УДК 004.896

Когортный анализ базы данных

Акентьев Данила Денисович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
Цель исследования - выполнить когортный анализ базы данных с помощью SQL. Для когортного анализа в SQL подключается и используется модуль Pandas и Numpy. Используется среда разработки Google Collab. В данной работе показан когортный анализ данных в SQL, на практическом примере.
Ключевые слова: Удержание, SQL, Когортный анализ, Pandas, Numpy.

Cohort analysis of the database

Akentev Danila Denisovich
Sholom-Aleichem Priamursky State University
Student

Abstract
The purpose of the study is to perform a cohort analysis of the database using SQL. For cohort analysis, the Pandas and Numpy module is connected and used in SQL. The Google Collab development environment is used. This paper shows the cohort analysis of data in SQL, using a practical example.
Keywords: Retention, SQL, Cohort analysis, Pandas, Numpy

1 Введение

1.1 Актуальность

На сегодняшний день когортный анализ является важным направлением применения современной вычислительной техники. Для аналитика данный способ необходим, чтобы понять как пользователи или клиенты ведут себя с течением времени, отслеживать эффективность рекламы.

1.2 Обзор исследований

В своей работе Н.А. Кондакова произвела оценку здоровья детей Вологодской области за 1995-2019 гг. с помощью метода когортного анализа данных [1]. Ю.И. Растова, Д.О. Яровой оценили влияние характера отношений с владельцами организации на эффективность ее деятельности с помощью когортного исследования [2]. В.А. Калашников, М.В. Иванов выяснили, что когортный анализ данных обладает преимуществами перед другими видами анализа [3]. В своей работе А.С. Тимофеева представила несколько примеров использование когортного анализа в интернет-маркетинге [4]. Г.А. Парахонская рассмотрела возможность применения

когортного анализа в изучении возрастных групп, провела сравнение данного метода с поколенным анализом и анализом жизненного пути [5]. П.А. Гришенков объяснил и на практике показал, как выполняется когортный анализ данных SQL [6]. В своей статье А. Зайчик объяснила, что такое когортный анализ и для чего он нужен [7].

1.3 Цель исследования

Цель исследования - выполнить когортный анализ базы данных с помощью SQL.

2 Материалы и методы

Для когортного анализа в SQL подключается и используется модуль Pandas и Numpy. Используется среда разработки Google Collab.

3 Результаты и обсуждения

Задание: В июле провели специальную акцию на улучшение ретеншена. Новым пользователям слали письмо с купоном. Проанализировать сработала ли акция?

Выполнение анализа происходит на сайте Google Collab. Перед началом работы требуется установить и импортировать sqlite3 (рис.1).

```
!gdown --id 1BSHIKQ7rFw5BpTq5nw1UZfjPK_7Mpnbi
!mv _sqlite3.cpython-37m-x86_64-linux-gnu.so /usr/lib/python3.7/lib-dynload/
import os
os.kill(os.getpid(), 9)
```

```
import sqlite3
```

Рисунок 1 – Установка и импортирование sqlite3

Для работы с базой данных осталось только поместить ее в папку content в Google Collab. После чего все готово для работы с файлом (рис.2)

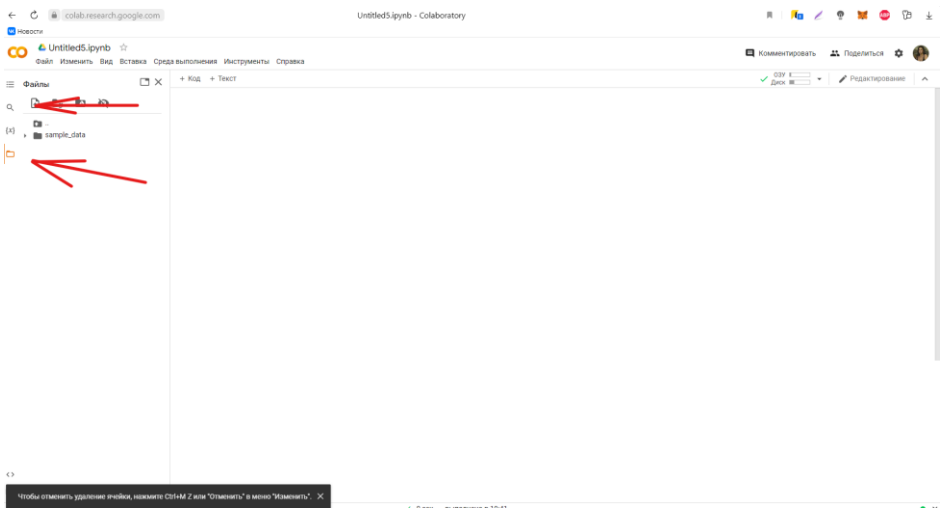

Рисунок 2 – Подключение Базы данных

Импортируем Pandas и Numpy для удобства (рис.3).

```
import pandas as pd
import numpy as np
```

Рисунок 3 – Импорт необходимой библиотеки

Для начала нужно создать соединение с базой данных (рис.4). Чтобы выполнять инструкции SQL и извлекать результаты из SQL-запросов подключаем курсор базы данных (рис.5).

```
[3]  con = sqlite3.connect('db')
```

Рисунок 4 – Соединение с базой данных

```
[4]  cur = con.cursor()
```

Рисунок 5 – Вызов курсора

Далее считываем оригинальный DataFrame (рис.6) И переводим даты к правильным типам (рис.7) данный перевод нужен для дальнейшего правильного выполнения кода, иначе можно потерять много данных.

```
df = pd.read_csv('/content/analyst_test_transactions_db.csv')
```

```
df.head()
```

	user_id	order_id	reg_date	transaction_date	revenue
0	57562	94545.0	2013-12-22	2015-07-16	1088
1	40047	81824.0	2013-11-24	2015-01-19	1027
2	48153	144851.0	2015-07-03	2015-10-29	601
3	54591	116111.0	2013-07-12	2015-04-30	881
4	39789	134943.0	2013-05-25	2015-07-21	984

Рисунок 6 – Считывание DataFrame

```
[6]  df['reg_date'] = pd.to_datetime(df['reg_date'],format='%Y-%m-%d')
     df['transaction_date'] = pd.to_datetime(df['transaction_date'],format='%Y-%m-%d')
```

Рисунок 7 – Перевод даты в правильный тип

Создаем базу данных SQL и подключаем таблицу к ней и сразу же проверим(рис.8).

```
def select(sql):
  return pd.read_sql(sql,con)
```

Рисунок 8.1 – Создание базы данных SQL

```
[7] df.to_sql('trans',con,index=False,if_exists='replace')

[8] sql = ''' select * from trans t'''

select(sql)
```

	user_id	order_id	reg_date	transaction_date	revenue
0	57562	94545.0	2013-12-22 00:00:00	2015-07-16 00:00:00	1088
1	40047	81824.0	2013-11-24 00:00:00	2015-01-19 00:00:00	1027
2	48153	144851.0	2015-07-03 00:00:00	2015-10-29 00:00:00	601
3	54591	116111.0	2013-07-12 00:00:00	2015-04-30 00:00:00	881
4	39789	134943.0	2013-05-25 00:00:00	2015-07-21 00:00:00	984
...	...	...	...	...	...
76852	36921	163682.0	2015-01-15 00:00:00	2015-03-28 00:00:00	1095
76853	31360	145146.0	2015-01-04 00:00:00	2015-08-09 00:00:00	547
76854	56195	8929.0	2012-09-18 00:00:00	2015-11-12 00:00:00	850
76855	46362	78673.0	2015-02-20 00:00:00	2015-07-21 00:00:00	356
76856	14872	106067.0	2015-03-13 00:00:00	2015-04-13 00:00:00	1114

76857 rows × 5 columns

Рисунок 8.2 – Подключение и проверка

Теперь делаем заготовку: пользователи с минимальной датой регистрации (переводя ее в начало месяца), и создаем агрегацию на столбце user_id, чтобы в будущем к ней присоединить транзакции.

```
[23] sql = ''' select t.user_id, date(min(reg_date),'start of month') as reg_month from trans t
     group by t.user_id'''

[24] select(sql)
```

	user_id	reg_month
0	-37599	2015-01-01
1	-32239	2015-11-01
2	-29822	2015-07-01
3	-29531	2014-09-01
4	-25337	2015-04-01
...	...	...
48209	128964	2015-10-01
48210	130617	2015-02-01
48211	131071	2012-04-01
48212	132484	2015-06-01
48213	138987	2014-10-01

48214 rows × 2 columns

Рисунок 9 – Создаем заготовку

С помощью команд на (рис.10) генерируем диапазон наших транзакций: находим максимальную и минимальную дату в столбце transaction _date.

```
[12] min = '''select date(min(t.transaction_date),'start of month') from trans t'''

     select(min)
```

	date(min(t.transaction_date),'start of month')
0	2015-01-01

Рисунок 10.1 – Находим минимальную дату

```
[14] max = '''select date(max(t.transaction_date),'start of month') from trans t'''

     select(max)
```

	date(max(t.transaction_date),'start of month')
0	2015-11-01

Рисунок 10.2 – Находим максимальную дату

После чего генерируем транзакции по месяцам в порядке возрастания от минимального к максимальному с помощью рекурсивной функции и f-string функции для упрощения кода. Для этого и нужны были минимальные и максимальные транзакции.

```
sql = f'''WITH RECURSIVE dates(date) AS (
  VALUES(({min}))
  UNION ALL
  SELECT date(date, '+1 month')
  FROM dates
  WHERE date < ({max})
)
SELECT date FROM dates;'''
```

```
[17] select(sql)
```

	date
0	2015-01-01
1	2015-02-01
2	2015-03-01
3	2015-04-01
4	2015-05-01
5	2015-06-01
6	2015-07-01
7	2015-08-01
8	2015-09-01
9	2015-10-01
10	2015-11-01

Рисунок 11 – Рекурсивная функция

Далее используем Common Table Expressions (Обобщенные табличные выражения). CROSS JOIN-им таблицу user_id и transaction_id или по другому начинаем сливать две таблицу в одну которая будет называться template. И убираем месяца пользователей в которых он еще не был зарегестрирован(reg_month)(рис.12).

Делаем еще одну табличку trans_month, что бы сгруппировать все транзакции на месяц и выводим количество транзакций каждым user_id, далее создаем и вычисляем lifetime и в конце концов ее CROSS JOIN-им к template(рис.13).

Остается создать сводную таблицу для этого фильтруем дату регистрации(ставим >=2015-01-01)(рис.14)

Теперь выполняем оператор PIVOT(рис.15)

```
sql = f'''with users as  (select t.user_id, date(min(reg_date),'start of month') as reg_month from trans t
group by t.user_id),

dates as (

WITH RECURSIVE dates(date) AS (
  VALUES(({min}))
  UNION ALL
  SELECT date(date, '+1 month')
  FROM dates
  WHERE date < ({max})
)
SELECT date FROM dates

),

template as (

select t.user_id, t.reg_month, d.date as month from users t
join dates d on d.date >= t.reg_month),
```

Рисунок 12 – CROSS JOIN template к dates

```
trans_month as (

select t.user_id,
date(t.transaction_date,'start of month') as month,

sum(t.revenue) as revenue_sum,
count(1) as transaction_cnt


 from trans t

 group by
t.user_id,
date(t.transaction_date,'start of month')

),

report as (
select t.*, tm.revenue_sum, tm.transaction_cnt,
case when tm.revenue_sum > 0 then 1 else 0 end as active,

round((julianday(t.month) - julianday(t.reg_month)) / 30) as lifetime

 from template t
left join trans_month tm on t.user_id = tm.user_id and t.month = tm.month)
```

Рисунок 13 – CROSS JOIN trans_month и report к template (создание и вычисление lifetime)

```
select t.reg_month, t.lifetime, avg(t.active) as retention from report t
where t.reg_month >= '2015-01-01'
group by t.reg_month, t.lifetime

'''
```

```
[29] t = select(sql)
```

```
[30] select(sql)
```

	reg_month	lifetime	retention
0	2015-01-01	0.0	0.116928
1	2015-01-01	1.0	0.116056
2	2015-01-01	2.0	0.129581
3	2015-01-01	3.0	0.116492
4	2015-01-01	4.0	0.115183
...	...	...	...
61	2015-09-01	1.0	0.381497
62	2015-09-01	2.0	0.221414
63	2015-10-01	0.0	0.561974
64	2015-10-01	1.0	0.448857
65	2015-11-01	0.0	1.000000

66 rows × 3 columns

Рисунок 14 – Создание сводной таблицы

```
t.pivot_table(index='reg_month',columns='lifetime',values='retention',aggfunc='max')
```

lifetime	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0
reg_month											
2015-01-01	0.116928	0.116056	0.129581	0.116492	0.115183	0.124782	0.118237	0.131326	0.126091	0.148778	0.114311
2015-02-01	0.132856	0.141543	0.135922	0.121615	0.122126	0.119571	0.123659	0.149208	0.163005	0.105263	NaN
2015-03-01	0.164114	0.138403	0.129650	0.127462	0.138403	0.145514	0.138950	0.170678	0.117068	NaN	NaN
2015-04-01	0.193627	0.158088	0.144608	0.136029	0.161152	0.137255	0.167892	0.127451	NaN	NaN	NaN
2015-05-01	0.246457	0.184252	0.154331	0.136220	0.152756	0.159055	0.132283	NaN	NaN	NaN	NaN
2015-06-01	0.257924	0.216283	0.180236	0.168428	0.189559	0.141703	NaN	NaN	NaN	NaN	NaN
2015-07-01	0.283110	0.331367	0.179088	0.181769	0.124397	NaN	NaN	NaN	NaN	NaN	NaN
2015-08-01	0.423358	0.268613	0.222628	0.157664	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2015-09-01	0.429314	0.381497	0.221414	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2015-10-01	0.561974	0.448857	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2015-11-01	1.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Рисунок 15 – Оператор PIVOT

По данной таблице можем увидеть, что в июле месяце, когда прошла акция retention начал расти до конца года. 2015-06-01 retention составляет 0.257924, а 2015-08-01 retention равен 0.429314. Из этого можно сделать вывод, что спустя месяц после проведения акции retention увеличился в более чем 1.5 раза, а уже через два месяца retention стал равен 0.561974, это говорит о том, что он увеличился более чем в 2 раза.

Из всех этих данных делаем вывод, что акция сработала.

4 Выводы

В данной работе показан когортный анализ данных в SQL, на практическом примере. Выполнили задание и ответили на поставленный вопрос. Были изучены оператор PIVOT, оператор CROSS JOIN, оператор SELECT, применение рекурсивных функций, создание сводных таблиц, перевод даты в правильный формат, создание и вычисление lifetime. Применили Common Table Expressions на практике. В ходе анализа данных было выяснено, что после акции в июле retention начал расти до конца года.

Библиографический список

1. Кондакова Н. А. Оценка здоровья детей Вологодской области за 1995-2019 гг.: метод когортного анализа // Социальные аспекты здоровья населения. – 2019. – Т. 65. – № 6. – С. 5.
2. Растова Ю. И. Когортный анализ эффективности корпоративного бизнеса // Известия Санкт-Петербургского государственного экономического университета. – 2019. – № 5-1(119). – С. 106-111.
3. Калашников В. А. Использование данных для когортного анализа при прогнозировании макроэкономических показателей методами машинного обучения // Экономика: вчера, сегодня, завтра. – 2022. – Т. 12. – № 8-1. – С. 7-15.
4. Тимофеева А. С. Когортный анализ в продуктовой и маркетинговой аналитике // Весенние дни науки ВШЭМ: Сборник докладов международной конференции студентов, аспирантов, молодых ученых, Екатеринбург, 20–22 апреля 2017 года. Екатеринбург: ООО "Издательство УМЦ УПИ", 2017. С. 502-506.
5. Парахонская Г. А. Когортный анализ в изучении возрастных групп // Вестник Тверского государственного университета. Серия: Экономика и управление. 2014. № 3. С. 89-97.
6. Гришенков П. Повторяем когортный анализ. Комплексный подход — Python, SQL, Power BI. URL:https://habr.com/ru/post/542626/ (дата обращения: 18.06.2024)
7. Зайчик А. Когортный анализ: зачем нужен бизнесу и как его проводить. URL: https://practicum.yandex.ru/blog/chto-takoe-kogortnyi-analiz/ (дата обращения: 18.06.2024)