**Министерство образования и науки Российской Федерации**

**Федеральное государственное автономное образовательное учреждение высшего образования   
«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КАФЕДРА ИНФОРМАЦИОННЫХ СИСТЕМ

Направление: 09.03.02 – Информационные системы и технологии

Профиль: Информационные системы в образовании

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

**АНАЛИЗ ТРАНЗАКЦИОННЫХ ДАННЫХ**

Студент 4-го курса  
группы 09-412   
"\_\_\_\_"\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г. Хуснетдинов Д.Р.

Научный руководитель   
к. ф.-м. н., доцент  
"\_\_\_\_"\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г. Гафаров Ф.М.

Заведующий кафедройд. т. н., профессор  
"\_\_\_\_"\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г. Сулейманов Д.Ш.

Казань – 2018

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc515404546)

[1 МАРКЕТПЛЕЙС 5](#_Toc515404547)

[2 АНАЛИЗ СО СТОРОНЫ КЛИЕНТОВ 10](#_Toc515404548)

[2.1 Парсинг 12](#_Toc515404549)

[2.2 Откидывание столбца событий 14](#_Toc515404550)

[2.3 Фильтрация данных за период 15](#_Toc515404551)

[2.4 Агрегация данных за период и избавление от аутлаеров 17](#_Toc515404552)

[2.5 Агрегация по группам 21](#_Toc515404553)

[2.6 Метрики и кластеризация 23](#_Toc515404554)

[3 АНАЛИЗ СО СТОРОНЫ МЕРЧАНТОВ 40](#_Toc515404555)

[3.1 Агрегация данных за период относительно мерчантов и избавление от аутлаеров 41](#_Toc515404556)

[3.2 Метрики и кластеризация мерчантов 45](#_Toc515404557)

[3.3 Пересечение клиентов и мерчантов 50](#_Toc515404558)

[ЗАКЛЮЧЕНИЕ 56](#_Toc515404559)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 57](#_Toc515404560)

[ПРИЛОЖЕНИЕ 58](#_Toc515404561)

[Скрипт connect\_setting.py: 58](#_Toc515404562)

[Скрипт Main\_Script.py: 59](#_Toc515404563)

[Скрипт CSV\_Export(Step№1).py: 62](#_Toc515404564)

[Скрипт Starting\_Table(Step№2).py: 64](#_Toc515404565)

[Скрипт Starting\_Table\_For\_The\_Period(Step№3).py: 66](#_Toc515404566)

[Скрипт Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Clients(Step№4.1).py: 67](#_Toc515404567)

[Скрипт Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Correct\_Clients(Step№4.2).py: 68](#_Toc515404568)

[Скрипт Transaction\_Table\_Of\_Correct\_Clients(Step№4.3).py: 70](#_Toc515404569)

[Скрипт Aggregation\_Count\_Of\_Trans\_On\_Groups(Step№5).py: 72](#_Toc515404570)

[Скрипт Create\_Procedure\_CountOfTransForMonthOnClients(Step№6.1).py: 73](#_Toc515404571)

[Скрипт Create\_Main\_Metric\_Table\_For\_Clients(Step№6.2).py: 75](#_Toc515404572)

[Скрипт Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Clients\_On\_Merchants(Step№7.1).py: 83](#_Toc515404573)

[Скрипт Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Correct\_Clients\_On\_Merchants(Step№7.2).py: 84](#_Toc515404574)

[Скрипт Transaction\_Table\_Of\_Correct\_Clients\_On\_Merchants(Step№7.3).py: 86](#_Toc515404575)

[Скрипт Create\_Procedure\_CountOfTransForMonthOnMerchants(Step№8.1).py: 88](#_Toc515404576)

[Скрипт Create\_Metric\_Tables\_On\_Merchants\_Extended(Step№8.2).py: 90](#_Toc515404577)

[Скрипт Create\_Comparison\_Tables(Step№9).py: 97](#_Toc515404578)

# ВВЕДЕНИЕ

Сейчас медлительным и неэффективным компаниям, не готовым к переменам, остаётся мало шансов на выживание. Компании должны эволюционировать, их важнейшими преимуществами, активами должны стать информация и знания.

Извлечением информации из данных с последующим формированием в знание занимается аналитика. Под аналитикой подразумевается широкое использование данных, количественного и статистического анализа, описательных и прогнозных моделей для принятия решений и действий, на основе реальных признаков и фактов.

Некоторые отрасли, больше других склонны к использованию аналитики. Если в бизнесе генерируется много данных о транзакциях – скажем, это финансовые услуги, перевозки, туризм или игорный бизнес – то конкуренция на основе аналитики является естественной и правильной стратегией. Хотя многие фирмы всё же пренебрегают аналитикой. Если в основе вашей бизнес-модели факторы трудноизмеримы – допустим, человеческие отношения, в том числе поиска необходимых кадров, или стиль, как в индустрии моды – для соперничества на основе аналитики необходимы намного более изобретательные приёмы.

Сотни миллионов транзакций ежедневно проходят через банки, поэтому на серверах накапливаются большие данные: информация о клиентах, шаблоны покупок, правила в целом. Таким образом, банки превращаются в IT-ориентированные компании, как это произошло с телекоммуникационными операторами. Они предоставляют все больше услуг и цифровых сервисов, а собираемые ими данные и извлекаемая из них информация активно используются в создании новых предложений, и тех же услуг, и сервисов.

Применить эту информацию можно во множестве сфер и приложений, таких как: классические задачи оптимизации, обработка транзакций, распознавание мошенничества, кибербезопасность, создание персональных финансовых ассистентов.

Современные банки прекрасно понимают, чем живут их клиенты, и могут моделировать их поведение: будь это социальная группа в конкретном городе, отдельная промышленность или индустрия, или страна в целом. Это информация помогает финансовым учреждениям управлять своими рисками и рисками своих клиентов.

С развитием Data Science(DS) в банках и IT-компаниях стало популярным концентрировать R&D отделы c DS. Концентрированная практика DS внутри компаний способствует более продуктивному созданию новых IT-продуктов, а также решений бизнес-задач, при этом не утрачивая и не отставая в технологиях по всему спектру и перечню связанных с DS задач.

# 1 МАРКЕТПЛЕЙС

Есть кейс: Банк предоставляет данные о транзакциях своих клиентов – 1 244 890 записей 22 794 клиентов. Необходимо выполнить анализ предоставленных данных и получить некоторую полезную информацию, которая поспособствует развитию банка. Ожидаемые результаты: предиктивная модель, выделение наиболее значимых признаков, описание процесса анализа, любые другие инсайты.

Зачем это нужно банку? Мотивация: Развитие банка как единого поля для продвижения товаров и услуг нефинансовых организаций является важной частью стратегии развития банка.

Эта цель может достигаться разными способами, но банк предлагает следующую методику: Определение наиболее популярных мерчантов (торговец, коммерсант, бизнес-лицо. Часто это название связанно со службой, которая предоставляет возможность принимать платежи с использованием банковской карты) среди клиентов банка для развития сотрудничества с данными организациями с целью проведения совместных маркетинговых кампаний и разработка совместных продуктов для привлечения клиентской базы.

Перед тем как решать данный кейс, необходимо ознакомиться с предоставленными данными. Данные представляют собой транзакции случайной выборки владельцев пластиковых карт банка за период с 2016.01–2017.06 гг.(файл расширения .csv). Среди транзакций выделены только успешные расходные операции, которые совершены в рамках рассматриваемых МСС групп(код категории продавца — представляет собой четырёхзначный номер, определяющий род деятельности торговой точки в операции оплаты с помощью банковских карт в торгово-сервисном предприятии). Записи в файле имеют следующие поля:

1. event\_id – идентификатор транзакции, тип данных long\_integer
2. cli\_id – идентификатор клиента, тип данных long\_integer
3. trans\_date – дата/время транзакции, тип данных datetime
4. trans\_amount\_rub – сумма операции в рублях, тип данных decimal(38,2)
5. document\_channel – канал совершения операции, тип данных integer
6. document\_channel\_group – группа канала документа, например: банкомат, покупки в интернете, касса и т.д. Тип данных integer
7. our\_device – терминал приема оплаты, предоставленный банком, тип данных boolean
8. merchant\_id – идентификатор мерчанта, тип данных long\_integer
9. code – MCC/SIC код мерчанта. Определяет основной вид деятельности торговой точки, тип данных integer
10. group – категория мерчанта. Группировка MCC/SIC кодов по схожим тематикам, например: рестораны, медицина, еда и т.д. Тип данных string.

Для анализа нам понадобятся следующие инструменты:

1. Microsoft SQL Server – для хранения данных в виде наборов DataSet в таблицах. Для манипуляции данными при помощи SQL-запросов и скриптов, созданных и вызывающихся динамически из программы
2. Python 3.6 и PyCharm IDE – для автоматизированного создания аналитической модели на любой машине с помощью написанных скриптов на языке Python, включая: парсинг файла, формирование и вызова SQL-скриптов, создание наборов данных, вычисление новых метрик, применение алгоритмов машинного обучения
3. Git – распределённая система контроля версий. Для быстрого разделения и слияния версий, включая инструменты навигации по нелинейной истории разработки
4. GitHub – хостинг проекта для совместной разработки. Любой желающий может скачать проект, присоединится к нему, или предложить свои улучшения по проекту, если вы дадите на то согласие. Проект размещён по следующему url: <https://github.com/KhusDM/BankDataAnalysisRelease>
5. GitKraken – кроссплатформенный графический Git-клиент для визуализации веток с удобным интерфейсом. Отображает историю разработки.

Наши данные содержат истории двух сущностей – клиентов и мерчантов. Поэтому анализ будет проводится как со стороны клиента, так и со стороны мерчанта. В ходе анализа мы выясним детально как связаны эти две сущности между собой, каково их взаимодействие, по каким правилам проводятся операции, обнаружим неявные закономерности. С учётом поставленных целей выработан предполагаемый порядок работы анализа:

1. Парсинг данных из файла формата .csv и их перенос в SQL Server
2. Знакомство с данными, валидация данных, описательная статистика транзакций во временной динамике
3. Поиск пропущенных/нелогичных значений
4. Отбор признаков для построения моделей
5. Избавление от аутлаеров относительно клиентов(выбросов, ложных точек)
6. Группировка транзакций по продуктовым группам относительно клиентов, описательная статистика по клиентам во временной динамике
7. Генерирование статистических метрик на основе признаков, рассмотренных во временной динамике(мат.ожидание, дисперсия, среднеквадратичное отклонение, коэффициент вариации и т.д.)
8. Кластеризация клиентов на основе метрик
9. Избавление от аутлаеров относительно мерчантов(выбросов, ложных точек)
10. Группировка транзакций по продуктовым группам относительно мерчантов, описательная статистика по мерчантам во временной динамике
11. Генерирование статистических метрик на основе признаков, рассмотренных во временной динамике(мат.ожидание, дисперсия, среднеквадратичное отклонение, коэффициент вариации и т.д.)
12. Кластеризация мерчантов на основе метрик
13. Пересечение мерчантов и клиентов, создание таблиц сопоставлений мерчантов с клиентами.

В результате у нас получатся таблицы сопоставлений мерчантов и клиентов по определенным продуктовым группам, с помощью которых банк может делать предложения по взаимодействию мерчантов со своими клиентами в зависимости от принадлежности к определённому кластеру как мерчантов, так и клиентов. По выполнению проделанных работ, каталог таблиц в SQL Server должен иметь следующий вид, изображённый на рисунке 1.1:

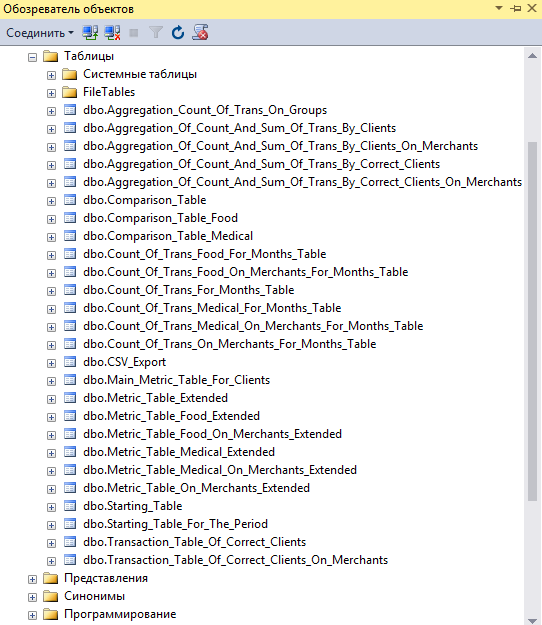


Рисунок 1.1 – Каталог таблиц в SQL Server

Выбранный порядок действий не является единственно верным. Всё зависит от поставленных целей и решаемых задач, от специалистов и их методик по анализу. В любом случае процесс стоит сделать постепенным и итеративным.

# 2 АНАЛИЗ СО СТОРОНЫ КЛИЕНТОВ

Прежде чем приступить непосредственно к анализу, необходимо сказать пару слов про строение проекта. Проект представляет собой набор скриптов на языке Python, которые вызываются друг за другом по очереди, запустив главный скрипт с названием Main\_Script.py. В Main\_Script.py определён порядок вызова скриптов, выполняющихся через подпроцессы. Таким образом проект имеет итеративный характер, каждый запускаемый скрипт – это определённый шаг на пути к построению модели. В каждом скрипте происходит обращение к SQL Server’y с помощью библиотеки pypyodbc, которая является адаптером, определяющая интерфейс DB API 2.0 – интерфейс прикладной программы с базой данных. Этот интерфейс, должен реализовывать все модули расширения, которые предназначаются для связи программ на языке Python с базами данных. Единый API позволяет абстрагироваться от марки используемой СУБД, при необходимости будет довольно просто поменять одну СУБД на другую, изучив всего один набор методов и функций.

Для корректного обращения программы к СУБД нужно задать настройки соединения. Для этого есть connect\_setting.py, являющийся по сути конфигурационным файлом, в котором задаются такие настройки соединения как DRIVER(драйвер, т.е. СУБД), SERVER(сервер, принадлежащий пользователю СУБД), DATABASE(БД, в которой выполняются все операции). Затем из этих настроек формируется строка соединения с нашей СУБД. Помимо этого, в конфигурационном файле присутствует имя файла с данными, который необходимо парсить. Также определены некоторые константы, которые будут использоваться в скриптах, о них будет упоминаться непосредственно при разборе шагов построения.

Общий каталог проекта выглядит следующим образом на рисунке 2.1:

1. Папка scripts содержит скрипты и папку connection\_setting, в которой содержится конфигурационный файл connect\_setting.py
2. Папка venv c некоторыми файлами создаётся по умолчанию. В последующем в эту папку будут добавляться некоторые файлы внешних библиотек, скаченных с помощью менеджера пакетов pip
3. Файл с набором данных Dataset\_for\_KFU.06\_07\_2017.csv
4. В External Libraries содержатся файлы внешних библиотек.

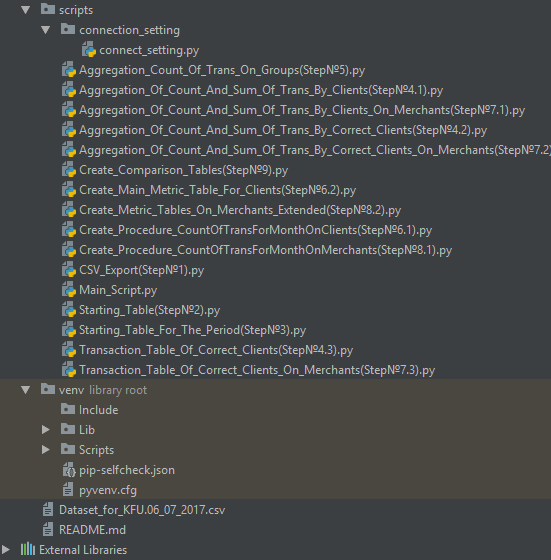


Рисунок 2.1 – Директория проекта

# 2.1 Парсинг

Первый шаг любого анализа данных это выгрузка данных. В рамках кейса, нам предоставлен файл с данными в формате .csv. На рисунке 2.1.1 можно увидеть содержимое файла и его структуру:

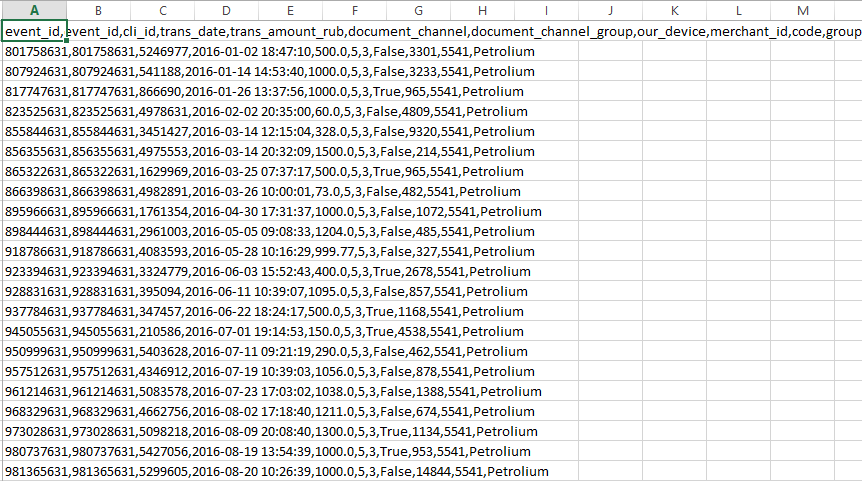


Рисунок 2.1.1 – Исходный файл с данными в формате .csv

Как видите, данные сложно интерпретировать в таком формате, не говоря о проведение над ними операций. Поэтому нужно произвести парсинг файла и перенести полученные данные в таблицу SQL Server’a. Парсинг файла производится в скрипте CSV\_Export(Step№1).py. В нём подключается ссылка на модуль pypyodbc, чтобы программа могла взаимодействовать с СУБД. Стоит сказать, что практически во всех скриптах производится подключение этого модуля, поэтому в дальнейшем будем опускать повествование об этом.

Также в скрипте подключается модуль pandas, который создан для обработки и анализа данных. pandas предоставляет специальные структуры данных в виде dataframe(датафрейм представлены в виде виртуальных таблиц, хранящихся в оперативной памяти во время runtime момента) и операции для управления таблицами и временными рядами.

После того как все необходимые модули подключены, дальше в скрипте происходит считывание данных, парсинг, небольшая обработка для дальнейшей вставки в таблицу в SQL Server’а. Также происходит подключение к самой СУБД, формирование динамического SQL-скрипта для создания таблицы с последующей вставкой данных, выполнение SQL-скрипта. В результате появилась таблица dbo.CSV\_Export. Код скрипта и реализация в целом всего проекта находится в разделе «Приложение».

Результат выполнение изображён на рисунке 2.1.2, таблица dbo.CSV\_Export:

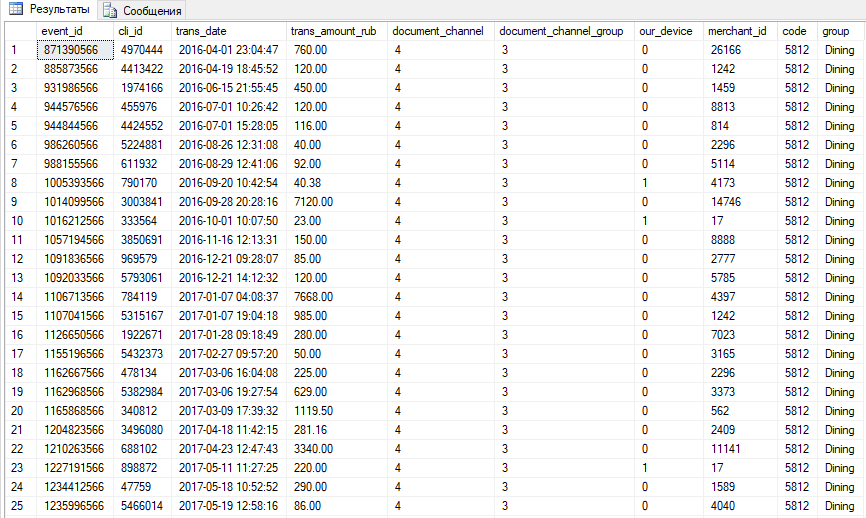


Рисунок 2.1.2 – Таблица dbo.CSV\_Export

# 2.2 Откидывание столбца событий

Изучив таблицу с импортированными данными, автор пришёл к выводу, что данные столбца event\_id не будут участвовать в проводящемся анализе(решается конкретная задача, поэтому данное решение не является единственно верным. Всё зависит от целей, задач и методик), поэтому, за счёт ненадобности, стоит откинуть этот столбец. Данное действие выполняется в скрипте Starting\_Table(Step№2).py. В скрипте происходит формирование SQL-скрипта на создание новой таблицы dbo.Starting\_Table на основе таблицы dbo.CSV\_Export, только при этом опускается столбец event\_id. Результат операции изображён на рисунке 2.2.1, таблица dbo.Starting\_Table:

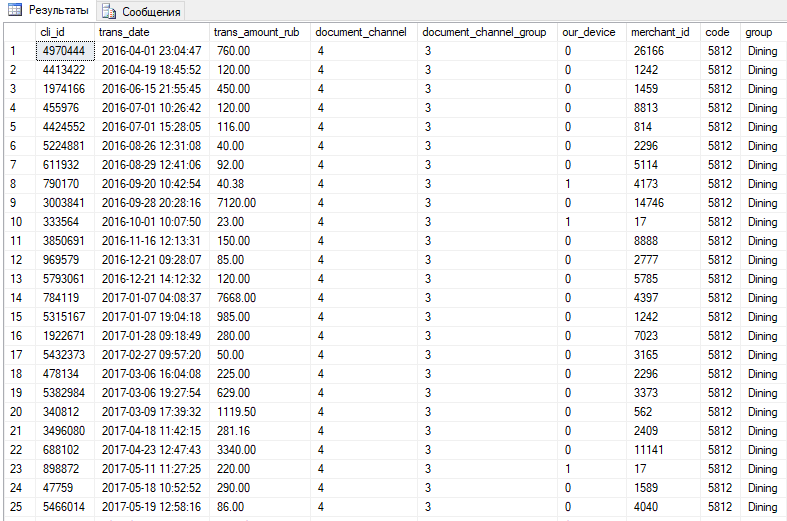


Рисунок 2.2.1 – Таблица dbo.Starting\_Table

# 2.3 Фильтрация данных за период

Следующим шагом будет отбор данных за определённый временной период, и в дальнейшем последующие шаги анализа будут проводиться в рамках этого периода. Нужные границы временного интервала задаются в конфигурационном файле connect\_setting.py, заполнением полей DATA\_BEGIN(начало) и DATA\_END(конец). Значение этих полей будут участвовать в построение запроса SQL. Значение задаётся в формате yyyy-mm-dd(год-месяц-день). Фильтрация данных происходит в скрипте с названием Starting\_Table\_For\_The\_Period(Step№3).py. В нём формируется динамический SQL-запрос, при выполнения которого у нас останутся данные за определённый временной интервал. В данном случае был выбран период за 2016.01.01-2017.05.31. Стоит сказать, что количество дней не будет играть роли, поскольку анализ во временной динамике будет проводиться по месяцам. Поэтому рекомендуется выбирать временные интервалы с учётом полных месяцев. Предоставленные нам данные содержат транзакции с 2016.01.01 по 2017.06.10. Как видите, получается, что за последний месяц неполные данные, если включать эти данные за неполный месяц как за равноценный полный месяц, то наш анализ ухудшится. Поэтому откидываются транзакции за этот неполный месяц, указав правую границу временного интервала – последний полный месяц. Таким образом анализ будет более чистым. Результаты данного шага изображены на рисунке 2.3.1, таблица dbo.Starting\_Table\_For\_The\_Period:

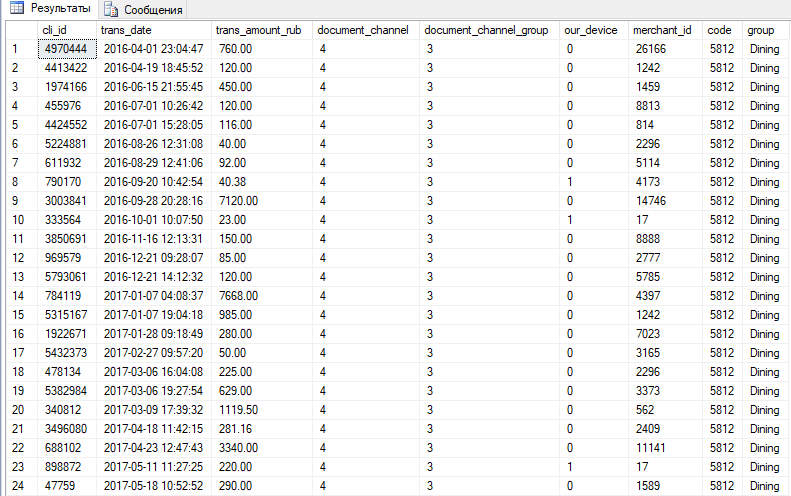


Рисунок 2.3.1 – Таблица dbo.Starting\_Table\_For\_The\_Period

# 2.4 Агрегация данных за период и избавление от аутлаеров

Теперь можно произвести агрегацию количества транзакций и суммы этих транзакций за весь рассматриваемый период, сгруппировав результаты по клиентам. Этот шаг необходим, чтобы выделить клиентов, на которых у нас есть информация. Дело в том, что в данных присутствуют записи о клиентах, которые совершили одну-две транзакции за весь период. По сути получается, что на этих клиентов недостаточно данных, следовательно, мы не можем моделировать, предсказывать их поведение и включать их в анализ, основываясь на неполных данных. Такие записи будут считаться как аутлаеры(выбросы, ложные точки). Во время анализа, важной частью является избавление от аутлаеров, поскольку они искажают результирующую модель. Показателем того, есть ли у нас информация по клиенту или нет, является количество транзакций. Чем больше клиент совершил транзакций, тем больше есть информации на него, следовательно, его можно включить в анализ.

Первоначальная агрегация выполняется в скрипте Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Clients(Step№4.1).py. Результат шага изображён на рисунке 2.4.1, таблица dbo.Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Clients:

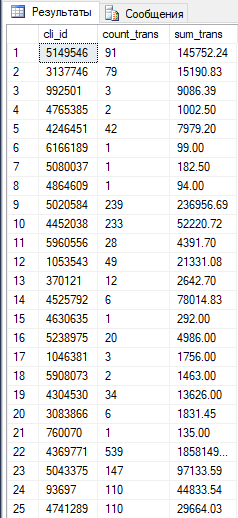


Рисунок 2.4.1 – Таблица dbo.Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Clients

В агрегированной таблице содержатся аутлаеры – записи с минимальным количеством транзакций. На данном этапе необходимо понять: какая должна быть нижняя граница у количества транзакций? Сколько минимально должно быть количество транзакций, чтоб клиент не считался аутлаером? Один из вариантов можно посчитать, что если клиент выполняет в среднем хотя бы одну транзакцию в неделю на протяжение всего периода, то можно считать, что на него есть данные. Временная динамика рассматривается по месяцам, следовательно, за месяц должно быть в среднем четыре транзакции. Таким образом, нижняя граница будет равна 4 транзакции \* количество месяцев в периоде. Все клиенты, у которых количество транзакций не удовлетворяют нижней границе, считаются аутлаерами. Это лишь один из примеров как может формироваться нижняя граница. Нижнюю границу можно рассчитывать по-разному.

Избавление от аутлаеров происходит в скрипте Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Correct\_Clients(Step№4.2). Результат изображён на рисунке 2.4.2, таблица dbo.Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Correct\_Clients:

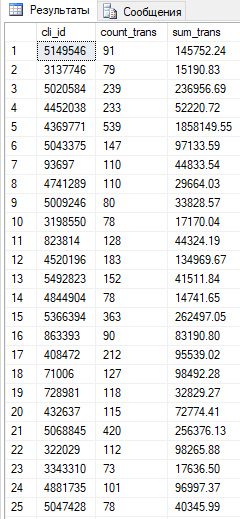


Рисунок 2.4.2 – Таблица dbo.Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Correct\_Clients

После этого в агрегирующей таблице остались только корректные клиенты. Теперь надо учесть этот момент в общей таблице транзакций dbo.Starting\_Table\_For\_The\_Period, найдя пересечение по клиентам. Таким образом, в общей таблице будут содержаться транзакции только корректных клиентов. Пересечение выполняется в скрипте Transaction\_Table\_Of\_Correct\_Clients(Step№4.3).py. Результат пересечения изображён на рисунке 2.4.3, таблица dbo.Transaction\_Table\_Of\_Correct\_Clients:

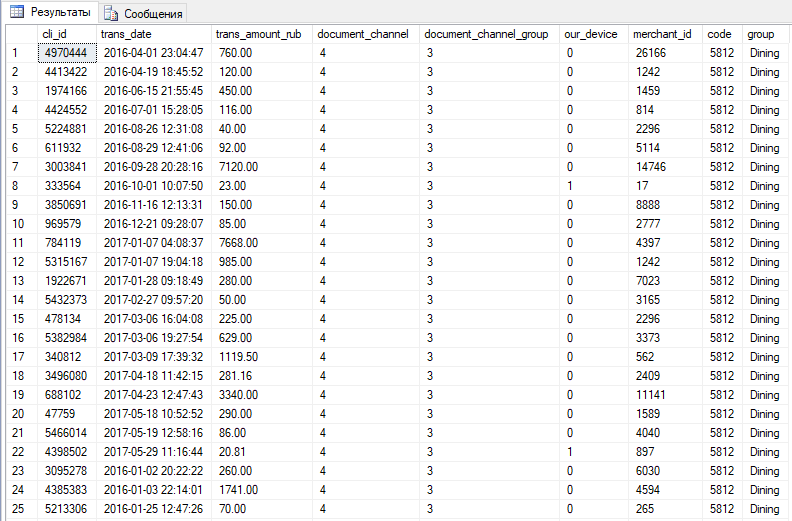


Рисунок 2.4.3 – Таблица dbo.Transaction\_Table\_Of\_Correct\_Clients

# 2.5 Агрегация по группам

Анализ клиентов в частности проводится в разрезе определённых продуктовых групп. В каких именно группах проводить анализ указывается в конфигурационном файле connect\_setting.py, перечислив значения в поле GROUPS. И тут возникает вопрос: какие именно группы необходимо указать, относительно каких групп должен происходить анализ? Чтобы определить наиболее значимые группы, которые востребованы у клиентов, необходимо агрегировать количество и суммы транзакций сгруппировав их по продуктовым группам. Агрегация по продуктовым группам выполняется в Aggregation\_Count\_Of\_Trans\_On\_Groups(Step№5).py. Результат выполнения скрипта изображён на рисунке 2.5.1, таблица dbo.Aggregation\_Count\_Of\_Trans\_On\_Groups:



Рисунок 2.5.1 – Таблица dbo.Aggregation\_Count\_Of\_Trans\_On\_Groups

Здесь, как и в таблице агрегирования по клиентам, важен столбец количества транзакций. Чем больше транзакций, тем группа считается востребованной клиентами банка, следовательно, возможно, стоит сконцентрироваться на группах с относительно высоким уровнем количества транзакций(если не стоит задача по привлечению клиентов в менее востребованные группы). Отсортировав агрегированную таблицу по количеству транзакций по убыванию, увидим группы, которые могут быть кандидатами на анализ, если, конечно, нет каких-либо других причин, почему не стоит проводить анализ в этих группах. В данном случаем были отобраны группы Food и Medical. Все последующие шаги будут выполняться относительно этих групп.

# 2.6 Метрики и кластеризация

Поскольку анализ проводится во временной динамике, где за единицу времени взят месяц, необходимо пронаблюдать как клиент совершает операции из месяца в месяц на протяжение всего наблюдаемого периода. Показателем за месяц всё также будет выступать количество транзакций, агрегированные по клиентам. Таким образом нашей целью становится создания таблицы, где каждый столбец будет содержать количество транзакций клиента за определённый месяц. Для создания такой таблицы удобно было бы иметь функцию, которая агрегировала транзакции. Поэтому создадим такую функцию. Создание функции происходит в скрипте Create\_Procedure\_CountOfTransForMonthOnClients(Step№6.1).py. В этом скрипте формируется SQL-запрос на создание хранимой процедуры, которая будет использоваться для агрегирования.

Следует сказать, что данный параграф будет объёмным по количеству проделанных операций. На данном этапе мы получим не только таблицы с прослеживанием количества операций по клиентам во временной динамике в продуктовых группах, но и ещё целый ряд таблиц, которые будут содержать в себе сгенерированные статистические метрики, а также принадлежность к кластеру, на основе предыдущих таблиц. Также получим таблицу, которая содержит данные по продуктовым группам в разрезе относительно данных по всем продуктовым группам. Все эти шаги будут подробно рассмотрены в дальнейшем. Данный этап выполняется в скрипте с названием Create\_Main\_Metric\_Table\_For\_Clients(Step№6.2).py.

После создание хранимой процедуры, можно приступить к созданию таблиц временной динамики. Создание таких таблиц являются частью алгоритма прописанного в скрипте Create\_Main\_Metric\_Table\_For\_Clients(Step№6.2).py. Такие таблицы создаются не только относительно продуктовых групп в отдельности, но и относительно всех продуктовых групп, а имена таблиц будут иметь следующий формат: Count\_Of\_Trans{0}\_For\_Months\_Table, где {0} – наименование продуктовой группы, если она есть.

Приведём часть результирующих таблиц данного шага. Количество появившихся таблиц равняется количеству продуктовых групп, указанных в поле GROUPS конфигурационного файла, и плюс таблица, включающая данные относительно всех продуктовых групп. Результаты изображены на рисунках 2.6.1 и 2.6.2, таблицы dbo.Count\_Of\_Trans\_For\_Months\_Table(по всем продуктовым группам) и dbo.Count\_Of\_Trans\_Food\_For\_Months\_Table(по группе Food в отдельности) соответственно:

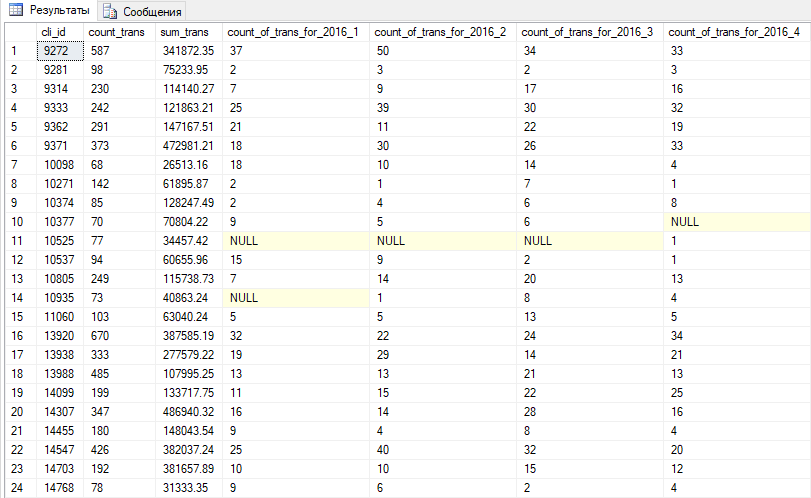


Рисунок 2.6.1 – Таблица dbo.Count\_Of\_Trans\_For\_Months\_Table

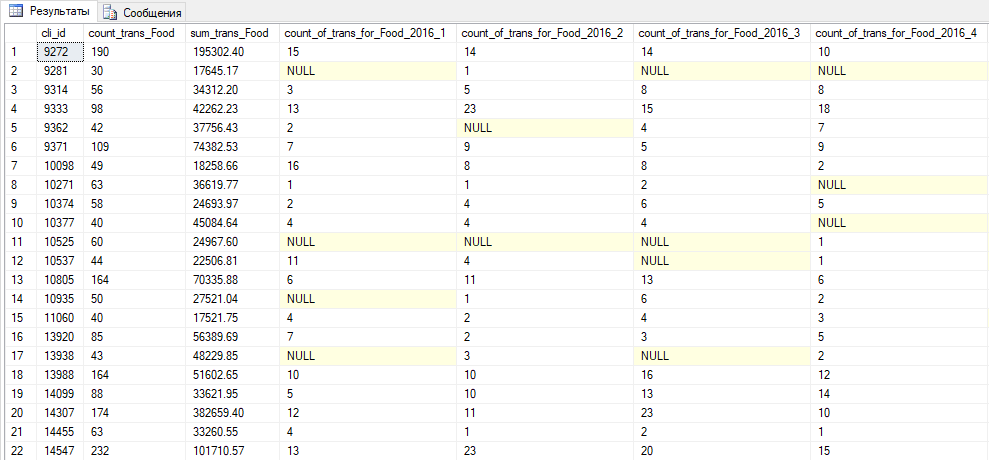


Рисунок 2.6.2 – Таблица dbo.Count\_Of\_Trans\_Food\_For\_Months\_Table

В данном случае, поскольку анализ проводится в двух продуктовых группах: Food и Medical, то создались три таблицы:

1. dbo.Count\_Of\_Trans\_For\_Months\_Table(по всем продуктовым группам)
2. dbo.Count\_Of\_Trans\_Food\_For\_Months\_Table(по группе Food)
3. dbo.Count\_Of\_Trans\_Medical\_For\_Months\_Table(по группе Medical)

Как видно по рисункам 2.6.1 и 2.6.2, помимо столбцов количества транзакций за месяцы присутствуют агрегированные столбцы количества транзакций за весь период и их общая сумма. В данном случае, с учётом выбранного временного интервала, столбцы отображают временную динамику на протяжении 17 месяцев.

Теперь, когда есть таблицы содержащие данные о динамике изменений, следующим шагом на основе этих показателей будет создание таблиц с сгенерированными метриками-вариациями, такими как: математическое ожидание, дисперсия, среднеквадратичное отклонение, коэффициент вариации, а также принадлежность к кластеру после процесса кластеризации. Данные таблицы будут иметь название следующего формата: Metric\_Table{0}\_Extended, где {0} – наименование продуктовой группы, если она есть.

Стоит уделить внимание метрикам и немного окунуться в курс математической статистики и теории вероятности. Теория вероятности изучает случайные величины. Для этого строятся различные характеристики, которые описывают их поведение. Математическое ожидание является одной из основных характеристик случайной величины являющееся своего рода центром, вокруг которого группируются остальные значения.

Формула мат.ожидания имеет следующий вид:

где – математическое ожидание, – это случайные величины, – их вероятности.

То есть, математическое ожидание случайной величины — это взвешенная сумма значений случайной величины, где веса равны соответствующим вероятностям. Математическая статистика предоставляет несколько вариантов оценки математического ожидания. Основное среди них – среднее арифметическое, обладающие рядом полезных свойств. Например, среднее арифметическое – это несмещенная оценка, т.е. мат.ожидание средней равно оцениваемому математическому ожиданию. Для нашего случая мат.ожидание будет являться средней арифметической количества транзакций за рассматриваемый период.

Следующей рассматриваемой метрикой является дисперсия случайной величины. Это очень значимый показатель, который используется в различных методах математической статистики (анализ причинно-следственных связей, проверка гипотез и др.). Дисперсия отражает меру разброса данных вокруг средней величины.

Как и мат.ожидание, дисперсия является важной характеристикой случайной величины. Если мат.ожидание отражает центр случайной величины, то дисперсия дает характеристику разброса данных вокруг этого центра.

В теории вероятности формула дисперсии имеет следующий вид:

То есть дисперсия – это математическое ожидание отклонений от математического ожидания.

При анализе выборок на практике математическое ожидание, как правило, не известно. Поэтому вместо него используют оценку – среднее арифметическое.

где - выборочная дисперсия, рассчитанная по данным наблюдениям, – отдельное рассматриваемое значение, среднее арифметическое по выборке.

Стоит отметить, что у такого способа расчета дисперсии есть недостаток – она получается смещенной, т.е. ее мат.ожидание не равно истинному значению дисперсии. Но ситуация исправляется при увеличении объема выборки, она все-таки приближается к своему теоретическому аналогу, т.е. является асимптотически не смещенной. Поэтому при работе с большими размерами выборок как у нас можно использовать формулу выше.

Получается, что дисперсия - это средний квадрат отклонений. То есть вначале рассчитывается среднее значение, затем берется разница между каждым рассматриваемым исходным и средним значением, разница возводится в квадрат, суммируется и затем делится на количество наблюдений в данной совокупности. Разница между отдельным рассматриваемым значением и средней отражает меру отклонения. В квадрат возводится только для того, чтобы все отклонения стали исключительно положительными числами, и чтобы избежать их взаимоуничтожения при суммировании отрицательных и положительных отклонений. Затем, имея сумму квадратов отклонений, просто рассчитываем среднюю арифметическую.

Однако в чистом виде, как, например, средняя арифметическая, дисперсия не используется. Это скорее вспомогательный и промежуточный показатель, который необходим для других видов метрик статистического анализа.

Следующая метрика – среднеквадратичное отклонение. Дабы использовать дисперсию в более приземленных целей, из нее извлекают квадратный корень. Получается так называемое среднеквадратичное(стандартное) отклонение.

Формула стандартного отклонения имеет следующий вид:

Для получения показателя по выборке используется формула:

Стандартное отклонение, также характеризует меру рассеяния данных, но теперь (в отличие от дисперсии) его можно сравнивать с исходными данными, так как единицы измерения у них одинаковые. Но и этот показатель в чистом виде не очень информативен, так как в нем заложено слишком много промежуточных расчетов, которые сбивают с толку. Тем не менее, со стандартным отклонением уже можно работать непосредственно, потому что свойства данного показателя хорошо изучены и известны. К примеру, есть такое правило трех сигм, которое гласит, что у нормально распределенных данных 997 значений из 1000 находятся в пределах ±3 сигмы от средней арифметической. Стандартное отклонение, как мера неопределенности, также участвует во многих статистических расчетах. С ее помощью устанавливают степень точности различных оценок и прогнозов. Если вариация очень большая, то стандартное отклонение тоже получится большим, следовательно, и прогноз будет неточным, что выразится, к примеру, в очень широких доверительных интервалах.

И напоследок остаётся коэффициент вариации. Стандартное отклонение дает абсолютную оценку меры разброса. Поэтому чтобы понять, насколько разброс велик относительно самих значений (т.е. независимо от их масштаба), требуется относительный показатель. И этим показателем называется коэффициент вариации**.**

Коэффициент вариации рассчитывается по следующей формуле:

где – коэффициент вариации, – стандартное отклонение, – среднее арифметическое по выборке.

Коэффициент вариации измеряется в процентах (если умножить на 100%). По этому показателю сравнивается однородность самых разных явлений независимо от их масштаба и единиц измерения. Это и делает коэффициент вариации столь популярным.

В статистике принято, что, если значение коэффициента вариации менее 33%, то совокупность считается однородной, если больше 33%, то – неоднородной. Это считается аксиомой.

Зачем нам нужно рассчитывать эти метрики? Посчитав эти вариации, особенно коэффициент вариации, мы сможем определять стабильность поведения клиентов. Посчитав коэффициент вариации по выборке количества транзакций во временной динамике, можно сказать стабильно ли клиент совершает транзакции или нет, или другими словами насколько он стабилен? Ведь в зависимости от этого фактора банк или мерчант могут предложить клиенту участие в какой-то маркетинговой кампании, карту с бонусами и т.д.. В таком случае у нас получается два вида клиентов: относительно стабильный и относительно нестабильный. Почему относительно? Потому что большинство клиентов будут иметь коэффициент вариации больше 33%, то есть основная масса является нестабильной. В такой ситуации можно смягчить нижнюю границу, повысив её, если это устраивает банк. Насколько повысить границу решает также банк.

После генерирование метрик, можно провести кластеризацию клиентов на основе метрик. Для начала, что такое кластеризация и зачем она нужна в нашем анализе?

Кластеризация (или кластерный анализ) — это задача разбиения множества объектов на группы, называемые кластерами. Внутри каждой группы должны оказаться «похожие» объекты, а объекты разных группы должны быть как можно более отличны.

Применение кластерного анализа в общем виде сводится к следующим этапам:

1. Отбор выборки объектов для кластеризации.
2. Определение множества признаков, по которым будут оцениваться объекты в выборке. При необходимости можно нормализовать значения признаков.
3. Вычисление значений меры сходства между объектами.
4. Применение метода кластерного анализа для создания групп сходных объектов (кластеров).
5. Представление результатов анализа.

После получения результатов анализа возможна корректировка выбранной метрики и метода кластеризации до получения оптимального результата.

Как же определяется схожесть объектов? Для начала необходимо составить вектор характеристик(признаков) для каждого объекта — как правило, это набор числовых значений, например, рост-вес человека. Однако существуют также алгоритмы, работающие с качественными характеристиками.

После того, как мы определили вектор характеристик, при необходимости можно провести нормализацию, чтобы все компоненты давали одинаковый вклад при расчете расстояния. В процессе нормализации все значения сводятся к некоторому диапазону, например, [-1, 1] или [0, 1].

Наконец, для каждой пары объектов измеряется расстояние между ними — степень похожести. Существует несколько способов высчитывания расстояния между парой объектов. Но приведём только один из них:

Евклидово расстояние - наиболее распространенная функция расстояния. Представляет собой геометрическое расстояние в многомерном пространстве:

где и – это координаты пар объектов по определенной оси.

Геометрическая интерпретация алгоритма изображена на рисунке 2.6.3:

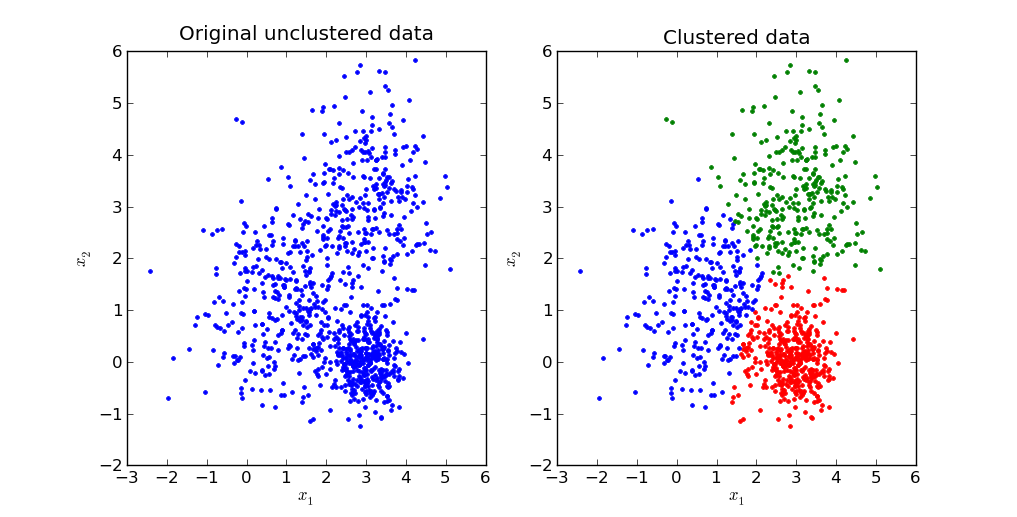


Рисунок 2.6.3 – Картинка слева – множество точек. Картинка справа – образованные кластеры в результате кластеризации

Алгоритмы кластеризации имеют многообразную классификацию. Но не будем углубляться в данную тему. Скажем только, что для нашего анализа был выбран алгоритм k-Means(k - Средних). Расскажем о нём подробнее.

Алгоритм k-Means кластеризует данные, пытаясь разделить выборку на n групп равной дисперсии, минимизируя критерий, известный как инерция или внутрикластерная сумма квадратов. Этот алгоритм требует указания количества кластеров. Алгоритм хорошо масштабируется на большом количестве наблюдений и был использован в широком диапазоне областей применения в различных сферах.

Алгоритм k-Means делит множество выборки X на K непересекающихся кластеров, каждый из которых описывается средним значением выборки в -ом кластере. Значение обычно называют центроидом кластера; стоит обратить внимание, что они, как правило, не являются точками из выборки X, хотя они живут в одном и том же пространстве. Алгоритм k-Means предназначен для выбора центроидов, минимизирующих инерцию, или критерий внутрикластерной суммы квадратов:

Инерцию, или критерий суммы квадратов внутри кластера, можно распознать как меру того, насколько внутренне кластеры когерентны(непротиворечивы). Алгоритм может страдать от различных недостатков:

Инерция делает предположение, что кластеры выпуклые и изотропные, что не всегда так. Плохо реагирует на удлиненные скопления или многообразия неправильной формы.

Инерция не является нормализованной метрикой: мы просто знаем, что более низкие значения лучше, а ноль оптимален. Но в очень многомерных пространствах Евклидовы расстояния имеют тенденцию раздуваться (это пример так называемого «проклятия размерности»). Запуск алгоритма уменьшения размерности такого как PCA до кластеризации, может облегчить эту проблему и ускорить вычисления.

k-Means часто называют алгоритмом Ллойда. В основных терминах алгоритм состоит из трех шагов. На первом шаге выбираются начальные центроиды, причем самым простым методом является выбор K выборок из набора данных X. После инициализации, k-Means проходит цикл между двумя шагами. На первом шаге каждому образцу присваивается ближайший центроид. Второй шаг создает новые центроиды, принимая среднее значение всех выборок, назначенных каждому предыдущему центроиду. Разница между старым и новым центроидами вычисляется, и алгоритм повторяет эти последние два шага, пока это значение не будет меньше порога. Другими словами, он повторяется до тех пор, пока центроиды не будут удовлетворять определённой ошибке или не перестанут двигаться.

k-Means эквивалентен алгоритму максимизации ожиданий с небольшой полностью равной диагональной ковариационной матрицей.

Алгоритм также может быть интерпретирован через концепцию диаграмм Вороного. Сначала в диаграмме Вороного точки вычисляются с использованием текущих центроидов. Каждый сегмент на диаграмме Вороного становится отдельным кластером. Во-вторых, центроиды обновляются до среднего значения каждого сегмента. Затем алгоритм повторяет это до тех пор, пока не будет выполнен критерий остановки. Обычно алгоритм останавливается, когда относительное уменьшение целевой функции между итерациями меньше заданного значения допуска.

При достаточном времени k-Means всегда будет сходиться, однако результат может оказаться локальным минимумом. Это сильно зависит от первоначального расположения центроидов. Поэтому вычисление часто выполнятся по нескольку раз с различными инициализациями центроидов. Одним из способов решения этой проблемы является схема k-Means++. Благодаря ей центроиды как можно дальше удаляют друг от друга, что приводит к доказуемо лучшим результатам, чем случайная инициализация центроидов.

Для кластеризации используются модули NumPy и Scikit-learn

NumPy — это расширение языка Python, добавляющее поддержку матриц и больших многомерных массивов, вместе с большой библиотекой высокоуровневых математических функций для операций с этими матрицами и массивами. Использование NumPy в проекте необходимо для преобразования формата входных характеристик, подающихся на кластеризацию. Всё дело в том, что функции библиотеки Scikit-learn принимают на вход массивы модуля NumPy.

Scikit-learn - бесплатная библиотека машинного обучения для языка программирования Python. Она имеет различные алгоритмы классификации, регрессии и кластеризации, включая алгоритмы опорных векторов, случайные леса, градиентный спуск, K-means и DBSCAN, и предназначена для взаимодействия с библиотеками Python NumPy и SciPy.

Для чего необходима кластеризация? У автора возникла гипотеза в полезности разделение клиентов на кластеры. Таким образом у нас получатся группы клиентов, имеющих общие поведение в совершение транзакции. Сформировав «похожих» клиентов по группам, банк или мерчантат могут выстроить единую стратегию по взаимоотношению со всеми клиентами в группе. Как было сказано в описании для метода k-Means нужно заранее определить количество кластеров. Обычно число кластеров находится или опытным путём, или знанием специфики, особенностей и нюансов данной предметной области, на основе которых можно выбрать какое-либо оптимальное число. Часто оба способа совмещают. В ходе опытов и умозаключений автором было предложено поделить клиентов на четыре следующих кластера:

1. Клиенты с малым количеством транзакций, совершённые на малые суммы
2. Клиенты с малым количеством транзакций, совершённые на большие суммы
3. Клиенты с большим количеством транзакций, совершённые на малые суммы
4. Клиенты с большим количеством транзакций, совершённые на большие суммы.

Из следующей классификации за основу характеристик кластеризации можно выбрать следующие признаки: мат.ожидание количества транзакций и мат.ожидание суммы транзакций. Метрики-вариации, о которых было сказано выше, генерируются перед кластеризацией. Передав на вход функции кластеризации матрицу признаков для всех клиентов, алгоритм совершит кластеризацию.

Стоит пояснить реализацию этого этапа. Участок кода отвечающий за кластеризацию:

kmeans = KMeans(n\_clusters=4, n\_init=25, max\_iter=2000)

kmeans.fit(np.array(signs\_of\_clustering))

labels = list(kmeans.labels\_)

Обратившись к документации по модулю Scikit-learn, узнаем значение этих строк. В первой строке создаётся экземпляр класса кластеризации. На вход конструктор принимает следующие аргументы:

1. n\_clusters – количество создаваемых кластеров, а также количество центроидов для генерации. Опционален, принимает значения типа int, по умолчанию n равняется 8
2. n\_init – количество попыток начального расположение центроидов для алгоритма. Выполнив n раз алгоритм, на выходе вернутся результаты лучшей попытки. Выше было сказано, что начальное расположение центроидов значительно влияет на результаты. Аргумент принимает значения типа int. Значение по умолчанию равно 10
3. max\_iter ­– максимальное число итераций алгоритма k-Means за одну попытку. Аргумент принимает значения типа int. Значение по умолчанию равно 300.

Во второй строке у экземпляра вызывается метод fit, с передачей ему необходимых признаков клиентов, которые содержатся в матрице signs\_of\_clustering, преобразованной в формат массива модуля Numpy, с помощью np.array. В третьей строке переменной labels присваиваются результаты алгоритма, преобразованные в список. После этого в переменной labels хранятся метки принадлежности к кластеру для каждого клиента.

Этот этап выполняется для всех таблиц вида Count\_Of\_Trans{0}\_For\_Months\_Table(где {0} – наименование продуктовой группы), описанных выше.

После того как были сгенерированы метрики вариаций, и проведена кластеризация, можно создать таблицы, которые будут хранить все эти значения. Как уже было выше сказано, таблицы будут иметь названия следующего формата: Metric\_Table{0}\_Extended, где {0} – это продуктовая группа, если она имеется.

Часть созданных таблиц изображены на рисунках 2.6.4 и 2.6.5:

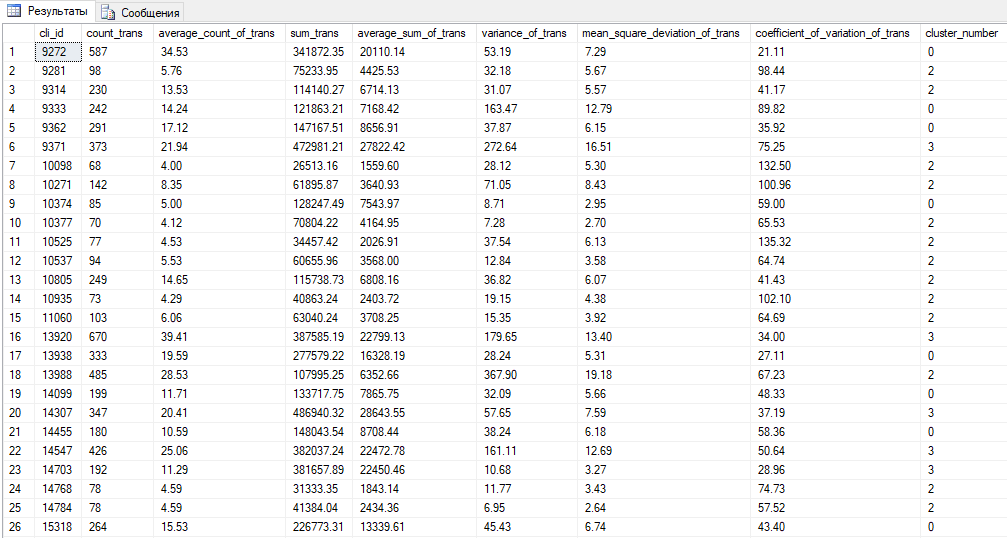


Рисунок 2.6.4 – Таблица dbo.Metric\_Table\_Extended

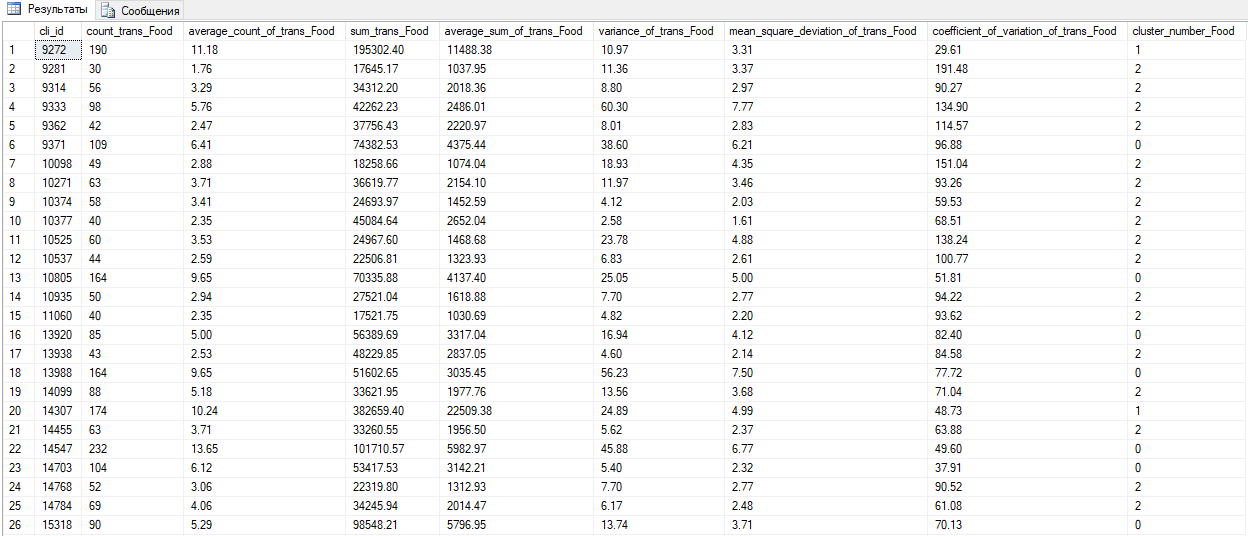


Рисунок 2.6.5 – Таблица dbo.Metric\_Table\_Food\_Extended

В данном случае было создано три таблицы:

1. dbo.Metric\_Table\_Extended(по всем продуктовым группам)
2. dbo.Metric\_Table\_Food\_Extended(по группе Food)
3. dbo.Metric\_Table\_Mediacl\_Extended(по группе Medical)

Имеет смысл объединить значения этих таблиц по клиентам(столбец cli\_id). Таким образом можно будет наглядно увидеть данные в разрезе по продуктовым группам. Это поможет взглянуть на данные с другой стороны, и, возможно, выдвинуть новые гипотезы. Результирующая таблица этого подпункта изображена на рисунках 2.6.6 и 2.6.7:

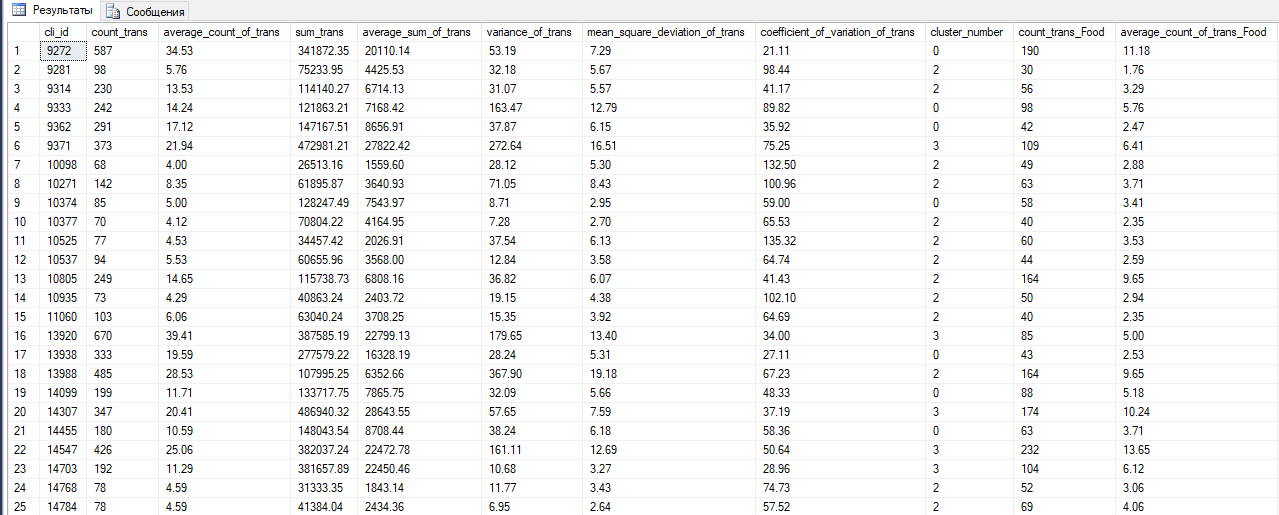


Рисунок 2.6.6 – Таблица dbo. Main\_Metric\_Table\_For\_Clients

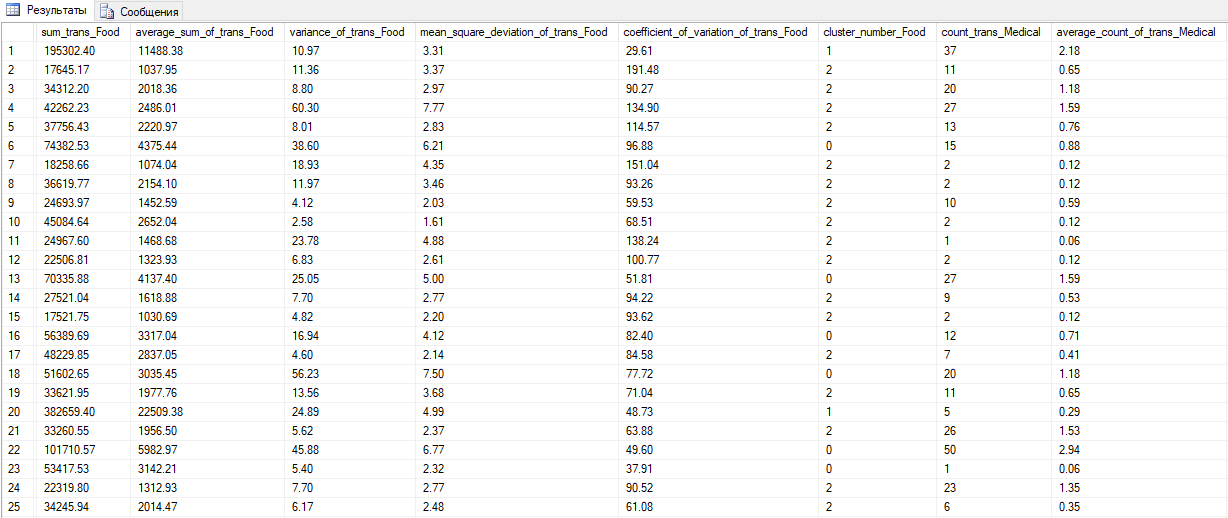


Рисунок 2.6.7 – Таблица dbo. Main\_Metric\_Table\_For\_Clients(продолжение)

Анализ со стороны клиентов завершён.

# 3 АНАЛИЗ СО СТОРОНЫ МЕРЧАНТОВ

В данном разделе будет произведена агрегация данных за рассматриваемый временной период относительно мерчантов, формирования таблиц временной динамики в разрезе продуктовых групп относительно мерчантов, генерирования метрик, кластеризация и формирования таблиц метрик в разрезе продуктовых групп относительно мерчантов. Как видите, данный этап мало чем отличается от этапа анализа со стороны клиентов, за исключением того, что все этапы будут проводиться относительно мерчантов. Приступим к пошаговому выполнению.

# 3.1 Агрегация данных за период относительно мерчантов и избавление от аутлаеров

Сначала проведём агрегацию данных за временной период относительно мерчантов. Данный этап выполняет скрипт Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Clients\_On\_Merchants(Step№7.1).py. Результатом выполнения скрипта будет таблица dbo.Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Clients\_On\_Merchants. Таблица изображена на рисунке 3.1.1:

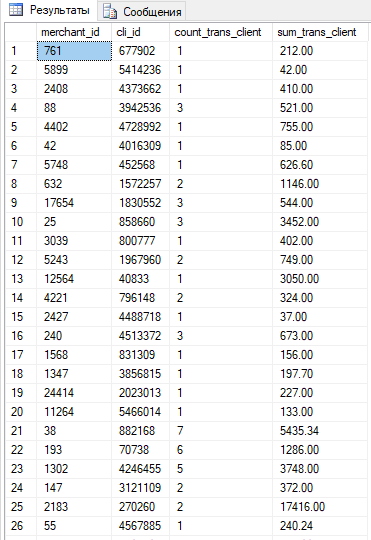
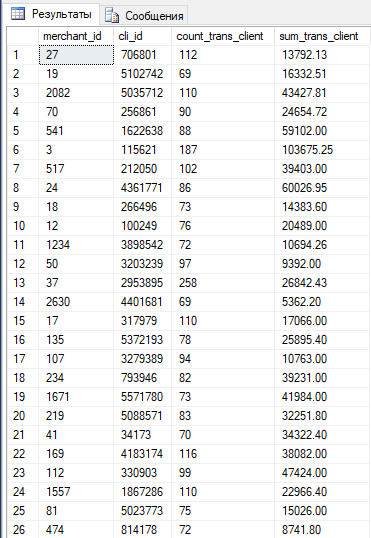


Рисунок 3.1.1 – Таблица dbo.Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Clients\_On\_Merchants

Данная таблица была сформирована на основе таблицы dbo.Starting\_Table\_For\_The\_Period(таблица, которая содержит записи за временной интервал, указанный в конфигурационном файле). Как было сказано, таблица агрегирует в себе данные относительно мерчантов, на самом деле, это не совсем так. Дело в том, что таблица представляет собой агрегацию данных относительно мерчантов и клиентов(столбец cli\_id), которые принадлежат данному мерчанту. Столбцы count\_trans\_client и sum\_trans\_client, как видно из названия, агрегированы по клиенту. Другими словами, таблица показывает клиента, количество транзакций клиента, сумма транзакций клиента, принадлежность клиента к определённому мерчанту. В этой таблице, как одному мерчанту могут принадлежать множество клиентов, так и один клиент может принадлежать множеству мерчантов.

Как видно из рисунка 3.1.1 мерчанту могут принадлежать клиенты, совершившие за всё время одну-две транзакции. Следует сказать, что данные записи характеризует клиентов как или случайных, или непостоянных. Оба случая не удовлетворяет нашему принципу и подходу, основанному на количестве и содержательности операций. Поэтому от таких записей следует избавиться из-за недостатка информации. Здесь руководствуемся той же логикой, которой придерживались на этапе анализа со стороны клиента. Мы должны оставить только те записи, у которых количество транзакций удовлетворяют нижней границе. Нижняя граница, как и на этапе с клиентами, формируется следующим образом: 4 транзакции(по одной транзакции в неделю) \* количество месяцев. В результате останутся записи о клиентах, про которых можно сказать, что по отношению к определённому мерчанту он является постоянным. Данный принцип исходит из того, что мерчант готов проводить маркетинговые кампании или предлагать бонусные программы клиентам, которых он считает постоянными. Избавление от аутлаеров(т.е. от непостоянных клиентов) происходит в скрипте Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Correct\_Clients\_On\_Merchants(Step№7.2).py. Результат изображён на рисунке 3.1.2:



3.1.2 – Таблица dbo. Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Correct\_Clients\_On\_Merchants

После того как мы избавились от аутлаеров, нужно учесть это в общей таблице транзакций. Для этого найдём пересечение у таблиц dbo.Starting\_Table\_For\_The\_Period и dbo. Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Correct\_Clients\_On\_Merchants по столбцам merchand\_id и cli\_id. Шаг выполняется в скрипте Transaction\_Table\_Of\_Correct\_Clients\_On\_Merchants(Step№7.3).py. Результат изображён на рисунке 3.1.3, таблица dbo.Transaction\_Table\_Of\_Correct\_Clients\_On\_Merchants:

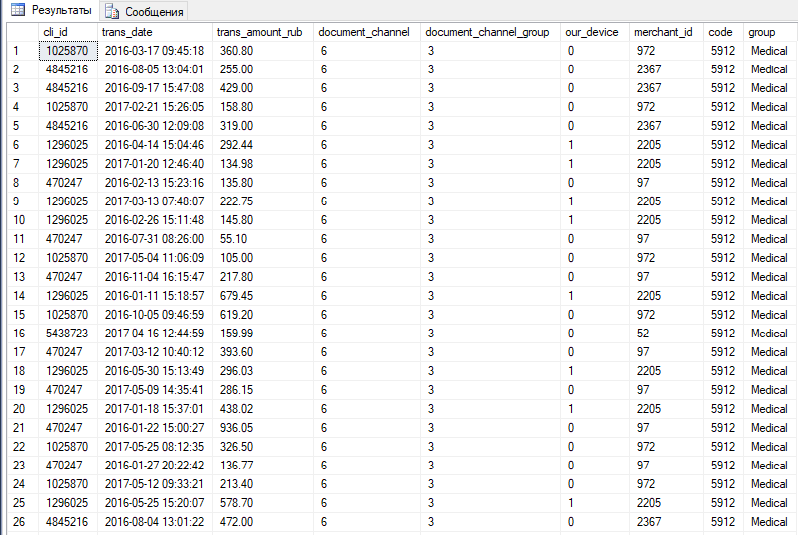


Рисунок 3.1.3 – dbo.Transaction\_Table\_Of\_Correct\_Clients\_On\_Merchants

# 3.2 Метрики и кластеризация мерчантов

Чтобы сгенерировать метрики вариации и провести кластеризацию по мерчантам, необходимо создать таблицы, в которых прослеживается совершение операций во временной динамике. Те же таблицы, что были в клиентском анализе, только в данном случае анализировать будем относительно мерчантов.

Для того чтобы создать таблицы, в которых прослеживается временная динамика совершённых операций, удобно было бы иметь вспомогательную процедуру, которая агрегировала количество транзакций у мерчанта за определённый месяц. Создадим хранимую процедуру. Создаёт процедуру скрипт Create\_Procedure\_CountOfTransForMonthOnMerchants(Step№8.1).py.

После создания процедуры, можно приступить к созданию таблиц временной динамике. Не будем заострять внимание на этом моменте, поскольку подобные шаги выполнялись при анализе со стороны клиентов. Скажем, что данные операции выполняет часть скрипта Create\_Metric\_Tables\_On\_Merchants\_Extended(Step№8.2).py, таблицы имеют название формата Count\_Of\_Trans{0}\_On\_Merchants\_For\_Months\_Table, где {0} – продуктовая группа, если она есть. Стоит напомнить, что анализ со стороны мерчантов проводиться относительно тех же продуктовых групп, что и в анализе со стороны клиентов, дабы потом найти пересечение клиентов и мерчантов в этих продуктовых группах. В результате создаются три таблицы:

1. dbo.Count\_Of\_Trans\_On\_Merchants\_For\_Months\_Table – по всем продуктовым группам
2. dbo.Count\_Of\_Trans\_Food\_On\_Merchants\_For\_Months\_Table ­– по группе «Food» в отдельности
3. dbo.Count\_Of\_Trans\_Medical\_On\_Merchants\_For\_Months\_Table – по группе «Medical» в отдельности

Две из выше перечисленных таблиц изображены на рисунках 3.2.1 и 3.2.2: dbo.Count\_Of\_Trans\_On\_Merchants\_For\_Months\_Table и dbo.Count\_Of\_Trans\_Food\_On\_Merchants\_For\_Months\_Table соответственно:

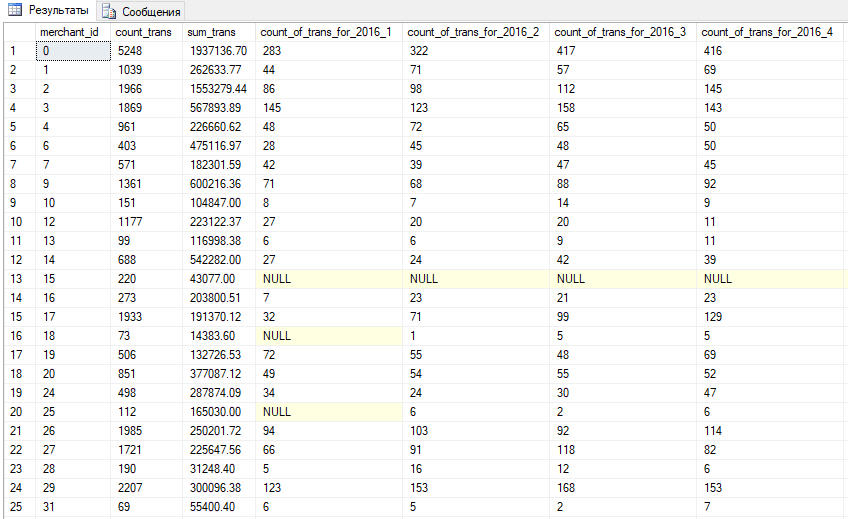


Рисунок 3.2.1 – Таблица dbo.Count\_Of\_Trans\_On\_Merchants\_For\_Months\_Table

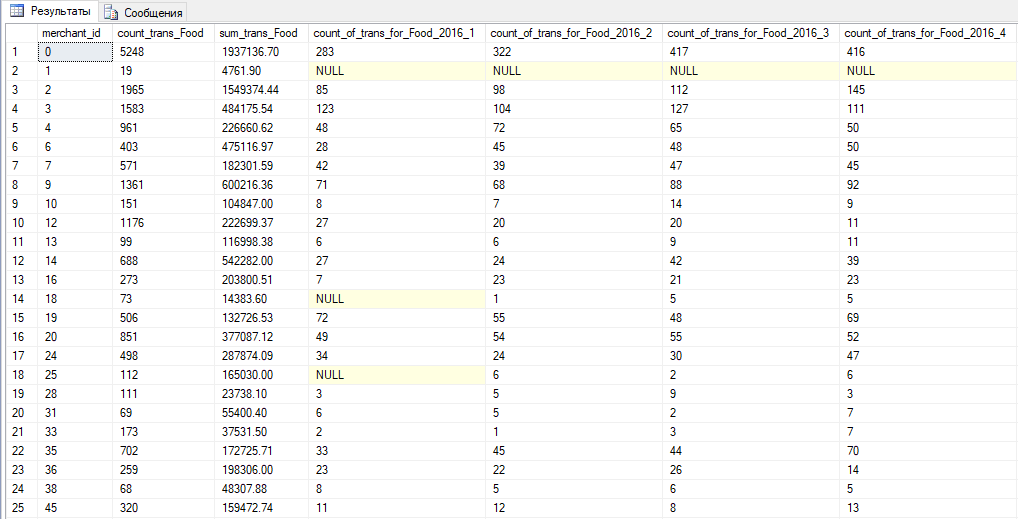


Рисунок 3.2.2 – dbo.Count\_Of\_Trans\_Food\_On\_Merchants\_For\_Months\_Table

Как видно на рисунках, помимо столбцов временной динамики, присутствуют агрегированные столбцы: количество транзакций за весь период и сумма транзакций за весь период.

Следующий шаг генерирование метрик на основе столбцов временной динамики с последующей кластеризацией. Напомним, что сгенерируются следующие метрики: мат.ожидание по количеству транзакций, мат.ожидание по сумме транзакций, дисперсия по количеству операций, среднеквадратичное отклонение по количеству операций, коэффициент вариации по количеству операции. Кластеризация будет происходить с расчётом на четыре группы мерчантов:

1. Мерчанты, у которых малое число транзакций на малые суммы
2. Мерчанты, у которых малое число транзакций на большие суммы
3. Мерчанты, у которых большое число транзакций на малые суммы
4. Мерчанты, у которых большое число транзакций на большие суммы

Данные шаги также выполняются в скрипте Create\_Metric\_Tables\_On\_Merchants\_Extended(Step№8.2).py. Названия созданных таблиц будет следующего формата: Metric\_Table{0}\_On\_Merchants\_Extended, где {0} – продуктовая группа, если она есть. В данном случае создаются три таблицы, две из ни изображены на рисунках 3.2.3 и 3.2.4, dbo.Metric\_Table\_On\_Merchants\_Extended и dbo.Metric\_Table\_Food\_On\_Merchants\_Extended:

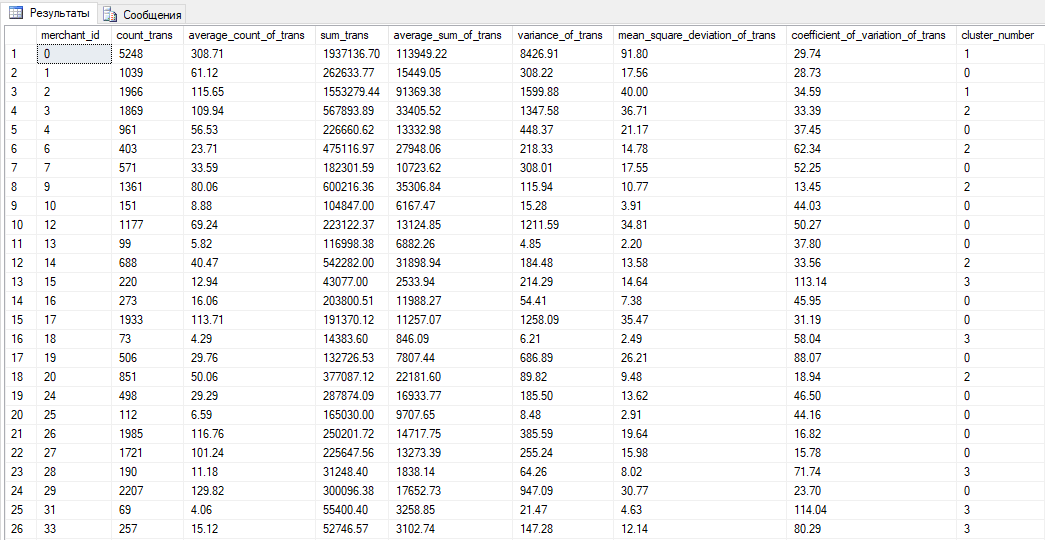


Рисунок 3.2.3 – Таблица dbo.Metric\_Table\_On\_Merchants\_Extended

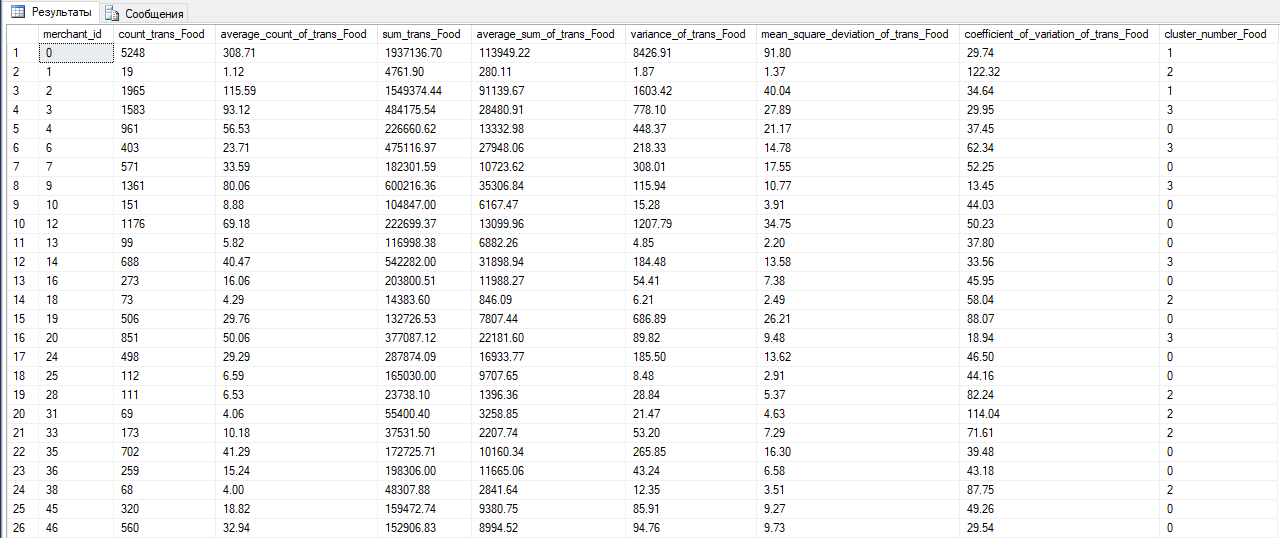


Рисунок 3.2.4 – Таблица dbo.Metric\_Table\_Food\_On\_Merchants\_Extended

# 3.3 Пересечение клиентов и мерчантов

Заключительным шагом анализа является пересечение клиентов и мерчантов, что по сути представляет собой таблицы сопоставления мерчантов с клиентами. Пересечение двух сущностей и создание таблиц в разрезе по продуктовым группам выполняется в скрипте Create\_Comparison\_Tables(Step№9).py. Таблицы имеют название формата Create\_Comparison\_Tables{0}, где {0} – продуктовая группа, если она есть. В данном случае, таких таблиц будет три:

1. dbo.Comparison\_Table – таблица сопоставления по всем продуктовым группам
2. dbo.Comparison\_Table\_Food – таблица сопоставления по группе «Food»
3. dbo.Comparison\_Table\_Medical – таблица сопоставления по группе «Medical».

Две из трёх таблиц изображены на рисунках 3.3.1 и 3.3.2, dbo.Comparison\_Table и dbo.Comparison\_Table\_Food соответственно:

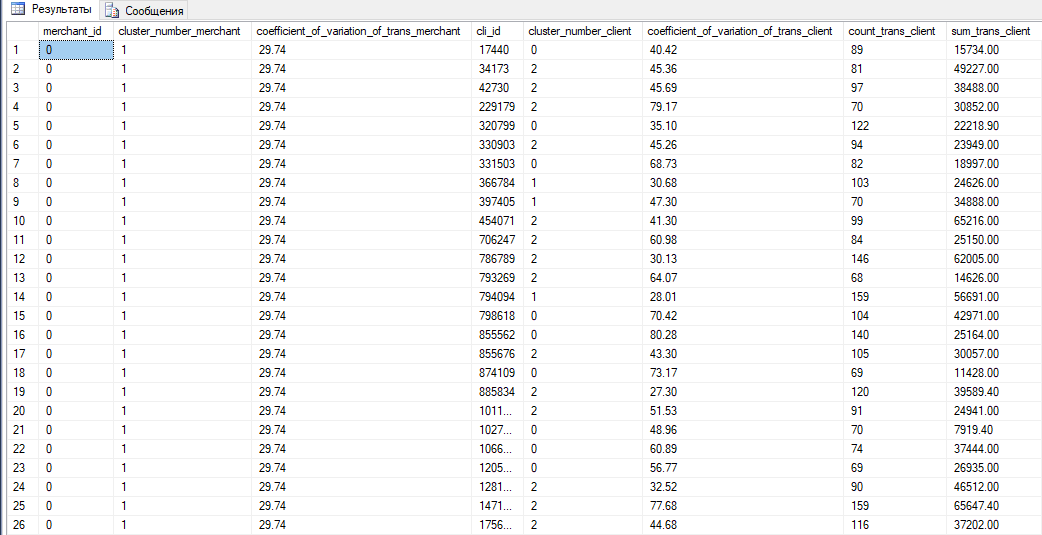


Рисунок 3.3.1 – Таблица dbo.Comparison\_Table

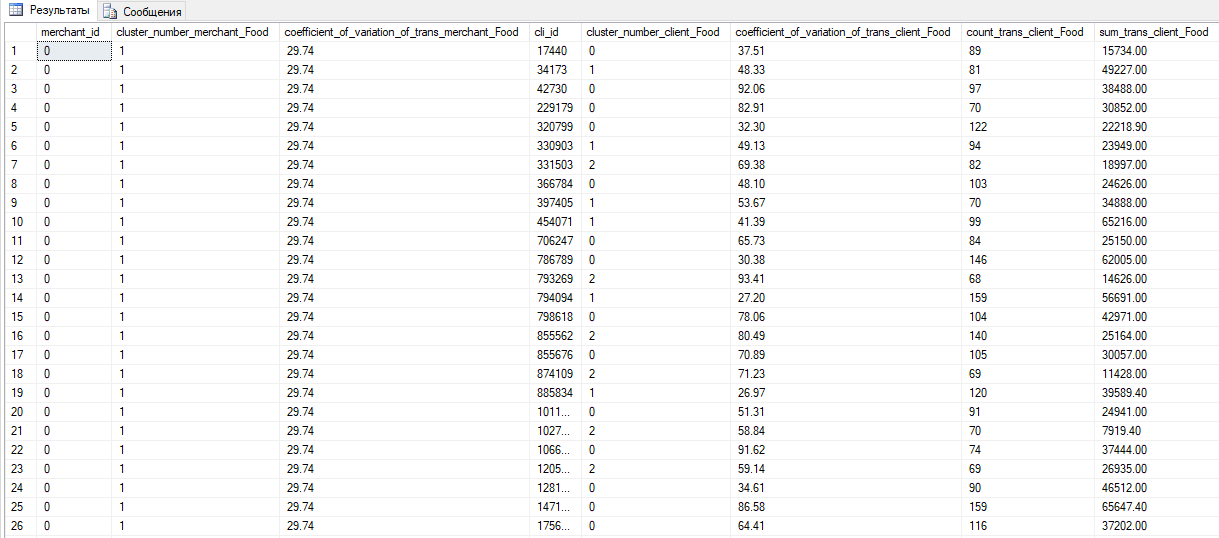


Рисунок 3.3.2 – Таблица dbo.Comparison\_Table\_Food

Стоит объяснить структуру данных таблиц и их содержимое, поскольку данные наборы являются заключительным результатом всей проведенной работы. Рассмотрим на примере dbo.Comparison\_Table. Данная таблица имеет столбец идентификатора мерчанта(merchant\_id), принадлежность мерчанта к одному из четырёх кластеров рассмотренных выше(cluster\_number\_merchant), коэффициент вариации мерчанта по транзакциям(coefficient\_of\_variation\_of\_trans\_merchant), идентификатор клиента(cli\_id), принадлежность клиента к одному из четырёх кластеров рассмотренных выше(cluster\_number\_client), коэффициент вариации клиента по транзакциям(coefficient\_of\_variation\_of\_trans\_client), количество транзакций клиента у определённого мерчанта за всё время(count\_trans\_client), сумма транзакций клиента у определённого мерчанта за всё время(sum\_trans\_client). Если таблица является разрезом относительно какой-то продуктовой группы, тогда все выше описанные столбцы будут браться из таблиц с метриками соответствующих продуктовых групп.

Как можно интерпретировать данный результат? Мерчанту, принадлежащий к одной из кластерных групп по мерчантам, соответствует клиент, принадлежащий к одной из кластерных групп по клиентам. На основе этих соответствий банк можно разработать стратегии и компании по взаимодействию как с мерчантом, так и с клиентом, или стать инициативным посредником по общему взаимодействию двух сущностей. Мерчант по отношению к клиенту использует относительно две стратегии: стратегия лояльности, когда мерчант пытается удержать у себя клиентов, чтобы они не перешли к конкуренту, и стратегия захвата, когда мерчант расширяет свою клиентскую базу, в том числе за счёт клиентов конкурента. Пример из стратегии лояльности: банк, основываясь на данных соответствиях может договориться с мерчантами(состоящих в определённой кластерной группе) на кэшбэк от покупок клиентов(состоящих в определённой кластерной группе), тем самым удержав у себя клиента. Пример из стратегии захвата: банк может завлекать клиентов(состоящих в определённой кластерной группе), у которых есть любимые места определённой тематики или наиболее востребованной продуктовой группы, к определённому мерчанту, который является представителем тематики или продуктовой группы, посредством смс-сообщений или телефонных звонков.

Подобных кейсов может быть достаточно много, и в каждом из них банку будет отходить определённый процент прибыли. В данной работе не описываются подобные схемы, потому что этими вопросами занимаются специализированные отделы. В рамках нашей работы проходит только анализ данных и интерпретация полученных моделей, c которыми работают другие подразделения банка. Поэтому снова вернёмся к данным.

Но всё ещё остаётся вопрос: как интерпретировать кластеры, какую определённую группу они характеризуют? Сразу на этот вопрос ответить затруднительно, стоит потратить некоторое время на изучение кластеров и их представителей. Для облегчения этой задачи рекомендуется использовать графические инструменты представления данных, таких как: Microsoft Power Bi, библиотеки Python для визуализации данных и т.д. Можно попробовать изучить кластеры «в лоб», сверяя клиента и его признаки, на основе которых проводилась кластеризация, с принадлежностью к кластеру. Стоит сказать, что после каждого перезапуска модели, стоит заново интерпретировать кластеры, поскольку процесс кластеризации основывается на случайном расположение первоначальных центроидов, а значит номера среди кластеров каждый раз будут распределяться по-разному.

Про коэффициенты вариации. Они подсчитаны, но пока нигде не используются, хотя это мощный признак, который характеризует поведение сущности в частоте совершения транзакций. Как было сказано, коэффициент вариации в частности отвечает на вопрос «стабильно или не стабильно поведение сущности?». Как вариант, в дальнейшем коэффициент вариации можно использовать как нижнюю границу, которая поделит сущности на стабильных и нестабильных. Сущности, которые расположены ниже относительной границы, считаются стабильными, сущности, которые находятся выше относительной границы, считаются нестабильными или хаотичными. Учитывая, что у нас проведена кластеризация по сущностям, то введя такую границу в виде коэффициента вариации, каждая наша кластер-группа разделится ещё на две подгруппы, допустим: группа, клиенты которых стабильно совершают большое число транзакций на большие суммы, и группа, клиенты которых нестабильно совершают большое число транзакций на большие суммы. Возможно стоит ввести столбец бинарных значений, который будет отвечать за принадлежность к стабильным и нестабильным категориям: 0 – нестабилен, 1 – стабилен.

Также у нас остались столбцы с количеством и суммой транзакций каждого клиента. Эти столбцы были внесены в результирующие наборы, чтобы их можно было использовать в качестве дополнительных фильтров, а также для наглядности классификации клиента.

# ЗАКЛЮЧЕНИЕ

В данной работе был рассмотрен процесс поэтапного создания аналитической модели для нужд банка, на основе транзакций клиентов банка. В ходе анализа были проведены манипуляции над данными, включая: парсинг исходного файла, перенос в хранилище, создания DataSet’ов, валидация данных, агрегирование данных, избавление от аутлаеров, прослеживание временных динамик, генерирование статистических метрик, кластеризация сущностей, сопоставление сущностей и т.д.. В последующем аналитическая модель может быть усовершенствована и доработана.

При выполнении работы использовались следующие инструменты: PyCharm, Python 3.6, Microsoft SQL Server, Git, GitHub, GitKraken. Проект представляет собой файл с данными, набор скриптов на языке Python 3.6, которые автоматизирует процесс создания аналитической модели, хранилище данных SQL Server’a, в которых хранятся данные в виде наборов DataSet.

Данная аналитическая модель поможет банку в выстраивании бизнес-отношений и в создании бизнес-стратегий по сущностям, рассматриваемыми в работе(мерчант и клиент).

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Том Дэвенпорт и Джон Харрис. Аналитика как конкурентное преимущество.
2. Блок о статистических метриках[Электронный ресурс]

//URL:<https://statanaliz.info/metody/opisanie-dannyx/11-dispersiya-standartnoe-otklonenie-koeffitsient-variatsii>

1. Документация по библиотеке языка Python scikit-learn[Электронный ресурс]

//URL:<http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>

1. Статья по работе с базой данных на языке Python[Электронный ресурс]

//URL: <https://habr.com/post/321510/>

1. Блок о кластеризации[Электронный ресурс]

//URL: <https://mubaris.com/2017/10/01/kmeans-clustering-in-python/>

# ПРИЛОЖЕНИЕ

# Скрипт connect\_setting.py:

DRIVER = ""#name of your DBMS

SQL\_SERVER = ""#name of your server

DATABASE = ""#name of your database

CONNECTION\_STRING = "DRIVER={0};SERVER={1};DATABASE={2}".format(DRIVER, SQL\_SERVER, DATABASE)

FILENAME = "Dataset\_for\_KFU.06\_07\_2017.csv"

DATE\_BEGIN = "2016-01-01"

DATE\_END = "2017-05-31"

GROUPS = ["Food", "Medical"]

# Скрипт Main\_Script.py:

import subprocess

import os

RELATIVE\_PATH = str(os.path.dirname(\_\_file\_\_))

INTERPRETER = RELATIVE\_PATH.replace(RELATIVE\_PATH[RELATIVE\_PATH.rfind("/") + 1:], "venv\Scripts\python.exe")

def run\_script(file\_path, step\_number, args=None):

program = [INTERPRETER, file\_path]

if (args != None):

for argument in args:

program.append(argument)

code = subprocess.check\_call(program)

if (code == 0):

print("Successful {0}".format(step\_number))

else:

raise Exception("Fail {0}".format(step\_number))

try:

file\_path = RELATIVE\_PATH + "/CSV\_Export(Step№1).py"

run\_script(file\_path, "Step№1")

file\_path = RELATIVE\_PATH + "/Starting\_Table(Step№2).py"

run\_script(file\_path, "Step№2")

file\_path = RELATIVE\_PATH + "/Starting\_Table\_For\_The\_Period(Step№3).py"

run\_script(file\_path, "Step№3")

file\_path = RELATIVE\_PATH + "/Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Clients(Step№4.1).py"

run\_script(file\_path, "Step№4.1")

file\_path = RELATIVE\_PATH + "/Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Correct\_Clients(Step№4.2).py"

run\_script(file\_path, "Step№4.2")

file\_path = RELATIVE\_PATH + "/Transaction\_Table\_Of\_Correct\_Clients(Step№4.3).py"

run\_script(file\_path, "Step№4.3")

file\_path = RELATIVE\_PATH + "/Aggregation\_Count\_Of\_Trans\_On\_Groups(Step№5).py"

run\_script(file\_path, "Step№5")

file\_path = RELATIVE\_PATH + "/Create\_Procedure\_CountOfTransForMonthOnClients(Step№6.1).py"

run\_script(file\_path, "Step№6.1")

file\_path = RELATIVE\_PATH + "/Create\_Main\_Metric\_Table\_For\_Clients(Step№6.2).py"

run\_script(file\_path, "Step№6.2")

file\_path = RELATIVE\_PATH + "/Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Clients\_On\_Merchants(Step№7.1).py"

run\_script(file\_path, "Step№7.1")

file\_path = RELATIVE\_PATH + "/Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Correct\_Clients\_On\_Merchants(Step№7.2).py"

run\_script(file\_path, "Step№7.2")

file\_path = RELATIVE\_PATH + "/Transaction\_Table\_Of\_Correct\_Clients\_On\_Merchants(Step№7.3).py"

run\_script(file\_path, "Step№7.3")

file\_path = RELATIVE\_PATH + "/Create\_Procedure\_CountOfTransForMonthOnMerchants(Step№8.1).py"

run\_script(file\_path, "Step№8.1")

file\_path = RELATIVE\_PATH + "/Create\_Metric\_Tables\_On\_Merchants\_Extended(Step№8.2).py"

run\_script(file\_path, "Step№8.2")

file\_path = RELATIVE\_PATH + "/Create\_Comparison\_Tables(Step№9).py"

run\_script(file\_path, "Step№9")

except Exception as exc:

print(exc)

# Скрипт CSV\_Export(Step№1).py:

import pandas

import pypyodbc as db

import os

from connection\_setting import connect\_setting

relative\_path = os.path.dirname(\_\_file\_\_)

FILENAME = str(relative\_path).replace(relative\_path[relative\_path.rfind("/") + 1:], connect\_setting.FILENAME)

connection\_string = connect\_setting.CONNECTION\_STRING

try:

connection = db.connect(connection\_string)

cursor = connection.cursor()

sql\_query = ("""

IF (OBJECT\_ID('CSV\_Export') IS NULL)

SELECT 0;

ELSE

SELECT 1;

""")

result = cursor.execute(sql\_query)

existence = list(result.fetchone())[0]

if (existence != 1):

df = pandas.read\_csv(FILENAME)

df.drop(df.columns[[0]], axis=1, inplace=True)

rows = list(tuple(row) for row in df.head(len(df)).values)

sql\_query = ("""

CREATE TABLE CSV\_Export (

event\_id BIGINT,

cli\_id BIGINT,

trans\_date NVARCHAR(50),

trans\_amount\_rub DECIMAL(38,2),

document\_channel INT,

document\_channel\_group INT,

our\_device BIT,

merchant\_id BIGINT,

code INT,

[group] NVARCHAR(50)

);

""")

cursor.execute(sql\_query)

cursor.commit()

sql\_query = ("""

INSERT INTO CSV\_Export(event\_id, cli\_id, trans\_date, trans\_amount\_rub, document\_channel,

document\_channel\_group, our\_device, merchant\_id, code, [group])

VALUES(?,?,?,?,?,?,?,?,?,?)

""")

cursor.executemany(sql\_query, rows)

cursor.commit()

except Exception as exc:

print(exc)

raise Exception(exc)

finally:

cursor.close()

connection.close()

# Скрипт Starting\_Table(Step№2).py:

import pypyodbc as db

from connection\_setting import connect\_setting

connection\_string = connect\_setting.CONNECTION\_STRING

try:

connection = db.connect(connection\_string)

cursor = connection.cursor()

sql\_query = ("""

IF (OBJECT\_ID('Starting\_Table') IS NULL)

BEGIN

SELECT

cli\_id,

trans\_date,

trans\_amount\_rub,

document\_channel,

document\_channel\_group,

our\_device,

merchant\_id,

code,

[group]

INTO Starting\_Table

FROM CSV\_Export

END

""")

cursor.execute(sql\_query)

cursor.commit()

except Exception as exc:

print(exc)

raise Exception(exc)

finally:

cursor.close()

connection.close()

# Скрипт Starting\_Table\_For\_The\_Period(Step№3).py:

import pypyodbc as db

from datetime import timedelta, datetime

from connection\_setting import connect\_setting

connection\_string = connect\_setting.CONNECTION\_STRING

try:

connection = db.connect(connection\_string)

cursor = connection.cursor()

date\_begin = connect\_setting.DATE\_BEGIN

date\_end = connect\_setting.DATE\_END

delta = timedelta(days=1)

date\_end = str(datetime.strptime(date\_end, '%Y-%m-%d').date() + delta)

sql\_query = ("""

IF(OBJECT\_ID('Starting\_Table\_For\_The\_Period') IS NULL)

BEGIN

SELECT \*

INTO Starting\_Table\_For\_The\_Period

FROM Starting\_Table

WHERE trans\_date BETWEEN '{0}' AND '{1}'

END

""".format(date\_begin, date\_end))

cursor.execute(sql\_query)

cursor.commit()

except Exception as exc:

print(exc)

raise Exception(exc)

finally:

cursor.close()

connection.close()

# Скрипт Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Clients(Step№4.1).py:

import pypyodbc as db

from connection\_setting import connect\_setting

connection\_string = connect\_setting.CONNECTION\_STRING

try:

connection = db.connect(connection\_string)

cursor = connection.cursor()

sql\_query = ("""

IF(OBJECT\_ID('Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Clients') IS NULL)

BEGIN

SELECT cli\_id, COUNT(trans\_amount\_rub) as count\_trans, SUM(trans\_amount\_rub) as sum\_trans

INTO Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Clients

FROM Starting\_Table\_For\_The\_Period

GROUP BY cli\_id

END

""")

cursor.execute(sql\_query)

cursor.commit()

except Exception as exc:

print(exc)

raise Exception(exc)

finally:

cursor.close()

connection.close()

# Скрипт Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Correct\_Clients(Step№4.2).py:

import pypyodbc as db

from datetime import datetime

from dateutil import relativedelta

from connection\_setting import connect\_setting

connection\_string = connect\_setting.CONNECTION\_STRING

try:

connection = db.connect(connection\_string)

cursor = connection.cursor()

sql\_query = ("""

IF(OBJECT\_ID('Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Correct\_Clients') IS NULL)

SELECT 0;

ELSE

SELECT 1;

""")

result = cursor.execute(sql\_query)

existence = list(result.fetchone())[0]

if (existence != 1):

date\_begin = connect\_setting.DATE\_BEGIN

date\_end = connect\_setting.DATE\_END

date1 = datetime.strptime(date\_begin, "%Y-%m-%d")

date2 = datetime.strptime(date\_end, "%Y-%m-%d")

difference = relativedelta.relativedelta(date2, date1)

months = 0

if (difference.days != 0):

months = abs(difference.years) \* 12 + abs(difference.months) + 1

else:

months = abs(difference.years) \* 12 + abs(difference.months)

sql\_query = ("""

SELECT cli\_id, count\_trans, sum\_trans

INTO Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Correct\_Clients

FROM Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Clients

WHERE count\_trans>=4\*{0}

""".format(months))

cursor.execute(sql\_query)

cursor.commit()

except Exception as exc:

print(exc)

raise Exception(exc)

finally:

cursor.close()

connection.close()

# Скрипт Transaction\_Table\_Of\_Correct\_Clients(Step№4.3).py:

import pypyodbc as db

from connection\_setting import connect\_setting

connection\_string = connect\_setting.CONNECTION\_STRING

try:

connection = db.connect(connection\_string)

cursor = connection.cursor()

sql\_query = ("""

IF(OBJECT\_ID('Transaction\_Table\_Of\_Correct\_Clients') IS NULL)

BEGIN

SELECT

A.cli\_id,

A.trans\_date,

A.trans\_amount\_rub,

A.document\_channel,

A.document\_channel\_group,

A.our\_device,

A.merchant\_id,

A.code,

A.[group]

INTO Transaction\_Table\_Of\_Correct\_Clients

FROM Starting\_Table\_For\_The\_Period AS A

INNER JOIN Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Correct\_Clients AS B

ON A.cli\_id=B.cli\_id

END

""")

cursor.execute(sql\_query)

cursor.commit()

except Exception as exc:

print(exc)

raise Exception(exc)

finally:

cursor.close()

connection.close()

# Скрипт Aggregation\_Count\_Of\_Trans\_On\_Groups(Step№5).py:

import pypyodbc as db

from connection\_setting import connect\_setting

connection\_string = connect\_setting.CONNECTION\_STRING

try:

connection = db.connect(connection\_string)

cursor = connection.cursor()

sql\_query = ("""

IF(OBJECT\_ID('Aggregation\_Count\_Of\_Trans\_On\_Groups') IS NULL)

BEGIN

SELECT [group], COUNT(trans\_amount\_rub) as count\_trans, SUM(trans\_amount\_rub) as sum\_trans

INTO Aggregation\_Count\_Of\_Trans\_On\_Groups

FROM Transaction\_Table\_Of\_Correct\_Clients

GROUP BY [group]

END

""")

cursor.execute(sql\_query)

cursor.commit()

except Exception as exc:

print(exc)

raise Exception(exc)

finally:

cursor.close()

connection.close()

# Скрипт Create\_Procedure\_CountOfTransForMonthOnClients(Step№6.1).py:

import pypyodbc as db

from connection\_setting import connect\_setting

connection\_string = connect\_setting.CONNECTION\_STRING

try:

connection = db.connect(connection\_string)

cursor = connection.cursor()

sql\_query = ("""

IF(OBJECT\_ID('CountOfTransForMonthOnClients') IS NULL)

SELECT 0;

ELSE

SELECT 1;

""")

result = cursor.execute(sql\_query)

existence = list(result.fetchone())[0]

if (existence != 1):

sql\_query = ("""

CREATE PROCEDURE CountOfTransForMonthOnClients @name NVARCHAR(100), @year INT, @month INT, @group NVARCHAR(100)

AS

BEGIN

DECLARE @SQL NVARCHAR(1000), @columnName NVARCHAR(250), @yearStr NVARCHAR(15), @monthStr NVARCHAR(10), @where NVARCHAR(250);

SET @yearStr=CAST(@year AS NVARCHAR(15));

SET @monthStr=CAST(@month AS NVARCHAR(10));

IF (@group='')

BEGIN

SET @columnName=CONCAT(@name, '\_', @yearStr, '\_', @monthStr);

SET @where=CONCAT('WHERE YEAR(trans\_date)=', @yearStr, ' AND ', 'MONTH(trans\_date)=', @monthStr);

END

ELSE

BEGIN

SET @columnName=CONCAT(@name, '\_', @group, '\_', @yearStr, '\_', @monthStr);

SET @where=CONCAT('WHERE YEAR(trans\_date)=', @yearStr, ' AND ', 'MONTH(trans\_date)=', @monthStr, ' AND ', '[group]=', CHAR(39), @group, CHAR(39))

END

SET @SQL=

CONCAT(

'SELECT cli\_id, COUNT(trans\_amount\_rub) AS ',

@columnName,

' INTO ',

@columnName,

' FROM Transaction\_Table\_Of\_Correct\_Clients ',

@where,

' GROUP BY cli\_id'

);

EXECUTE(@SQL);

END

""")

cursor.execute(sql\_query)

cursor.commit()

except Exception as exc:

print(exc)

raise Exception(exc)

finally:

cursor.close()

connection.close()

# Скрипт Create\_Main\_Metric\_Table\_For\_Clients(Step№6.2).py:

import pypyodbc as db

import numpy as np

import math

from datetime import datetime

from dateutil import relativedelta

from sklearn.cluster import KMeans

from connection\_setting import connect\_setting

connection\_string = connect\_setting.CONNECTION\_STRING

try:

connection = db.connect(connection\_string)

cursor = connection.cursor()

date\_begin = datetime.strptime(connect\_setting.DATE\_BEGIN, '%Y-%m-%d').date()

date\_end = datetime.strptime(connect\_setting.DATE\_END, '%Y-%m-%d').date()

difference = relativedelta.relativedelta(date\_end, date\_begin)

months = 0

if (difference.days != 0):

months = abs(difference.years) \* 12 + abs(difference.months) + 1

else:

months = abs(difference.years) \* 12 + abs(difference.months)

delta = relativedelta.relativedelta(months=1)

groups = connect\_setting.GROUPS

groups.insert(0, "")

main\_metrics\_table\_sql\_query = ""

columns\_metrics\_trans = ""

left\_joins\_metrics = ""

for group in groups:

into\_table = ""

postfix\_for\_group = ""

if (group != ""):

postfix\_for\_group = "\_{0}".format(group)

into\_table = "Count\_Of\_Trans{0}\_For\_Months\_Table".format(postfix\_for\_group)

sql\_query = ("""

IF(OBJECT\_ID('{0}') IS NULL)

SELECT 0;

ELSE

SELECT 1;

""".format(into\_table))

result = cursor.execute(sql\_query)

existence = list(result.fetchone())[0]

if (existence != 1):

sql\_query = ""

columns\_trans = ""

left\_joins = ""

date\_current = date\_begin

while ((date\_current.year < date\_end.year) or (

date\_current.year == date\_end.year and date\_current.month <= date\_end.month)):

year\_current = date\_current.year

month\_current = date\_current.month

sql\_query += "EXECUTE CountOfTransForMonthOnClients @name='count\_of\_trans\_for', @year={0}, @month={1}, @group='{2}';\n".format(

year\_current, month\_current, group)

columns\_trans += "\t\tcount\_of\_trans\_for{0}\_{1}\_{2},\n".format(postfix\_for\_group, year\_current,

month\_current)

left\_joins += "LEFT JOIN count\_of\_trans\_for{0}\_{1}\_{2} AS for\_{1}\_{2} ON aggregation\_table.cli\_id=for\_{1}\_{2}.cli\_id\n".format(

postfix\_for\_group, year\_current, month\_current)

date\_current += delta

columns\_trans = columns\_trans[:len(columns\_trans) - 2] + "\n"

sql\_query += "\n"

sql\_query += "SELECT\n\t\taggregation\_table.cli\_id,\n\t\taggregation\_table.count\_trans{0},\n\t\taggregation\_table.sum\_trans{0},\n".format(

postfix\_for\_group)

sql\_query += columns\_trans

sql\_query += "INTO {0}\n".format(into\_table)

subquery = ""

subquery += "SELECT\n\t\tcli\_id,\n\t\tCOUNT(trans\_amount\_rub) AS count\_trans{0},\n\t\tSUM(trans\_amount\_rub) AS sum\_trans{0}\n".format(

postfix\_for\_group)

subquery += "\t\tFROM Transaction\_Table\_Of\_Correct\_Clients\n"

if (group != ""):

subquery += "\t\tWHERE [group]='{0}'\n".format(group)

subquery += "\t\tGROUP BY cli\_id\n\t"

sql\_query += "FROM ({0}) AS aggregation\_table\n".format(subquery)

sql\_query += left\_joins

sql\_query += "\n"

sql\_query += "DROP TABLE\n"

sql\_query += columns\_trans

cursor.execute(sql\_query)

cursor.commit()

metrics\_table\_name = "Metric\_Table{0}".format(postfix\_for\_group)

sql\_query = ("""

IF(OBJECT\_ID('{0}\_Extended') IS NULL)

SELECT 0;

ELSE

SELECT 1;

""".format(metrics\_table\_name))

result = cursor.execute(sql\_query)

existence = list(result.fetchone())[0]

if (existence != 1):

sql\_query = "SELECT \* FROM {0}".format(into\_table)

rows = cursor.execute(sql\_query)

rows = rows.fetchall()

signs\_of\_clustering = []

metrics\_table = []

for i in range(0, len(rows)):

row = list(rows[i])

cli\_id = M\_count = M\_sum = D\_count = S\_count = V\_count = 0

count\_trans = 0

for j in range(3, len(row)):

if (row[j] != None):

count\_trans += row[j]

else:

count\_trans += 0

metrics = []

cli\_id = row[0]

if (count\_trans != 0):

M\_count = count\_trans / months

sum = 0

for j in range(3, len(row)):

if (row[j] != None):

sum += pow(row[j] - M\_count, 2)

else:

sum += pow(0 - M\_count, 2)

M\_count = round(M\_count, 2)

M\_sum = round(float(row[2]) / months, 2)

D\_count = round(sum / months, 2)

S\_count = round(math.sqrt(D\_count), 2)

V\_count = round(S\_count / M\_count \* 100, 2)

else:

M\_count = 0

M\_sum = 0

D\_count = 0

S\_count = 0

V\_count = 0

metrics.append(cli\_id)

metrics.append(M\_count)

metrics.append(M\_sum)

metrics.append(D\_count)

metrics.append(S\_count)

metrics.append(V\_count)

metrics\_table.append(metrics)

signs\_of\_clustering.append([M\_count, M\_sum])

kmeans = KMeans(n\_clusters=4, n\_init=25, max\_iter=2000)

kmeans.fit(np.array(signs\_of\_clustering))

labels = list(kmeans.labels\_)

for i in range(len(labels)):

metrics\_table[i].append(int(labels[i]))

metrics\_table[i] = tuple(metrics\_table[i])

sql\_query = ("""

CREATE TABLE {0} (

cli\_id BIGINT,

average\_count\_of\_trans{1} DECIMAL(38,2),

average\_sum\_of\_trans{1} DECIMAL(38,2),

variance\_of\_trans{1} DECIMAL(38,2),

mean\_square\_deviation\_of\_trans{1} DECIMAL(38,2),

coefficient\_of\_variation\_of\_trans{1} DECIMAL(38,2),

cluster\_number{1} INT

)""".format(metrics\_table\_name, postfix\_for\_group))

cursor.execute(sql\_query)

cursor.commit()

sql\_query = ("""

INSERT INTO {0}( cli\_id, average\_count\_of\_trans{1}, average\_sum\_of\_trans{1}, variance\_of\_trans{1},

mean\_square\_deviation\_of\_trans{1}, coefficient\_of\_variation\_of\_trans{1}, cluster\_number{1})

VALUES(?,?,?,?,?,?,?)

""".format(metrics\_table\_name, postfix\_for\_group))

cursor.executemany(sql\_query, metrics\_table)

cursor.commit()

sql\_query = ("""

SELECT

A.cli\_id,

A.count\_trans{2},

B.average\_count\_of\_trans{2},

A.sum\_trans{2},

B.average\_sum\_of\_trans{2},

B.variance\_of\_trans{2},

B.mean\_square\_deviation\_of\_trans{2},

B.coefficient\_of\_variation\_of\_trans{2},

B.cluster\_number{2}

INTO {1}\_Extended

FROM {0} AS A, {1} AS B

WHERE A.cli\_id=B.cli\_id

""".format(into\_table, metrics\_table\_name, postfix\_for\_group))

cursor.execute(sql\_query)

cursor.commit()

sql\_query = ("""DROP TABLE {0}""".format(metrics\_table\_name))

cursor.execute(sql\_query)

cursor.commit()

if (group != ""):

columns\_metrics\_trans += ",\n\t\tcount\_trans{0},\n\t\taverage\_count\_of\_trans{0},\n\t\tsum\_trans{0},\n\t\taverage\_sum\_of\_trans{0},\n\t\tvariance\_of\_trans{0},\n\t\tmean\_square\_deviation\_of\_trans{0},\n\t\tcoefficient\_of\_variation\_of\_trans{0},\n\t\tcluster\_number{0}\n".format(

postfix\_for\_group)

left\_joins\_metrics += "LEFT JOIN {0}\_Extended AS Extended{1} ON Extended{1}.cli\_id=A.cli\_id\n".format(

metrics\_table\_name, postfix\_for\_group)

else:

columns\_metrics\_trans += "A.cli\_id,\n\t\tcount\_trans,\n\t\taverage\_count\_of\_trans,\n\t\tsum\_trans,\n\t\taverage\_sum\_of\_trans,\n\t\tvariance\_of\_trans,\n\t\tmean\_square\_deviation\_of\_trans,\n\t\tcoefficient\_of\_variation\_of\_trans,\n\t\tcluster\_number\n"

sql\_query = ("""

IF(OBJECT\_ID('Main\_Metric\_Table\_For\_Clients') IS NULL)

SELECT 0;

ELSE

SELECT 1;

""")

result = cursor.execute(sql\_query)

existence = list(result.fetchone())[0]

if (existence != 1):

columns\_metrics\_trans = columns\_metrics\_trans[:len(columns\_metrics\_trans) - 1]

main\_metrics\_table\_sql\_query = ("""

SELECT

{0}

INTO Main\_Metric\_Table\_For\_Clients

FROM Metric\_Table\_Extended AS A

{1}

""".format(columns\_metrics\_trans, left\_joins\_metrics))

cursor.execute(main\_metrics\_table\_sql\_query)

cursor.commit()

except Exception as exc:

print(exc)

raise Exception(exc)

finally:

cursor.close()

connection.close()

# Скрипт Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Clients\_On\_Merchants(Step№7.1).py:

import pypyodbc as db

from connection\_setting import connect\_setting

connection\_string = connect\_setting.CONNECTION\_STRING

try:

connection = db.connect(connection\_string)

cursor = connection.cursor()

sql\_query = ("""

IF(OBJECT\_ID('Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Clients\_On\_Merchants') IS NULL)

BEGIN

SELECT merchant\_id, cli\_id, COUNT(trans\_amount\_rub) as count\_trans\_client, SUM(trans\_amount\_rub) as sum\_trans\_client

INTO Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Clients\_On\_Merchants

FROM Starting\_Table\_For\_The\_Period

GROUP BY merchant\_id, cli\_id

END

""")

cursor.execute(sql\_query)

cursor.commit()

except Exception as exc:

print(exc)

raise Exception(exc)

finally:

cursor.close()

connection.close()

# Скрипт Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Correct\_Clients\_On\_Merchants(Step№7.2).py:

import pypyodbc as db

from datetime import datetime

from dateutil import relativedelta

from connection\_setting import connect\_setting

connection\_string = connect\_setting.CONNECTION\_STRING

try:

connection = db.connect(connection\_string)

cursor = connection.cursor()

sql\_query = ("""

IF(OBJECT\_ID('Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Correct\_Clients\_On\_Merchants') IS NULL)

SELECT 0;

ELSE

SELECT 1;

""")

result = cursor.execute(sql\_query)

existence = list(result.fetchone())[0]

if (existence != 1):

date\_begin = connect\_setting.DATE\_BEGIN

date\_end = connect\_setting.DATE\_END

date1 = datetime.strptime(date\_begin, "%Y-%m-%d")

date2 = datetime.strptime(date\_end, "%Y-%m-%d")

difference = relativedelta.relativedelta(date2, date1)

months = 0

if (difference.days != 0):

months = abs(difference.years) \* 12 + abs(difference.months) + 1

else:

months = abs(difference.years) \* 12 + abs(difference.months)

sql\_query = ("""

SELECT merchant\_id, cli\_id, count\_trans\_client, sum\_trans\_client

INTO Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Correct\_Clients\_On\_Merchants

FROM Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Clients\_On\_Merchants

WHERE count\_trans\_client>=4\*{0}

""".format(months))

cursor.execute(sql\_query)

cursor.commit()

except Exception as exc:

print(exc)

raise Exception(exc)

finally:

cursor.close()

connection.close()

# Скрипт Transaction\_Table\_Of\_Correct\_Clients\_On\_Merchants(Step№7.3).py:

import pypyodbc as db

from connection\_setting import connect\_setting

connection\_string = connect\_setting.CONNECTION\_STRING

try:

connection = db.connect(connection\_string)

cursor = connection.cursor()

sql\_query = ("""

IF(OBJECT\_ID('Transaction\_Table\_Of\_Correct\_Clients\_On\_Merchants') IS NULL)

BEGIN

SELECT

A.cli\_id,

A.trans\_date,

A.trans\_amount\_rub,

A.document\_channel,

A.document\_channel\_group,

A.our\_device,

A.merchant\_id,

A.code,

A.[group]

INTO Transaction\_Table\_Of\_Correct\_Clients\_On\_Merchants

FROM Starting\_Table\_For\_The\_Period AS A

INNER JOIN Aggregation\_Of\_Count\_And\_Sum\_Of\_Trans\_By\_Correct\_Clients\_On\_Merchants AS B

ON A.merchant\_id=B.merchant\_id AND A.cli\_id=B.cli\_id

END

""")

cursor.execute(sql\_query)

cursor.commit()

except Exception as exc:

print(exc)

raise Exception(exc)

finally:

cursor.close()

connection.close()

# Скрипт Create\_Procedure\_CountOfTransForMonthOnMerchants(Step№8.1).py:

import pypyodbc as db

from connection\_setting import connect\_setting

connection\_string = connect\_setting.CONNECTION\_STRING

try:

connection = db.connect(connection\_string)

cursor = connection.cursor()

sql\_query = ("""

IF(OBJECT\_ID('CountOfTransForMonthOnMerchants') IS NULL)

SELECT 0;

ELSE

SELECT 1;

""")

result = cursor.execute(sql\_query)

existence = list(result.fetchone())[0]

if (existence != 1):

sql\_query = ("""

CREATE PROCEDURE CountOfTransForMonthOnMerchants @name NVARCHAR(110), @year INT, @month INT, @group NVARCHAR(100)

AS

BEGIN

DECLARE @SQL NVARCHAR(1010), @columnName NVARCHAR(260), @yearStr NVARCHAR(15), @monthStr NVARCHAR(10), @where NVARCHAR(250);

SET @yearStr=CAST(@year AS NVARCHAR(15));

SET @monthStr=CAST(@month AS NVARCHAR(10));

IF (@group='')

BEGIN

SET @columnName=CONCAT(@name, '\_', @yearStr, '\_', @monthStr);

SET @where=CONCAT('WHERE YEAR(trans\_date)=', @yearStr, ' AND ', 'MONTH(trans\_date)=', @monthStr);

END

ELSE

BEGIN

SET @columnName=CONCAT(@name, '\_', @group, '\_', @yearStr, '\_', @monthStr);

SET @where=CONCAT('WHERE YEAR(trans\_date)=', @yearStr, ' AND ', 'MONTH(trans\_date)=', @monthStr, ' AND ', '[group]=', CHAR(39), @group, CHAR(39))

END

SET @SQL=

CONCAT(

'SELECT merchant\_id, COUNT(trans\_amount\_rub) AS ',

@columnName,

' INTO ',

@columnName,

' FROM Transaction\_Table\_Of\_Correct\_Clients\_On\_Merchants ',

@where,

' GROUP BY merchant\_id'

);

EXECUTE(@SQL);

END

""")

cursor.execute(sql\_query)

cursor.commit()

except Exception as exc:

print(exc)

raise Exception(exc)

finally:

cursor.close()

connection.close()

# Скрипт Create\_Metric\_Tables\_On\_Merchants\_Extended(Step№8.2).py:

import pypyodbc as db

import numpy as np

import math

from datetime import datetime

from dateutil import relativedelta

from sklearn.cluster import KMeans

from connection\_setting import connect\_setting

connection\_string = connect\_setting.CONNECTION\_STRING

try:

connection = db.connect(connection\_string)

cursor = connection.cursor()

date\_begin = datetime.strptime(connect\_setting.DATE\_BEGIN, '%Y-%m-%d').date()

date\_end = datetime.strptime(connect\_setting.DATE\_END, '%Y-%m-%d').date()

difference = relativedelta.relativedelta(date\_end, date\_begin)

months = 0

if (difference.days != 0):

months = abs(difference.years) \* 12 + abs(difference.months) + 1

else:

months = abs(difference.years) \* 12 + abs(difference.months)

delta = relativedelta.relativedelta(months=1)

groups = connect\_setting.GROUPS

groups.insert(0, "")

columns\_metrics\_trans = ""

left\_joins\_metrics = ""

for group in groups:

into\_table = ""

postfix\_for\_group = ""

if (group != ""):

postfix\_for\_group = "\_{0}".format(group)

into\_table = "Count\_Of\_Trans{0}\_On\_Merchants\_For\_Months\_Table".format(postfix\_for\_group)

sql\_query = ("""

IF(OBJECT\_ID('{0}') IS NULL)

SELECT 0;

ELSE

SELECT 1;

""".format(into\_table))

result = cursor.execute(sql\_query)

existence = list(result.fetchone())[0]

if (existence != 1):

sql\_query = ""

columns\_trans = ""

left\_joins = ""

date\_current = date\_begin

while ((date\_current.year < date\_end.year) or (

date\_current.year == date\_end.year and date\_current.month <= date\_end.month)):

year\_current = date\_current.year

month\_current = date\_current.month

sql\_query += "EXECUTE CountOfTransForMonthOnMerchants @name='count\_of\_trans\_for', @year={0}, @month={1}, @group='{2}';\n".format(

year\_current, month\_current, group)

columns\_trans += "\t\tcount\_of\_trans\_for{0}\_{1}\_{2},\n".format(postfix\_for\_group, year\_current,

month\_current)

left\_joins += "LEFT JOIN count\_of\_trans\_for{0}\_{1}\_{2} AS for\_{1}\_{2} ON aggregation\_table.merchant\_id=for\_{1}\_{2}.merchant\_id\n".format(

postfix\_for\_group, year\_current, month\_current)

date\_current += delta

columns\_trans = columns\_trans[:len(columns\_trans) - 2] + "\n"

sql\_query += "\n"

sql\_query += "SELECT\n\t\taggregation\_table.merchant\_id,\n\t\taggregation\_table.count\_trans{0},\n\t\taggregation\_table.sum\_trans{0},\n".format(

postfix\_for\_group)

sql\_query += columns\_trans

sql\_query += "INTO {0}\n".format(into\_table)

subquery = ""

subquery += "SELECT\n\t\tmerchant\_id,\n\t\tCOUNT(trans\_amount\_rub) AS count\_trans{0},\n\t\tSUM(trans\_amount\_rub) AS sum\_trans{0}\n".format(

postfix\_for\_group)

subquery += "\t\tFROM Transaction\_Table\_Of\_Correct\_Clients\_On\_Merchants\n"

if (group != ""):

subquery += "\t\tWHERE [group]='{0}'\n".format(group)

subquery += "\t\tGROUP BY merchant\_id\n\t"

sql\_query += "FROM ({0}) AS aggregation\_table\n".format(subquery)

sql\_query += left\_joins

sql\_query += "\n"

sql\_query += "DROP TABLE\n"

sql\_query += columns\_trans

cursor.execute(sql\_query)

cursor.commit()

metrics\_table\_name = "Metric\_Table{0}\_On\_Merchants".format(postfix\_for\_group)

sql\_query = ("""

IF(OBJECT\_ID('{0}\_Extended') IS NULL)

SELECT 0;

ELSE

SELECT 1;

""".format(metrics\_table\_name))

result = cursor.execute(sql\_query)

existence = list(result.fetchone())[0]

if (existence != 1):

sql\_query = "SELECT \* FROM {0}".format(into\_table)

rows = cursor.execute(sql\_query)

rows = rows.fetchall()

signs\_of\_clustering = []

metrics\_table = []

for i in range(0, len(rows)):

row = list(rows[i])

merchant\_id = M\_count = M\_sum = D\_count = S\_count = V\_count = 0

count\_trans = 0

for j in range(3, len(row)):

if (row[j] != None):

count\_trans += row[j]

else:

count\_trans += 0

metrics = []

merchant\_id = row[0]

if (count\_trans != 0):

M\_count = count\_trans / months

sum = 0

for j in range(3, len(row)):

if (row[j] != None):

sum += pow(row[j] - M\_count, 2)

else:

sum += pow(0 - M\_count, 2)

M\_count = round(M\_count, 2)

M\_sum = round(float(row[2]) / months, 2)

D\_count = round(sum / months, 2)

S\_count = round(math.sqrt(D\_count), 2)

V\_count = round(S\_count / M\_count \* 100, 2)

else:

M\_count = 0

M\_sum = 0

D\_count = 0

S\_count = 0

V\_count = 0

metrics.append(merchant\_id)

metrics.append(M\_count)

metrics.append(M\_sum)

metrics.append(D\_count)

metrics.append(S\_count)

metrics.append(V\_count)

metrics\_table.append(metrics)

signs\_of\_clustering.append([M\_count, M\_sum])

kmeans = KMeans(n\_clusters=4, n\_init=25, max\_iter=2000)

kmeans.fit(np.array(signs\_of\_clustering))

labels = list(kmeans.labels\_)

for i in range(len(labels)):

metrics\_table[i].append(int(labels[i]))

metrics\_table[i] = tuple(metrics\_table[i])

sql\_query = ("""

CREATE TABLE {0} (

merchant\_id BIGINT,

average\_count\_of\_trans{1} DECIMAL(38,2),

average\_sum\_of\_trans{1} DECIMAL(38,2),

variance\_of\_trans{1} DECIMAL(38,2),

mean\_square\_deviation\_of\_trans{1} DECIMAL(38,2),

coefficient\_of\_variation\_of\_trans{1} DECIMAL(38,2),

cluster\_number{1} INT

)""".format(metrics\_table\_name, postfix\_for\_group))

cursor.execute(sql\_query)

cursor.commit()

sql\_query = ("""

INSERT INTO {0}( merchant\_id, average\_count\_of\_trans{1}, average\_sum\_of\_trans{1}, variance\_of\_trans{1},

mean\_square\_deviation\_of\_trans{1}, coefficient\_of\_variation\_of\_trans{1}, cluster\_number{1})

VALUES(?,?,?,?,?,?,?)

""".format(metrics\_table\_name, postfix\_for\_group))

cursor.executemany(sql\_query, metrics\_table)

cursor.commit()

sql\_query = ("""

SELECT

A.merchant\_id,

A.count\_trans{2},

B.average\_count\_of\_trans{2},

A.sum\_trans{2},

B.average\_sum\_of\_trans{2},

B.variance\_of\_trans{2},

B.mean\_square\_deviation\_of\_trans{2},

B.coefficient\_of\_variation\_of\_trans{2},

B.cluster\_number{2}

INTO {1}\_Extended

FROM {0} AS A, {1} AS B

WHERE A.merchant\_id=B.merchant\_id

""".format(into\_table, metrics\_table\_name, postfix\_for\_group))

cursor.execute(sql\_query)

cursor.commit()

sql\_query = ("""DROP TABLE {0}""".format(metrics\_table\_name))

cursor.execute(sql\_query)

cursor.commit()

except Exception as exc:

print(exc)

raise Exception(exc)

finally:

cursor.close()

connection.close()

# Скрипт Create\_Comparison\_Tables(Step№9).py:

import pypyodbc as db

from connection\_setting import connect\_setting

connection\_string = connect\_setting.CONNECTION\_STRING

try:

connection = db.connect(connection\_string)

cursor = connection.cursor()

groups = connect\_setting.GROUPS

groups.insert(0, "")

for group in groups:

into\_table = ""

postfix\_for\_group = ""

where\_sql = ""

if (group != ""):

postfix\_for\_group = "\_{0}".format(group)

where\_sql = "AND [group]='{0}'".format(group)

into\_table = "Comparison\_Table{0}".format(postfix\_for\_group)

sql\_query = ("""

IF(OBJECT\_ID('{0}') IS NULL)

SELECT 0;

ELSE

SELECT 1;

""".format(into\_table))

result = cursor.execute(sql\_query)

existence = list(result.fetchone())[0]

if (existence != 1):

sql\_query = ("""

SELECT DISTINCT

A.merchant\_id,

A.cluster\_number{1} AS cluster\_number\_merchant{1},

A.coefficient\_of\_variation\_of\_trans{1} AS coefficient\_of\_variation\_of\_trans\_merchant{1},

B.cli\_id,

B.cluster\_number{1} AS cluster\_number\_client{1},

B.coefficient\_of\_variation\_of\_trans{1} AS coefficient\_of\_variation\_of\_trans\_client{1},

COUNT(C.trans\_amount\_rub) AS count\_trans\_client{1},

SUM(C.trans\_amount\_rub) AS sum\_trans\_client{1}

INTO {0}

FROM

Metric\_Table{1}\_On\_Merchants\_Extended AS A,

Metric\_Table{1}\_Extended AS B,

Transaction\_Table\_Of\_Correct\_Clients\_On\_Merchants AS C

WHERE A.merchant\_id=C.merchant\_id AND B.cli\_id=C.cli\_id {2}

GROUP BY

A.merchant\_id,

A.cluster\_number{1},

A.coefficient\_of\_variation\_of\_trans{1},

B.cli\_id,

B.cluster\_number{1},

B.coefficient\_of\_variation\_of\_trans{1}

""".format(into\_table, postfix\_for\_group, where\_sql))

cursor.execute(sql\_query)

cursor.commit()

except Exception as exc:

print(exc)

raise Exception(exc)

finally:

cursor.close()

connection.close()