Składowanie danych w systemach Big Data - sprawozdanie

Fijałkowski Paweł, Szmajdziński Szymon, Żółkowski Artur

06.01.2023

1 Wstęp

Celem niniejszego projektu jest stworzenie i zbudowanie architektury rozwiązania Big Data do badania problemu społecznego o dowolnej tematyce. W ramach tego projektu będziemy analizować zależność pomiędzy ilością zgłoszeń do straży pożarnej w Seattle (USA), a warunkami pogodowymi w tym regionie. Do realizacji tego celu wykorzystamy architekturę lambda oraz najlepsze (pochodzące od `Apache Software Foundation`) narzędzia do przetwarzania i składowania dużych zbiorów danych. W rezultacie udostępnimy użytkownikom systemu możliwość analizy zależności pomiędzy wymienionymi zmiennymi. Kod źródłowy opisanych rozwiązań jest umieszczony na repozytorium github: github.com/pablo2811/team-intelligence-big-data.

2 Dane

2.1 Visual Crossing Weather API

Otwarte `Visual Crossing Weather API` pozwalające na zapytania dotyczące warunków pogodowych w regionie Seattle. Można przy jego użyciu otrzymać informację dotyczące temperatury, widoczności czy ciśnienia w danym miejscu i czasie. Dane w systemie źródłowym aktualizują się w 3-godzinnych cyklach i są udostępniane w formacie `json`. Proces aktualizacji danych pogodowych do stanu w systemie źródłowym przebiega raz na dobę, za dzień poprzedni. Link do dokumentacji źródła danych.

Listing 1: Przykład odpowiedzi z Visual Crossing Weather API

```
[
  {
    "datetime": "19:00:00",
    "day": "2023-01-04",
    "temp": 9.3,
    "humidity": 50.92,
```

```
    "windgust": 41.4,
    "pressure": 1002.2,
    "visibility": 16.0
  },
  {
    "datetime": "20:00:00",
    "day": "2023-01-04",
    "temp": 10.1,
    "humidity": 45.85,
    "windgust": 46.4,
    "pressure": 1000.6,
    "visibility": 16.0
  },
  {
    "datetime": "21:00:00",
    "day": "2023-01-04",
    "temp": 10.5,
    "humidity": 44.35,
    "windgust": 45.4,
    "pressure": 1000.6,
    "visibility": 16.0
  }
]
```

2.2 Seattle Real Time Fire 911 Calls

Otwarte API dotyczące zgłoszeń do straży pożarnej w Seattle jest głównym źródłem danych strumieniowych w omawianym projekcie. Źródło to dostarcza w formacie json informację dotyczące czasu i miejsca zgłoszenia (adres i współrzędne) jak i typ zgłoszenia i jego identyfikator. System źródłowy aktualizuje nowe zgłoszenia w 5-minutowych cyklach, można więc założyć że jest ono strumieniowe. Proces pobierania danych będzie odbywa się w cyklach 10-minutowych. Link do dokumentacji źródła danych.

Listing 2: Przykład odpowiedzi z Seattle Real Time Fire 911 API

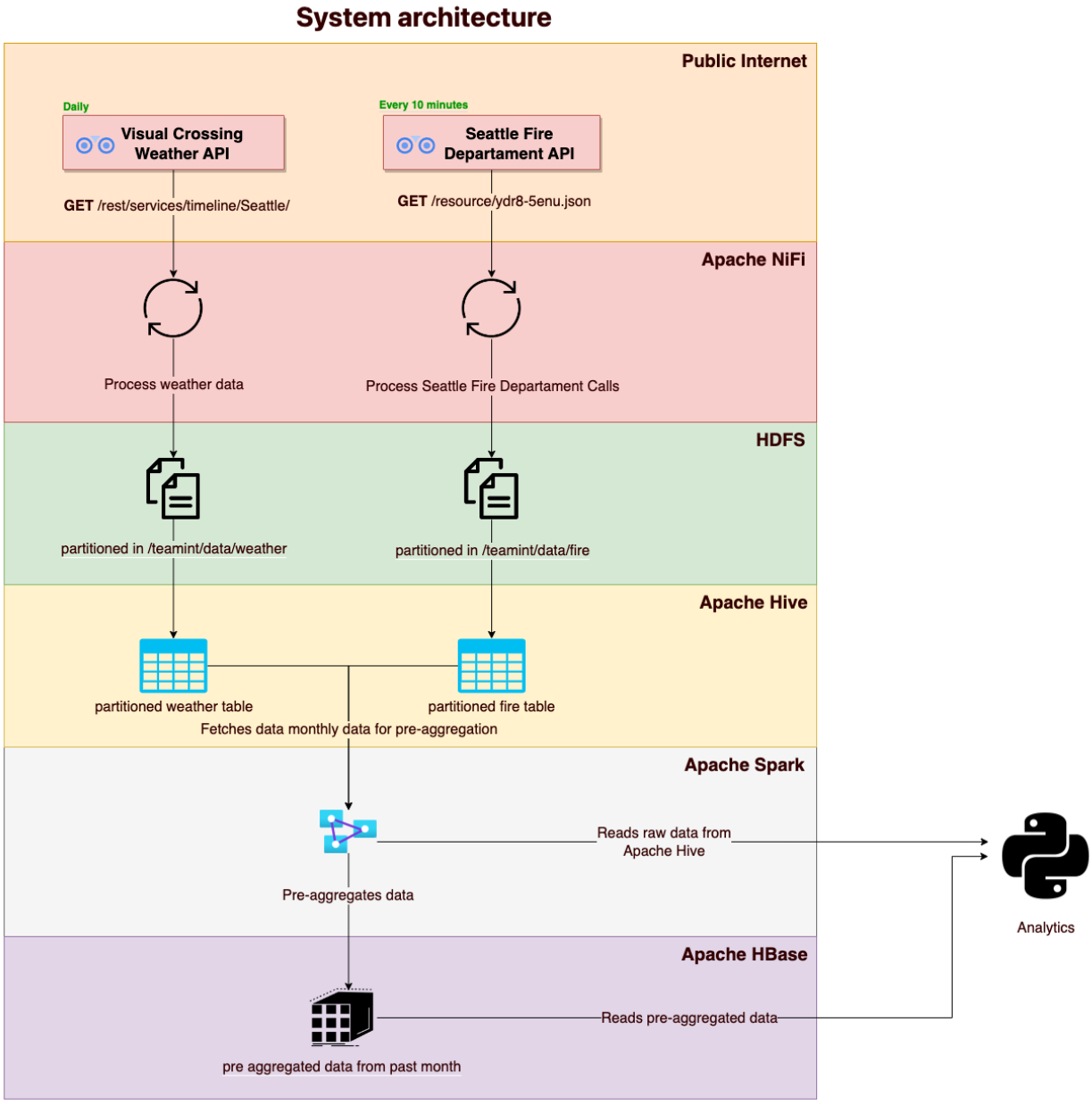
```
[
  {
    "address": "3300 Ne 125th St",
    "type": "Investigate Out Of Service",
    "datetime": "2023-01-06T02:19:00.000",
    "latitude": "47.719265",
    "longitude": "-122.29273",
    "report_location": {
      "type": "Point",
      "coordinates": [
```

```
        -122.29273,
        47.719265
      ]
    },
    "incident_number": "F230002201"
  },
  {
    "address": "1511 3rd Ave",
    "type": "Medic Response- Overdose",
    "datetime": "2023-01-06T02:17:00.000",
    "latitude": "47.609851",
    "longitude": "-122.337886",
    "report_location": {
      "type": "Point",
      "coordinates": [
        -122.337886,
        47.609851
      ]
    },
    "incident_number": "F230002200"
  },
  {
    "address": "2215 1st Ave",
    "type": "Aid Response",
    "datetime": "2023-01-06T02:16:00.000",
    "latitude": "47.612716",
    "longitude": "-122.345399",
    "report_location": {
      "type": "Point",
      "coordinates": [
        -122.345399,
        47.612716
      ]
    },
    "incident_number": "F230002198"
  }
]
```


3 Architektura

Diagram 4 przedstawia architekturę rozwiązania. Zautomatyzowany przepływ danych jest w pełni realizowany z wykorzystaniem `Apache NiFi`. Pobierane dane są partycjonowane po dzcie i zapisywane w systemie `Apache Hadoop` w formie `parquet` ze zdefiniowanym dostępem z poziomy `Apache Hive`. Następnie dane są

regularnie agregowane przy użyciu `Apache Spark`, a następnie tak przetworzone dane są zapisywane w `Apache HBase`.



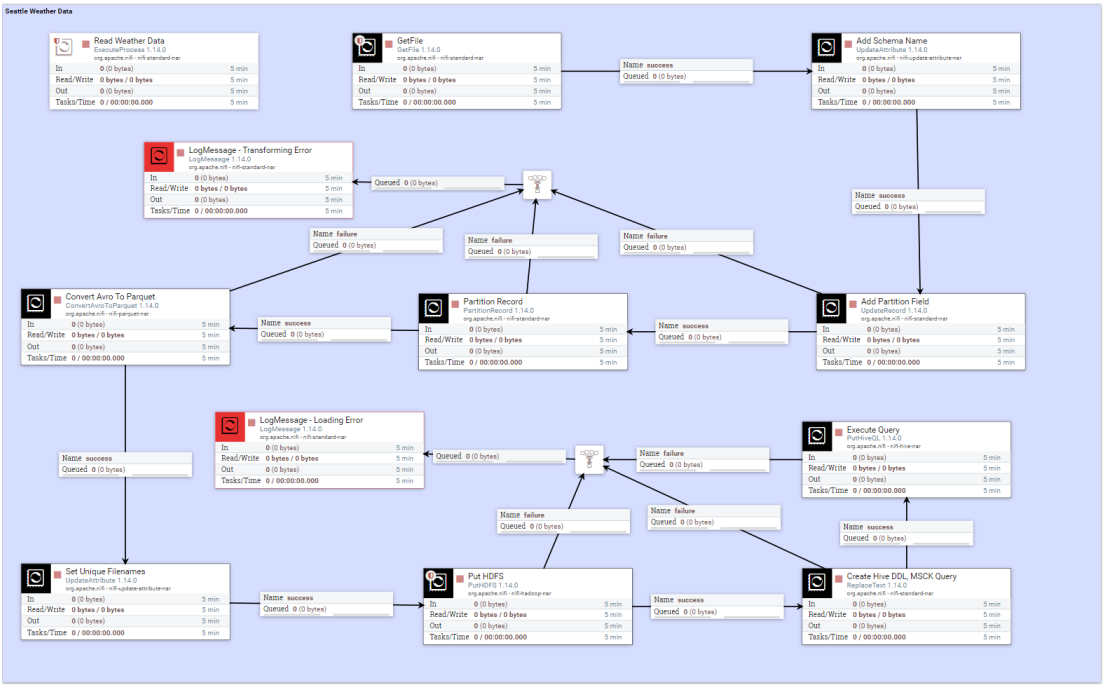
Rysunek 1: Diagram architektury systemu

3.1 Przetwarzanie danych pogodowych

Dane pogodowe są pobierane skryptem w języku `Python` przy użyciu processora `NiFi` – `ExecuteProcess`. Skrypt jest uruchamiany codziennie o 09:00 czasu Polskiego, co przekłada się na 00:00 czasu Seattle. Pobierane są wtedy dane pogodowe za dzień poprzedni. Dane zapisywane są w formacie `json` w lokalnym systemie plików pod ścieżką: `./data/weather/seattle-weather.json`. Następnie, celem umieszczenia w warstwie wsadowej (`HDFS`) i serwującej (`Hive`), dane przetwarzane są w następujący sposób:

1. `GetFile` – Pobranie pliku `json` z danymi pogodowymi
2. `UpdateAttribute` – Dodanie do `FlowFile` atrybut `schema.name` o wartości sch
3. `Set Dynamic Partition Property` – Zdefiniowanie schematu danych (`sch`) i konwersja do `avro`
4. `Partition Record` – Stwórzenie partycji na podstawie treści pliku `FlowFile` (partycja zawiera plik z pogodą z danego dnia poprzedniego w formacie *yyyy-MM-dd*)
5. `Convert Avro To Parquet` – Konwersja typu `avro` do `parquet`
6. `Customize File Name` – Nadanie unikalną nazwę (`UUID`) stworzonych plików z rozszerzeniem `.parquet`
7. `PutHDFS` – Umieszczenie w odpowiednich partycjach plików w `HDFS`
8. `Create Hive SQL Insertion Query` – Zbudowanie komendy do stworzenie partycjonowanej tabeli `weather` w `Apache Hive` i umieszczenia w niej nowo-pobranych danych
9. `Execute Records Insertion Query` – Wykonanie stworzonej komendy, stworzenie tabeli jeśli nie istnieje, aktualizacja danych

Potencjalne błędy na etapie wczytywania, transformacji czy ładowania danych są logowane w celu łatwej weryfikacji.

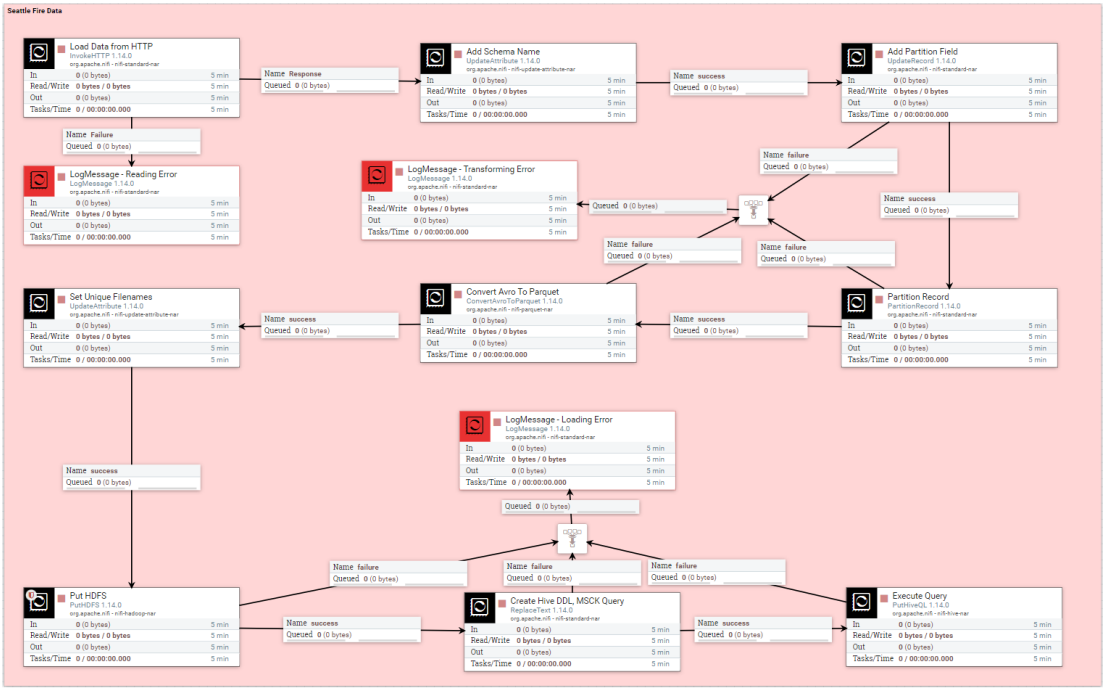


Rysunek 2: Przetwarzanie danych pogodowych w Seattle przy użyciu `Apache NiFi`

3.2 Przetwarzanie danych zgłoszeń do straży pożarnej

Dane dotyczące zgłoszeń do straży pożarnej w Seattle są pobierane ze źródła przy użyciu natywnego narzędzia `Apache NiFi - InvokeHTTP`. Źródło odświeża dane w 5-minutowych cyklach, więc, żeby zapewnić jak najwyższą dokładność, przedstawiane rozwiązanie będzie konsumować napływające zgłoszenia co 10 minut. Wykonując przy użyciu `Apache NiFi` odpowiednie przekształcenia na napływających danych, umieścimy je w warstwie *batch layer* (`Apache Hadoop`) i warstwie *serving layer* (`Apache Hive`).

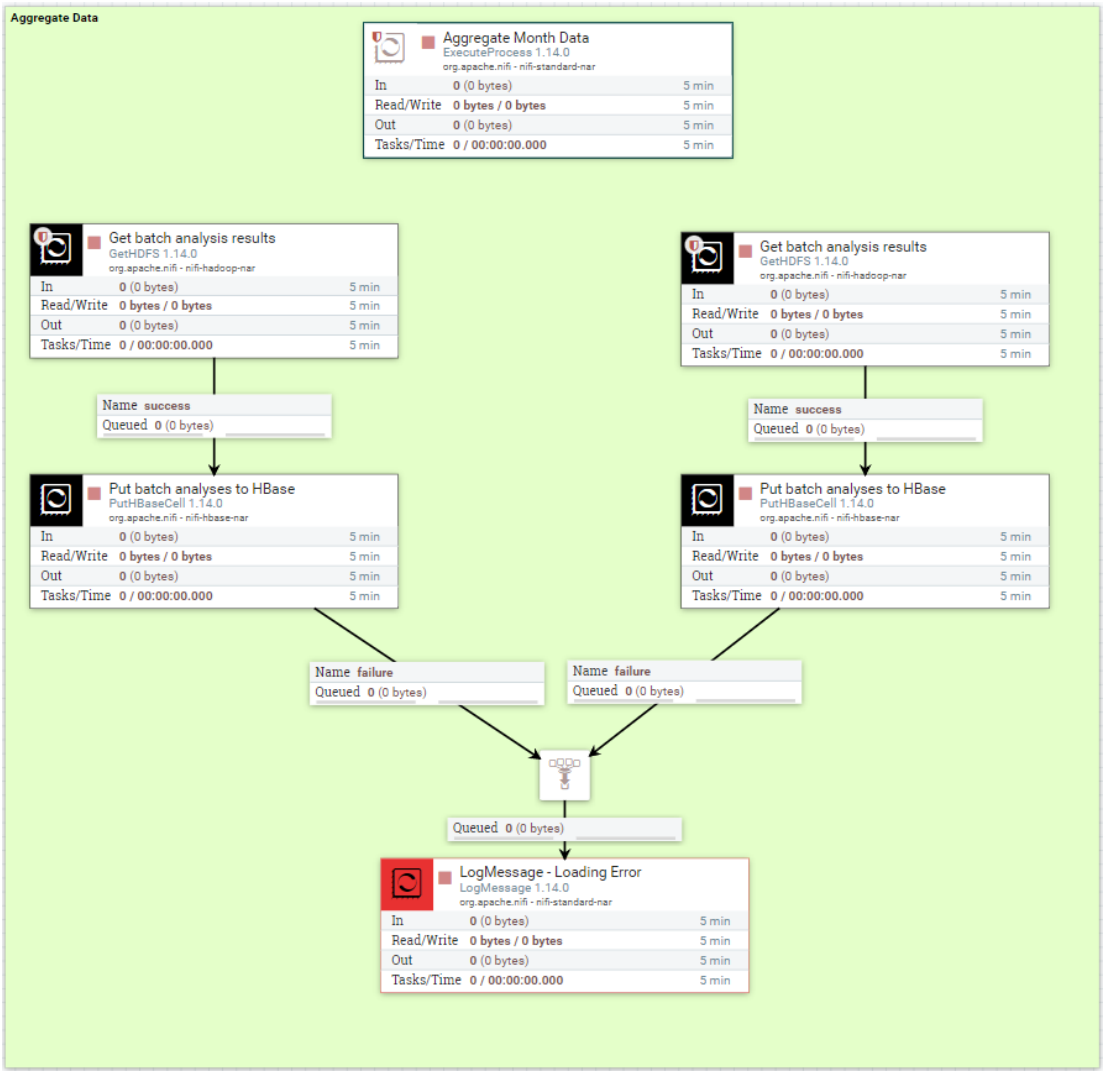
1. `Load Data via REST API` – Wykonanie zapytania `HTTP Get` do `Seattle Fire 911 Calls API`
2. `UpdateAttribute` – Dodanie do `FlowFile` atrybut `schema.name` o wartości sch (potrzebny do zdefiniowania schematu danych pochodzących ze źródła)
3. `Set Dynamic Partition Property` – Zdefiniowanie schematu danych pochodzących ze źródła (`sch`) i konwersja do `avro`
4. `Partition Record` – Stworzenie partycji na podstawie treści pliku `FlowFile` (partycja zawiera wszystkie zgłoszenia z danego dnia w formacie *yyyy-MM-dd*)
5. `Convert Avro To Parquet` – Konwersja typu `avro` do `parquet`
6. `Customize File Name` – Nadanie unikalną nazwę (`UUID`) stworzonych plików z rozszerzeniem `.parquet`
7. `PutHDFS` – Umieszczenie w odpowiednich partycjach plików w `HDFS`
8. `Create Hive SQL Insertion Query` – Zbudowanie komendy do stworzenie partycjonowanej tabeli `fire` w `Apache Hive` i umieszczenia w niej nowo-pobranych danych
9. `Execute Records Insertion Query` – Wykonanie stworzonej komendy, stworzenie tabeli jeśli nie istnieje, aktualizacja danych Potencjalne błędy na etapie wczytywania, transformacji czy ładowania danych są logowane w celu łatwej weryfikacji.



Rysunek 3: Przetwarzanie danych zgłoszeń do straży pożarnej w Seattle przy użyciu `Apache NiFi`

3.3 Agregacja danych

Celem przyśpieszenia warstwy analityki, dane pogodowe i zgłoszeń do straży pożarnej są agregowane przy użyciu `Apache Spark` i umieszczane w `HBase` w cyklach dobowych. Na zagregowane dane, stworzone zostały dwie tabele, pierwsza w podziale na typ zgłoszenia i dzień, druga na typ i godzinę zgłoszenia. Wyliczone agregację zawierają średnie, minimalne i maksymalne statystyki pogodowe w danej grupie.



Rysunek 4: Agregacja danych przy użyciu PySpark i umieszczenie w HBase.

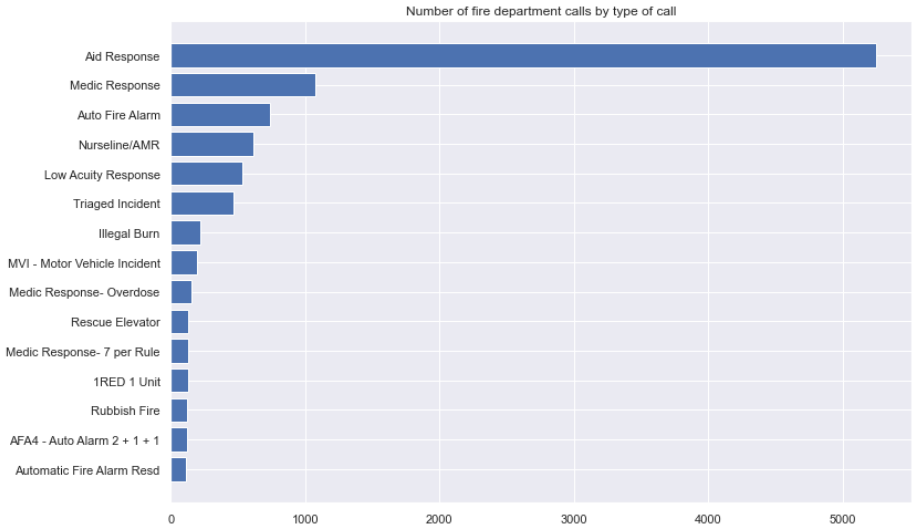
4 Warstwa analityczna

Jako warstwę analityczną przygotowaliśmy ‘jupyter notebook‘ zawirający wizualizacje stworzone przy użyciu danych, których używaliśmy w tym projekcie. Jako przykład posłużyliśmy się danymi z grudnia 2022 roku. Wizualizacje powstały na podstwie stworzonych we wcześniejszej warstwie tabel zagregowanych.

4.1 Analiza telefonów alarmowych

Zanim przeszliśmy do analizy tego jak pogoda może wpływać na liczbę i typy zgłoszeń, przyjrzeliśmy się najpierw jak wyglądają same dane dotyczące zgłoszeń na numer alarmowy. Uznaliśmy, że dane z przeszłości mogą również nieść dużo informacji potrzebnych do tworzenia predykcji liczby ewentualnych zgłoszeń w przyszłości.

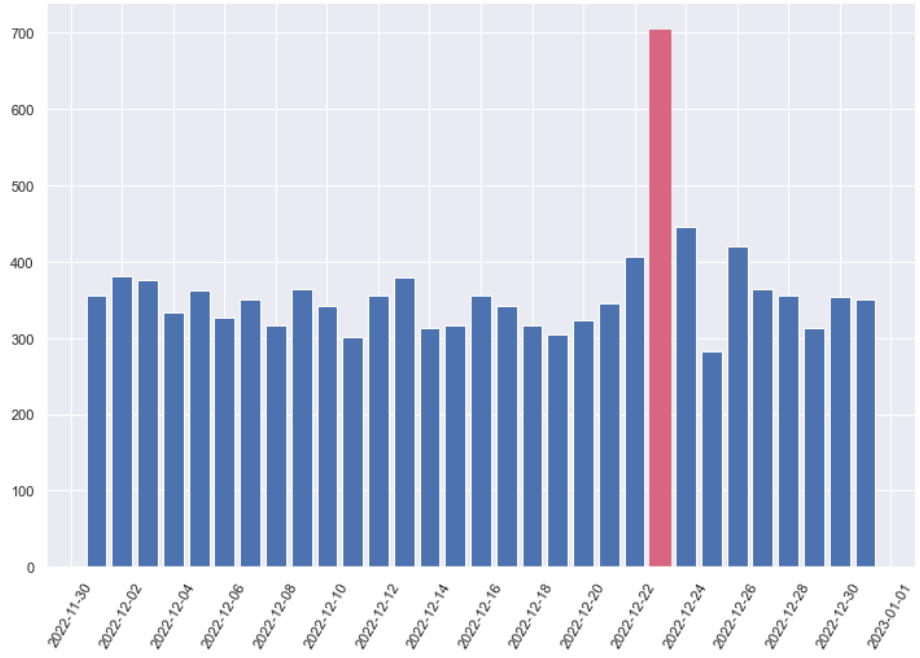
Na początku sprawdziliśmy, których z typów zgłoszeń było najwięcej. W tym celu stworzyliśmy prosty wykres słupkowy. Na poniższym wykresie widać że jeden typ zgłoszeń był zdecydowanie dominujący.



Rysunek 5: Najczęstrze typy zgłoszeń

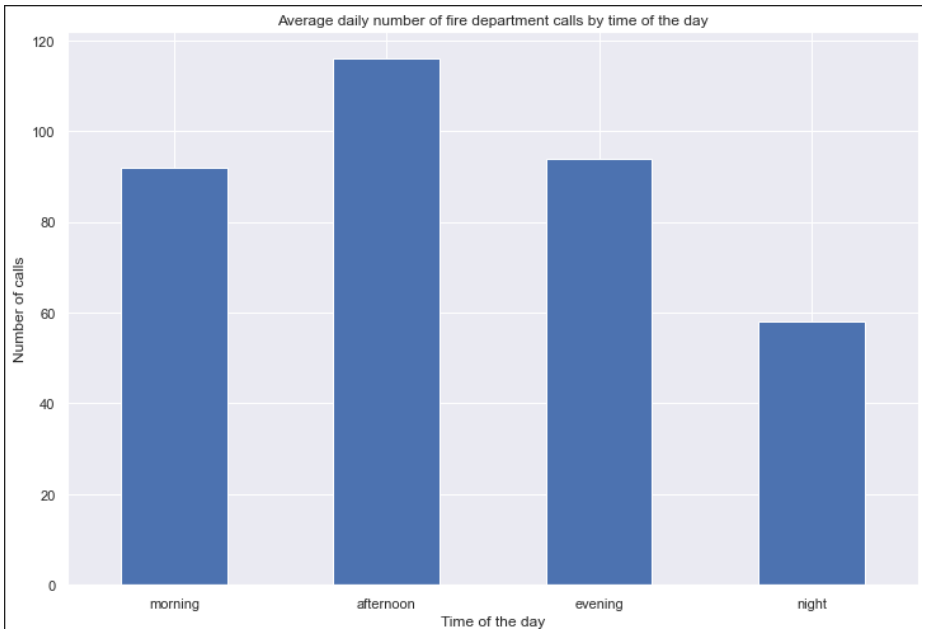
Następnie przyjrzelismy się jak wygląda rozkład liczby zgłoszeń w ciągu całego miesiąca. Na poniższym wykresie widać ile zgłoszeń było w poszczególnych dniach w grudniu 2022 roku. widać wyraźnie jaki wpływ na liczbęzgłoszeń miały święta Bożego Narodzenia. Zdecydowanie najwięcej zgłoszeń było 23.12. Prawdopodobnie dużo osób tego dnia przemieszczało się w inne miejsca co sprzyja wypadką. Możliwe, że dużo osób było pijanych tego dnia. Duży spadek w zgło-

szeniach został odnotowany 25.12. Można założyć, że tego dnia wiele osób odpoczywało po świętach.



Rysunek 6: Liczba zgłoszeń w grudniu 2022 roku

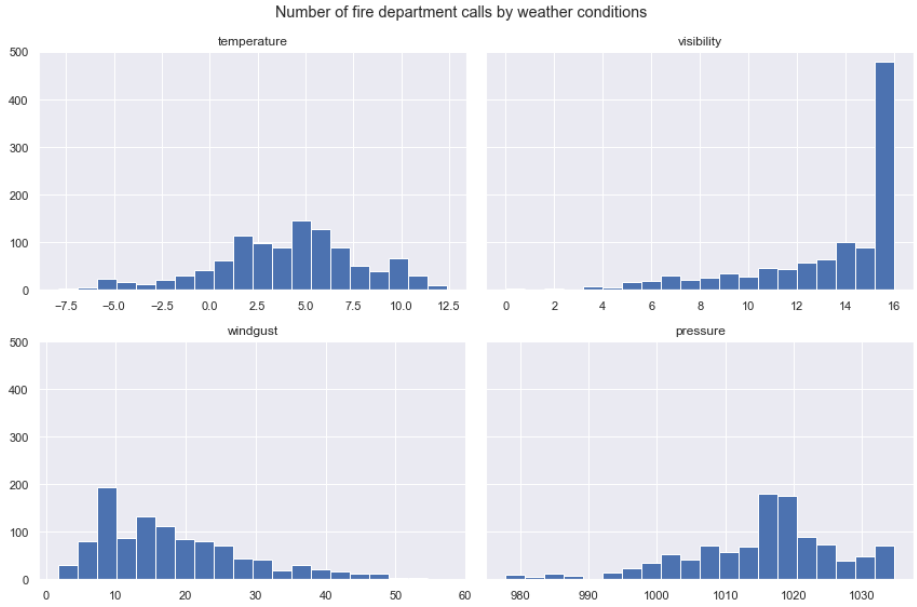
Stworzyliśmy również podobną wizualizację dla dnia. Sprawdziliśmy jak zmienia się liczba zgłoszeń w ciągu doby. pogrupowanliśmy godziny w cztery kategorie. Noc (24:00 - 06:00), poranek (06:00 - 12:00), popołudnie (12:00 - 18:00), wieczór (18:00 - 24:00). Widać wyraźnie, że najwięcej zgłoszeń jest popołudniu, a najmniej w nocy. Nie jest to nic odkrywczego, ale zgadza się z naszą intuicją.



Rysunek 7: Liczba w ciągu dnia

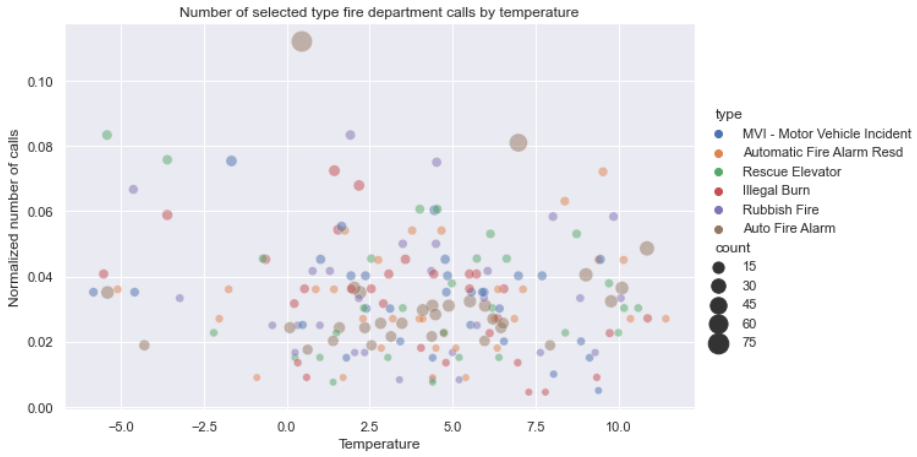
4.2 Liczba zgłoszeń a pogoda

Następnie zbadaliśmy jaki wpływ na liczbę zgłoszeń mają warunki pogodowe. Sprawdziliśmy jak wyglądają rozkłady liczby zgłoszeń w zależności od warunków pogodowych. Wyniki zamieściliśmy na poniższym wykresie. Widać na nim, że rozkłady dla widoczności i siły wiatru mają ciekawe zależności. Co ciekawe najwięcej zgłoszeń było przy najlepszej widoczności. Prawdopodobnie dlatego, że dobr widoczność występuje najczęściej.

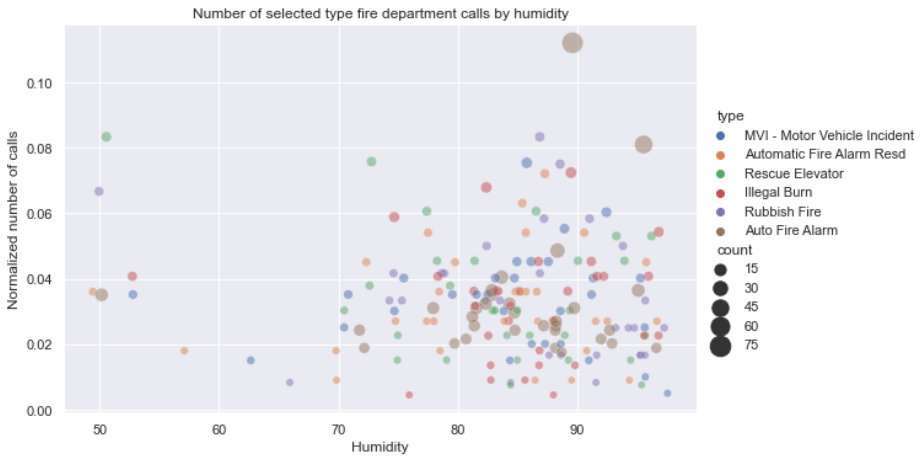


Rysunek 8: Liczba zgłoszeń w zależności od warunków pogodowych

Dodatkowo stworzyliśmy również dwa wykresy punktowe które pokazują jak wilgotność i temperatura wpływa na poszczególne typy zgłoszeń.



Rysunek 9: Wpływ temperatury na zgłoszenia



Rysunek 10: Wpływ wilgotności na zgłoszenia

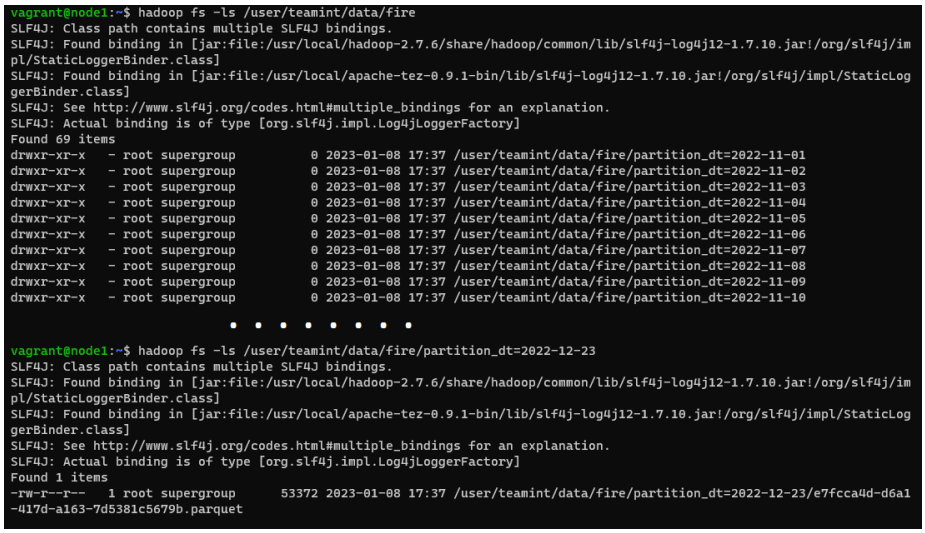
5 Testy

Opis testu	Weryfikacja zapisu spartycjonowanych po dacie danych zgłoszeń w `HDFS` w formacie `parquet`
Przebieg	Uruchomienie procesu pobierającego dane zgłoszeń i umieszczającego je w `HDFS`
Oczekiwany wynik	Pliki znajdują się w `HDFS`

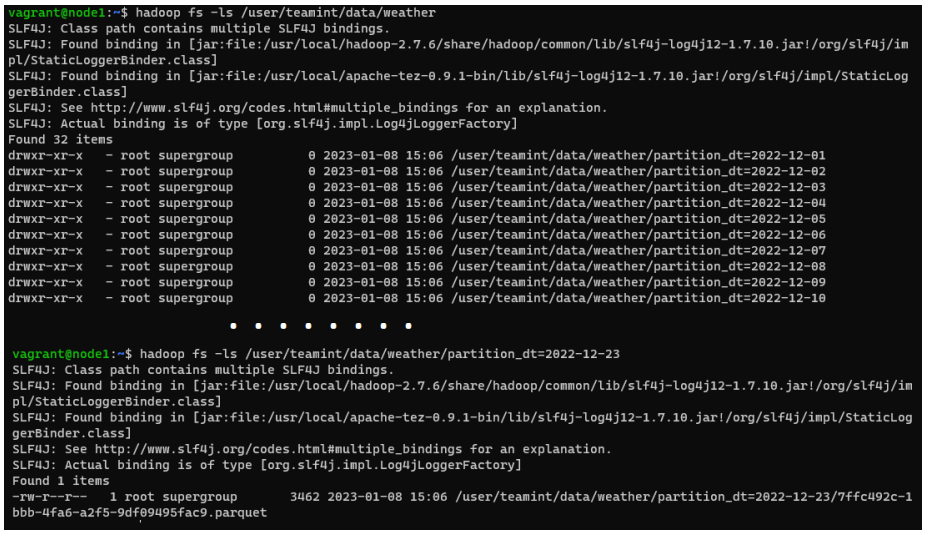
Tabela 1: Test umieszczenia spartycjonowanych danych zgłoszeń w `HDFS`

Opis testu	Weryfikacja zapisu spartycjonowanych po dacie danych pogodowych w `HDFS` w formacie `parquet`
Przebieg	Uruchomienie procesu pobierającego dane pogodowe i umieszczającego je w `HDFS`
Oczekiwany wynik	Pliki znajdują się w `HDFS`

Tabela 2: Test umieszczenia spartycjonowanych danych pogodowych w `HDFS`



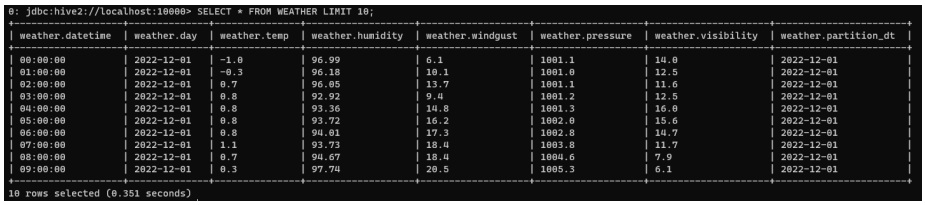
Rysunek 11: Potwierdzenie testu 1



Rysunek 12: Potwierdzenie testu 2

Opis testu	Weryfikacja zapisu do tabeli `weather` w `Apache Hive`
Przebieg	Uruchomienie procesu pobierającego dane pogodowe i umieszczającego je w `Apache Hive`
Oczekiwany wynik	Zapytanie w `Apache Hive` zwraca oczekiwane dane

Tabela 3: Test umieszczenia danych w tabeli `weather` w `Apache Hive`

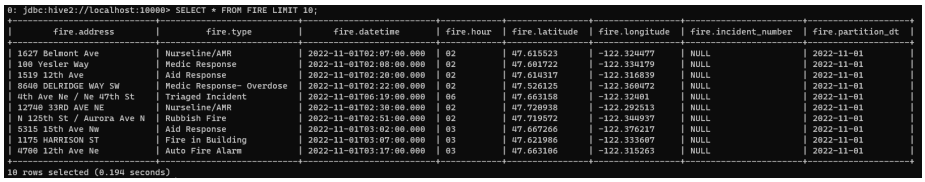


```
0: jdbc:hive2://localhost:10000> SELECT * FROM WEATHER LIMIT 10;
+---------------------+--------------+---------------+-----------------------+-----------------------+-----------------------+-----------------------+---------------------------+
| weather.datetime    | weather.day  | weather.temp  | weather.humidity      | weather.windgust      | weather.pressure      | weather.visibility    | weather.partition_dt      |
+---------------------+--------------+---------------+-----------------------+-----------------------+-----------------------+-----------------------+---------------------------+
| 00:00:00            | 2022-12-01   | -1.0          | 96.99                 | 6.1                   | 1001.1                | 14.0                  | 2022-12-01                |
| 01:00:00            | 2022-12-01   | -0.3          | 96.18                 | 10.1                  | 1001.0                | 12.5                  | 2022-12-01                |
| 02:00:00            | 2022-12-01   | 0.7           | 96.85                 | 13.7                  | 1001.1                | 11.6                  | 2022-12-01                |
| 03:00:00            | 2022-12-01   | 0.8           | 92.92                 | 9.4                   | 1001.2                | 12.5                  | 2022-12-01                |
| 04:00:00            | 2022-12-01   | 0.8           | 93.36                 | 14.8                  | 1001.3                | 16.0                  | 2022-12-01                |
| 05:00:00            | 2022-12-01   | 0.8           | 93.72                 | 16.2                  | 1002.0                | 15.6                  | 2022-12-01                |
| 06:00:00            | 2022-12-01   | 0.8           | 94.01                 | 17.3                  | 1002.8                | 14.7                  | 2022-12-01                |
| 07:00:00            | 2022-12-01   | 1.1           | 93.73                 | 18.4                  | 1003.8                | 11.7                  | 2022-12-01                |
| 08:00:00            | 2022-12-01   | 0.7           | 94.67                 | 18.4                  | 1004.6                | 7.9                   | 2022-12-01                |
| 09:00:00            | 2022-12-01   | 0.3           | 97.74                 | 20.5                  | 1005.3                | 6.1                   | 2022-12-01                |
+---------------------+--------------+---------------+-----------------------+-----------------------+-----------------------+-----------------------+---------------------------+
10 rows selected (0.351 seconds)
```

Rysunek 13: Potwierdzenie testu 3

Opis testu	Weryfikacja zapisu do tabeli `fire` w `Apache Hive`
Przebieg	Uruchomienie procesu pobierającego dane zgłoszeń do straży pożarnej i umieszczającego je w `Apache Hive`
Oczekiwany wynik	Zapytanie w `Apache Hive` zwraca oczekiwane dane

Tabela 4: Test umieszczenia danych w tabeli `fire` w `Apache Hive`



```
0: jdbc:hive2://localhost:10000> SELECT * FROM FIRE LIMIT 10;
+-------------------------+-------------------------+-------------------------+-----------------+-----------------------+-----------------------+-----------------------+---------------------------+
|      fire.address       |        fire.type        |     fire.datetime       | fire.hour       | fire.latitude         | fire.longitude        | fire.incident_number  | fire.partition_dt         |
+-------------------------+-------------------------+-------------------------+-----------------+-----------------------+-----------------------+-----------------------+---------------------------+
| 1627 Belmont Ave        | Nurseline/AMR           | 2022-11-01T02:07:00.000 | 02              | 47.615523             | -122.324477           | NULL                  | 2022-11-01                |
| 100 Yesler Way          | Medic Response          | 2022-11-01T02:08:00.000 | 02              | 47.601722             | -122.334179           | NULL                  | 2022-11-01                |
| 1519 12th Ave           | Aid Response            | 2022-11-01T02:20:00.000 | 02              | 47.614317             | -122.316839           | NULL                  | 2022-11-01                |
| 8640 DELRIDGE WAY SW    | Medic Response- Overdose| 2022-11-01T02:22:00.000 | 02              | 47.526125             | -122.360472           | NULL                  | 2022-11-01                |
| 4th Ave Ne / Ne 47th St | Triaged Incident        | 2022-11-01T06:19:00.000 | 06              | 47.663158             | -122.32401            | NULL                  | 2022-11-01                |
| 12740 33RD AVE NE       | Nurseline/AMR           | 2022-11-01T02:30:00.000 | 02              | 47.720938             | -122.292513           | NULL                  | 2022-11-01                |
| N 125th St / Aurora Ave N | Rubbish Fire          | 2022-11-01T02:51:00.000 | 02              | 47.719572             | -122.344937           | NULL                  | 2022-11-01                |
| 5315 15th Ave Nw        | Aid Response            | 2022-11-01T03:02:00.000 | 03              | 47.667266             | -122.376217           | NULL                  | 2022-11-01                |
| 1175 HARRISON ST        | Fire in Building        | 2022-11-01T03:07:00.000 | 03              | 47.621986             | -122.333607           | NULL                  | 2022-11-01                |
| 4700 12th Ave Ne        | Auto Fire Alarm         | 2022-11-01T03:17:00.000 | 03              | 47.663106             | -122.315263           | NULL                  | 2022-11-01                |
+-------------------------+-------------------------+-------------------------+-----------------+-----------------------+-----------------------+-----------------------+---------------------------+
10 rows selected (0.194 seconds)
```

Rysunek 14: Potwierdzenie testu 4

Opis testu	Weryfikacja zapisu zagregowanych danych miesięcznych w tabelach `HBase`
Przebieg	Uruchomienie procesu agregującego dane (dzień, godzina)
Oczekiwany wynik	Dane umieszczone są w `HBase`

Tabela 5: Test agregacji danych miesięcznych i zapisu w tabelach `HBase`

```
hbase(main):013:0> describe "aggregated_type_day"
Table aggregated_type_day is ENABLED
aggregated_type_day
COLUMN FAMILIES DESCRIPTION
{NAME => 'analysis', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', VERSIONS => '1'
, KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', COMPRESSION => '
NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE =>
 '65536', REPLICATION_SCOPE => '0'}
```

Rysunek 15: Potwierdzenie testu 5 - tabela `aggregated_type_day`

```
hbase(main):014:0> describe "aggregated_type_hour"
Table aggregated_type_hour is ENABLED
aggregated_type_hour
COLUMN FAMILIES DESCRIPTION
{NAME => 'analysis', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', VERSIONS => '1'
, KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', COMPRESSION => '
NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE =>
 '65536', REPLICATION_SCOPE => '0'}
```

Rysunek 16: Potwierdzenie testu 5 - tabela `aggregated_type_hour`

Opis testu	Weryfikacja możliwości odczytu zawartości tabeli `weather` z poziomu `Apache Spark`
Przebieg	Dostęp do danych z `Hive` przy użyciu klienta `PySpark`
Oczekiwany wynik	Dane są dostępne gotowe do dalszej analizy w notatniku

Tabela 6: Test dostępności danych z `Apache Hive` z poziomu `Apache Spark`

```
hive_context = HiveContext(spark.sparkContext)
weather = hive_context.table("weather")
weather.show(10)
```

```
+--------+----------+----+--------+--------+--------+----------+------------+
|datetime|       day|temp|humidity|windgust|pressure|visibility|partition_dt|
+--------+----------+----+--------+--------+--------+----------+------------+
|00:00:00|2022-12-23|-3.9|    62.3|    38.9|  1019.7|      16.0|  2022-12-23|
|01:00:00|2022-12-23|-4.2|   66.65|    29.5|  1018.3|      16.0|  2022-12-23|
|02:00:00|2022-12-23|-3.8|    72.3|    42.5|  1017.3|      10.6|  2022-12-23|
|03:00:00|2022-12-23|-3.8|   81.47|    37.1|  1016.5|       8.9|  2022-12-23|
|04:00:00|2022-12-23|-3.0|   82.66|    20.5|  1015.2|       7.3|  2022-12-23|
|05:00:00|2022-12-23|-2.2|   88.38|    20.5|  1015.1|       6.3|  2022-12-23|
|06:00:00|2022-12-23|-2.6|   90.06|    27.7|  1015.1|       5.7|  2022-12-23|
|07:00:00|2022-12-23|-2.7|   89.22|    20.5|  1014.4|      11.1|  2022-12-23|
|08:00:00|2022-12-23|-2.2|    85.8|    29.5|  1014.6|      14.8|  2022-12-23|
|09:00:00|2022-12-23|-2.0|   90.67|    20.5|  1015.5|       5.0|  2022-12-23|
+--------+----------+----+--------+--------+--------+----------+------------+
only showing top 10 rows
```

Rysunek 17: Potwierdzenie testu 6

Opis testu	Weryfikacja możliwości odczytu zawartości tabeli `fire` z poziomu `Apache Spark`
Przebieg	Dostęp do danych z `Hive` przy użyciu klienta `PySpark`
Oczekiwany wynik	Dane są dostępne gotowe do dalszej analizy w notatniku

Tabela 7: Test dostępności danych `fire` z `Apache Hive` z poziomu `Apache Spark`

```
hive_context = HiveContext(spark.sparkContext)
fire = hive_context.table("fire")
fire.show(10)
```

```
+--------------------+--------------------+--------------------+----+---------+-----------+------------+
|             address|                type|            datetime|hour| latitude|  longitude|partition_dt|
+--------------------+--------------------+--------------------+----+---------+-----------+------------+
|         325 9th Ave|     Auto Fire Alarm|2022-12-23T18:35:...|  18|47.603588|-122.322949|  2022-12-23|
|     4539 32nd Ave W|        Aid Response|2022-12-23T15:28:...|  15|47.661802| -122.39825|  2022-12-23|
|  605 Boylston Ave E|        Aid Response|2022-12-23T15:29:...|  15|47.624307|-122.323039|  2022-12-23|
|1311 S Massachuse...|        Aid Response|2022-12-23T15:30:...|  15|47.588437|-122.315855|  2022-12-23|
|   4101 Beacon Ave S|AFA4 - Auto Alarm...|2022-12-23T15:32:...|  15|47.566528|-122.306499|  2022-12-23|
|    3242 35th Ave Sw|Activated CO Dete...|2022-12-23T15:34:...|  15|47.574078|-122.376004|  2022-12-23|
|    8501 12th Ave Nw| Low Acuity Response|2022-12-23T15:38:...|  15|47.690632| -122.37144|  2022-12-23|
|   7301 Beacon Ave S|        Aid Response|2022-12-23T15:39:...|  15|47.536762|-122.293169|  2022-12-23|
|    1825 Harvard Ave|        Aid Response|2022-12-23T15:40:...|  15| 47.61785|-122.322173|  2022-12-23|
|    9829 26th Ave Sw|     Mutual Aid- Aid|2022-12-23T15:41:...|  15|47.515256| -122.36589|  2022-12-23|
+--------------------+--------------------+--------------------+----+---------+-----------+------------+
only showing top 10 rows
```

Rysunek 18: Potwierdzenie testu 7

Opis testu	Weryfikacja możliwości odczytu zawartości tabeli `aggregated_type_day` i `aggregated_type_hour` przy użyciu `happybase`
Przebieg	Użycie `happybase` do odczytu zagregowanych danych z `HBase`
Oczekiwany wynik	Zagregowane dane są dostępne gotowe do dalszej analizy w notatniku

Tabela 8: Test dostępności zagregowanych danych z `Apache HBase` przy użyciu `happybase`

```
df_day = load_hbase_to_df("aggregated_type_day")
df_hour = load_hbase_to_df("aggregated_type_hour")
```

```
df_day.head()
```

	type	day	count	max_temp	max_humidity	max_windgust	max_pressure	n
0	Brush Fire	2022-12-03	1	1.9	71.13	23.0	1020.1	
1	MVI - Motor Vehicle Incident	2022-12-05	4	4.2	88.22	11.9	1018.5	
2	1RED 1 Unit	2022-12-07	7	6.8	94.67	20.9	1021.9	
3	Unk Odor	2022-12-18	1	1.9	81.38	14.8	1015.1	
4	Automatic Fire Alarm False	2022-12-20	4	2.1	97.16	31.7	1022.9	

```
df_hour.head()
```

	type	hour	count
0	Rescue Lock In/Out	16:00:00	3
1	Aid Response Yellow	08:00:00	3
2	AFA4 - Auto Alarm 2 + 1 + 1	03:00:00	2
3	Illegal Burn	05:00:00	14
4	Crisis Center	18:00:00	2

Rysunek 19: Potwierdzenie testu 8

6 Podsumowanie

Dzięki opisanemu w tym dokumencie projektowi, udało się nam stworzyć rozwiązanie do przetwarzania i składowania danych dużej skali. Dostarczenie danych do warstwy *Serving Layer* i odpowiednia analityka w wartstwie prezentacji/analityki, sprawiły, że udało się rozwiązać opisany problem z perspektywy użytkownika końcowego. Co jednak najważniejsze, dzięki projektowi semestralnemu z przedmiotu *Składowanie danych w systemach Big Data*, mieliśmy okazaję zaznajomić się lepiej z szeroko używanymi w branży narzędziami open-source `Apache Software Foundation` takimi jak `Hadoop, NiFi, Hive, Spark`.