (Elbow Curve)

      # In[29]:


      # Elbow-curve

      ssd = []
      range_n_clusters = range(2, 12)
      for num_clusters in range_n_clusters:
          kmeans = KMeans(n_clusters=num_clusters, max_iter=100)
          kmeans.fit(rfm_df_scaled)
          ssd.append(kmeans.inertia_)

      plt.figure(figsize=(10, 10))
      plt.plot(ssd, 'b+-')


      # #### Silhouette Analysis
      #

      # * Значение диапазона оценки находится в диапазоне от -1 до 1.
      # * Оценка, близкая к 1, означает, что точка очень похожа на другой набор
точек в кластере.
      # * Оценка, близкая к -1, указывает на то, что точка не похожа на набор точек
в своем кластере.

      # In[30]:


      # Silhouette analysis
      range_n_clusters = range(2, 12)
      silhouette_avg = []

      for num_clusters in range_n_clusters:

          # intialise kmeans
          kmeans = KMeans(n_clusters=num_clusters, max_iter=100)
          kmeans.fit(rfm_df_scaled)

          cluster_labels = kmeans.labels_
          silhouette = silhouette_score(rfm_df_scaled, cluster_labels)

          # silhouette score
          silhouette_avg.append(silhouette)
          print("For n_clusters={0}, the silhouette score is
{1}".format(num_clusters, silhouette))


      # In[31]:


      plt.figure(figsize=(10, 10))
      plt.plot(range_n_clusters, silhouette_avg, 'b+-')


      # In[32]:
```

```
        # Окончательный результат при k=3
        kmeans = KMeans(n_clusters=3, max_iter=100)
        kmeans.fit(rfm_df_scaled)


        # In[33]:


        kmeans.labels_


        # In[34]:


        rfm['ClusterID'] = kmeans.labels_
        rfm.head()


        # #### Визуализация кластеров (K-Means)

        # In[35]:


        fig = plt.figure(figsize=(20,20))
        ax = fig.add_subplot(111, projection='3d')
        sc = ax.scatter(xs=rfm['Recency'], ys=rfm['Frequency'], zs=rfm['Monetary'],
c=rfm['ClusterID'])

        plt.title('RFM Clustering')
        ax.set_xlabel('Recency')
        ax.set_ylabel('Frequency')
        ax.set_zlabel('Monetary')
        plt.legend(*sc.legend_elements(), bbox_to_anchor=(1,1), loc=2)


        # #### С понижением размерности до 2D

        # In[36]:


        rfm_df_scaled.head()


        # In[38]:


        pca = PCA(n_components = 2)
        pca_rfm = pca.fit_transform(rfm_df_scaled.to_numpy())


        # In[39]:


        plt.figure(figsize=(20,20))
        plt.scatter(x=pca_rfm.T[0], y=pca_rfm.T[1], c=kmeans.labels_)


        # ### DBSCAN
```

```
	# Метод основан на плотности данных в пространстве признаков. Он выполняется следующим образом:
	# - Алгоритм группирует плотнорасположенные объекты вместе.
	# - Объекты, находящиеся дальше определенного расстояния, считает шумовыми точками.
	# - Алгоритма начинается с произвольной точки, которая не была рассмотрена ранее.
	# - Выбирается окрестность точки, определенного радиуса. Если она содержит достаточное количество точек, чтобы объединить их в один кластер, происходит объединение. Если найдена точка, принадлежащая какому-либо кластеру, то ее окружность тоже является частью кластера. Следовательно, все точки, находящиеся в ее области, тоже являются частью этого кластера. Этот процесс продолжается пока не останется свободных точек. Все точки будут либо принадлежать одному кластеру, либо являться шумовыми точками.

	# In[40]:


	dbscan = DBSCAN(min_samples=7, metric='euclidean')
	dbscan.fit(rfm_df_scaled)


	# In[41]:


	dbscan.labels_


	# In[42]:


	from sklearn.neighbors import NearestNeighbors
	nearest_neighbors = NearestNeighbors(n_neighbors=5)
	nearest_neighbors.fit(rfm_df_scaled)
	distances, indices = nearest_neighbors.kneighbors(rfm_df_scaled)
	distances = np.sort(distances, axis=0)[:, 1]
	print(distances)
	plt.plot(distances)
	plt.show()


	# In[43]:


	dbscan = DBSCAN(eps=0.3, min_samples=5, metric='euclidean')
	dbscan.fit(rfm_df_scaled)


	# In[44]:


	dbscan.labels_


	# In[45]:


	rfm['ClusterID_DBSCAN'] = dbscan.labels_
	rfm.head()
```

```
# #### Визуализация кластеров (DBSCAN)

# In[46]:


fig = plt.figure(figsize=(20,20))
ax = fig.add_subplot(111, projection='3d')
sc = ax.scatter(xs=rfm['Recency'], ys=rfm['Frequency'], zs=rfm['Monetary'],
c=rfm['ClusterID_DBSCAN'])
ax.set_xlabel('Recency')
ax.set_ylabel('Frequency')
ax.set_zlabel('Monetary')
plt.legend(*sc.legend_elements(), bbox_to_anchor=(1,1), loc=2)


# #### С понижением размерности до 2D

# In[47]:


pca = PCA(n_components = 2)
pca_rfm = pca.fit_transform(rfm_df_scaled.to_numpy())


# In[48]:


plt.figure(figsize=(20, 20))
plt.scatter(x=pca_rfm.T[0], y=pca_rfm.T[1], c=dbscan.labels_)


# ### Иерархическая кластеризация

# #### Агломеративная иерархическая кластеризация

# In[49]:


rfm_df.head()


# In[50]:


def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack([model.children_, model.distances_,
                                      counts]).astype(float)
    plt.figure(figsize=(20,10))
```

```
            dendrogram(linkage_matrix, **kwargs)


        # #### Метод одиночной связи

        # In[51]:


        agg_clustering = AgglomerativeClustering(distance_threshold=0,
n_clusters=None, linkage='single')
        agg_clustering.fit(rfm_df)
        agg_clustering.labels_


        # In[52]:


        plot_dendrogram(agg_clustering, truncate_mode='level', p=5)


        # #### Метод полной связи

        # In[53]:


        agg_clustering = AgglomerativeClustering(distance_threshold=0,
n_clusters=None, linkage='complete')
        agg_clustering.fit(rfm_df)
        agg_clustering.labels_


        # In[54]:


        plot_dendrogram(agg_clustering, truncate_mode='level', p=5)


        # #### Метод средней связи

        # In[55]:


        agg_clustering = AgglomerativeClustering(distance_threshold=0,
n_clusters=None, linkage='average')
        agg_clustering.fit(rfm_df)
        agg_clustering.labels_


        # In[56]:


        plot_dendrogram(agg_clustering, truncate_mode='level', p=5)


        # #### Метод Уорда

        # In[57]:


        agg_clustering = AgglomerativeClustering(distance_threshold=0,
n_clusters=None)
```

```
agg_clustering.fit(rfm_df)
agg_clustering.labels_


# In[58]:


plot_dendrogram(agg_clustering, truncate_mode='level', p=5)


# #### Ограничим алгоритм 3 кластерами

# In[59]:


agg_clustering = AgglomerativeClustering(n_clusters=3)
agg_clustering.fit(rfm_df)
agg_clustering.labels_


# In[60]:


fig = plt.figure(figsize=(20,20))
ax = fig.add_subplot(111, projection='3d')
sc = ax.scatter(xs=rfm['Recency'], ys=rfm['Frequency'], zs=rfm['Monetary'],
c=agg_clustering.labels_)

plt.title('RFM Clustering')
ax.set_xlabel('Recency')
ax.set_ylabel('Frequency')
ax.set_zlabel('Monetary')
plt.legend(*sc.legend_elements(), bbox_to_anchor=(1,1), loc=2)


# ## Результат

# In[61]:


fig = plt.figure(figsize=(20,20))
ax = fig.add_subplot(111, projection='3d')
sc = ax.scatter(xs=rfm['Recency'], ys=rfm['Frequency'], zs=rfm['Monetary'],
c=rfm['ClusterID'])

plt.title('RFM Clustering')
ax.set_xlabel('Recency')
ax.set_ylabel('Frequency')
ax.set_zlabel('Monetary')
plt.legend(*sc.legend_elements(), bbox_to_anchor=(1,1), loc=2)


# In[62]:


rfm = rfm[['CustomerID', 'Recency', 'Frequency', 'Monetary', 'ClusterID']]
rfm.head()


# In[63]:
```

```
      analysis_rfm = rfm.groupby('ClusterID', as_index=False)['Monetary'].sum()
      analysis_rfm.columns = ['ClusterID', 'Revenue']
      analysis_rfm.head()


      # In[64]:


      analysis_rfm['CustomerCount'] = rfm['ClusterID'].value_counts()
      analysis_rfm['Customer%'] = 100 * analysis_rfm['CustomerCount'] /
sum(analysis_rfm['CustomerCount'])
      analysis_rfm.head()


      # In[65]:


      analysis_rfm['MeanRecency'] = rfm.groupby('ClusterID',
as_index=False)['Recency'].mean()['Recency']
      analysis_rfm.head()


      # In[66]:


      analysis_rfm['TransactionCount'] = rfm.groupby('ClusterID',
as_index=False)['Frequency'].sum()['Frequency']
      analysis_rfm['Transaction%'] = 100 * analysis_rfm['TransactionCount'] /
sum(analysis_rfm['TransactionCount'])
      analysis_rfm.head()


      # In[67]:


      analysis_rfm['MeanFrequency'] = analysis_rfm['TransactionCount'] /
analysis_rfm['CustomerCount']


      # In[68]:


      analysis_rfm['Revenue%'] = 100 * analysis_rfm['Revenue'] /
sum(analysis_rfm['Revenue'])


      # In[69]:


      analysis_rfm['AvgPerTransaction'] = analysis_rfm['Revenue'] /
analysis_rfm['TransactionCount']


      # In[70]:


      analysis_rfm['AvgPerCustomer'] = analysis_rfm['Revenue'] /
analysis_rfm['CustomerCount']
```

```
# In[71]:


analysis_rfm = analysis_rfm[['ClusterID', 'CustomerCount', 'Customer%',
'MeanRecency', 'MeanFrequency', 'Revenue', 'Revenue%','TransactionCount',
'Transaction%', 'AvgPerTransaction', 'AvgPerCustomer']]


# In[72]:


analysis_rfm.head()


# K-Means Clustering (k = 3)
# * Кластер 0 - обычные клиенты
#     - Они составляют большую часть (около 63%) от всех клиентов и это
главный по величине источник дохода (47%).
#     - Средняя давность около 48 дней назад.
#     - Они часто делают покупки.
# * Кластер 1 - ценные клиенты / оптовики.
#     - Есть только 494 клиента кластера 0, которые составляют всего 11% от
общего числа клиентов и 42% от общего числа транзакций и более 44% общего дохода
поступает от них.
#     - Они также любят делать покупки и в целом тратят гораздо больше. Их
средняя сумма покупки за транзакцию составляет около 17 долларов, что является самым
высоким показателем среди всех трех групп клиентов. Их средняя общая сумма расходов
составляет более 5002 долларов.
#     - Они покупают довольно часто со средней давностью 22 дня.
# * Кластер 2 - разовые, нечастые клиенты.
#     - Они составляют 24% от всех, но приносят лишь немногим более 8% от
общего дохода. Их сумма транзакции по счету - 16 долларов.
#     - В среднем они последний раз совершали покупки в Интернете около 248
дней назад.
#     - Они совершают покупки лишь изредка
```