Zespół 1408

Bartosz Rożek, Anna Urbala

Cel projektu

Celem projektu jest stworzenie systemu do automatycznego pobierania danych o tramwajach w Warszawie i analizy ich w kontekście osiąganych prędkości. Takie informacje mogą być przydatne szczególnie dla osób o mniejszej sprawności fizycznej, gdyż wysokie prędkości osiągane w środkach transportu publicznego mogą być dla nich niebezpieczne. Dzięki zagregowanym danym mogliby się dowiedzieć m.in. które tramwaje są najszybsze.

Zbiory danych

Położenie tramwajów

Dane są pobierane cyklicznie z użyciem API https://api.um.warszawa.pl/. Dane są w formie JSON, odświeżane są co ok. 10 sekund, zawierają informacje o aktualnym położeniu tramwajów, w jednym batchu znajduje się kilkaset rekordów.
Dostępne pola:

- Lines (string) - numer linii
- VehicleNumber (string) - numer pojazdu
- Brigade (string) - brygada
- Time (string) - czas pomiaru
- Lon (double) - długość geograficzna
- Lat (double) - szerokość geograficzna

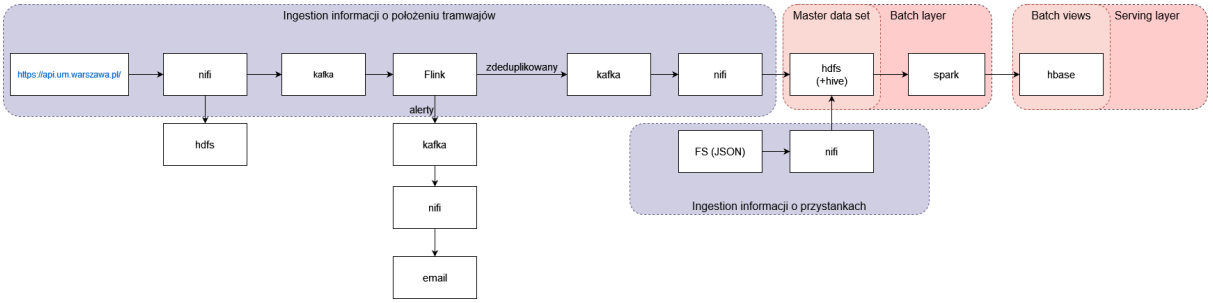
Przystanki

Dane również pochodzą ze strony https://api.um.warszawa.pl/, jednak zostały przez nas zdumpowane w statycznym pliku JSON ze względu na to, że jednak bardzo rzadko ulegają zmianom (plik można oczywiście w każdym momencie zaktualizować, używając komendy np. curl). Ich format to również JSON, ale znacznie mniej przyjazny dla użytkownika - każda informacja o 1 przystanku to tablica słowników z 2 kluczami: “key” i “value”, gdzie wartościami są odpowiednio nazwa i wartość pola. Na szczęście schemat zawsze jest taki sam i finalne pola to (wszystkie źródłowo są typem string):

- zespol - numer zespołu przystankowego (grupy przystanków)
- slupek - numer identyfikujący słupek
- nazwa_zespolu - nazwa zespołu przystankowego
- id_ulicy - identyfikator ulicy
- szer_geo - szerokość geograficzna położenia słupka

- dług_geo - długość geograficzna położenia słupka
- kierunek - kierunek jazdy pojazdów zatrzymujących się przy danym słupku
- obowiazuje_od - od kiedy funkcjonuje dany przystanek

Architektura



Podstawą naszego rozwiązania jest dość rozbudowane ingestion informacji o położeniach tramwajów. NiFi w regularnych odstępach czasu odpytuje endpoint, pobiera zestaw ostatnich lokalizacji i pojedynczo wrzuca je na Kafkę. W celu uniknięcia duplikatów (które są spowodowane charakterystyką API, które po prostu zwraca zestaw ostatnich lokalizacji, więc jeśli odpytujemy API częściej niż odświeżają się lokalizacje, to możemy dostać tę samą informację kilka razy) zaimplementowaliśmy deduplikację w Apache Flink. Pozwoliliśmy sobie tutaj też na dodanie “micro speed layer” (autorska nazwa)*. Zdeduplikowane we Flinku rekordy są znowu wrzucane na Kafkę, następnie pobierane przez NiFi i zapisywane w external table w Hive. Warto zauważyć, że dane po drodze nie ulegają praktycznie żadnym przekształceniom poza zmianą formatu z JSON na Parquet.

Ogólna logika ingestion przystanków jest dość prosta. Jeśli mamy potrzebę zaktualizowania listy przystanków, wystarczy, że pobierzemy plik (co można zrobić z poziomu NiFi) i uruchomimy flow, które przetwarza i zapisuje dane na HDFSie.

W batch layer użyliśmy narzędzia Apache Spark. Skrypty pysparkowe są wywoływane w NiFi cyklicznie lub na żądanie. Takich skryptów mamy 2. Każdy odpowiada za liczenie innych statystyk. Pierwszy wylicza chwilową prędkość tramwaju i łączy ją z pobliskim zespołem przystanków. Drugi wylicza dzienne statystyki linii - średnią prędkość, godziny kursowania, liczbę wypuszczonych brygad.

Dane procesowane przez Sparka zapisywane są w bazie HBase. Ze względu na charakterystykę statystyk (mają różne grupowanie) każda statystyka znajduje się w oddzielnej tabeli.

* Flink poza deduplikacją również przelicza chwilową prędkość tramwajów i jeśli przekroczona została prędkość 50 km/h, wrzuca informację o takim tramwaju na Kafkę. Taka informacja jest pobierana przez NiFi i wysyłana na maila.

Operacje na danych wejściowych

Położenie tramwajów

Dane o położeniu tramwajów zdobywamy poprzez regularne wywołanie API. Na podstawie JSONa wyodrębniamy numer linii i przy wrzucaniu rekordu na Kafkę używamy numeru linii jako klucza (jest to ważne w przypadku, gdy zależy nam na zachowaniu kolejności eventów dla danej linii - a zależy przy Flinku). Dodatkowo wszystkie eventy źródłowe są dumpowane na HDFSie. Następnie Flink pobiera rekordy, deduplikuje je i przyrównuje bieżące eventy do poprzednich w celu wyliczenia obecnej prędkości. Jeśli prędkość 50km/h jest przekroczona, używamy side outputu do przekazania tej wiadomości na osobny topic Kafki. Wszystkie eventy po deduplikacji również są wrzucane na Kafkę, następnie NiFi je sczytuje, zmienia format na Parquet, merguje w większe pliki i zapisuje na HDFSie, partycjonując po dacie. Do tej lokalizacji odwołuje się tabela externalna w Hive.

Przystanki

Przypadek przystanków jest mniej złożony w kontekście wykorzystania różnych technologii, ale ma bardziej skomplikowane flow, ze względu na mało przyjazny format wejściowy. Przykładowy rekord:


```
{"values":[{"value":"1001","key":"zespol"},{"value":"01","key":"sl
upek"},{"value":"Kijowska","key":"nazwa_zespolu"},{"value":"2201",
"key":"id_ulicy"},{"value":"52.248455","key":"szer_geo"},{"value":
"21.044827","key":"dlug_geo"},{"value":"al.Zieleniecka","key":"kie
runek"},{"value":"2021-09-01 00:00:00.0","key":"obowiazuje_od"}]}
```


Ten format wymagał od nas podzielenia jsona na pojedyncze rekordy, wyodrębnienia wartości dla każdego pola (na szczęście kolejność pól w słowniku jest stała), zapisania ich do csv, zmergowania zawartości kolejnych FlowFile i finalnie skonwertowania do Avro i zapisania na HDFSie w tabeli externalnej.

Opis widoków wsadowych

W wyniku pracy powstały dwa widoki wsadowe:

- **tramsStats** - tabela jest indeksowana numerem linii tramwajowej i posiada trzy rodziny kolumn:
    - “infoTram": ["vehiclenumber","brigade"],
    - "infoTeam": ["zespol","nazwa_zespolu"],
    - "stats": ["velocity", "lon", "lat", "distance", "time", "timeDiff"]

  Kolejne kroki do stworzenia widoku:
    1. W tabeli *trams* dodajemy nowe kolumny zawierające poprzednie położenie i czas tramwaju.
    2. Obliczamy przemieszczenie [metry] i zmianę czasu [milisekundy zamienione na sekundy], na podstawie których wyznaczamy prędkość chwilową (zakładamy, że czas między pomiarami jest niewielki, więc można pominąć wpływ zakrzywienia trasy na wyniki obliczeń).

3. Tabelę *stops* ograniczamy do obszaru na którym poruszają się tramwaje (wiedza ekspercka) i wyznaczamy środek ciężkości położenia danego zespołu obsługującego przystanki.
4. Używamy *crossjoin* aby połączyć tabele.
5. Na połączonej tabeli wyznaczamy odległość geograficzną między danym pomiarem chwilowym a każdym zespołem.
6. Dla każdego pomiaru chwilowego wybieramy z tabeli rekord z najmniejszą odległością policzoną w poprzednim kroku.

W wyniku tego otrzymujemy tabelę, która zawiera pomiary chwilowe tramwajów, wraz z prędkością i przebytym dystansem a także informacje o zespole na terenie którego wykonany był pomiar. Wszystkie kolumny to:

- numer pojazdu
- id brygady
- id zespołu
- nazwa zespolu
- prędkość chwilowa
- szerokość geograficzna
- długość geograficzna
- dystans przebyty od ostatniego pomiaru
- czas pomiaru
- czas, który upłynął od ostatniego pomiaru

```
hbase(main):005:0> get 'tramsStats', '9'
COLUMN                                CELL

 infoTeam:nazwa_zespolu               timestamp=2022-01-19T20:34:10.760, value='Krucza'
 infoTeam:zespol                      timestamp=2022-01-19T20:34:10.760, value='7033'
 infoTram:brigade                     timestamp=2022-01-19T20:34:10.760, value='35'
 infoTram:vehiclenumber               timestamp=2022-01-19T20:34:10.760, value='3119'
 stats:distance                       timestamp=2022-01-19T20:34:10.760, value='236.97762808297793'
 stats:lat                            timestamp=2022-01-19T20:34:10.760, value='52.23097'
 stats:lon                            timestamp=2022-01-19T20:34:10.760, value='21.01685'
 stats:time                           timestamp=2022-01-19T20:34:10.760, value='1642025929000.0'
 stats:timeDiff                       timestamp=2022-01-19T20:34:10.760, value='32.0'
 stats:velocity                       timestamp=2022-01-19T20:34:10.760, value='7.40555087759306'
```

- **linesStats** - tabela jest indeksowana numerem linii i posiada dwie rodziny kolumn:
  - "stats": ["mean_velocity","number_of_brigades"],
  - "dayStats": ["min_time","max_time", "day"]

  Kroki do stworzenia widoku:
  1. Kroki 1 i 2 tak jak w poprzednim widoku.
  2. Agregujemy tabelę po linii tramwajowej, biorąc średnią z prędkości, liczbę brygad, minimalny czas, maksymalny czas, mimalny dzień (co w naszym przypadku oznacza po prostu dzień pomiaru, ponieważ agregujemy wyniki tylko z jednego dnia).
  3. Zmiana minimalnego i maksymalnego czasu z formatu timestamp na format *yyyy-MM-dd HH:mm:ss.*

  Dzięki tym obliczeniom uzyskujemy tabelę, która agreguje pomiary dla danej linii tramwajowej:
  - średnią prędkość
  - liczbę brygad
  - pierwszy pomiar z dnia
  - ostatni pomiar z dnia
  - dzień pomiaru

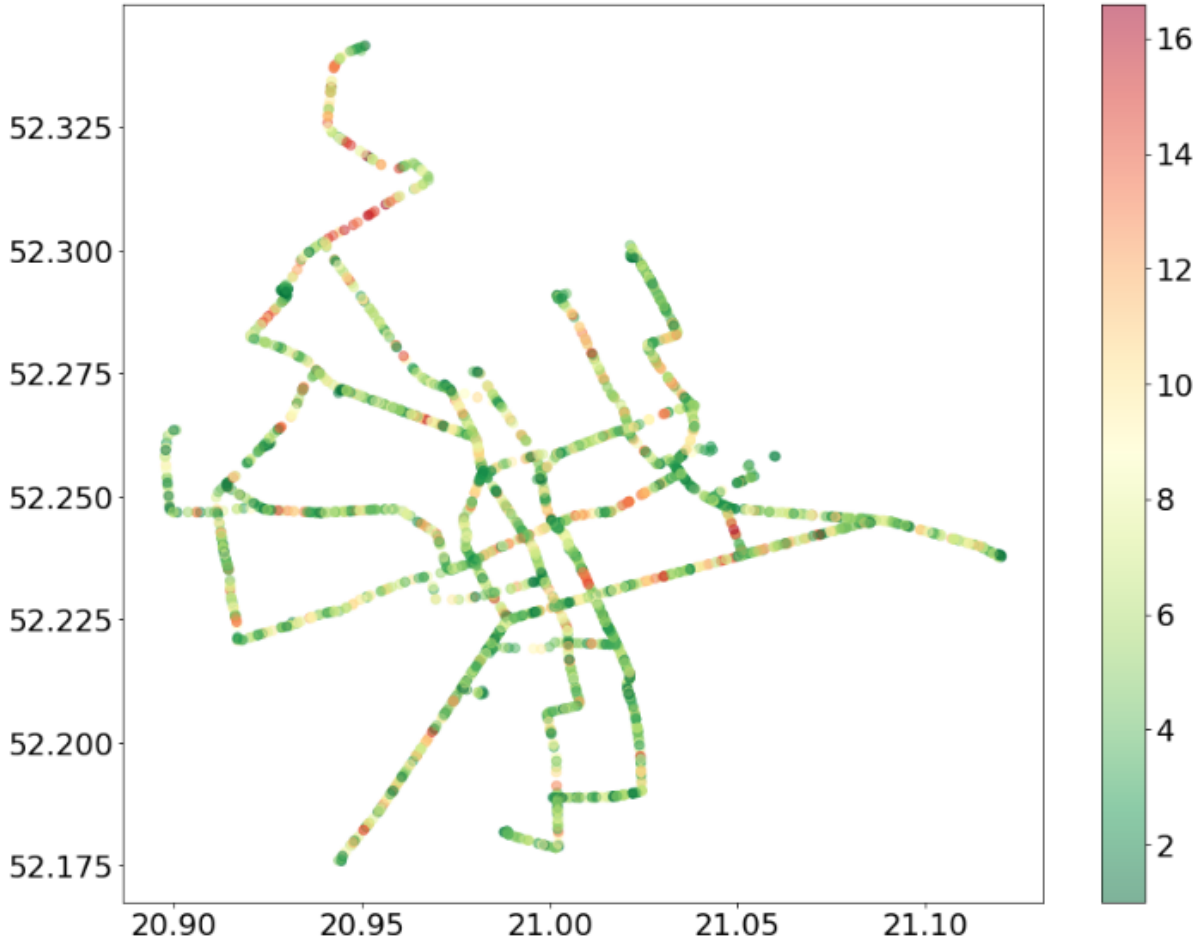
```
hbase(main):020:0> get 'linesStats', '9'
COLUMN                                          CELL

 dayStats:day                                   timestamp=2022-01-20T20:40:41.113, value=2022-01-19
 dayStats:max_time                              timestamp=2022-01-20T20:40:41.113, value=2022-01-20 04:12:15
 dayStats:min_time                              timestamp=2022-01-20T20:40:41.113, value=2022-01-20 01:12:20
 stats:mean_velocity                            timestamp=2022-01-20T20:40:41.113, value=4.025956956174726
 stats:number_of_brigades                       timestamp=2022-01-20T20:40:41.113, value=5602
```

Przykłady danych dostępnych dla warstwy prezentacyjnej

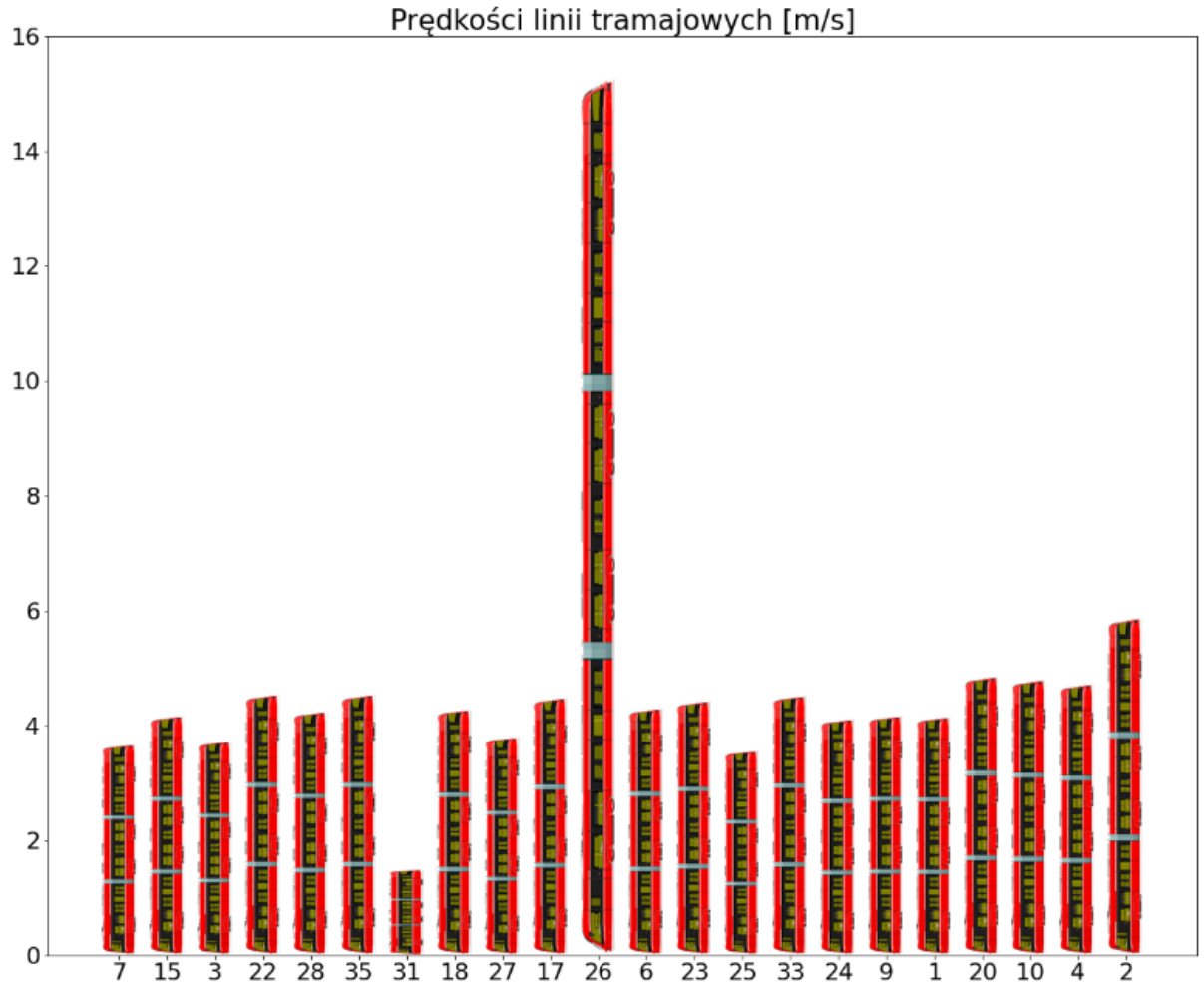
Wizualizacja 1:

Wizualizacja powstała przy użyciu danych z tabeli HBase *tramsStats*. Przedstawia położenie danych pomiarów chwilowych razem z osiągniętą prędkością. Jak można było przypuszczać, tramwaje największe prędkości rozwijają na prostych odcinkach.

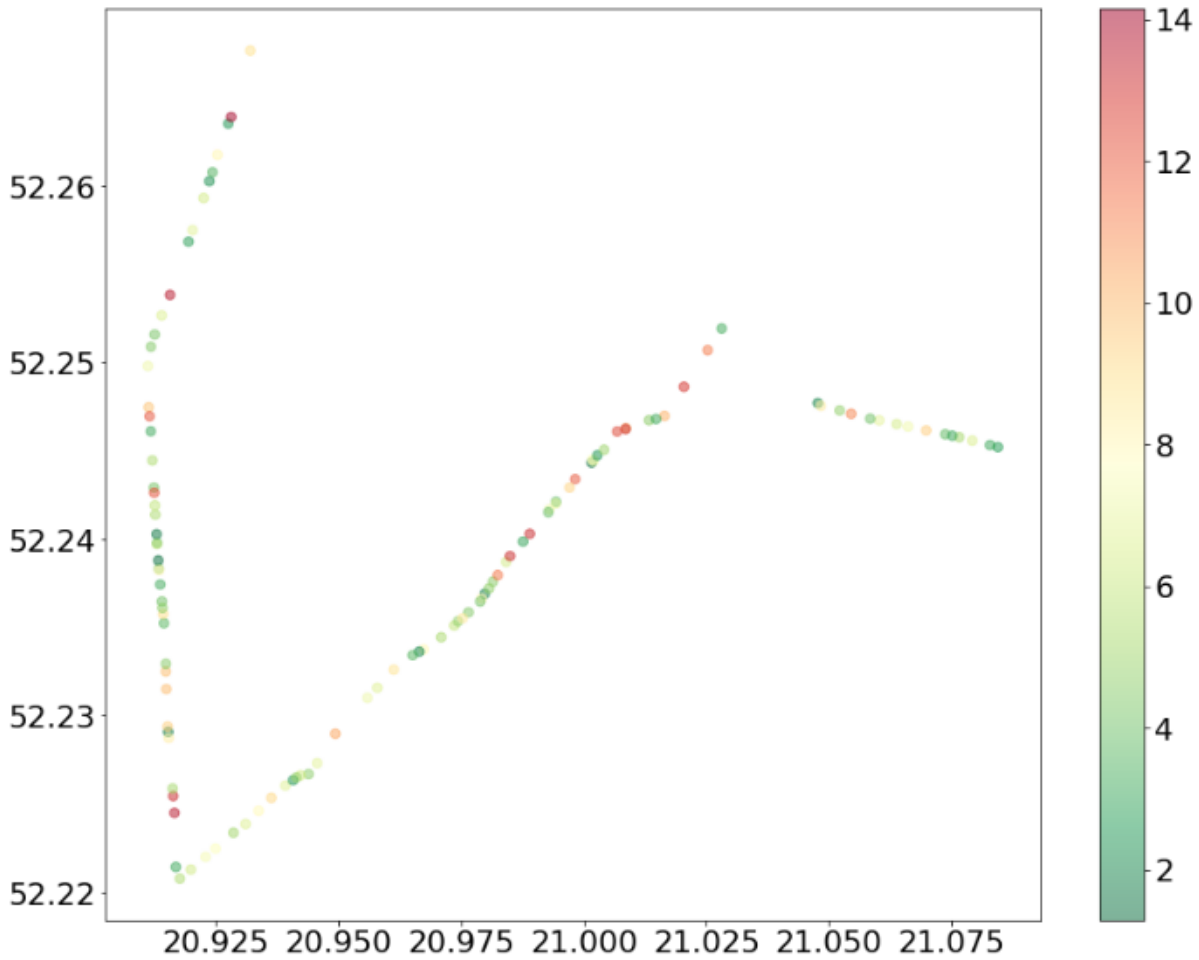


Wizualizacja 2:

Na wykresie widzimy zestawienie porównujące średnie prędkości z dnia dla danej linii tramwajowej. Te dane dostępne są w HBase w tabeli *linesStats*.

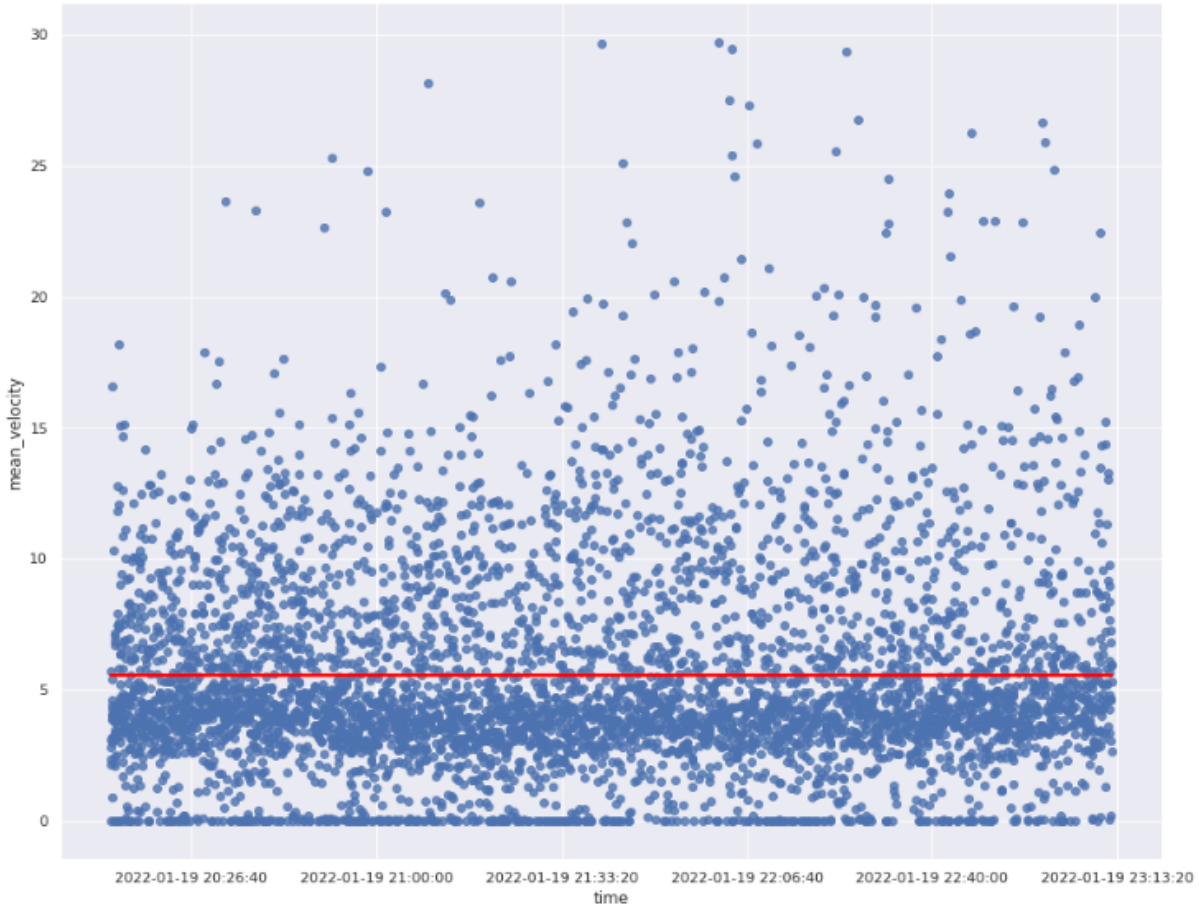


Jak widzimy, linia 26 wyróżnia się znacznie wyższą średnią. Postanowiliśmy to sprwadzić.



Wykres przedstawia trasę lini 26 i jak możemy zaobserwować, wcześniejszy wynik wynika z tego, że jej trasa składa się praktycznie z trzech prostych, więc tramwaj traci mniej czasu na skrzyżowaniach.

Wizualizacja 3:



Wykres przedstawia średnią prędkość tramwajów na dany pomiar chwilowy, dodatkowo do danych została dopasowana regresja liniowa. Jak widzimy, czas pomiaru nie ma wpływu na prędkość tramwaju, co pokazuje, że jest to dobra alternatywa do autobusów, które znacząco zależą od aktualnego ruchu komunikacyjnego.

Podsumowanie

Naszym zdaniem rozwiązanie jest całkiem *eleganckie* i wydajne. Jest możliwość rozszerzenia go o inne dane, gdyby zaszła taka konieczność. Gdyby inne miasta wystawiły analogiczne dane, łatwo można przeskalować rozwiązanie, aby wydajność była wciąż satysfakcjonująca. Dodatkowo temat, choć może wydawać się błahy, jest naszym zdaniem całkiem ciekawy.

Podział pracy w zespole

Bartosz Rożek - batch processing w sparku, warstwa prezentacyjna, dokumentacja
Anna Urbala - pobieranie danych, architektura, deduplikacja, dokumentacja