

Big Data

Dokumentacja projektu

Jakub Kozieł, Tomasz Krupiński, Jakub Lis

Spis treści

1	Cel projektu	2
2	Zbiory danych	2
3	Architektura	4
3.1	Pobieranie danych <i>Station Status</i>	5
3.2	Pobieranie danych <i>Station Information</i>	6
3.3	Pobieranie danych <i>Daily Weather</i>	7
3.4	Pobieranie danych <i>Hourly Weather</i>	8
3.5	Konwersja plików	9
3.6	Zapisywanie plików	10
3.7	Autmatyczne wywoływanie tworzenia się i aktualizacji batchviews	10
4	Testy	11
4.1	Odkładanie plików w HDFS	11
4.2	Batchview	13
4.3	Zapisywanie analiz do HDFS	15
5	Wizualizacje	16
5.1	Wykres dostępności rowerów na danej stacji	16
5.2	Mapa ze stacjami	17
5.3	Wykres dostępności docków na danej stacji	20
5.4	Wykresy parametrów pogodowych	21

1 Cel projektu

Celem projektu było przygotowanie mechanizmu pozyskiwania i wstępnego przetwarzania danych oraz wykonanie analiz wsadowych. Nasze rozwiązanie bazuje na architekturze Lambda. Zbierane przez nas dane dotyczyły wypożyczeń rowerów oraz pogody w Nowym Jorku.

Dostarczane przez nas rozwiązanie, zapewnia szereg benefitów z biznesowego punktu widzenia. Prowadzenie wypożyczeń rowerów przez firmę CitiBike jest usługą nastawioną na generowanie przychodów dla firmy oraz podwyższanie satysfakcji użytkowników.

Aby ułatwić trafne podejmowanie strategicznych decyzji, chcemy zaoferować rozwiązanie pozwalające na przechowywanie danych historycznych wielkiej skali oraz umożliwić łatwy wgląd w nie poprzez zaoferowanie szeregu analiz. Ich szybkie wykonywanie będzie natomiast zagwarantowane dzięki możliwościom zastosowanej architektury Lambda. Korzyści zostaną wskazane przy prezentowaniu analiz, aby łatwiej je zauważyć oraz dać podstawy pod istnienie takowej analizy.

Kod rozwiązania został umieszczony w repozytorium na Githubie: <https://github.com/jakubkoziel/BikesBigData>.

2 Zbiory danych

Zdecydowaliśmy się użyć następujących zbiorów danych:

1. https://github.com/MobilityData/gbfs/blob/master/gbfs.md#station_statusjson
Zbiór ten zawiera dane dotyczące stacji rowerowych w Nowym Jorku. Odświeżany jest co pięć sekund, więc odpytywanie go właśnie z taką częstotliwością w naszym projekcie pozwoliłoby nam na pokazanie "na żywo" zachodzących zmian. Ze względu na zasoby pamięciowe naszych komputerów zdecydowaliśmy się jednak na odświeżanie pobierania danych co 12 minut, z wyjątkiem paru godzin, w których rzeczywiście pobieraliśmy dane co 10 sekund, aby mieć możliwość dokładniejszych analiz. Dostępne API zwraca między innymi takie informacje jak:

- station_id - identyfikator stacji
- num_vehicles_available - liczba rowerów dostępnych do wypożyczenia na tej stacji
- num_vehicles_disabled - liczba rowerów niedostępnych, ale nadal zadokowanych na tej stacji (np. zespuste rowery)
- num_docks_available - liczba wolnych dostępnych miejsc na rowery
- num_docks_disabled - liczba wyłączenych z użytku, zesputych doków

Pozostałe kolumny pozwalają m.in. na uzyskanie informacji, czy dana stacja jest aktywna. Osobno śledzona jest dostępność rowerów elektrycznych.

2. https://github.com/MobilityData/gbfs/blob/master/gbfs.md#station_informationjson
Zbiór ten należy do tego samego źródła, co poprzedni. Jest on również odświeżany co pięć sekund, jednak pobieramy go raz dziennie. Argumentowaliśmy to danymi, na których nam zależało:

- station_id - identyfikator stacji

- name - nazwa stacji
- lat - szerokość geograficzna stacji
- lon - długość geograficzna stacji
- capacity - liczba miejsc do zatrzymania rowerów

Uznaлиmy więc, że te dane nie będą zmieniać się częściej niż raz dziennie.

3. https://open-meteo.com/en/docs#latitude=40.71&longitude=-74.01&daily=temperature_2m_max,temperature_2m_min,apparent_temperature_max,apparent_temperature_min,precipitation_sum,rain_sum,showers_sum,snowfall_sum,windspeed_10m_max,windgusts_10m_max&timezone=America%2FNew_York

Jest to zbiór danych z drugiego źródła, zawierający historyczne dane pogodowe oraz prognozy. Pobieramy codziennie historyczne dane na poprzedni dzień. API zwraca następujące informacje:

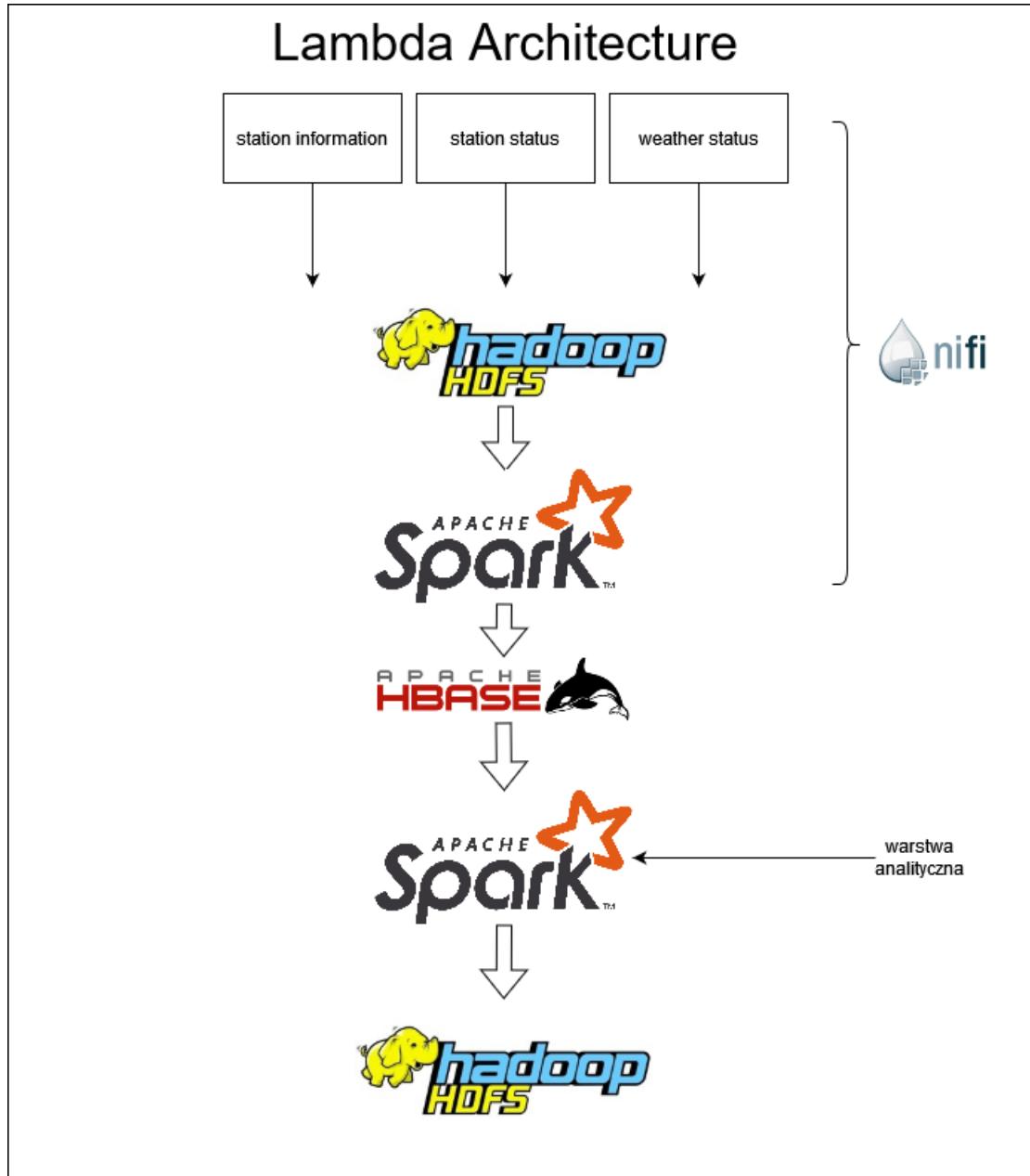
- Maximum Temperature (2 m) - maksymalna temperatura w danym dniu (2 metry nad poziomem gruntu)
- Minimum Temperature (2 m) - minimalna temperatura
- Maximum Apparent Temperature (2 m) - maksymalna odczuwalna temperatura
- Minimum Apparent Temperature (2 m) - minimalna odczuwalna temperatura
- Rain Sum - suma opadu deszczu
- Showers Sum - suma przelotnego opadu deszczu
- Snowfall Sum - suma opadów śniegu
- Precipitation Sum - suma opadów (deszczu, przelotnego deszczu i śniegu)
- Maximum Wind Speed (10 m) - maksymalna prędkość powietrza
- Maximum Wind Gusts (10 m) - maksymalna prędkość podmuchu powietrza (czyli, chwilowej zmiany prędkości)

4. https://open-meteo.com/en/docs#latitude=40.71&longitude=-74.01&hourly=temperature_2m,apparent_temperature,precipitation,rain,showers,snowfall,windspeed_10m,windgusts_10m&timezone=America%2FNew_York

Analogicznie jak poprzednio, jednak tym razem są to dane co godzinne i tak właśnie je odświeżamy.

- Temperature (2 m) - temperatura w danej godzinie (2 metry nad poziomem gruntu)
- Apparent Temperature - odczuwalna temperatura
- Rain - opady deszczu
- Showers - opady przelotnego deszczu
- Snowfall - opady śniegu
- Precipitation (rain + showers + snow) - suma opadów (deszczu, przelotnego deszczu i śniegu) w danej godzinie
- Wind Speed (10 m) - maksymalna prędkość powietrza
- Wind Gusts (10 m) - maksymalna prędkość podmuchu powietrza (czyli, chwilowej zmiany prędkości)

3 Architektura

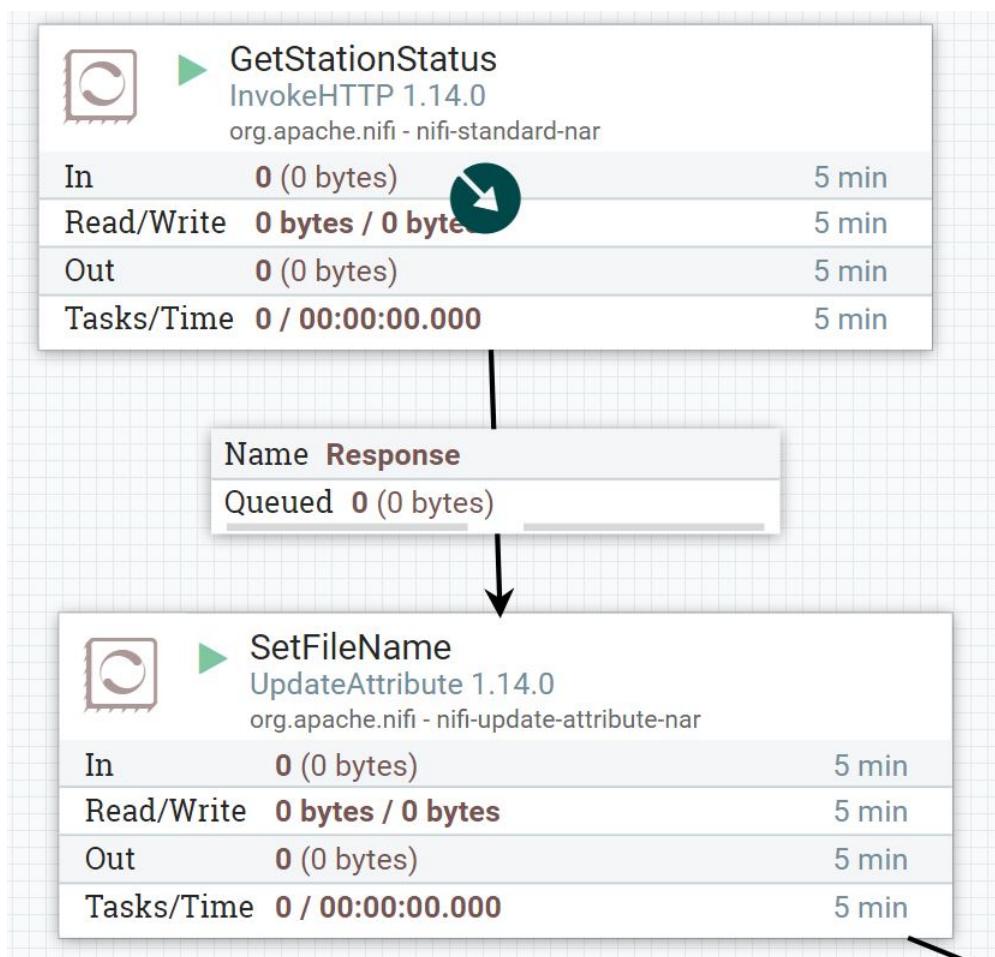


Projekt został zaimplementowany zgodnie z architekturą Lambda. Najpierw dane są pozyskiwane z wskazanych w poprzedniej sekcji endpointów. Przepływ danych oparty o technologię Apache Nifi kieruje dane do Master datasetu. Format danych ulega natomiast zmianie i przechowujemy dane o rozszerzeniu Parquet, nie JSON. Taki format zapewni wydajność przetwarzania dla danych wielkiej

skali, z którymi mamy do czynienia, szczególnie w trakcie querowania. Dodatkowo mamy wsparcie dla zagnieżdżonych, skomplikowanych struktur wewnętrz kolumn, a takie występują w naszych źródłach. Na tym etapie gotowy jest Master dataset. Z użyciem Apache Spark zostały stworzone Batch views i umieszczone w Serving layer, zarządzany przez Apache Hbase. Jak wcześniej wspomniane, wprowadzamy osobne tabele zarówno zorientowane na dni, jak i godziny. Następnie została dokonana wsadowa analiza danych w Apache Spark z wykorzystaniem uprzednio przygotowanych danych spoczywających w Serving layer. Jej wyniki zostały przedstawione w warstwie prezentacji, a na koniec umieszczone w HDFSie.

3.1 Pobieranie danych *Station Status*

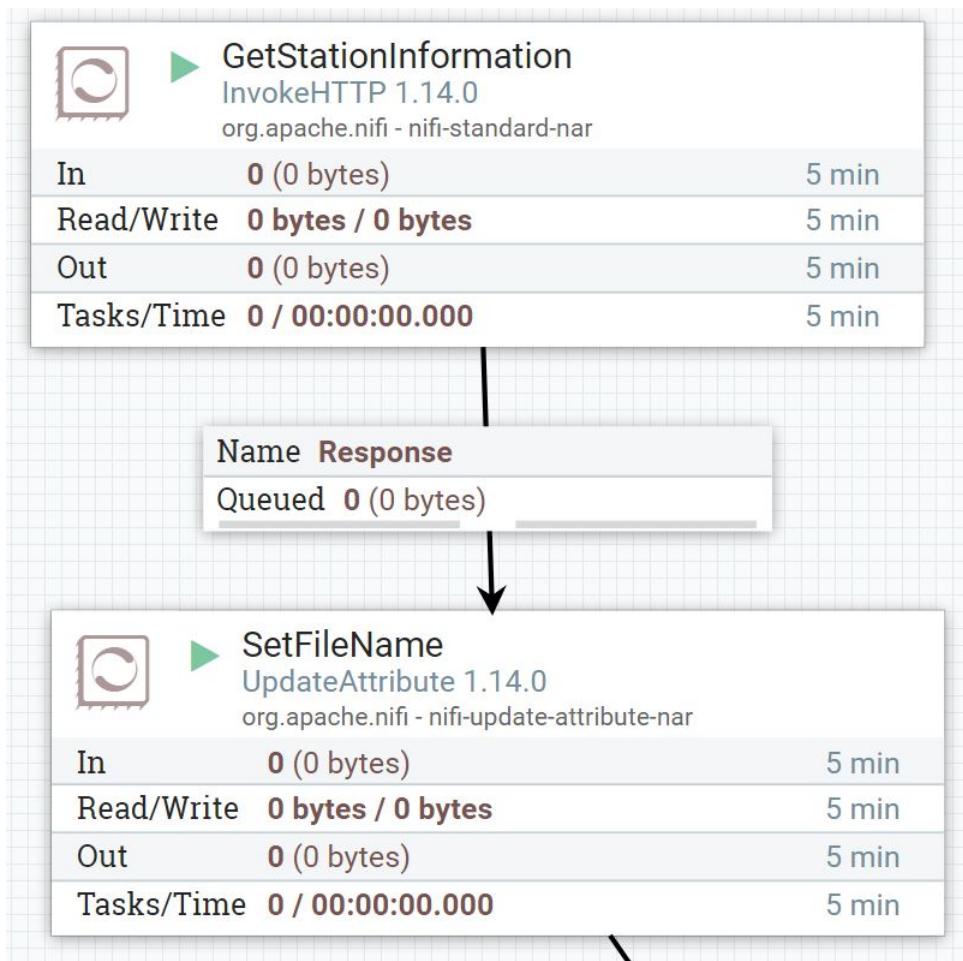
Jest to dwublokowa kolejka, która pobiera dane odnośnie aktualnego statusu stacji. Blok ten wykonywany jest co 720 sekund (12 minut).



- **GetStationStatus** - wykonuje zapytanie HTTP GET, do endpoint'u: https://gbfs.citibikenyc.com/gbfs/en/station_status.json. Następnie przekazuje te dane dalej.
- **SetFileName** - dla wygody debugowania oraz pracy, ustala nazwę nowopobranego pliku na: `station_status.json`.

3.2 Pobieranie danych *Station Information*

Jest to dwublokowa kolejka, która pobiera dane odnośnie aktualnych informacji na temat stacji. Wywoływana jest według CRON'a: `0 1 5 1/1 * ? *`, czyli o godzinie 5:01 każdego dnia według czasu na VM. Jest to tożsame z minutą po północy czasu nowojorskiego.

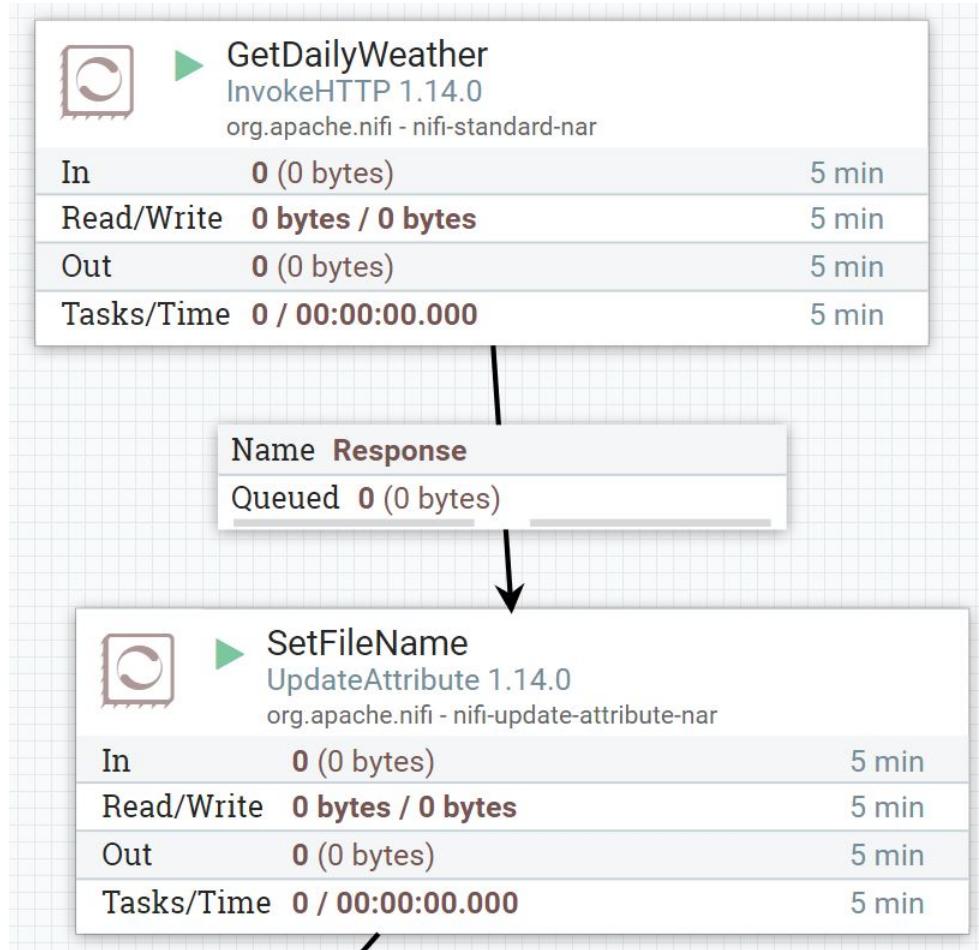


- **GetStationInformation** - wykonuje zapytanie HTTP GET, do endpoint'u: https://gbfs.citibikenyc.com/gbfs/en/station_information.json. Następnie przekazuje te dane dalej.

- **SetFileName** - dla wygody debugowania oraz pracy, ustala nazwę nowopobranego pliku na: *station_information.json*.

3.3 Pobieranie danych *Daily Weather*

Jest to dwublokowa kolejka, która pobiera dane odnośnie dzisiejszej pogody. Wywoływana jest według identycznego jak wyżej CRON'a: *0 1 5 1/1 * ? **.



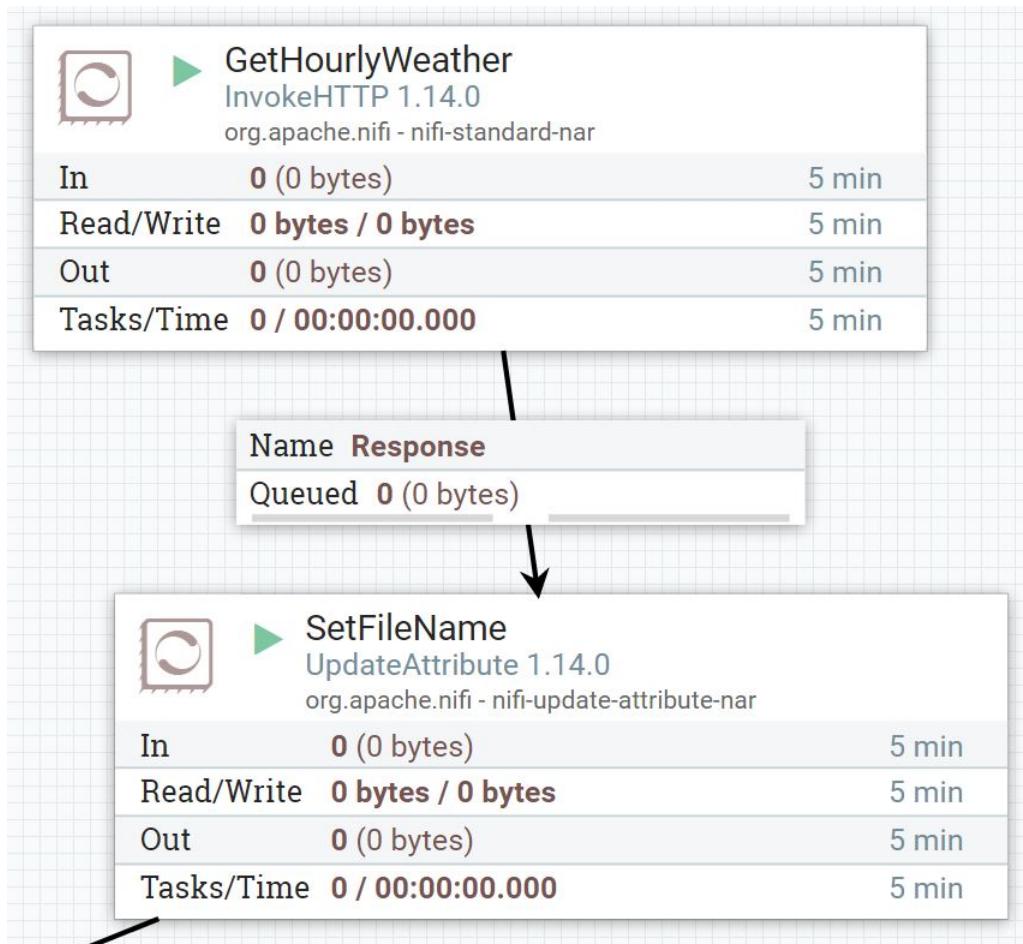
- **GetDailyWeather** - wykonuje zapytanie HTTP GET, do endpoint'u:

w Nowym Jorku. Ta sama wartość jest również przekazywana jako *end_date*. Następnie przekazuje te dane dalej.

- **SetFileName** - dla wygody debugowania oraz pracy, ustala nazwę nowopobranego pliku na: *daily_weather.json*.

3.4 Pobieranie danych *Hourly Weather*

Jest to dwublokowa kolejka, która pobiera dane odnośnie dzisiejszej pogody, zgraniuowanej na godziny. Wywoływana jest według CRON'a: *0 1 0/1 1/1 * ? **, czyli minutę po każdej pełnej godzinie.

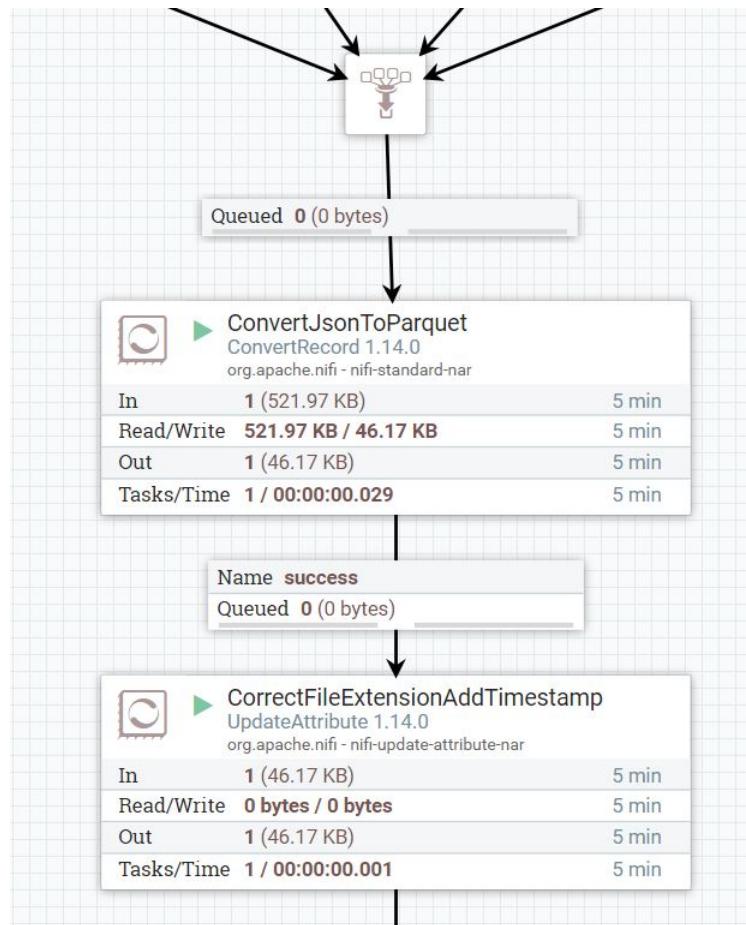


- **GetHourlyWeather** - wykonuje zapytanie HTTP GET, do endpoint'u:

`date=${now():minus(18000000):format("yyyy-MM-dd")}`. Ważnym jest, że `$now():minus(18000000):format("yyyy-MM-dd")` oznacza aktualną datę co do dnia, cofniętą o 5 godzin, żeby odzwierciedlać godzinę w Nowym Jorku. Ta sama wartość jest również przekazywana jako `end_date`. Następnie przekazuje te dane dalej.

- **SetFileName** - dla wygody debugowania oraz pracy, ustala nazwę nowopobranego pliku na: `hourly_weather.json`.

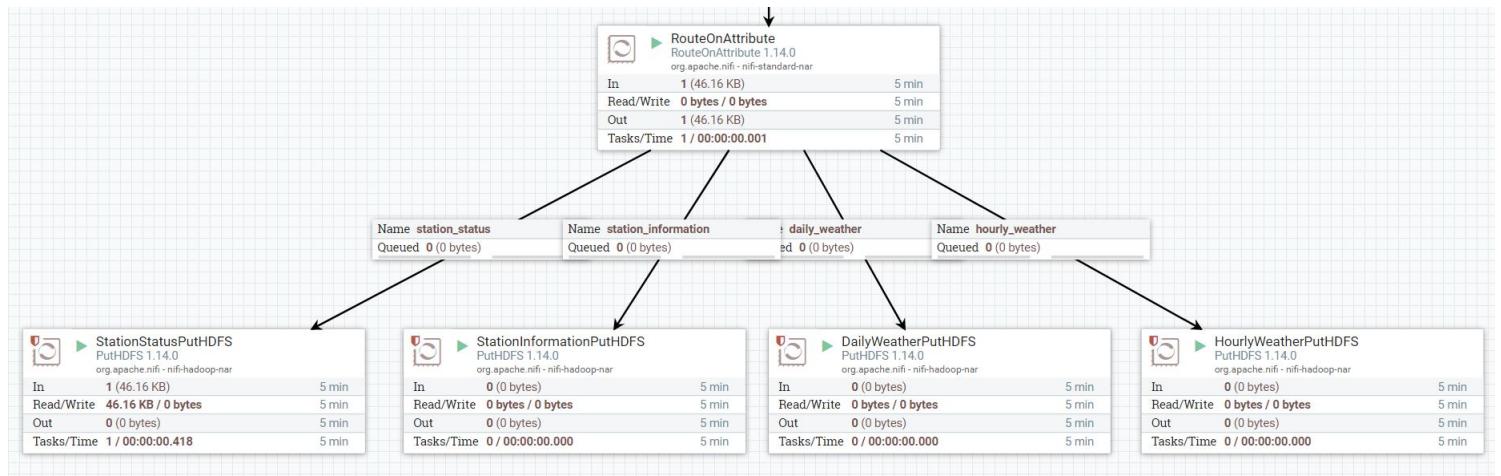
3.5 Konwersja plików



- **ConvertJsonToParquet** - za pomocą `JsonTreeReader` oraz `ParquetRecordSetWriter` konwertuje każdy plik .json na .parquet.
- **CorrectFileExtensionAddTimestamp** - zmienia nazwę pliku wraz z rozszerzeniem na: `filename:substringBefore('.').$now():minus(18000000):format("yyyy-MM-dd_HH-mm-ss-SSS").parquet`
 - `filename:substringBefore('.') - nazwa pliku bez rozszerzenia`

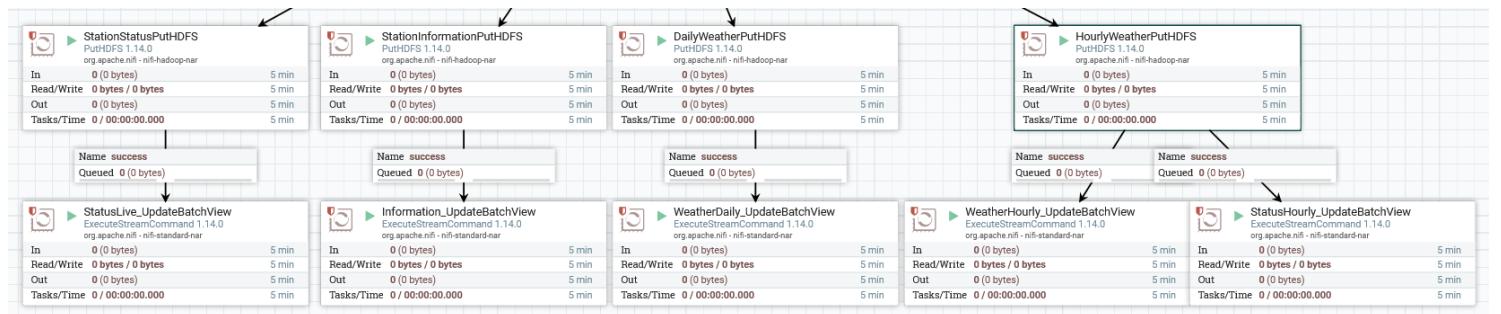
- `$now():minus(18000000):format("yyyy-MM-dd HH-mm-ss-SSS")` - aktualna data co do milisekund cofnięta o 5 godzin.

3.6 Zapisywanie plików



- **RouteOnAttribute** - bloczek ten rozdziela odpowiednie pliki do odpowiednich bloczków zapisujących pliki
- **StationStatusPutHDFS** - zapisuje plik, odpowiadający nazwie bloczka, do odpowiadającej ścieżki
- **StationInformationPutHDFS** - analogicznie
- **DailyWeatherPutHDFS** - analogicznie
- **HourlyWeatherPutHDFS** - analogicznie

3.7 Autmatyczne wywoływanie tworzenia się i aktualizacji batchviews



Na koniec, aby w pełni zautomatyzować kolejny etap przepływu danych w architekturze naszego rozwiązania, dodaliśmy bloki, które będą odpowiadały za aktualizację lub tworzenie batchviews. Nasze batchviews, jak deklarowane wcześniej, są tworzone z użyciem Sparka. Kod tworzenia widoków wsadowych został podzielony na 5 osobnych skryptów w ten sposób, aby stanowił on odrębną całość i możliwe oraz sensowne było jego samodzielne wywołanie tak, aby konieczne operacje zostały wykonane z jak najmniejszym narzutem obliczeniowym. Wyseparowane kody zawierają niezbędne modyfikacje (konfiguracyjce), aby mogły zostać wywołane z poziomu Nifi. Niemniej jednak zapewniamy także jupyter notebook, który zawiera zbiorczo zawartość każdego z tych skryptów i umożliwia interaktywną pracę z batchami.

- **StatusLive_UpdateBatchView** - zaktualizowanie wraz z napłynięciem nowego pliku widoku, który przedstawia najnowsze dane na temat statusu naszej stacji. Widok ten posłuży dalej w celach utrzymania często odświeżanej mapy obrazującej dostępność rowerów na stacjach. Udostępnienie takich informacji użytkownikom rowerów citibike może pozwolić na lepsze planowanie podróży i uniknięcie niepotrzebnej frustracji w przypadku braku dostępnych rowerów po dotarciu do stacji. Więcej o tej analizie w dalszej części dokumentu.
- **Information_UpdateBatchView** - wywołanie sparkowego joba, który dokona dodania informacji o stacjach wraz z rozpoczęciem nowego dnia. Umożliwi to śledzenie stacji, które np. stają się niedostępne.
- **WeatherDaily_UpdateBatchView** - uzupełnienie w widoku wsadowym statystyk pogodowych dla danego dnia wraz z napłynięciem odpowiedniego pliku.
- **WeatherHourly_UpdateBatchView** - uzupełnienie w widoku wsadowym statystyk pogodowych dla danej godziny wraz z napłynięciem odpowiedniego pliku.
- **StatusHourly_UpdateBatchView** - dopływający cogodzinny plik (CRON) wywoła uruchomienie przetwarzania danych dotyczących stacji z ubiegłej godziny.

Command Arguments	?	/user/hive/station_status/\${filename}
Command Path	?	/home/vagrant/BikesProject/status_live.py

Aby zapewnić większą odporność na błędy, do każdego skryptu pysparkowego przekazywany jest jako argument nazwa pliku, którego przetwarzanie ma dotyczyć.

4 Testy

4.1 Odkładanie plików w HDFS

Poniższe testy sprawdzają, czy procesy wykonywane w NiFi odpowiednio odkładają pliki - do odpowiednich folderów oraz czy odpowiednio zostają nazwane. Także sprawdzamy, czy wykonują się zgodnie z ustaloną częstotliwością.

```
vagrant@node1:~$ hdfs dfs -ls /user/hive/station_information
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop-2.7.6/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/apache-tez-0.9.1-bin/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 1 items
-rw-r--r-- 1 root supergroup 204404 2023-01-08 05:01 /user/hive/station_information/station_information_2023-01-08_00-01-01-032.parquet
vagrant@node1:~$ _
```

Oczekiwany wynikiem są pliki station_information_YYYY-MM-DD_00-01-ss-sss.parquet w folderze station_infromation pojawiające się raz dziennie. Zgodnie z planowaniem, zapis pliku wykonał się minutę po pierwszej godzinie czasu nowojorskiego, ma odpowiednią nazwę i rozszerzenie parquet.

```
-rw-r--r-- 1 root supergroup 47271 2023-01-08 21:22 /user/hive/station_status/station_status_2023-01-08_16-22-28-810.parquet
-rw-r--r-- 1 root supergroup 47278 2023-01-08 21:34 /user/hive/station_status/station_status_2023-01-08_16-34-29-373.parquet
-rw-r--r-- 1 root supergroup 47267 2023-01-08 21:46 /user/hive/station_status/station_status_2023-01-08_16-46-29-950.parquet
-rw-r--r-- 1 root supergroup 47287 2023-01-08 21:58 /user/hive/station_status/station_status_2023-01-08_16-58-30-410.parquet
-rw-r--r-- 1 root supergroup 47303 2023-01-08 22:10 /user/hive/station_status/station_status_2023-01-08_17-10-30-935.parquet
-rw-r--r-- 1 root supergroup 47308 2023-01-08 22:22 /user/hive/station_status/station_status_2023-01-08_17-22-31-432.parquet
-rw-r--r-- 1 root supergroup 47268 2023-01-08 22:34 /user/hive/station_status/station_status_2023-01-08_17-34-31-950.parquet
-rw-r--r-- 1 root supergroup 47285 2023-01-08 22:46 /user/hive/station_status/station_status_2023-01-08_17-46-32-465.parquet
```

Oczekiwany wynikiem były pliki dotyczące statusu stacji zapisywane, co 12 minut. Widzimy na screenie, że odstępy czasowe są odpowiednie. Pliki zapisują się do folderu station_status i mają odpowiednie nazwy.

```
vagrant@node1:~$ hdfs dfs -ls /user/hive/daily_weather
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop-2.7.6/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/apache-tez-0.9.1-bin/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 1 items
-rw-r--r-- 1 root supergroup 8617 2023-01-08 05:01 /user/hive/daily_weather/daily_weather_2023-01-08_00-01-00-322.parquet
```

Oczekiwany wynikiem są pliki odkładane raz dziennie, godzinę po północy czasu nowojorskiego. Widzimy, że wynik jest zgodny z oczekiwaniemi, tzn. że plik zapisał się o godzinie 1:00, ma odpowiedni format i znajduje się w odpowiednim folderze.

```
-rw-r--r-- 1 root supergroup 8610 2023-01-08 13:01 /user/hive/hourly_weather/hourly_weather_2023-01-08_08-01-00-274.parquet
-rw-r--r-- 1 root supergroup 8599 2023-01-08 14:01 /user/hive/hourly_weather/hourly_weather_2023-01-08_09-01-00-259.parquet
-rw-r--r-- 1 root supergroup 8610 2023-01-08 15:01 /user/hive/hourly_weather/hourly_weather_2023-01-08_10-01-00-258.parquet
-rw-r--r-- 1 root supergroup 8641 2023-01-08 16:01 /user/hive/hourly_weather/hourly_weather_2023-01-08_11-01-00-268.parquet
-rw-r--r-- 1 root supergroup 8641 2023-01-08 17:01 /user/hive/hourly_weather/hourly_weather_2023-01-08_12-01-00-291.parquet
-rw-r--r-- 1 root supergroup 8625 2023-01-08 18:01 /user/hive/hourly_weather/hourly_weather_2023-01-08_13-01-00-268.parquet
```

Zgodnie z oczekiwaniemi, pliki zapisują się co godzinę w folderze hourly_weather.

4.2 Batchview

Poniższe testy sprawdzają zapis batchview do HBase. Do połączenia używamy biblioteki happybase w Pythonie. W każdym screenie odpowiadającym testowi widać także przyjęty przez nas podział na rodziny kolumn.

Warto zauważyć, że klucze zostały dobrane z zachowaniem dobrych zasad projektowania klucza w tej platformie. Zapewniona jest nie tylko unikalność, ale konstrukcja gwarantuje m.in. że dane dotyczące np. konkretnego dnia będą wymagały mniejszej liczby regionów do przeskanowania, gdyż będą znajdowały się obok siebie.

```
In [9]: weather_hourly = connection.table('weather_hourly')
show_batchview_from_hbase(weather_hourly)

KEY: b'2023-01-08T17:00'
COLUMN: b'precipitation:precipitation' VALUE: b'0.0'
COLUMN: b'precipitation:rain' VALUE: b'0.0'
COLUMN: b'precipitation:showers' VALUE: b'0.0'
COLUMN: b'precipitation:snowfall' VALUE: b'0.0'
COLUMN: b'temperature:apparent_temperature' VALUE: b'-2.2'
COLUMN: b'temperature:temperature_2m' VALUE: b'1.7'
COLUMN: b'time:time' VALUE: b'2023-01-08T17:00'
COLUMN: b'wind:windgusts_10m' VALUE: b'13.7'
COLUMN: b'wind:windspeed_10m' VALUE: b'6.9'

KEY: b'2023-01-08T18:00'
COLUMN: b'precipitation:precipitation' VALUE: b'0.0'
COLUMN: b'precipitation:rain' VALUE: b'0.0'
COLUMN: b'precipitation:showers' VALUE: b'0.0'
COLUMN: b'precipitation:snowfall' VALUE: b'0.0'
COLUMN: b'temperature:apparent_temperature' VALUE: b'-3.4'
COLUMN: b'temperature:temperature_2m' VALUE: b'0.6'
COLUMN: b'time:time' VALUE: b'2023-01-08T18:00'
COLUMN: b'wind:windgusts_10m' VALUE: b'12.2'
COLUMN: b'wind:windspeed_10m' VALUE: b'7.7'
```

Powyżej jest przykładowe wywołanie batchview, pokazującego dane pogodowe dla różnych kolejnych godzin. Kluczem jest data wraz z godziną.

```
In [10]: weather_daily = connection.table('weather_daily')
show_batchview_from_hbase(weather_daily)

KEY: b'2023-01-08'
COLUMN: b'precipitation:precipitation_sum' VALUE: b'0.0'
COLUMN: b'precipitation:rain_sum' VALUE: b'0.0'
COLUMN: b'precipitation:showers_sum' VALUE: b'0.0'
COLUMN: b'precipitation:snowfall_sum' VALUE: b'0.0'
COLUMN: b'temperature:apparent_temperature_max' VALUE: b'1.1'
COLUMN: b'temperature:apparent_temperature_min' VALUE: b'-4.7'
COLUMN: b'temperature:temperature_2m_max' VALUE: b'4.8'
COLUMN: b'temperature:temperature_2m_min' VALUE: b'-0.5'
COLUMN: b'time:time' VALUE: b'2023-01-08'
COLUMN: b'wind:windgusts_10m_max' VALUE: b'32.0'
COLUMN: b'wind:windspeed_10m_max' VALUE: b'11.4'
```

Tym razem, mamy zgregowane dane w batchview, odnoszące się do różnych metryk pogodowych w ciągu całego dnia. Tutaj kluczem jest tylko data.

Na przykładzie tego testu pragniemy wskazać inną ogromnie istotną rzecz. Jest to automatyczne wywołanie w Nifi pliku z kodem pysparkowym, który odświeżył widok wzbogacając go o najnowsze napływające dane.

```
In [11]: information_daily = connection.table('information_daily')
show_batchview_from_hbase(information_daily)

KEY: b'2023-01-08_116'
COLUMN: b'detail:capacity' VALUE: b'74'
COLUMN: b'detail:electric_bike_surcharge_waiver' VALUE: b'False'
COLUMN: b'detail:has_kiosk' VALUE: b'True'
COLUMN: b'detail:name' VALUE: b'W 17 St & 8 Ave'
COLUMN: b'detail:short_name' VALUE: b'6148.02'
COLUMN: b'detail:station_type' VALUE: b'classic'
COLUMN: b'geo:lat' VALUE: b'40.74177603'
COLUMN: b'geo:lon' VALUE: b'-74.00149746'
COLUMN: b'id:external_id' VALUE: b'66db28b5-0aca-11e7-82f6-3863bb44ef7c'
COLUMN: b'id:legacy_id' VALUE: b'116'
COLUMN: b'id:region_id' VALUE: b'71'
COLUMN: b'id:station_id' VALUE: b'116'

KEY: b'2023-01-08_119'
COLUMN: b'detail:capacity' VALUE: b'53'
COLUMN: b'detail:electric_bike_surcharge_waiver' VALUE: b'False'
COLUMN: b'detail:has_kiosk' VALUE: b'True'
COLUMN: b'detail:name' VALUE: b'Park Ave & St Edwards St'
COLUMN: b'detail:short_name' VALUE: b'4700.06'
```

Teraz, wyświetlony został batchview z informacjami ogólnymi o stacjach, z aktualnego dnia. Kluczem jest data połączona za pomocą "_" z id stacji.

```
In [12]: status_hourly = connection.table('status_hourly')
show_batchview_from_hbase(status_hourly)

KEY: b'2023-01-08_16_116'
COLUMN: b'id:legacy_id' VALUE: b'116'
COLUMN: b'id:station_id' VALUE: b'116'
COLUMN: b'metric:num_bikes_available' VALUE: b'39.0'
COLUMN: b'metric:num_bikes_disabled' VALUE: b'2.0'
COLUMN: b'metric:num_docks_available' VALUE: b'32.0'
COLUMN: b'metric:num_docks_disabled' VALUE: b'0.0'
COLUMN: b'metric:num_ebikes_available' VALUE: b'0.6666666666666666'

KEY: b'2023-01-08_16_119'
COLUMN: b'id:legacy_id' VALUE: b'119'
COLUMN: b'id:station_id' VALUE: b'119'
COLUMN: b'metric:num_bikes_available' VALUE: b'44.0'
COLUMN: b'metric:num_bikes_disabled' VALUE: b'1.0'
COLUMN: b'metric:num_docks_available' VALUE: b'7.0'
COLUMN: b'metric:num_docks_disabled' VALUE: b'0.0'
COLUMN: b'metric:num_ebikes_available' VALUE: b'0.0'

KEY: b'2023-01-08_16_120'
COLUMN: b'id:legacy_id' VALUE: b'120'
COLUMN: b'id:station_id' VALUE: b'120'
COLUMN: b'metric:num_bikes_available' VALUE: b'38.0'
COLUMN: b'metric:num_bikes_disabled' VALUE: b'2.0'
COLUMN: b'metric:num_docks_available' VALUE: b'32.0'
COLUMN: b'metric:num_docks_disabled' VALUE: b'0.0'
COLUMN: b'metric:num_ebikes_available' VALUE: b'0.0'
```

Powyżej jest batchview, przedstawiający dane zagregowane średnią odnośnie danych o sytuacji każdej stacji, jak np. ilość dostępnych rowerów. Analogicznie jak ostatnio, kluczem jest data, wraz z następującą godziną oraz id stacji, oddzielone za pomocą znaku ”_”.

```
In [13]: status_live = connection.table('status_live')
show_batchview_from_hbase(status_live)

KEY: b'116'
COLUMN: b'id:last_reported' VALUE: b'1673217928'
COLUMN: b'id:legacy_id' VALUE: b'116'
COLUMN: b'id:station_id' VALUE: b'116'
COLUMN: b'metric:num_bikes_available' VALUE: b'38'
COLUMN: b'metric:num_bikes_disabled' VALUE: b'2'
COLUMN: b'metric:num_docks_available' VALUE: b'32'
COLUMN: b'metric:num_docks_disabled' VALUE: b'0'
COLUMN: b'metric:num_ebikes_available' VALUE: b'0'
COLUMN: b'status:is_installed' VALUE: b'1'
COLUMN: b'status:is_renting' VALUE: b'1'
COLUMN: b'status:is_returning' VALUE: b'1'
COLUMN: b'status:station_status' VALUE: b'active'

KEY: b'119'
COLUMN: b'id:last_reported' VALUE: b'1673216802'
COLUMN: b'id:legacy_id' VALUE: b'119'
COLUMN: b'id:station_id' VALUE: b'119'
COLUMN: b'metric:num_bikes_available' VALUE: b'45'
COLUMN: b'metric:num_bikes_disabled' VALUE: b'1'
```

Analogicznie jak poprzednio, z tym, że teraz są aktualne dane na temat każdej stacji. Kluczami są id stacji.

4.3 Zapisywanie analiz do HDFS

Poniższy kod przedstawia przykładowy kod realizujący zapis wyników analizy wsadowej do HDFS.

```

ts = str(time.time()).replace('.', '-')
file_name = 'regions_avail_' + ts + '.parquet'
df = spark.createDataFrame(data)
df.write.save('/user/hive/analysis_data/' + file_name, format='parquet', mode='append')

```

```

vagrant@node1:~/BikesProject$ hdfs dfs -ls /user/hive/analysis_data
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop-2.7.6/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/apache-tez-0.9.1-bin/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 16 items
drwxr-xr-x - vagrant supergroup          0 2023-01-09 09:09 /user/hive/analysis_data/regions_avail_1673255369-5013855.parquet
drwxr-xr-x - vagrant supergroup          0 2023-01-09 08:25 /user/hive/analysis_data/regions_map_1673252756-6148047.parquet
drwxr-xr-x - vagrant supergroup          0 2023-01-09 08:47 /user/hive/analysis_data/regions_map_1673254046-3788772.parquet
drwxr-xr-x - vagrant supergroup          0 2023-01-09 08:48 /user/hive/analysis_data/regions_map_1673254101-450543.parquet
drwxr-xr-x - vagrant supergroup          0 2023-01-09 08:48 /user/hive/analysis_data/regions_map_1673254121-1328173.parquet
drwxr-xr-x - vagrant supergroup          0 2023-01-09 07:48 /user/hive/analysis_data/wiz_1_1673250513.1075814.parquet
drwxr-xr-x - vagrant supergroup          0 2023-01-09 07:51 /user/hive/analysis_data/wiz_1_1673250693-4767199.parquet
drwxr-xr-x - vagrant supergroup          0 2023-01-09 08:00 /user/hive/analysis_data/wiz_1_1673251244-980342.parquet
drwxr-xr-x - vagrant supergroup          0 2023-01-09 08:11 /user/hive/analysis_data/wiz_1_1673251887-8429081.parquet
drwxr-xr-x - vagrant supergroup          0 2023-01-09 09:18 /user/hive/analysis_data/wiz_1_1673255912-675942.parquet
drwxr-xr-x - vagrant supergroup          0 2023-01-09 09:19 /user/hive/analysis_data/wiz_1_1673255992-2496393.parquet
drwxr-xr-x - vagrant supergroup          0 2023-01-09 09:20 /user/hive/analysis_data/wiz_1_1673256002-7675579.parquet
drwxr-xr-x - vagrant supergroup          0 2023-01-09 09:21 /user/hive/analysis_data/wiz_1_1673256077-157724.parquet
drwxr-xr-x - vagrant supergroup          0 2023-01-09 08:11 /user/hive/analysis_data/wiz_map_1673251903-4523103.parquet
drwxr-xr-x - vagrant supergroup          0 2023-01-09 08:25 /user/hive/analysis_data/wiz_map_1673252736-4692383.parquet
drwxr-xr-x - vagrant supergroup          0 2023-01-09 09:21 /user/hive/analysis_data/wiz_weather_1673256076-1958005.parquet

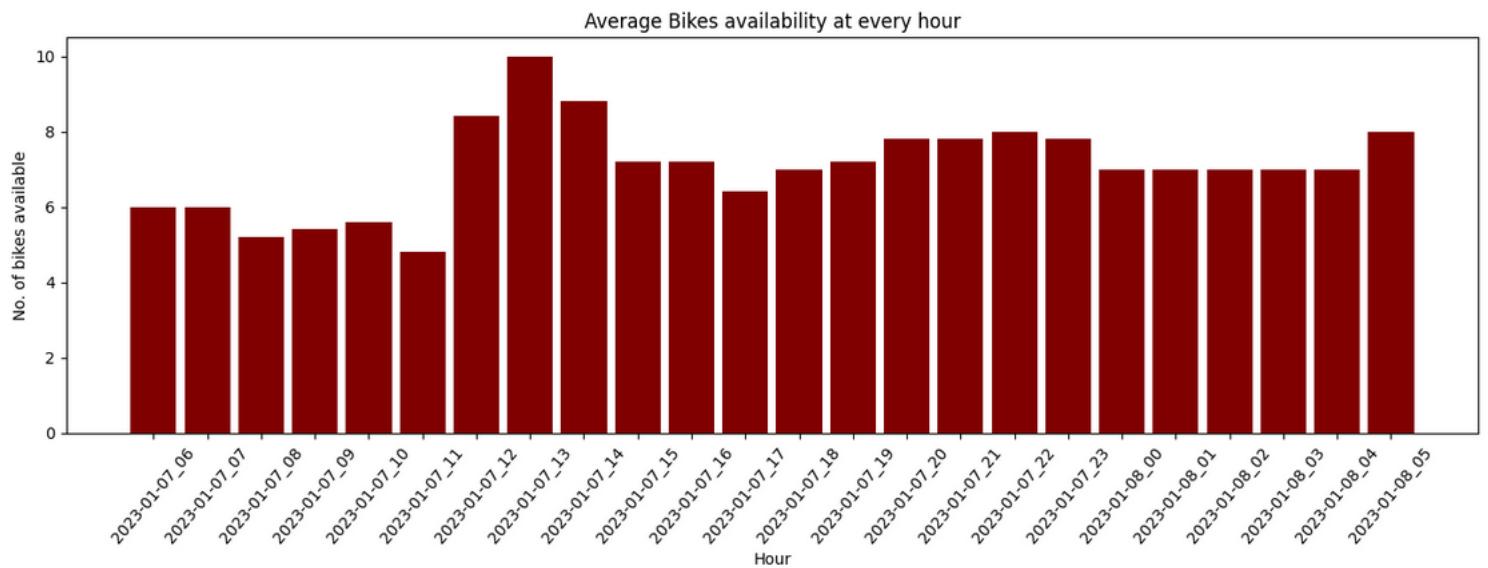
```

Widzimy, że dane do każdej z wizualizacji się zapisyły oraz posiadają odpowiedni timestamp i nazwę jednoznacznie identyfikującą ją z daną wizualizacją.

5 Wizualizacje

5.1 Wykres dostępności rowerów na danej stacji

Wyświetla średnią dostępność rewerów na danej stacji, wybieranej poprzez jej id. Dane są zagregowane do godzin oraz użytkownik może wybrać ile z ostatnich godzin chce on by się wyświetliło. Domyslnie jest to 24h.

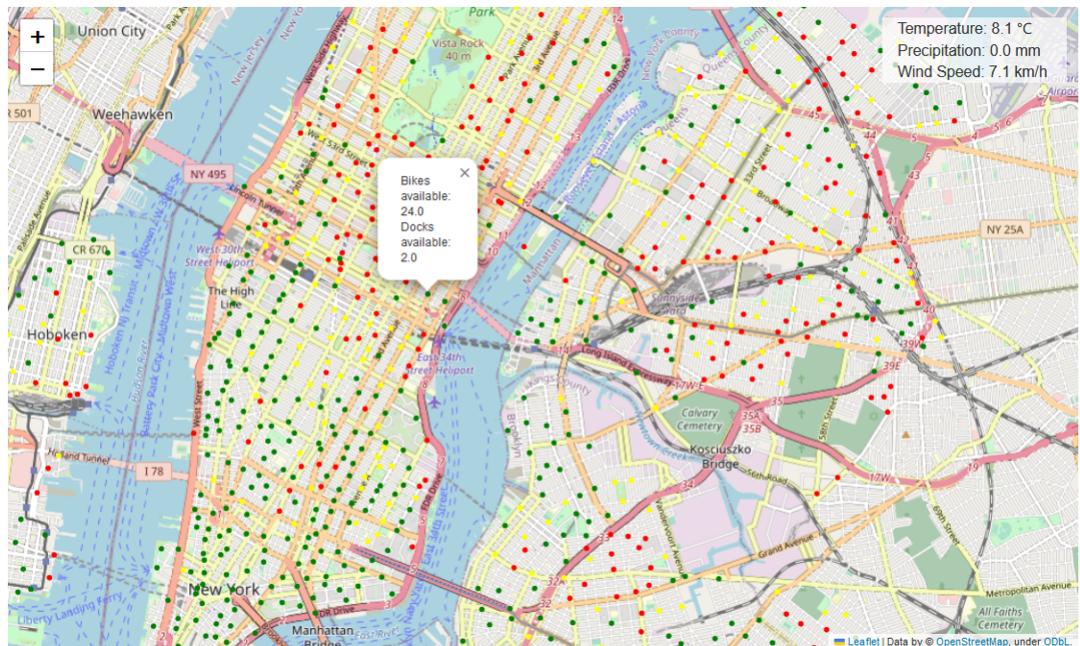
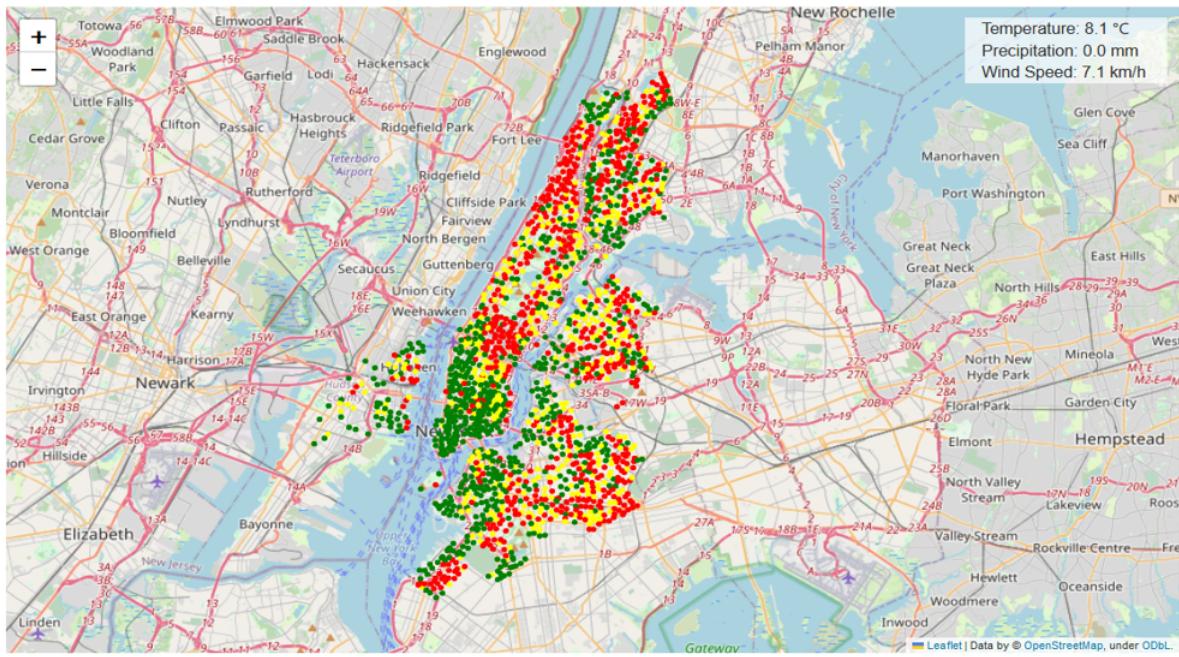


5.2 Mapa ze stacjami

Są nałożone na mapę, wszystkie dostępne stacje. Również, mają one swoje kolory, gdzie każdy kolor oznacza procentową dostępność rowerów na tej stacji:

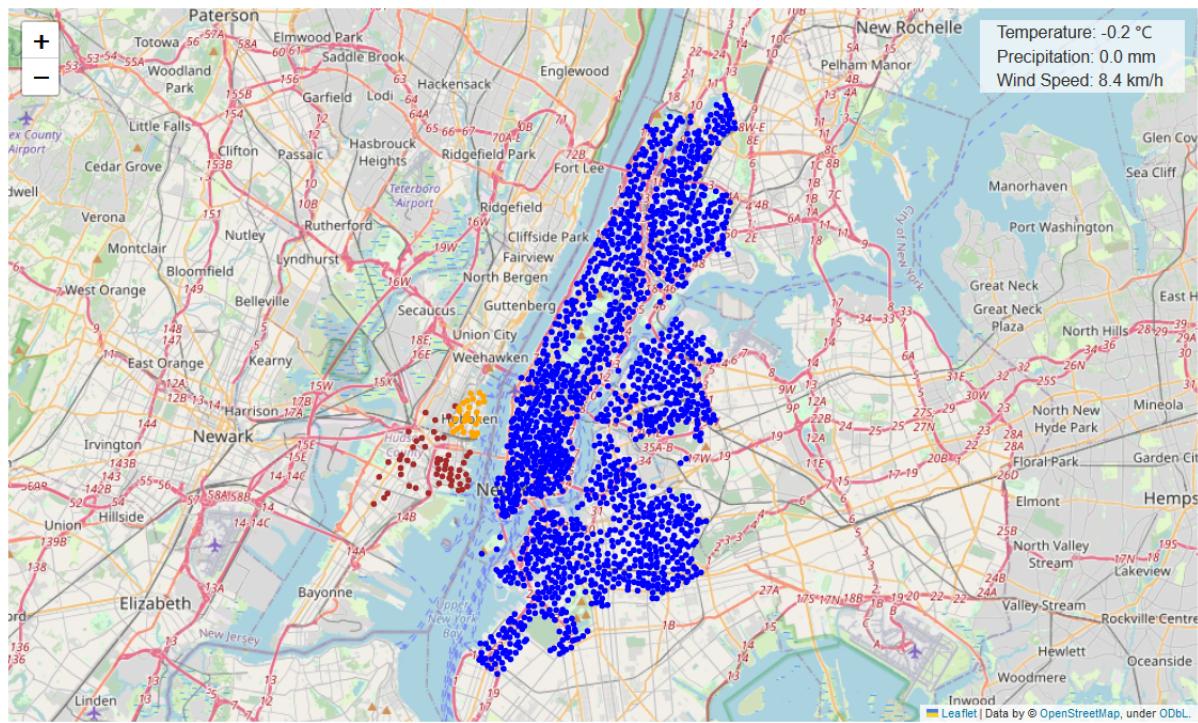
- zielony - jest ponad 50% rowerów
- żółty - jest pomiędzy 25% a 50% rowerów
- czerwony - jest mniej niż 25% rowerów

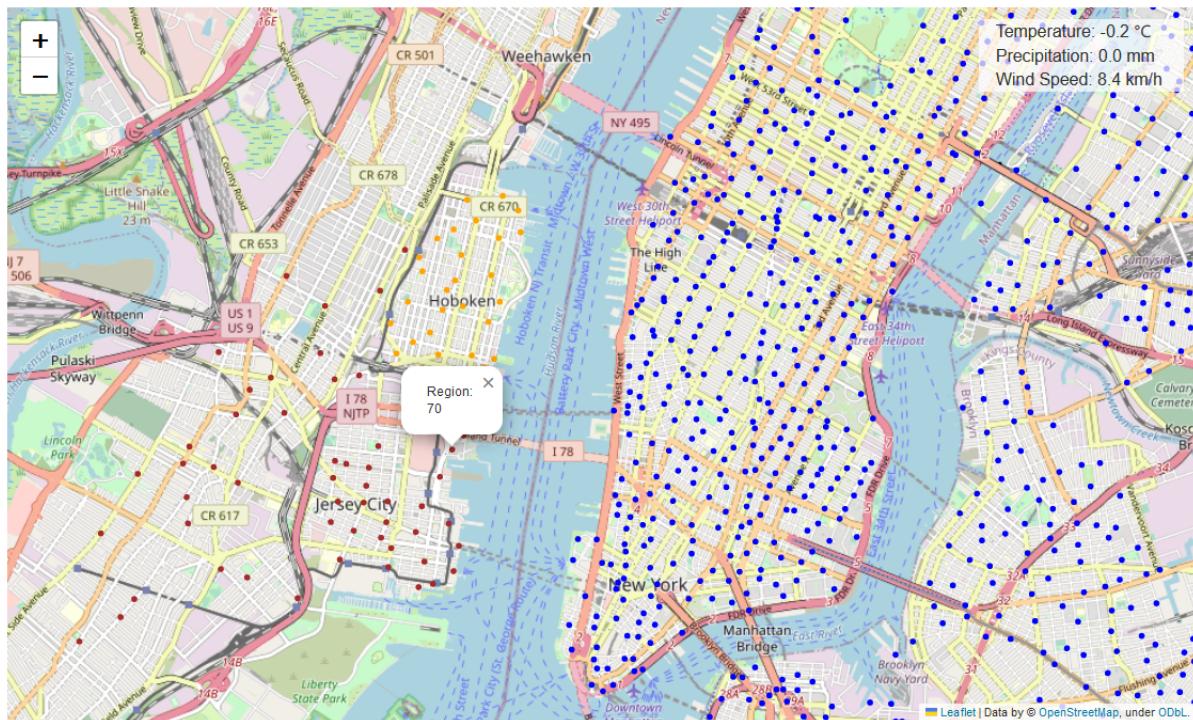
Po naciśnięciu na stację pokazuje się tooltip z dokładnymi liczbami. Może być online oraz z batcha ze zagregowanymi godzinowo danymi. Wizualizowanie i analiza mapy z danymi online pozwoli m.in. szybciej reagować na aktualne potrzeby użytkowników.



Również, przygotowana została wizualizacja przyporządkowania stacji do regionów. Bazuje ona na danych odświeżanych raz dziennie. W przypadku dodania nowej stacji, czy zmiany przypisania regionów będzie to zauważone. Mapa ta pozwala na lepsze analizowanie wykresów dotyczących regionów, w naszym przypadku wykresu przedstawiającego średnią dostępną liczbę docków w każdym

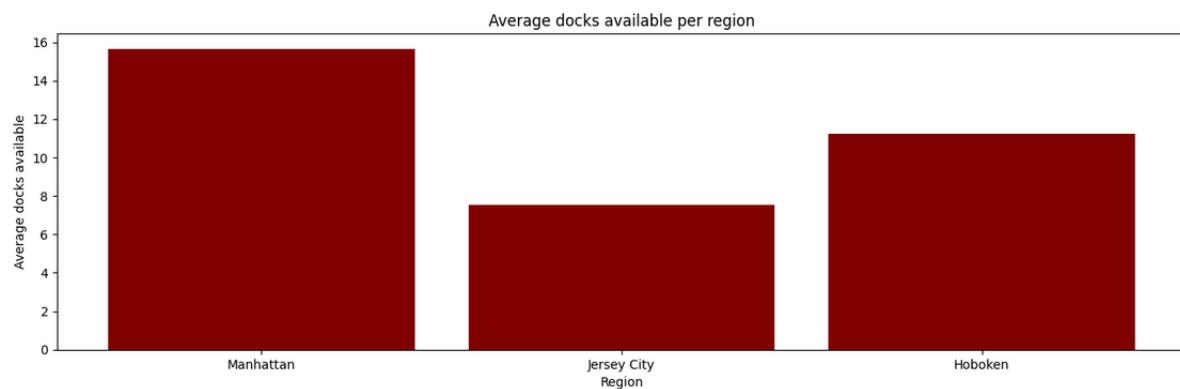
z regionów.





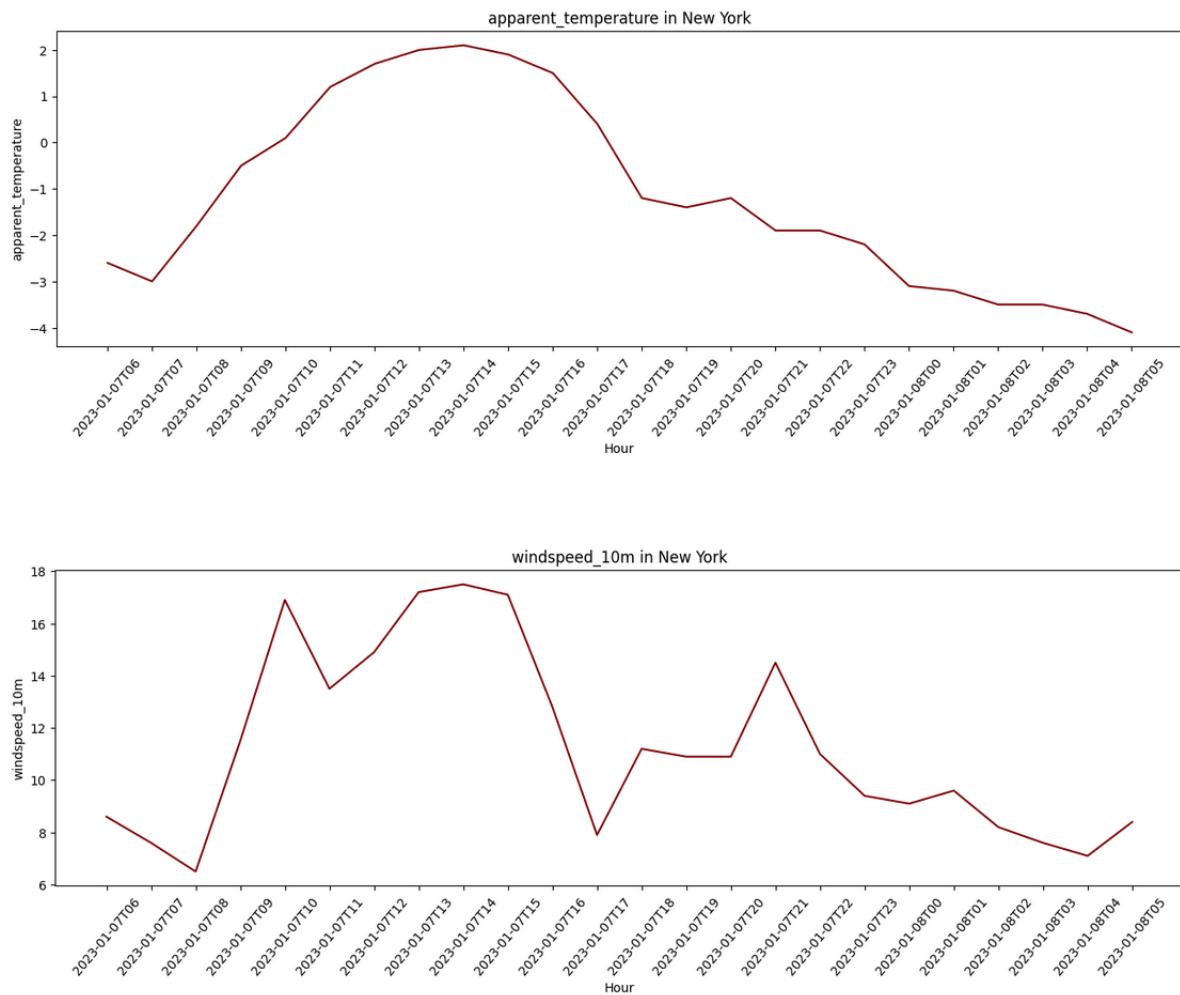
5.3 Wykres dostępności docków na danej stacji

Ważne jest utrzymanie możliwości oddawania rowerów w każdym regionie. Poniższy wykres przedstawia średnią liczbę wolnych docków, pozwalających zwrócić rower. W przypadku zauważalnie niskiej wartości mamy informację, że powinny zostać zwolnione miejsca poprzez na przykład przewiezienie części rowerów do regionu, w którym wolnych docków jest zbyt dużo.



5.4 Wykresy parametrów pogodowych

Przygotowane funkcje w Jupyter Notebooku pozwalają na wybranie parametru pogodowego, który nas interesuje oraz dowolnego dnia (aktualnego lub z przeszłości). Wówczas otrzymujemy wykres pogodowy z dokładnością co do godziny. Poniższe wykresy prezentują odczuwalną temperaturę w Nowym Jorku 7 stycznia 2023 oraz prędkość wiatru (10m nad gruntem) tego samego dnia.



Wierzymy, że wzbogacenie dostępnych danych na temat aktywności użytkowników o dane pogodowe pozwoli lepiej zrozumieć obserwowane zjawiska. Jeśli przyczyną gorszej rentowności firmy w danym miesiącu są niekorzystne warunki pogodowe, a nie złe zarządzanie, sposób radzenia sobie z problemem będzie odmienny.