NYC-Yellow-Taxi + Spark

Zadania:

ETL – obraz czasu rzeczywistego

Utrzymywanie agregacji na poziomie dnia i dzielnicy. Wartości agregatów to:

- liczba wyjazdów
- liczba przyjazdów
- sumaryczna osób wyjeżdżających
- sumaryczna osób przyjeżdżających

Wykrywanie "anomalii"

Wykrywanie "anomalii" ma polegać na wykrywaniu dużej różnicy w liczbie osób wyjeżdżających z danej dzielnicy w stosunku do liczby przyjeżdżających do danej dzielnicy w określonym czasie. Program ma być parametryzowany przez:

- D - długość okresu czasu wyrażoną w godzinach
- L - liczbę osób (minimalna)

Wykrywanie anomalii ma być dokonywane co godzinę.
Przykładowo, dla parametrów D=4, L=10000 program co godzinę będzie raportował te dzielnice, w których w ciągu ostatnich 4 godzin liczba osób "zmniejszyła się" o co najmniej 10 tysięcy osób. Raportowane dane mają zawierać:

- analizowany okres - okno (start i stop)
- nazwę dzielnicy
- liczbę osób wyjeżdżających
- liczbę osób przyjeżdżających
- różnicę w powyższych liczbach

Uruchomienie krok po kroku:

Uwaga!
Najpierw przeczytaj szczegółowy opis, który znajduje się w dalszej części. Po przeczytaniu opisu wróć do tego fragmentu. Ten rozdział to jedynie to podsumowanie, które pomoże ci w uruchomieniu projektu.

Będą potrzebne 4 terminale - jeden dla producenta, drugi dla przetwarzania Spark, trzeci dla odczytywania obrazów czasu rzeczywistego, czwarty dla odczytywania anomalii.

1. Skrypt ustawiający środowisko.

```
source setup.sh <pathToYellowTripDataResult> <pathToTaxiZoneLookup>
```

Przykładowo:

```
source setup.sh gs://pbd-23-AA/projekt2/yellow_tripdata_result gs://pbd-23-AA/projekt2/taxi_zone_lookup.csv
```

2. Skrypt tworzący tematy Kafki.

```
./manage-topics.sh up
```

3. Skrypt tworzący ujście ETL.

```
./manage-sink.sh up
```

4. Skrypt uruchamiający przetwarzanie.

```
./run-processing.sh A 4 1000
```

Poczekaj chwilę, aż Spark prawidłowo się uruchomi. Nie powinno zająć to dużo czasu, ale chwilę może potrwać - jeżeli pokazują się logi, które dość intensywnie się przewijają i informują o sprawdzaniu tematu Kafki to znaczy że możesz przejść dalej.

5. Skrypt uruchamiający producenta.

```
./run-producer.sh
```

6. Skrypt odczytujący wyniki przetwarzania.

```
./run-consumer-etl.sh
./run-consumer-anomalies.sh
```

7. Skrypt czyszczący środowisko.

```
./clean.sh
```

Po skrypcie czyszczącym środowisko wróć do punktu 2. aby uruchomić przetwarzanie ponownie (np. z innymi parametrami dla skryptu `run-processing.sh`)

Producent i skrypty inicjujące i zasilające

- Utwórz klaster na platformie GCP przy użyciu poniższego polecenia

```
gcloud dataproc clusters create ${CLUSTER_NAME} \
--enable-component-gateway --bucket ${BUCKET_NAME} \
--region ${REGION} --subnet default \
--master-machine-type n1-standard-4 --master-boot-disk-size 50 \
--num-workers 2 --worker-machine-type n1-standard-2 --worker-boot-disk-size 50 \
--image-version 2.1-debian11 --optional-components FLINK,DOCKER,ZOOKEEPER \
--project ${PROJECT_ID} --max-age=3h \
--metadata "run-on-master=true" \
--initialization-actions \
gs://goog-dataproc-initialization-actions-${REGION}/kafka/kafka.sh
```

- Otwórz terminal SSH do maszyny master i wgraj na nią pliki z projektu:
    - setup.sh
    - manage-topics.sh
    - manage-sink.sh (oraz create_tables.sql)
    - run-processing.sh (oraz main.py)
    - run-producer.sh (oraz KafkaProducer.jar)
    - run-consumer-etl.sh
    - run-consumer-anomalies.sh
    - clean.sh
- Skrypt inicjalizujący środowisko. Pobiera on niezbędne biblioteki, dane wejściowe do projektu oraz ustawia zmienne środowiskowe (dlatego uruchamiamy poprzez polecenie source). Skrypt przyjmuje dwa parametry: pierwszy to folder w usłudze Cloud Storage, w którym znajdują się pliki główne (strumieniowe - zbiór pierwszy), a drugi to lokalizacja pliku statycznego (zbiór drugi) w usłudze Cloud Storage.

```
source setup.sh <pathToYellowTripDataResult> <pathToTaxiZoneLookup>
```

Przykładowo:

```
source setup.sh gs://pbd-23-AA/projekt2/yellow_tripdata_result gs://pbd-23-
AA/projekt2/taxi_zone_lookup.csv
```

Wyjście po uruchomieniu skryptu powinno wyglądać mniej więcej tak:

```
Dodano CLUSTER_NAME do .bashrc
Dodano TAXI_STREAM_DATA_PATH do .bashrc
Dodano TAXI_STATIC_DATA_PATH do .bashrc
Dodano KAFKA_TOPIC_PRODUCER do .bashrc
Dodano KAFKA_TOPIC_ANOMALIES do .bashrc
Dodano PGPASSWORD do .bashrc
Tworzenie folderu data
Kopiowanie plików strumieniowych z usługi Cloud Storage...
```

Później logi z kopiowania plików... a zakończenie wyjścia tak:

```
Tworzenie folderu w HDFS do przechowania statycznego pliku...
Kopiowanie pliku statycznego do HDFS...
Pobieranie JDBC do Postgres...
--2024-06-08 19:29:52--  https://jdbc.postgresql.org/download/postgresql-
42.6.0.jar
Resolving jdbc.postgresql.org (jdbc.postgresql.org)... 72.32.157.228, 200
1:4800:3e1:1::228
Connecting to jdbc.postgresql.org (jdbc.postgresql.org)|72.32.157.228|:44
3... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1081604 (1.0M) [application/java-archive]
Saving to: 'postgresql-42.6.0.jar'

postgresql-42.6.0. 100%[==========>]   1.03M  1.82MB/s    in 0.6s

2024-06-08 19:29:53 (1.82 MB/s) - 'postgresql-42.6.0.jar' saved [1081604/
1081604]

Instalowanie tree...
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  tree
0 upgraded, 1 newly installed, 0 to remove and 6 not upgraded.
Need to get 49.6 kB of archives.
After this operation, 118 kB of additional disk space will be used.
Get:1 https://deb.debian.org/debian bullseye/main amd64 tree amd64 1.8.0-
1+b1 [49.6 kB]
Fetched 49.6 kB in 0s (725 kB/s)
Selecting previously unselected package tree.
(Reading database ... 170750 files and directories currently installed.)
Preparing to unpack .../tree_1.8.0-1+b1_amd64.deb ...
Unpacking tree (1.8.0-1+b1) ...
Setting up tree (1.8.0-1+b1) ...
Processing triggers for man-db (2.9.4-2) ...
Ustawianie uprawnień dla skryptów z projektu...
Done!
@pbd-cluster-m:~$
```

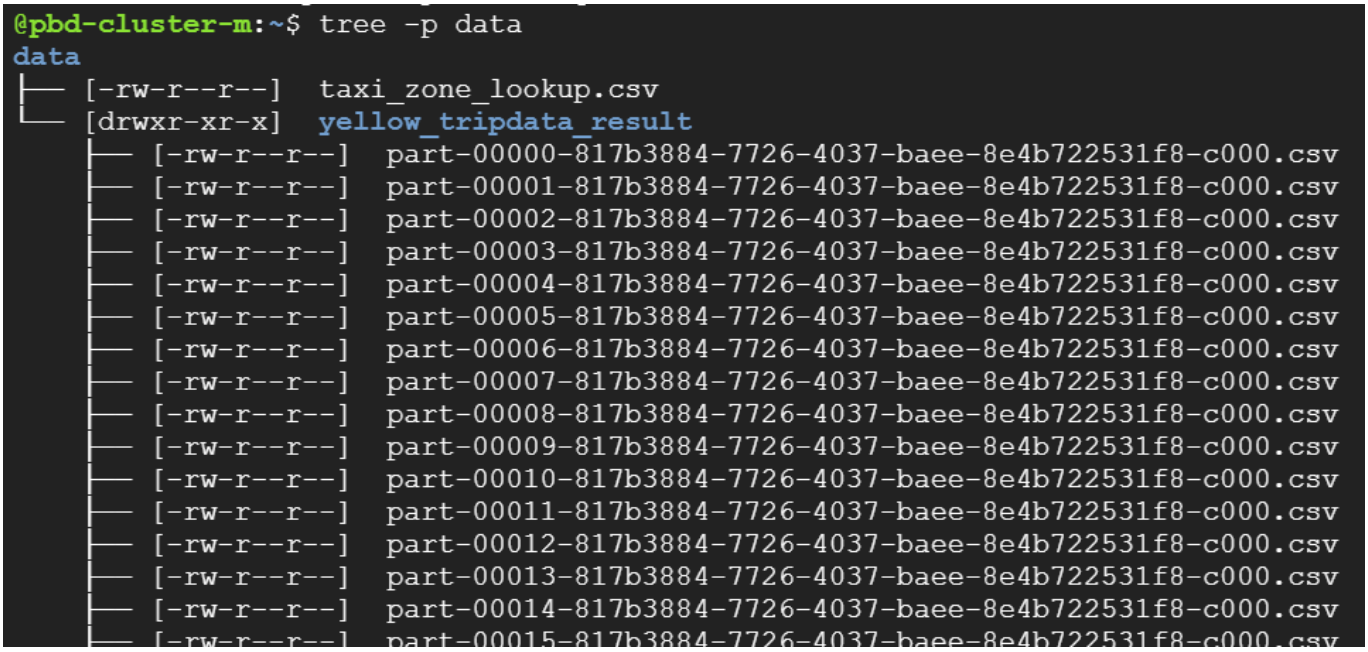
Jeżeli pojawiły się błędy przy pobieraniu pakietu tree (czasami zdarza się tak jeżeli klaster nie zdążył się w pełni zainicjalizować) to nie ma powodu do obaw, ponieważ nie jest on niezbędny. Możesz jednak go doinstalować poleceniem:

```
sudo apt-get install tree
```

Aby sprawdzić czy pliki zostały poprawnie pobrane na maszynę z usługi Cloud Storage możesz użyć polecenia:

```
tree -p data
```

Wynik powinien być dokładnie taki:

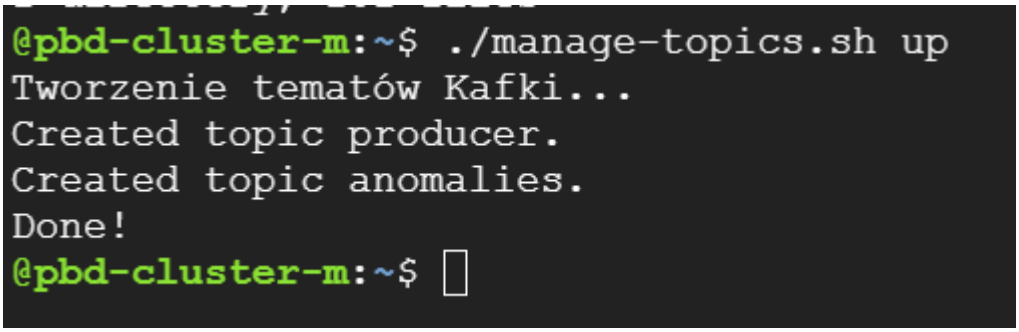


Na końcu polecenia powinieneś/powinnaś zobaczyć napis `1 directory, 101 files`. Inny wynik oznacza, że źle został uruchomiony skrypt inicjujący (ze złymi parametrami). Aby to poprawić uruchom go jeszcze raz z poprawnymi (jest on przystosowany do wielokrotnego uruchamiania i nie powinny wystąpić żadne błędy, a niepoprawnie zapisane pliki w katalogu `data` zostaną nadpisane).

- Skrypt tworzący tematy Kafki (źródłowy temat oraz temat dla anomalii).

```
./manage-topics.sh <up|down>
```

Przykład działania:



- Skrypt czyszczący środowisko.

```
./clean.sh
```

Utrzymanie obrazu czasu rzeczywistego.

Fragmenty kodu programu odpowiadającego za wyliczenia obrazu czasu rzeczywistego od poziomu źródła

```
# ------------------------------------------------------------------------------
---------------------------------- #
# STREAM
# Wczytanie danych z Kafki
ds1 = spark.readStream
.format("kafka")
.option("kafka.bootstrap.servers", f"{host_name}:9092")
.option("subscribe", os.getenv("KAFKA_TOPIC_PRODUCER"))
.load()

# Parsujemy dane aby utworzyć z nich DataFrame z prawidłowymi typami danych
# Metoda to_timestamp zamienia nam czas eventu, który jest typu string na typ
timestamp
valuesDF = ds1.selectExpr("CAST(value as string)")
schema = "tripID STRING, start_stop INT, timestamp STRING, locationID INT,
passenger_count INT, trip_distance DOUBLE,"
" payment_type INT, amount DOUBLE, VendorID STRING"
parsedDF = valuesDF.select(from_csv(valuesDF.value,
schema).alias("data")).select("data.*")
parsedDF = parsedDF.withColumn("timestamp", to_timestamp(col("timestamp"), "yyyy-
MM-dd'T'HH:mm:ss.SSS'Z'"))

# ------------------------------------------------------------------------------
---------------------------------- #
# STATIC
# Wczytujemy dane statyczne z HDFS
taxi_zone_lookup = spark.read.csv(os.getenv("TAXI_STATIC_DATA_PATH"), header=True,
inferSchema=True)
# ColumnRenamed służy temu, aby w dalszej części łatwo połączyć ze sobą dwa
DataFrames
taxi_zone_lookup = taxi_zone_lookup.withColumnRenamed("LocationID", "locationID")

# ------------------------------------------------------------------------------
---------------------------------- #
# JOIN
# Łączenie danych strumieniowych oraz statycznych
joinedDF = parsedDF.join(taxi_zone_lookup, "locationID")

# ------------------------------------------------------------------------------
---------------------------------- #
# WATERMARK
# Dodajemy watermark, ponieważ dane mogą być opóźnione o 1 dzień
watermarkedDF = joinedDF.withWatermark("timestamp", "1 day")
```

```
# ------------------------------------------------------------------------------
---------------------------------- #
# ETL

# AGREGACJE
# Grupujemy po oknie jednodniowym oraz po kolumnie borough
# Następnie wyliczamy agregaty - dwa razy count oraz dwa razy sum. Funkcja
coalesce służy zapobieganiu wartości NULL,
# które występowały kiedy w danej grupie nie było np. odjazdów (nie było żadnego
eventu z wartością start_stop = 0).
# Funkcja lit(0) służy wpisaniu w takim przypadku wartości 0 do tej kolumny.
# - num_departures: zliczamy ile razy wystąpiła wartość start_stop równa 0
# - num_arrivals: zliczamy ile razy wystąpiła wartość start_stop równa 1
# - total_departing_passengers: sumujemy ile pasażerów było w eventach gdzie
start_stop równe 0
# - total_arriving_passengers: sumujemy ile pasażerów było w eventach gdzie
start_stop równe 1
aggregatedDF = watermarkedDF.groupBy(
    window(col("timestamp"), "1 day").alias("date"),
    "Borough"
).agg(
    coalesce(count(when(col("start_stop") == 0, True)),
lit(0)).alias("num_departures"),
    coalesce(count(when(col("start_stop") == 1, True)),
lit(0)).alias("num_arrivals"),
    coalesce(_sum(when(col("start_stop") == 0, col("passenger_count"))),
lit(0)).alias("total_departing_passengers"),
    coalesce(_sum(when(col("start_stop") == 1, col("passenger_count"))),
lit(0)).alias("total_arriving_passengers")
)
```

Utrzymanie obrazu czasu rzeczywistego - obsługa trybu A i C

```
# UJŚCIE ETL
# W zależności od wybranego trybu działania
# - A - tryb update (aktualizacja danych)
# - C - tryb append (dopisywanie danych)
output_mode = None
if mode == 'A':
    output_mode = "update"
elif mode == 'C':
    output_mode = "append"

# Korzystamy z metody foreachBatch i dla każdego batcha danych wybieramy
odpowiednie dane (przy okazji zmieniamy datę
# na string, ponieważ taki typ jest w bazie danych dla tej kolumny) oraz nazwę
Borough na borough, żeby wszystkie
```

```
# kolumny zaczynały się małą literą (trzymamy się jednej konwencji nazewnictwa).

# W późniejszej części wysyłamy dane do postgresa, który dostępny jest na porcie
54320 hosta.

# Dodatkowo zapisujemy checkpoint, aby w przypadku awarii przetwarzanie zaczęło
się od odpowiedniego momentu.
query_etl = aggregatedDF.writeStream.outputMode(output_mode).foreachBatch(
    lambda df_batch, batch_id:
    df_batch.select(
        col("date").cast("string").alias("day"),
        col("Borough").alias("borough"),
        col("num_departures"),
        col("num_arrivals"),
        col("total_departing_passengers"),
        col("total_arriving_passengers")
    ).write
    .format("jdbc")
    .mode("append")
    .option("url", f"jdbc:postgresql://{host_name}:54320/streamoutput")
    .option("dbtable", "taxi_etl")
    .option("user", "postgres")
    .option("password", os.getenv("PGPASSWORD"))
    .save()
).option("checkpointLocation", "/tmp/etl").start()
```

Wykrywanie anomalii

```
# AGREGACJE
# Podobnie jak w przetwarzaniu ETL najpierw grupujemy, a później wyliczamy
agregaty.
# W tym przypadku grupujemy po oknie o długości zależnej od parametru D oraz z
przeskokiem co 1 godzinę.
# Dodatkowo grupujemy też po atrybucie Borough.
# Następnie dokonujemy sumy podobnie jak dla ETL pasażerów którzy wyjeżdżają i
przyjeżdżają w danej grupie.

# W dalszej części wyliczamy kolumnę "difference", która jest różnicą pomiędzy
pasażerami wyjeżdżającymi i przyjeżdżającymi
# oraz filtrujemy, aby otrzymać tylko takie przypadki dla których wartość
difference jest większa od
# zadanego progu - parametru L
anomalyDF = watermarkedDF.groupBy(
    window(col("timestamp"), f"{D} hours", "1 hour").alias("anomaly_window"),
    "Borough"
).agg(
    coalesce(_sum(when(col("start_stop") == 0, col("passenger_count"))),
lit(0)).alias("total_departing_passengers"),
    coalesce(_sum(when(col("start_stop") == 1, col("passenger_count"))),
lit(0)).alias("total_arriving_passengers")
```

```
).withColumn(
    "difference", col("total_departing_passengers") -
col("total_arriving_passengers")
).filter(
    col("difference") >= L
)

# Wybieramy odpowiednie dane (być może wystarczyłby tutaj zwykły select, a nie
selectExpr, ale wydaje się działać więc już nie zmieniam :)
# W dalszej części pakujemy dane do jsona i wysyłamy do kafki.

# Dodany jest również checkpoint
query_anomaly = anomalyDF.selectExpr(
    "anomaly_window",
    "Borough AS borough",
    "total_departing_passengers",
    "total_arriving_passengers",
    "difference"
).selectExpr(
    "to_json(struct(*)) AS value"
).writeStream
.format("kafka")
.option("kafka.bootstrap.servers", f"{host_name}:9092")
.option("topic", os.getenv("KAFKA_TOPIC_ANOMALIES"))
.option("checkpointLocation", "/tmp/anomalies")
.start()
```

Program przetwarzający strumienie danych, skrypt uruchamiający

- Skrypt przetwarzający dane w wersji A
    - Anomalie nie występują

```
./run-processing.sh A 1 1000000
```

- Uruchom skrypt przetwarzający dane w wersji C
    - Anomalie występują stosunkowo często

```
./run-processing.sh C 2 1000
```

Miejsce utrzymywania obrazów czasu rzeczywistego – skrypt tworzący

```
./manage-sink.sh up
```

Przykład poprawnego działania:

```
@pbd-cluster-m:~$ ./manage-sink.sh up
Tworzenie ujścia ETL...
Unable to find image 'postgres:latest' locally
latest: Pulling from library/postgres
09f376ebb190: Pull complete
119215dfb3e3: Pull complete
e02bbc8c8252: Pull complete
061f31803c55: Pull complete
accd4903f49a: Pull complete
2016ff8e6e3a: Pull complete
088e651df7e9: Pull complete
ed155773e5e0: Pull complete
ffebb35d2904: Pull complete
293f0bec643a: Pull complete
1655a257a5b5: Pull complete
4ddba458499d: Pull complete
90e48ae03559: Pull complete
822c1a513e6a: Pull complete
Digest: sha256:1bf73ccae25238fa555100080042f0b2f9be08eb757e200fe6afc1fc413a1b3c
Status: Downloaded newer image for postgres:latest
8a4ba71b9f855fbf03a2e266eef3ddf259ecf8e123a82b49df476199fdcc241e
Done!
```

Możesz sprawdzić czy ujście dla ETL działa prawidłowo poleceniem:

```
docker ps
```

Przykład poprawnego działania:

```
@pbd-cluster-m:~$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED              STATUS              PORTS                                         NAMES
8a4ba71b9f85   postgres   "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0.0:54320->5432/tcp, :::54320->5432/tcp   postgresik
@pbd-cluster-m:~$
```

Miejsce utrzymywania obrazów czasu rzeczywistego – cechy

Jako miejsce utrzymywania obrazów czasu rzeczywistego zdecydowałem się na wybranie systemu zarządzania bazą danych Postgres. Cechy:

- zdecydowana większość narzędzi (programów czy języków programowania) posiada możliwość integracji z tą bazą danych i dzięki temu rozszerzenie projektu o dodatkowe funkcjonalności nie stanowiłoby problemu
- PostgreSQL jest jednym z najlepszych systemów zarządzania bazą danych (zgodnie z rankingiem db-engines https://db-engines.com/en/ranking)
- PostgreSQL jest oprogramowaniem typu OpenSource
- obsługa dużych zbiorów danych jest możliwa w PostgreSQL dzięki mechanizmom partycjonowania tabel. Pozwalają one na dzielenie dużych tabel na mniejsze oraz łatwiejsze do zarządzania części.

Konsument: skrypt odczytujący wyniki przetwarzania

Obraz czasu rzeczywistego:

```
./run-consumer-etl.sh
```

Przykład działania:

```
@pbd-cluster-m:~$ ./run-consumer-etl.sh
                   day                    |    borough    | num_departures | num_arrivals | total_departing_passengers | total_arriving_passengers
------------------------------------------+---------------+----------------+--------------+----------------------------+--------------------------
 {2008-12-31 00:00:00, 2009-01-01 00:00:00} | Queens        |             13 |            3 |                         18 |                        3
 {2008-12-31 00:00:00, 2009-01-01 00:00:00} | Unknown       |              1 |            3 |                          1 |                        3
 {2009-01-01 00:00:00, 2009-01-02 00:00:00} | Queens        |              8 |            2 |                          8 |                        2
 {2009-01-01 00:00:00, 2009-01-02 00:00:00} | Manhattan     |             29 |           26 |                         53 |                       45
 {2003-12-23 00:00:00, 2003-12-24 00:00:00} | Manhattan     |              1 |            1 |                          1 |                        1
 {2008-12-31 00:00:00, 2009-01-01 00:00:00} | Manhattan     |             33 |           30 |                         75 |                       69
 {2009-01-01 00:00:00, 2009-01-02 00:00:00} | Brooklyn      |              0 |            3 |                          0 |                        3
 {2008-12-31 00:00:00, 2009-01-01 00:00:00} | Bronx         |              1 |            1 |                          1 |                        1
 {2009-01-01 00:00:00, 2009-01-02 00:00:00} | Unknown       |              1 |            0 |                          1 |                        0
 {2008-12-31 00:00:00, 2009-01-01 00:00:00} | Brooklyn      |              1 |            0 |                          1 |                        0
 {2008-12-31 00:00:00, 2009-01-01 00:00:00} | EWR           |              0 |            1 |                          0 |                        1
 {2018-11-02 00:00:00, 2018-11-03 00:00:00} | Brooklyn      |            718 |         2965 |                       1080 |                     4569
 {2018-11-02 00:00:00, 2018-11-03 00:00:00} | Staten Island |              2 |           17 |                          2 |                       29
 {2018-10-31 00:00:00, 2018-11-01 00:00:00} | Unknown       |              1 |            0 |                          1 |                        0
 {2018-11-01 00:00:00, 2018-11-02 00:00:00} | Queens        |          18487 |        13876 |                      29378 |                    21383
 {2018-11-01 00:00:00, 2018-11-02 00:00:00} | Bronx         |            488 |         2024 |                        744 |                     3146
 {2018-11-01 00:00:00, 2018-11-02 00:00:00} | Unknown       |           5354 |         5321 |                       7209 |                     7181
 {2018-11-01 00:00:00, 2018-11-02 00:00:00} | Staten Island |             17 |           91 |                         26 |                      152
 {2018-11-02 00:00:00, 2018-11-03 00:00:00} | Bronx         |             90 |          426 |                        134 |                      667
 {2009-01-01 00:00:00, 2009-01-02 00:00:00} | Queens        |             11 |            7 |                         15 |                       11
 {2018-10-31 00:00:00, 2018-11-01 00:00:00} | Queens        |             12 |            2 |                         33 |                       11
 {2018-11-01 00:00:00, 2018-11-02 00:00:00} | Manhattan     |         284238 |       274434 |                     441989 |                   427012
 {2018-11-01 00:00:00, 2018-11-02 00:00:00} | Brooklyn      |           4338 |        13487 |                       6651 |                    21087
 {2009-01-01 00:00:00, 2009-01-02 00:00:00} | Manhattan     |             71 |           80 |                        104 |                      117
 {2018-10-28 00:00:00, 2018-10-29 00:00:00} | Manhattan     |              0 |            1 |                          0 |                        1
 {2018-11-02 00:00:00, 2018-11-03 00:00:00} | EWR           |              3 |           86 |                          2 |                      147
 {2009-01-02 00:00:00, 2009-01-03 00:00:00} | Manhattan     |              0 |            1 |                          0 |                        2
 {2018-10-31 00:00:00, 2018-11-01 00:00:00} | Manhattan     |            249 |           72 |                        454 |                      135
 {2018-11-02 00:00:00, 2018-11-03 00:00:00} | Queens        |           1892 |         2911 |                       2867 |                     4615
 {2018-10-31 00:00:00, 2018-11-01 00:00:00} | Brooklyn      |              7 |            2 |                         12 |                        3
 {2009-01-01 00:00:00, 2009-01-02 00:00:00} | Brooklyn      |              2 |            6 |                          6 |                       11
 {2018-11-02 00:00:00, 2018-11-03 00:00:00} | Unknown       |            497 |          575 |                        718 |                      817
 {2009-01-01 00:00:00, 2009-01-02 00:00:00} | Unknown       |              1 |            2 |                          1 |                        4
 {2018-11-01 00:00:00, 2018-11-02 00:00:00} | EWR           |             27 |          515 |                         46 |                      840
 {2018-11-02 00:00:00, 2018-11-03 00:00:00} | Manhattan     |          29451 |        26526 |                      46415 |                    41876
```

Wykrywanie anomalii:

```
./run-consumer-anomalies.sh
```