Итоговая аттестация
по курсу Data Engineer, осенний поток 2023

Ларин Петр Михайлович

Проект №5 – «Служба такси»

Задание

Входные данные: Служба такси

Есть таблица, состоящая из поездок такси в Нью-Йорке:

Поле	Описание
VendorId	ИД компании
Trep_pickup_datetime	Время и дата, когда пассажир сел в такси
Trep_dropoff_datetime	Время и дата, когда пассажир вышел из такси
Passanger_count	Количество пассажиров
Trip_distance	Пройденное расстояние
Ratecodeid	Код скорости
Store_and_fwd_flag	Флаг, отвечающий за сохранение записи поездки перед ее отправкой поставщику
PulocationId	Широта, где была начата поездка
Dolocationid	Долгота, где была начата поездка
Payment_type	Тип оплаты
Fare_amount	Стоимость поездки
Mta_tax	Комиссия автопарка
Tip_amount	Чаевые
Tools_amount	Оплата за платные дороги
Improvement_surchange	Доплата за страховку
Total_amount	Полная стоимость поездки
Congestion_surchange	Дополнительный сбор

Необходимо, используя таблицу поездок для каждого дня, рассчитать процент поездок по количеству человек в машине (без пассажиров, 1, 2, 3, 4 и более пассажиров). По итогу должна получиться таблица (parquet) с колонками date, percentage_zero, percentage_1p, percentage_2p, percentage_3p, percentage_4p_plus. Технологический стек — sql, scala (что-то одно).
Также добавить столбцы к предыдущим результатам с самой дорогой и самой дешевой поездкой для каждой группы.

Дополнительно: также провести аналитику и построить график на тему «Как пройденное расстояние и количество пассажиров влияет на чаевые» в любом удобном инструменте.

Используемые технологии с обоснованием

Я решил выполнить задание двумя разными способами независимо друг от друга, чтобы получить больше практики и иметь возможность сравнить результаты:

1-й способ:

- PostgreSQL
- Docker
- pgAdmin

Преимущества:

- Знакомый мне продукт и язык

Недостатки:

- Не полностью отвечает заданию, т.к. непосредственно не поддерживает parquet
- Ограниченные возможности по масштабированию

2-й способ:

- PySpark (без использования SQL-запросов)
- Jupyter Notebook

Преимущества:

- Поддерживает parquet
- Масштабируется

Недостатки:

- Менее знакомая мне технология, менее наглядная, чем SQL

Дополнительное задание:

(аналитика чаевых)

- Jupyter Notebook
- Pandas
- Matplotlib, Seaborn

Качество входных данных и предобработка

- **Значения NULL.** Хотелось бы отметить высокое качество входных данных: из общего количества в 6.4 млн строк в датасете оказалось всего 65K строк (1%) с NULL, причем в одних и тех же столбцах. Данные строки пришлось удалить, т.к. NULL присутствовали в необходимых для анализа столбцах (в частности, кол-во пассажиров).
- **Даты поездок.** По датасету было понятно, что он представляет собой сведения за январь 2020 – примерно 200K поездок в день. Однако присутствовали единичные поездки на случайные даты за период с 2003 по 2021 г. Я оставил те поездки, которые хотя бы частично пересекались с январем 2020.
- **Стоимость поездки.** В то же время возникли сложности с интерпретацией нулевых и отрицательных значений стоимости поездки. Скорее всего, отрицательные значения соответствовали возврату денежных средств. С нулевыми менее понятно, возможно это были бонусные поездки. Проблема состояла в том, что при сохранении данных строк значения минимума стоимости поездки оказывались отрицательными. Поэтому я решил эти поездки также удалить.
- **Итоговый датасет.** После проведенной предобработки датасет сократился с 6,405,008 до 6,318,308 строк, или на 1.35%.
- **Расстояние поездки.** Для исследования зависимости чаевых от расстояния в дополнительном задании я также удалил поездки с нулевым расстоянием, т.к. непонятно, что это означает, возможно, информация была потеряна. После удаления таких поездок датасет сократился до 6,254,202 строк, или на 2.35% от первоначальной величины.

Результаты разработки

PostgreSQL

- SQL-скрипт **datamart_postgres.sql**
- Витрина данных **datamart_postgres.csv**

Время работы скрипта при первоначальном запуске – около 20 с. (в Докере)

PySpark

- Jupyter Notebook **datamart_spark.ipynb**
- Витрина данных **datamart_spark.parquet**
- Витрина данных **datamart_spark.csv**

Время работы скрипта при первоначальном запуске – около 53 с.

Система: i5-9400F CPU @ 2.90GHz, 128G ОЗУ, Win10

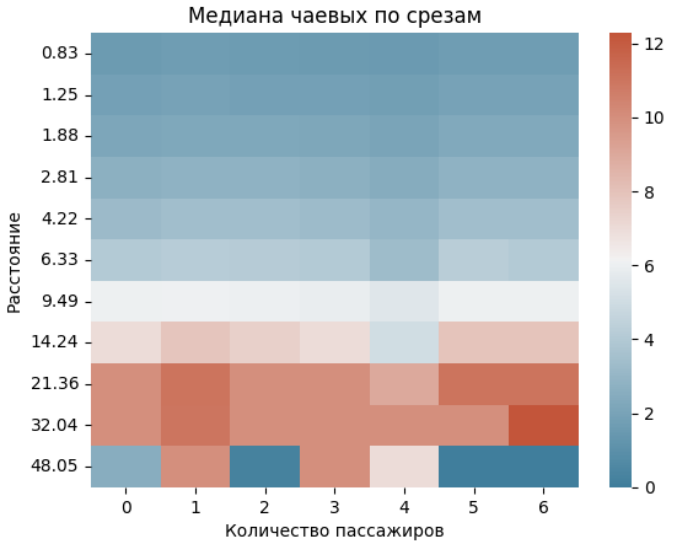
PySpark отработал медленнее, вероятно, из-за больших накладных расходов, которые становятся заметны при таком малом объеме данных, а также из-за массированного применения UDF.

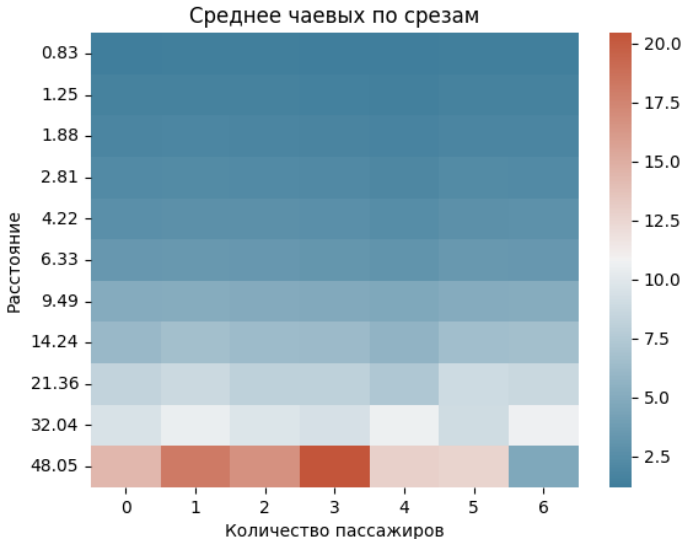
Витрины данных в формате csv совпали на 100%.

Витрина данных

pickup_date	per_0p	per_1p	per_2p	per_3p	per_4p+	min_0p	max_0p	min_1p	max_1p	min_2p	max_2p	min_3p	max_3p	min_4p+	max_4p+
2019-12-31	0.0	57.14	17.46	4.76	20.63	NULL	NULL	5.8	71.62	9.8	41.8	16.0	34.42	6.8	54.36
2020-01-01	1.52	63.51	19.48	5.75	9.74	0.3	145.55	0.3	465.3	0.3	281.42	0.3	433.04	0.3	350.42
2020-01-02	1.7	68.81	16.01	4.74	8.74	0.3	174.36	0.3	492.8	0.3	328.04	0.3	215.54	3.3	352.3
2020-01-03	1.77	68.87	16.31	4.58	8.47	0.3	187.42	0.3	1242.3	0.3	370.3	0.31	409.59	0.3	348.3
2020-01-04	1.69	65.65	18.38	5.14	9.14	0.3	152.54	0.3	965.8	0.3	481.3	0.3	313.3	0.3	577.8
2020-01-05	1.7	67.69	17.35	4.77	8.49	0.3	435.42	0.3	596.42	0.3	375.3	0.31	333.34	3.3	264.36
2020-01-06	1.82	73.79	13.53	3.69	7.17	0.3	275.42	0.3	1040.39	0.3	276.36	3.3	200.3	0.6	185.9
2020-01-07	1.76	73.99	13.66	3.46	7.12	0.3	114.8	0.01	410.67	0.31	375.3	3.3	343.3	3.3	242.8
2020-01-08	1.83	74.03	13.53	3.51	7.1	0.3	319.03	0.3	914.55	0.3	282.36	0.3	350.8	0.3	253.67
2020-01-09	1.86	74.12	13.54	3.48	7.01	0.3	186.05	0.3	620.3	0.3	231.65	0.3	230.3	3.3	404.3
2020-01-10	1.82	72.01	14.85	3.92	7.4	2.8	253.3	0.3	362.3	0.3	245.05	3.3	187.55	3.3	284.76
2020-01-11	1.68	67.41	17.99	4.65	8.28	0.3	143.45	0.3	416.15	0.3	228.02	0.3	250.3	3.3	424.29
2020-01-12	1.66	68.88	17.0	4.56	7.91	0.3	161.8	0.3	450.3	0.3	431.6	0.31	331.42	0.3	658.35
2020-01-13	1.8	74.78	13.02	3.51	6.9	0.3	276.29	0.3	534.12	0.3	475.17	3.3	311.3	3.3	312.46
2020-01-14	1.84	74.49	13.16	3.43	7.08	0.3	300.3	0.3	340.01	0.3	420.3	0.3	300.3	1.6	412.09
2020-01-15	1.83	74.5	13.35	3.34	6.99	0.3	322.8	0.01	480.36	0.3	408.95	0.3	190.3	0.3	179.16
2020-01-16	1.84	74.1	13.52	3.44	7.09	0.3	112.36	0.3	434.3	0.3	291.35	3.3	207.95	1.3	156.04
2020-01-17	1.81	72.25	14.67	3.95	7.33	0.3	281.3	0.02	769.8	0.3	360.96	3.3	420.3	0.02	351.29
2020-01-18	1.74	67.0	17.86	4.7	8.69	0.31	166.92	0.3	499.8	0.3	387.6	0.3	435.54	3.3	384.66
2020-01-19	1.86	67.2	17.59	4.9	8.46	0.8	160.92	0.3	464.8	0.02	407.8	0.3	499.56	3.3	222.36
2020-01-20	1.84	70.86	15.51	4.23	7.56	0.3	226.3	0.3	480.3	0.3	617.3	0.31	409.89	3.3	270.96
2020-01-21	1.94	74.34	13.22	3.39	7.11	0.3	300.3	0.3	4268.3	0.3	495.3	0.3	498.66	0.3	730.3
2020-01-22	1.91	74.63	13.14	3.31	7.0	0.31	400.0	0.3	1110.8	0.3	294.96	3.3	400.3	3.3	264.2
2020-01-23	1.87	74.11	13.39	3.41	7.23	0.31	194.29	0.3	553.14	0.3	312.88	0.3	400.3	0.3	367.35
2020-01-24	1.96	72.1	14.74	3.8	7.4	0.3	165.3	0.3	735.17	0.3	308.47	0.3	275.69	0.3	367.3
2020-01-25	1.72	66.94	18.04	4.72	8.58	0.3	315.3	0.3	468.22	0.3	173.72	3.3	480.36	3.3	153.8
2020-01-26	1.78	68.53	17.09	4.56	8.04	0.3	186.35	0.3	1722.3	0.3	345.34	3.3	220.8	0.3	217.86
2020-01-27	1.91	74.46	13.06	3.48	7.08	0.3	260.3	0.3	425.92	0.3	240.3	3.15	318.29	0.3	238.55
2020-01-28	1.89	74.62	13.17	3.25	7.08	0.31	143.05	0.01	587.3	0.3	298.12	0.99	260.31	3.3	294.8
2020-01-29	1.84	74.84	13.01	3.27	7.03	3.3	209.17	0.3	600.36	0.3	454.19	3.3	245.3	3.3	227.66
2020-01-30	1.89	74.57	13.13	3.38	7.03	3.3	137.35	0.3	600.3	0.3	243.49	0.3	220.3	0.3	411.82
2020-01-31	1.81	72.73	14.58	3.72	7.15	0.3	159.77	0.3	906.35	0.3	360.36	3.3	135.36	3.3	293.16

Дополнительное задание – аналитика чаевых





Выводы:

- Для поездок с числом пассажиров от 7 данных для анализа недостаточно. Для поездок с числом пассажиров менее 7 величина чаевых практически не зависит от количества пассажиров. Исключения: четверо пассажиров оставляют меньше чаевых (на 5-25%), а один пассажир оставляет больше чаевых (до 10%).
- Для поездок до ~30 миль (для медианы) и до ~50 миль (для среднего) прослеживается прямая зависимость чаевых от расстояния, а дальше начинает сказываться нехватка данных.
- Более подробно см. тетрадку **tip_analysis.ipynb**