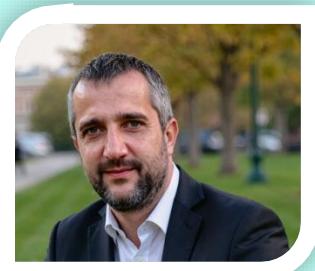


Hurtownia danych dla logistyki



**Podstawy hurtowni danych
Projekt**



Błażej Strus, 164489

 b.strus@gmail.com

 [LinkedIn](#)

Spis treści

Streszczenie.....	4
Wstęp	4
Hurtownia danych	4
Przykładowe analizy	5
Bezpieczeństwo danych i skalowalność	5
Bezpieczeństwo danych	5
Skalowalność.....	5
Modele koncepcyjne	6
Model logiczny	7
Model fizyczny.....	7
Wybór silnika – analiza i porównanie	9
Narzędzia, technologie i biblioteki.....	10
ETL	10
Instalacja	10
Prezentacja analiz	11
Szczegółowe analizy	11
Analiza 1. Rozkład incydentów w czasie (Analizy czasowe)	12
Analiza 2. Trendy czasów dostaw według miesiąca (Analizy czasowe).....	13
Analiza 3. Trendy efektywności kierowców (Analizy czasowe)	14
Analiza 4. Zmiany czasów dostaw w miesiącach (Analizy czasowe)	15
Analiza 5. Średni koszt paliwa według typu drogi i miesiąca (Analizy czasowe)	16
Analiza 6. Koszty paliwa według użycia pojazdu (Analizy transportowe)	17
Analiza 7. Najbardziej ryzykowne trasy według incydentów (Analizy transportowe)	18
Analiza 8. Najczęściej używani kierowcy (Analizy transportowe)	19
Analiza 9. Najdroższe przesyłki (Analizy transportowe)	20
Analiza 10. Odległości tras (geograficzne) (Analizy transportowe)	21

Analiza 11. Percentyle czasów dostaw według typu pojazdu (Analizy transportowe)	22
Analiza 12. Ranking kierowców według czasu dostawy (Analizy transportowe)	23
Analiza 13. Średni czas dostawy według miasta (Analizy transportowe)	24
Analiza 14. Całkowita waga według magazynu (Analizy magazynowe)	25
Analiza 15. Incydenty według typu (Analizy incydentów)	26
Analiza 16. Rozkład geograficzny magazynów (Analizy geograficzne)	27
Analiza 17. Rozkład geograficzny tras (Analizy geograficzne)	28
Schemat 1. Schemat tabeli Fact Shipments (Schematy).....	29
Schemat 2. Schemat tabeli Fact Vehicle Usage (Schematy)	30
Schemat 3. Schemat tabeli Fact Warehouse Activity (Schematy).....	31
Literatura	32

Streszczenie

Projekt polega na stworzeniu hurtowni danych w oparciu o bazę PostgreSQL, zaprojektowanej do przechowywania i analizy danych logistycznych. System obejmuje generowanie danych wymiarowych i faktów, takich jak klienci, trasy, pojazdy, kierowcy, magazyny, produkty, daty, paliwa, incydenty, przesyłki, użycie pojazdów oraz aktywności magazynowe. Dane są generowane za pomocą biblioteki **Faker** w języku Python, a następnie zapisywane do bazy danych przy użyciu SQLAlchemy. Proces ETL (*Extract, Transform, Load*) został zaimplementowany w celu przygotowania danych do analizy. Projekt umożliwia przechowywanie danych w sposób strukturyzowany, co pozwala na łatwe tworzenie raportów i analiz biznesowych związanych z logistiką.

Wstęp

W dobie cyfryzacji i rosnącej liczby danych generowanych przez firmy logistyczne, efektywne zarządzanie danymi staje się kluczowe dla poprawy procesów operacyjnych, optymalizacji kosztów i zwiększenia konkurencyjności. Ten projekt dostarcza solidnej podstawy do przechowywania i analizy danych logistycznych, co pozwala na podejmowanie decyzji opartych na danych (*data-driven decision-making*). Dzięki hurtowni danych firmy mogą lepiej zrozumieć swoje operacje, identyfikować wąskie gardła (np. opóźnienia w dostawach), optymalizować trasy i zasoby oraz poprawiać obsługę Klienta. W kontekście ogólnym projekt wpisuje się w trend Industry 4.0, gdzie dane i analityka odgrywają kluczową rolę w transformacji cyfrowej przedsiębiorstw.

Hurtownia danych

Hurtownia danych to skoncentrowana baza danych zaprojektowana do przechowywania dużych ilości danych historycznych w sposób zoptymalizowany do analiz i raportowania. W tym projekcie hurtownia danych została zbudowana w oparciu o model gwiazdy (*star schema*), gdzie dane są podzielone na tabele wymiarowe (np. **dim_customer**, **dim_route**) i tabele faktów (np. **fact_shipments**, **fact_vehicle_usage**). Tabele wymiarowe przechowują informacje opisowe (np. dane o klientach), a tabele faktów przechowują dane ilościowe (np. waga przesyłki, koszt transportu).

Hurtownia danych umożliwia:

- **scentralizowane przechowywanie danych** – wszystkie dane są w jednym miejscu, co ułatwia dostęp i zarządzanie,
- **optymalizację analiz** – struktura gwiazdy pozwala na szybkie wykonywanie zapytań analitycznych, nawet na dużych zbiorach danych,
- **integrację danych** – dane z różnych źródeł (np. systemów ERP, GPS pojazdów) mogą być zintegrowane i ujednolicone,
- **wsparcie decyzji biznesowych** – dzięki analityce danych firmy mogą podejmować bardziej świadome decyzje.

Przykładowe analizy

1. **Analiza efektywności tras** – obliczanie średniego czasu i kosztu transportu na różnych trasach.
2. **Analiza wykorzystania pojazdów** – monitorowanie zużycia paliwa i kosztów utrzymania pojazdów.
3. **Analiza incydentów** – identyfikacja najczęściej występujących problemów (np. opóźnień, uszkodzeń) i ich wpływu na koszty.
4. **Analiza aktywności magazynowych** – śledzenie ruchów magazynowych, np. ilości towarów przyjętych i wydanych.
5. **Analiza klientów** – segmentacja klientów na podstawie typu i lokalizacji.
6. **Analiza trendów czasowych** – badanie sezonowości w przesyłkach i incydentach.

Bezpieczeństwo danych i skalowalność

Bezpieczeństwo danych

W obecnym projekcie bezpieczeństwo danych nie zostało w pełni zaimplementowane, ale można je zapewnić poprzez:

- **autoryzację i uwierzytelnianie** – użycie ról i użytkowników w PostgreSQL (np. `ship_me_user` z ograniczonymi uprawnieniami),
- **szifrowanie danych** – zastosowanie SSL dla połączeń z bazą danych i szifrowanie danych wrażliwych (np. danych klientów),
- **kopie zapasowe** – regularne tworzenie backupów bazy danych za pomocą `pg_dump`,
- **logowanie i audit** – monitorowanie dostępu do danych za pomocą logów PostgreSQL.

Skalowalność

- **pozioma** – PostgreSQL wspiera replikację (np. replikacja strumieniowa), co pozwala na rozproszenie obciążenia na wiele serwerów,

- **pionowa** – zwiększenie zasobów serwera (CPU, RAM, dyski SSD) może poprawić wydajność przy większej ilości danych,
- **optymalizacja zapytań** – użycie indeksów (np. B-tree, GIN) oraz partycjonowanie tabel (np. **fact_shipments** według daty) pozwala na skalowanie przy dużych wolumenach danych,
- **ETL i przetwarzanie wsadowe** – proces ETL można zoptymalizować, dzieląc dane na partie (*batch processing*) i używając bardziej efektywnych metod zapisu (np. **COPY** zamiast **session.add()**).

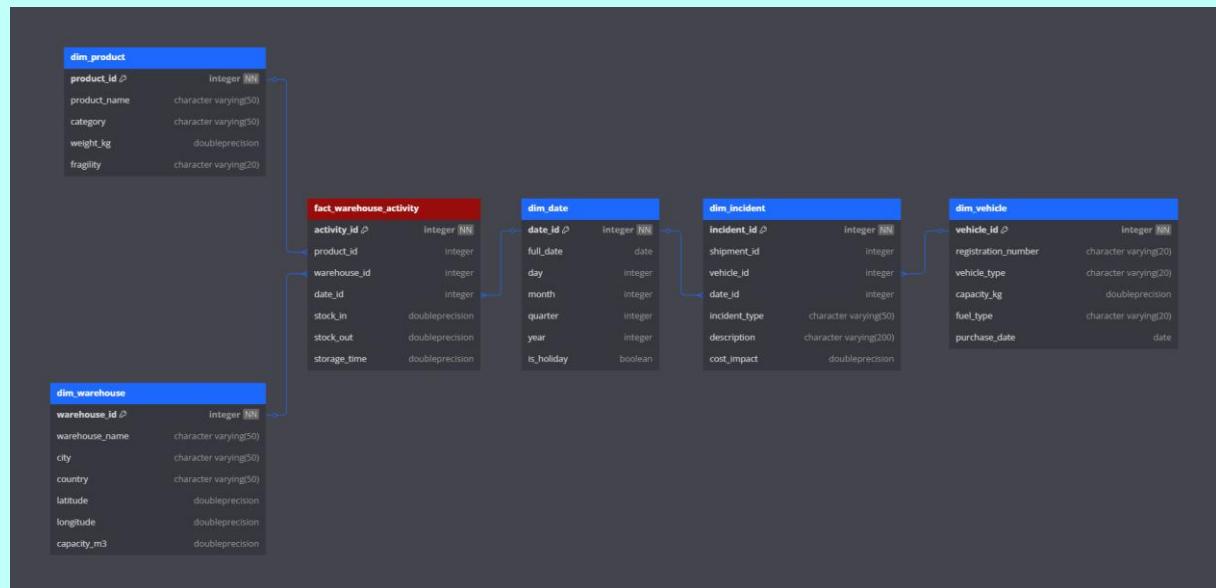
Modele koncepcyjne

Model koncepcyjny definiuje wysokopoziomową strukturę danych i relacje między encjami bez szczegółów technicznych. W tym projekcie model koncepcyjny obejmuje:

- **encje wymiarowe** – Klienci, Trasy, Pojazdy, Kierowcy, Magazyny, Produkty, Daty, Paliwa,
- **encje faktów** – Przesyłki, Użycie pojazdów, Aktywności magazynowe, Incydenty.
- **relacje** – Przesyłki łączą klientów, trasy, pojazdy, kierowców, magazyny i daty. Incydenty są powiązane z przesyłkami i pojazdami.

Przykłady relacji:

Tabela 1. Przykładowe relacje



- **1:N**: jeden klient (**dim_customer**) może mieć wiele przesyłek (**fact_shipments**),
- **1:N**: jedna trasa (**dim_route**) może być użyta w wielu przesyłkach (**fact_shipments**).

Model logiczny

Model logiczny przekształca model koncepcyjny w strukturę relacyjną z tabelami, kluczami głównymi i obcymi. W projekcie zastosowano schemat gwiazdy:

- **tabele wymiarowe** – każda tabela wymiarowa ma klucz główny (np. `customer_id`, `route_id`) i atrybuty opisowe (np. `first_name`, `start_location`),
- **tabele faktów** (np. `fact_shipments`) zawierają klucze obce do tabel wymiarowych oraz miary (np. `weight`, `shipping_cost`).

Przykłady:

Tabela 2. `dim_customer`

dim_customer	
<code>customer_id</code> ↳	integer NN
<code>first_name</code>	character varying(50)
<code>last_name</code>	character varying(50)
<code>address</code>	character varying(100)
<code>city</code>	character varying(50)
<code>country</code>	character varying(50)
<code>customer_type</code>	character varying(20)

Tabela 3. `fact_shipments`

fact_shipments	
<code>shipment_id</code> ↳	integer NN
<code>customer_id</code> ↳	integer
<code>route_id</code> ↳	integer
<code>vehicle_id</code> ↳	integer
<code>driver_id</code> ↳	integer
<code>warehouse_id</code> ↳	integer
<code>date_id</code> ↳	integer
<code>weight</code>	doubleprecision
<code>shipping_cost</code>	doubleprecision
<code>delivery_time</code>	doubleprecision

Model fizyczny

Model fizyczny definiuje implementację w konkretnej bazie danych (PostgreSQL). Obejmuje:

Błażej Strus, 164489

- **tabele i kolumny** – zdefiniowane w plikach modelu (`src.models`) za pomocą SQLAlchemy ORM,
- **typy danych** – użyto odpowiednich typów danych (np. INTEGER dla ID, VARCHAR dla nazw, FLOAT dla miar, DATE dla dat),
- **indeksy** – dodano indeksy bazujące na zapytaniach i na często używanych kolumnach (np. `customer_id` w `fact_shipments`)

```
# Indeksy dla fact_shipments
op.create_index('idx_fact_shipments_customer_id', 'fact_shipments', ['customer_id'], unique=False)
op.create_index('idx_fact_shipments_route_id', 'fact_shipments', ['route_id'], unique=False)
op.create_index('idx_fact_shipments_driver_id', 'fact_shipments', ['driver_id'], unique=False)
op.create_index('idx_fact_shipments_date_id', 'fact_shipments', ['date_id'], unique=False)
op.create_index('idx_fact_shipments_warehouse_id', 'fact_shipments', ['warehouse_id'],
unique=False)
op.create_index('idx_fact_shipments_vehicle_id', 'fact_shipments', ['vehicle_id'], unique=False)
op.create_index('idx_fact_shipments_shipping_cost', 'fact_shipments', ['shipping_cost'],
unique=False)
# Indeksy dla dim_incident
op.create_index('idx_dim_incident_shipment_id', 'dim_incident', ['shipment_id'], unique=False)
op.create_index('idx_dim_incident_date_id', 'dim_incident', ['date_id'], unique=False)
op.create_index('idx_dim_incident_incident_type', 'dim_incident', ['incident_type'], unique=False)
# Indeksy dla fact_vehicle_usage
op.create_index('idx_fact_vehicle_usage_vehicle_id', 'fact_vehicle_usage', ['vehicle_id'],
unique=False)
op.create_index('idx_fact_vehicle_usage_route_id', 'fact_vehicle_usage', ['route_id'],
unique=False)
op.create_index('idx_fact_vehicle_usage_fuel_id', 'fact_vehicle_usage', ['fuel_id'], unique=False)
op.create_index('idx_fact_vehicle_usage_date_id', 'fact_vehicle_usage', ['date_id'], unique=False)
# Indeksy dla fact_warehouse_activity
op.create_index('idx_fact_warehouse_activity_warehouse_id', 'fact_warehouse_activity',
['warehouse_id'], unique=False)
op.create_index('idx_fact_warehouse_activity_product_id', 'fact_warehouse_activity',
['product_id'], unique=False)
# Opcjonalne indeksy dla tabel wymiarów
op.create_index('idx_dim_date_month_year', 'dim_date', ['month', 'year'], unique=False)
op.create_index('idx_dim_date_full_date', 'dim_date', ['full_date'], unique=False)
op.create_index('idx_dim_vehicle_vehicle_type', 'dim_vehicle', ['vehicle_type'], unique=False)
op.create_index('idx_dim_route_road_type', 'dim_route', ['road_type'], unique=False)
```

- klucze obce – zdefiniowane w modelu `SQLAlchemy`, np. `customer_id` w `fact_shipments` wskazuje na `dim_customer(customer_id)`.

Wybór silnika – analiza i porównanie

Kryterium	PostgreSQL	MySQL	MSSQL	BigQuery
Licencja	Open Source (PostgreSQL)	Open Source (GPL)	Komercyjna (Microsoft)	Komercyjna (Google Cloud)
Wsparcie dla SQL	Pełne, zaawansowane	Pełne, ale mniej zaawansowane	Pełne, z rozszerzeniami T-SQL	SQL-like, zoptymalizowany pod analizy
Skalowalność	Dobra (replikacja, partycje)	Dobra (replikacja)	Dobra (klastry)	Bardzo dobra (chmura)
Obsługa JSON	Zaawansowana (jsonb)	Dobra (JSON)	Dobra (JSON)	Dobra
Geoprzestrzeń	Zaawansowana (PostGIS)	Ograniczona	Ograniczona	Dobra
Koszty	Bezpłatny	Bezpłatny (w wersji Community)	Płatny	Płatny (model pay-as-you-go)

Wybrałem PostgreSQL, gdyż:

- **jest Open Source** – PostgreSQL jest darmowy i ma aktywną społeczność, co obniża koszty i zapewnia wsparcie,
- **ma zaawansowane funkcje** – wspiera PostGIS (dane geoprzestrzenne), co jest istotne dla danych logistycznych (np. współrzędne tras); obsługuje też jsonb, który w tym momencie co prawda nie jest używany, ale można łatwo wyobrazić sobie dodatkowe funkcjonalności (np. opis przesyłki), w których ma zastosowanie; obsługuje zaawansowane indeksy,
- **jest skalowalny i wydajny** – PostgreSQL dobrze radzi sobie z dużymi ilościami danych dzięki replikacji i partycjonowaniu,
- **jest elastyczny** – idealny do prototypowania i dalszego rozwoju projektu,
- **jest kompatybilny z SQLAlchemy** – SQLAlchemy ma świetne wsparcie dla PostgreSQL, co ułatwia integrację.

Narzędzia, technologie i biblioteki

- **Python** – język programowania wybrany ze względu na prostotę, szeroką społeczność i dostępne biblioteki,
- **PostgreSQL** – baza danych wybrana ze względu na jej zaawansowane funkcje i skalowalność,
- **Docker** – użyty do uruchomienia PostgreSQL w kontenerze ([shipme-postgres-1](#)), co ułatwia konfigurację i zarządzanie środowiskiem.
- **SQLAlchemy** – ORM (*Object-Relational Mapping*) do definiowania modeli danych i zarządzania sesjami bazy danych – wybrany ze względu na łatwość użycia i możliwość mapowania obiektów Python na tabele w bazie,
- **Faker** – biblioteka do generowania realistycznych danych testowych (np. imiona, adresy) – wybrana ze względu na prostotę i wsparcie dla języka polskiego ([pl_PL](#)).

ETL

ETL (Extract, Transform, Load) to proces:

- **Extract** – pobieranie danych z różnych źródeł (w projekcie dane są generowane za pomocą Faker, co symuluje proces pobierania danych z systemów źródłowych),
- **Transform** – przekształcanie danych, np. czyszczenie, normalizacja, agregacja (dane są przekształcane, np. obcinanie stringów za pomocą [truncate_string](#), generowanie miar takich jak [weight](#) czy [shipping_cost](#)),
- **Load** – zapis danych do docelowej bazy danych – hurtowni danych (dane są zapisywane do PostgreSQL za pomocą SQLAlchemy – [session.add\(\)](#) i [session.commit\(\)](#)).

Projekt jest ściśle powiązany z ETL, ponieważ jego celem jest przygotowanie danych w formie odpowiedniej do analizy w hurtowni danych. Proces ETL umożliwia integrację danych, ich standaryzację i przygotowanie do dalszych analiz biznesowych.

Instalacja

Proces instalacji systemu obejmuje konfigurację środowiska deweloperskiego, bazy danych oraz uruchomienie skryptów generujących dane. Poniżej opisuję kroki instalacji:

Wymagania wstępne

- Python 3.8+ zainstalowany na systemie.
- Docker i Docker Compose (do uruchomienia PostgreSQL w kontenerze).

- Git (opcjonalnie, do klonowania repozytorium projektu).
- System operacyjny: Linux, macOS lub Windows z WSL (Windows Subsystem for Linux).

Pełna instrukcja instalacji jest w pliku [README.md](#).

Prezentacja analiz

Jak łatwo tworzyć analizy?

Dzięki zastosowaniu schematu gwiazdy w hurtowni danych tworzenie analiz jest bardzo proste i efektywne:

- **proste zapytania SQL** – struktura gwiazdy pozwala na łatwe łączenie tabel wymiarowych z tabelami faktów za pomocą kluczy obcych,
- **integracja z narzędziami BI** – dane można łatwo zaimportować do narzędzi takich jak Power BI, Tableau czy Metabase, co umożliwia tworzenie interaktywnych dashboardów,
- **agregacje i filtry** – możliwość szybkiego agregowania danych (np. SUM, AVG) i filtrowania według wymiarów (np. daty, typy incydentów),
- **Python i biblioteki analityczne** – dane można analizować w Pythonie za pomocą bibliotek takich jak Pandas i Matplotlib.

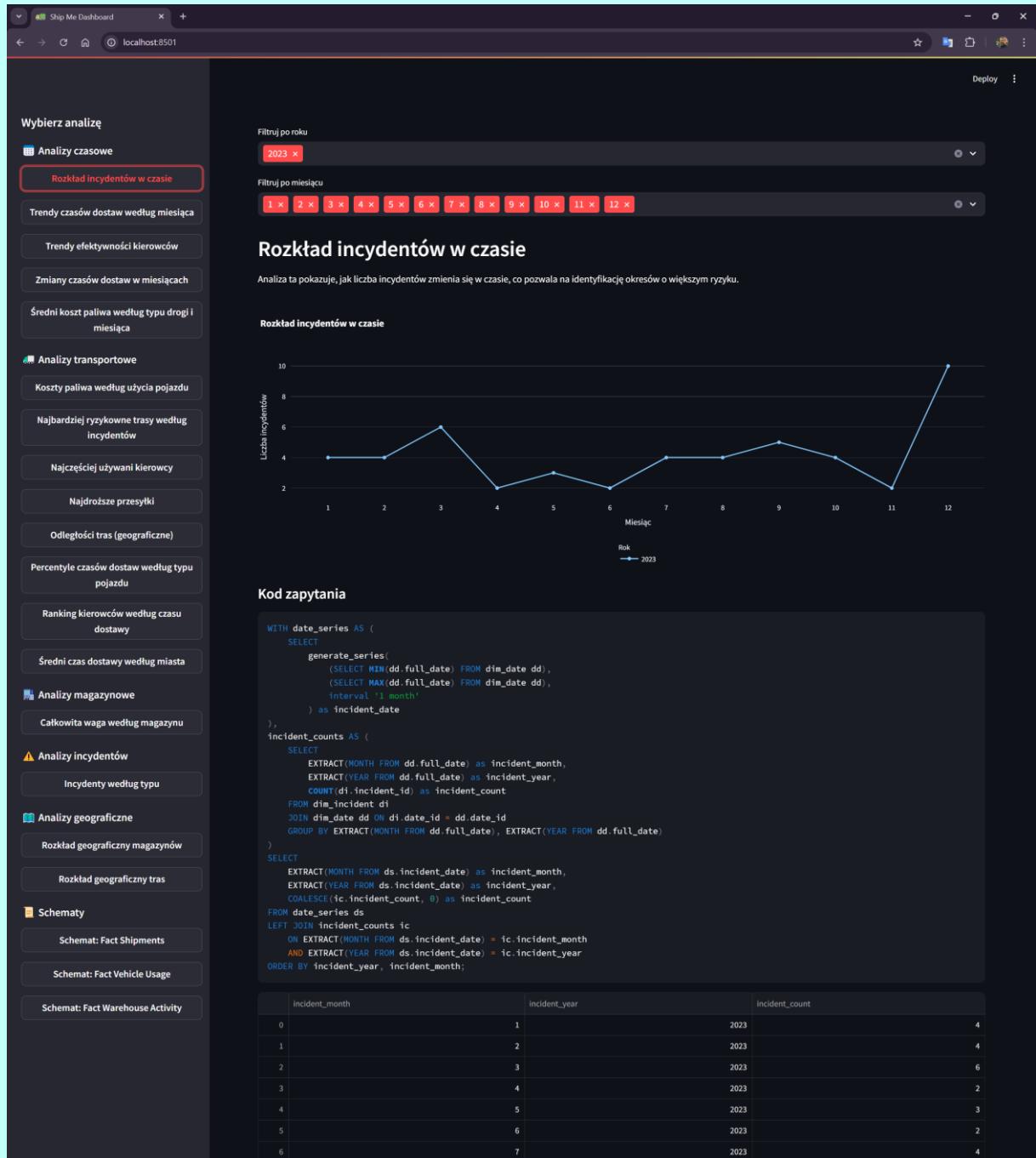
Szczegółowe analizy

Poniżej opisuję szczegółowe przykłady analiz, które można wykonać na danych w hurtowni danych w tym projekcie. Zostały przeze mnie zaimplementowane. Każda analiza zawiera zapytanie SQL oraz krótki opis potencjalnych wniosków biznesowych.

Po analizach przedstawiam i omawiam trzy schematy – dla każdej tabeli faktów po jednym.

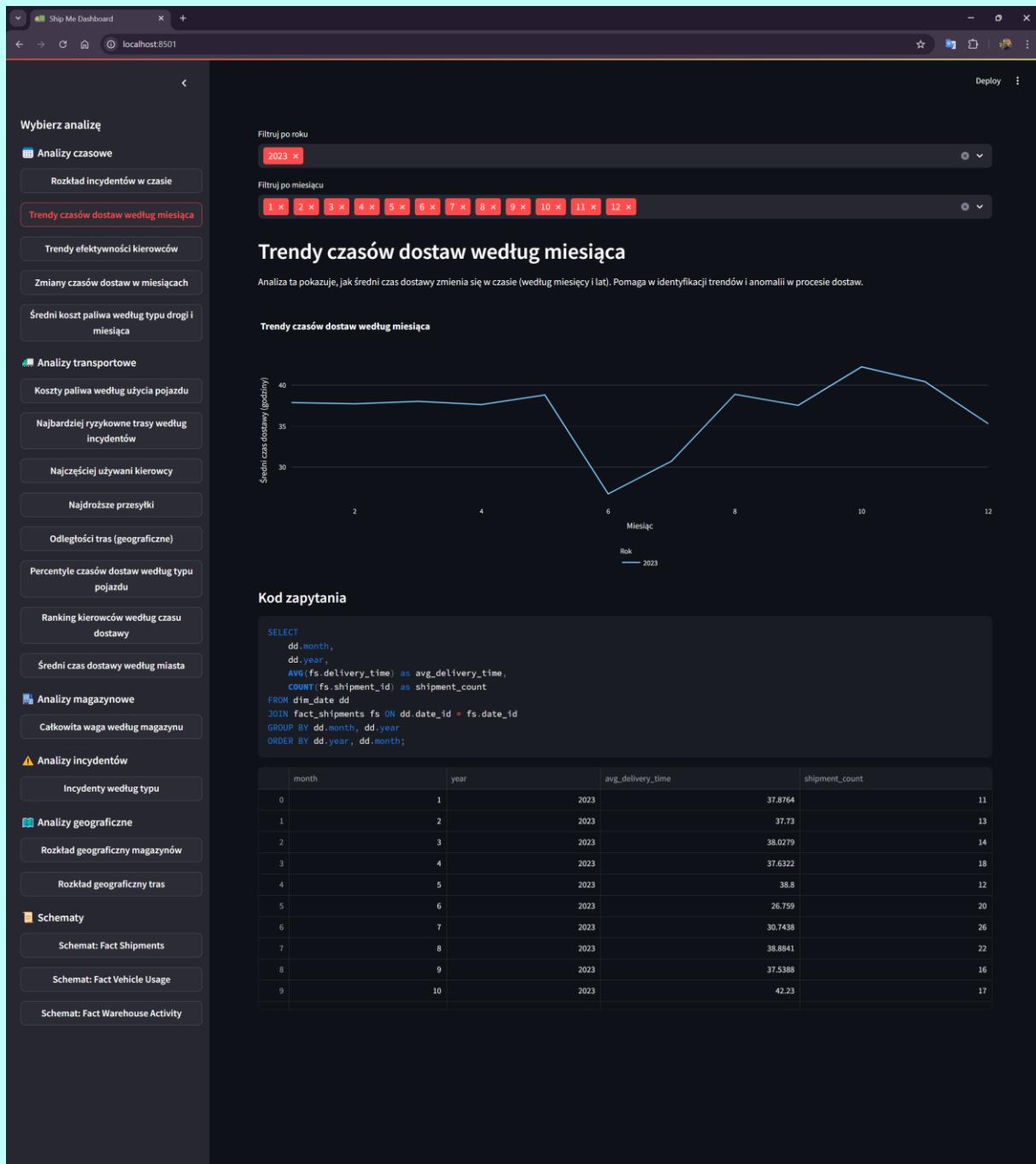
Analiza 1. Rozkład incydentów w czasie (Analizy czasowe)

Analiza ta pokazuje, jak liczba incydentów zmienia się w czasie, co pozwala na identyfikację okresów o większym ryzyku.



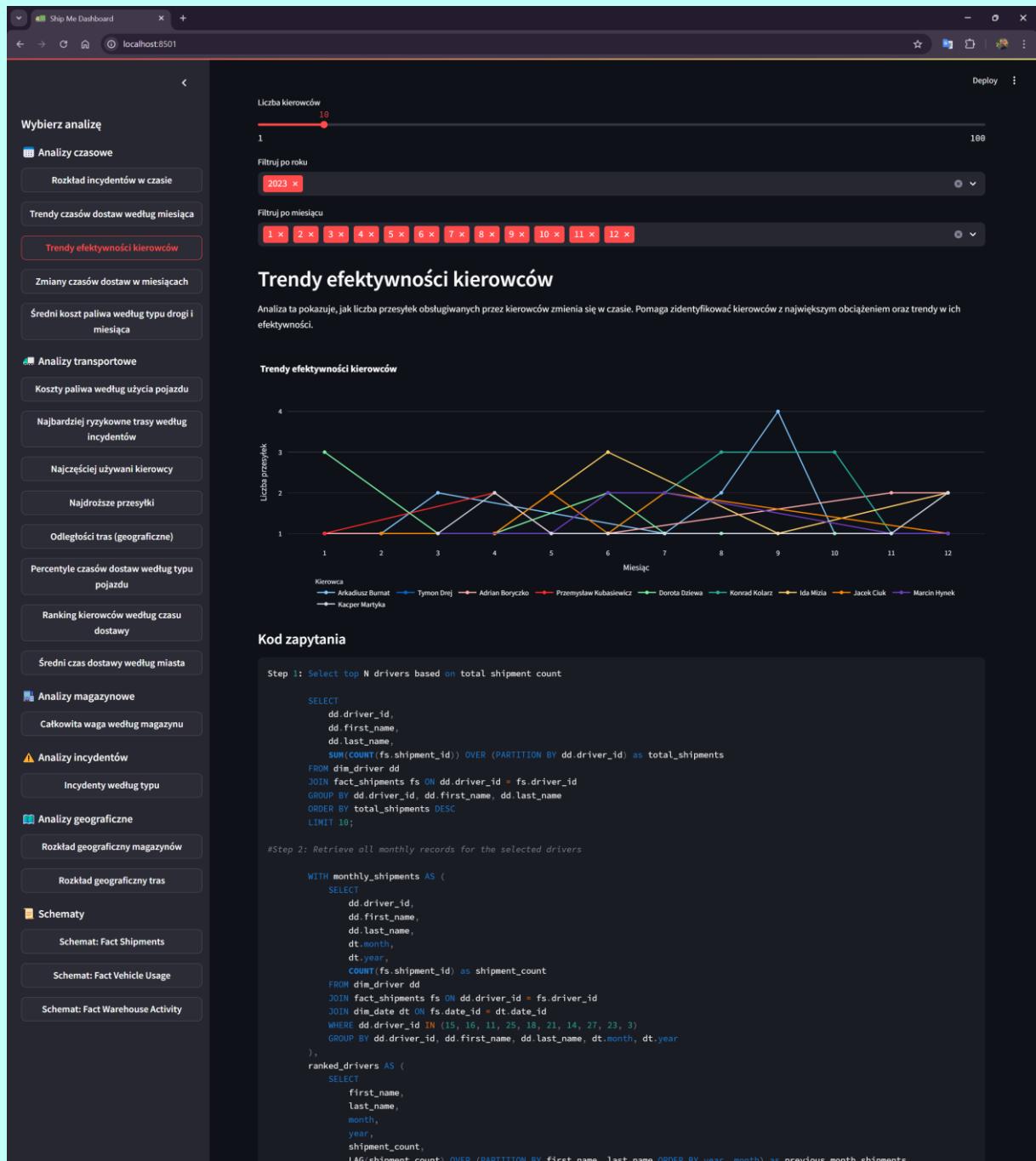
Analiza 2. Trendy czasów dostaw według miesiąca (Analizy czasowe)

Analiza ta pokazuje, jak średni czas dostawy zmienia się w czasie (według miesięcy i lat). Pomaga w identyfikacji trendów i anomalii w procesie dostaw.



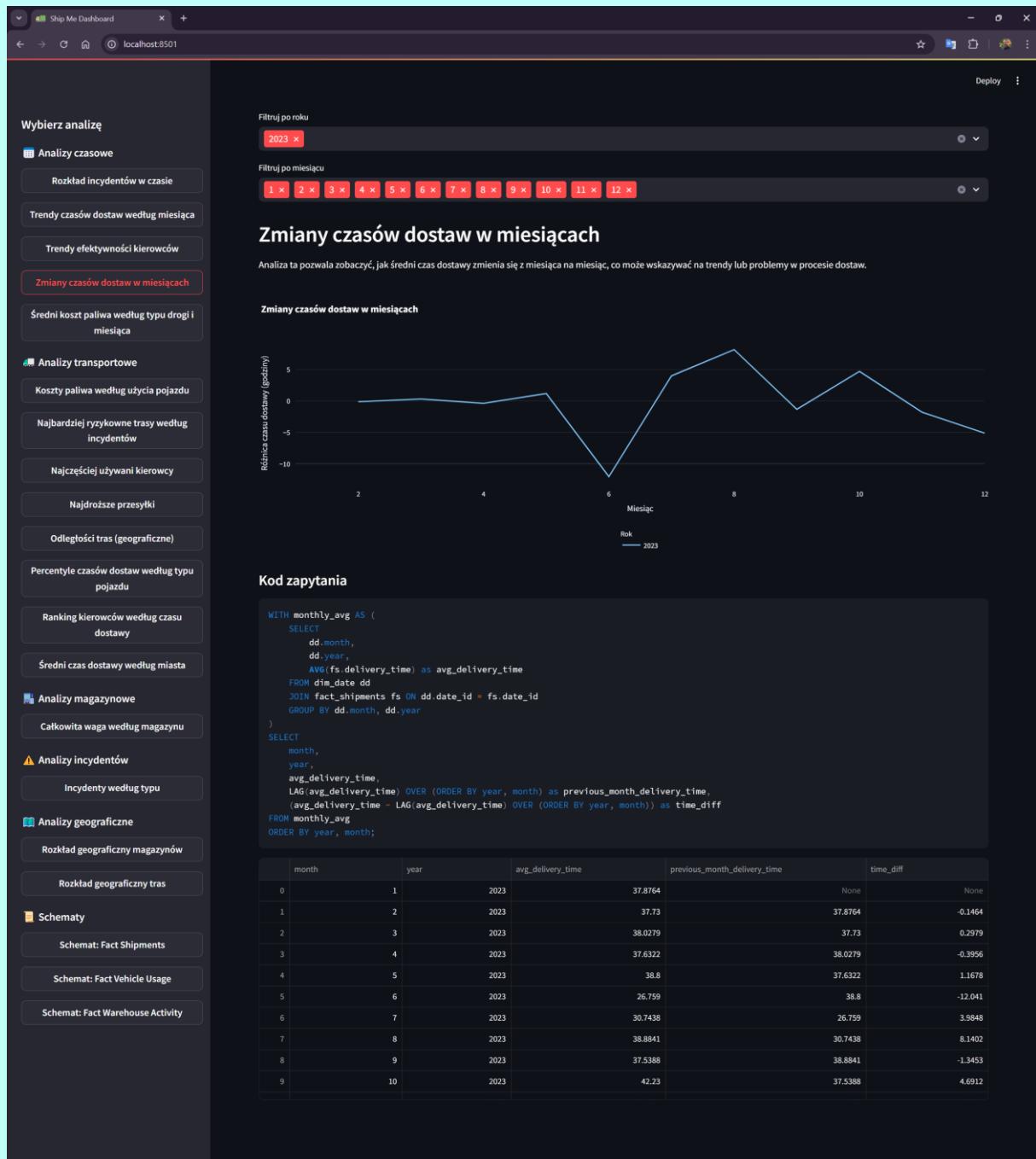
Analiza 3. Trendy efektywności kierowców (Analizy czasowe)

Analiza ta pokazuje, jak liczba przesyłek obsługiwanych przez kierowców zmienia się w czasie. Pomaga zidentyfikować kierowców z największym obciążeniem oraz trendy w ich efektywności.



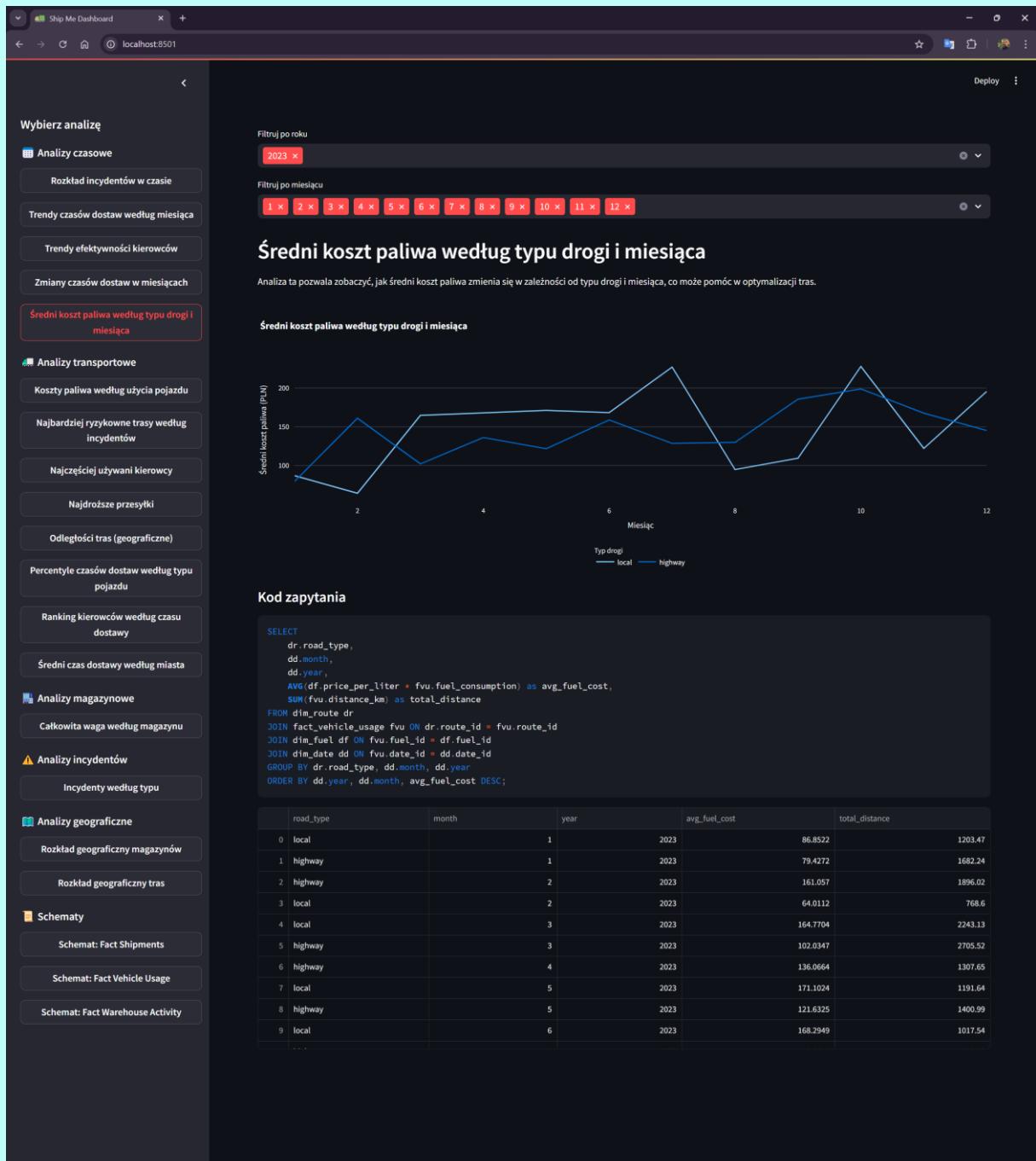
Analiza 4. Zmiany czasów dostaw w miesiącach (Analizy czasowe)

Analiza ta pozwala zobaczyć, jak średni czas dostawy zmienia się z miesiąca na miesiąc, co może wskazywać na trendy lub problemy w procesie dostaw.



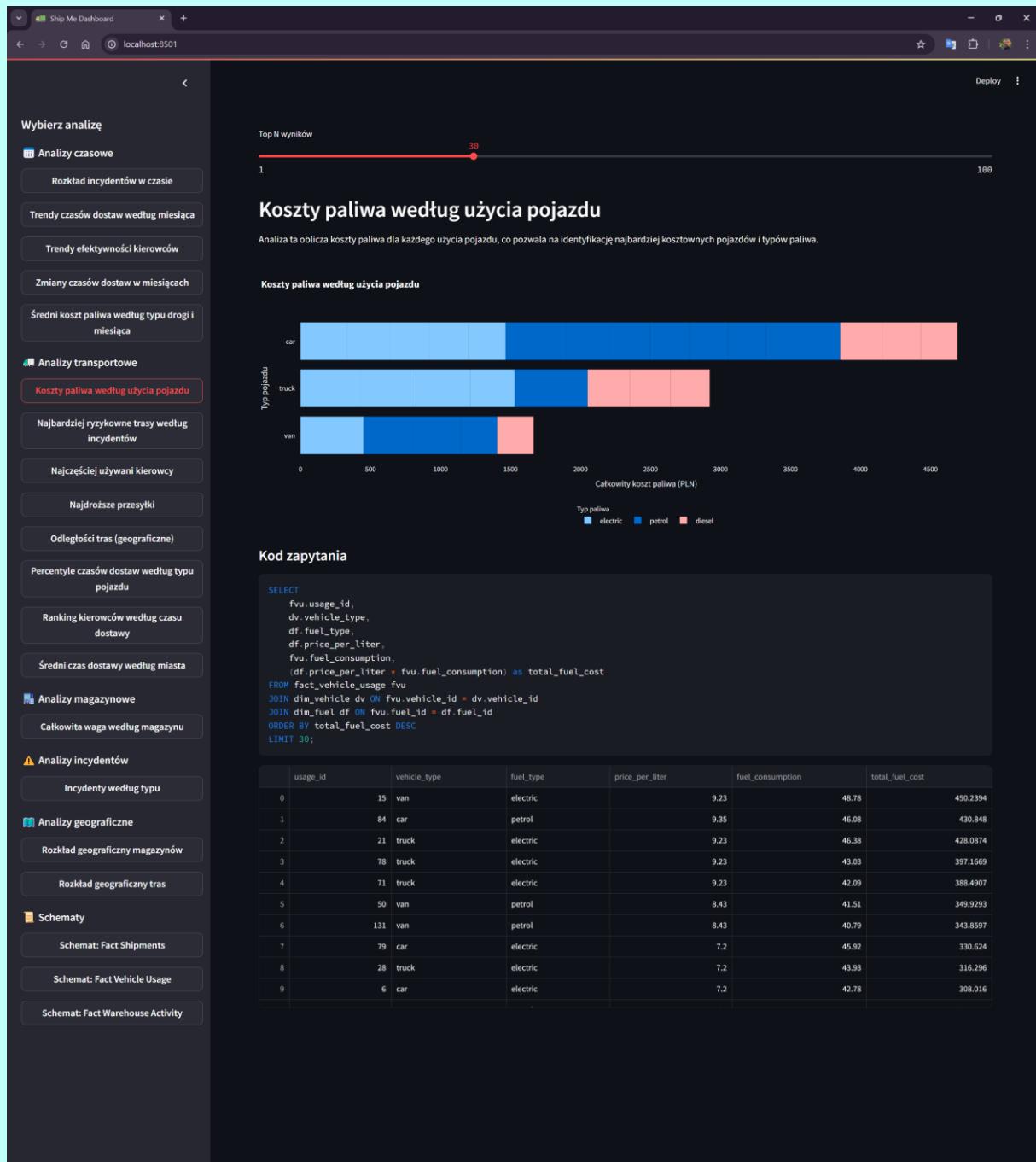
Analiza 5. Średni koszt paliwa według typu drogi i miesiąca (Analizy czasowe)

Analiza ta pozwala zobaczyć, jak średni koszt paliwa zmienia się w zależności od typu drogi i miesiąca, co może pomóc w optymalizacji tras.



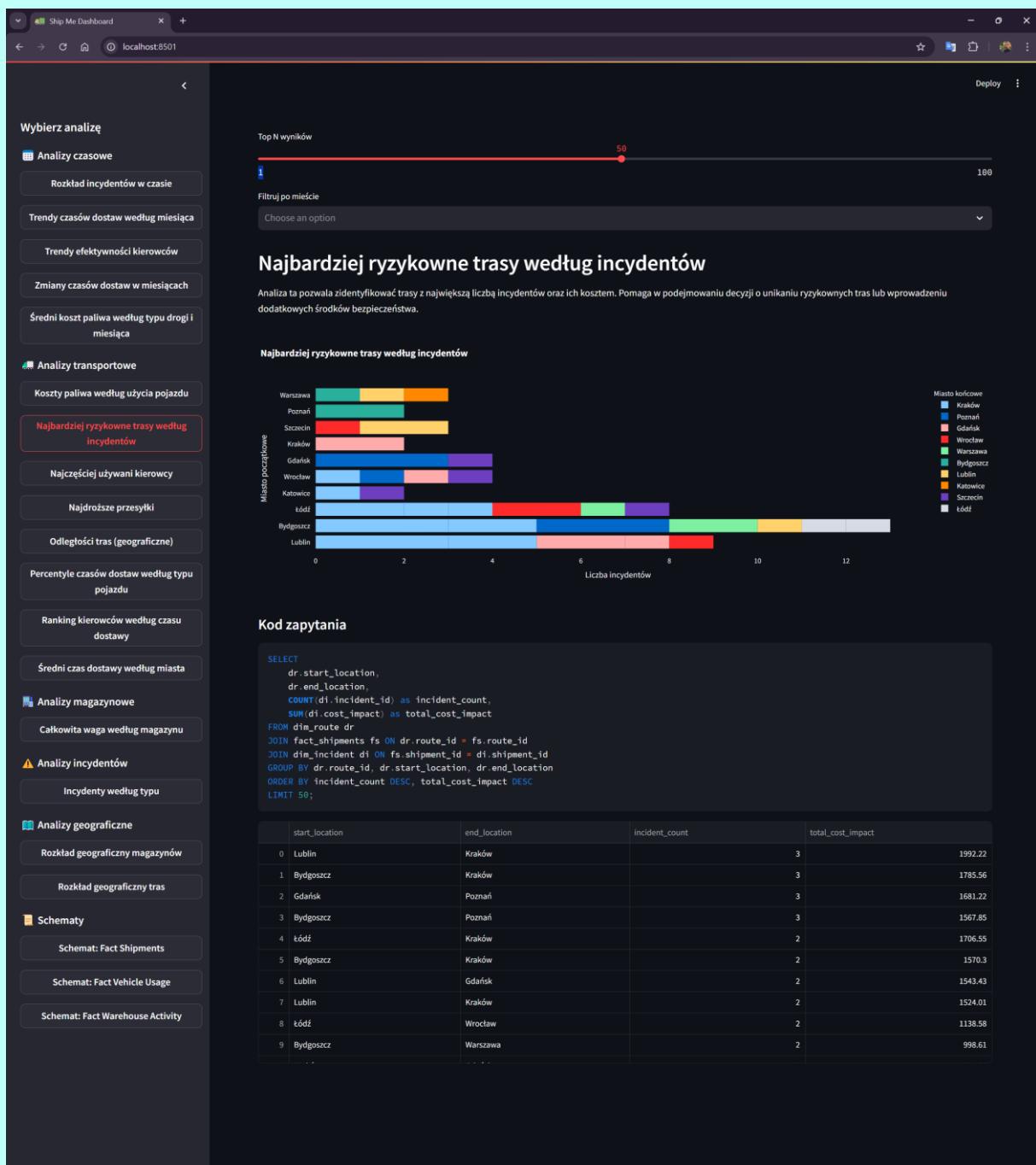
Analiza 6. Koszty paliwa według użycia pojazdu (Analizy transportowe)

Analiza ta oblicza koszty paliwa dla każdego użycia pojazdu, co pozwala na identyfikację najbardziej kosztownych pojazdów i typów paliwa.



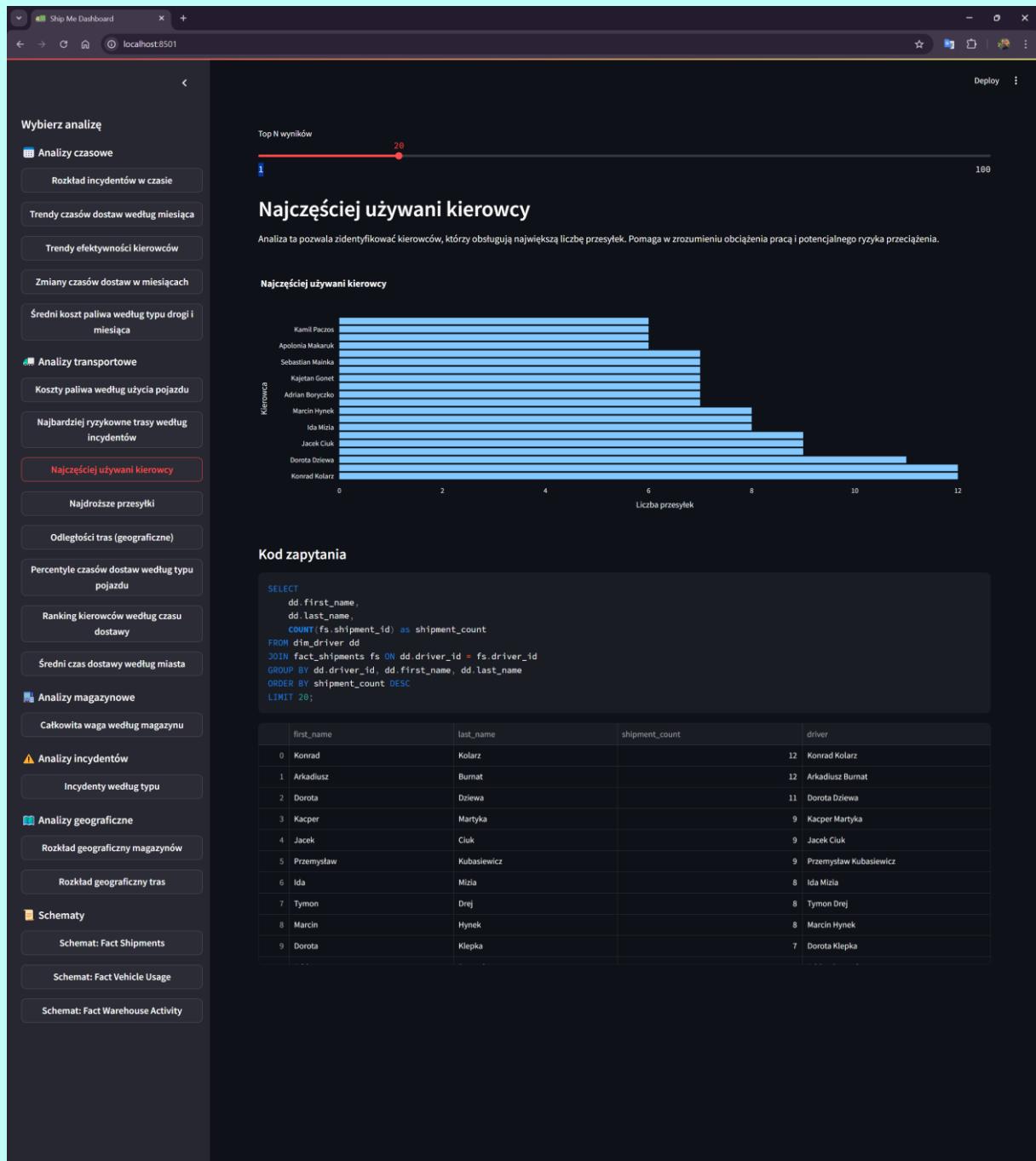
Analiza 7. Najbardziej ryzykowne trasy według incydentów (Analizy transportowe)

Analiza ta pozwala zidentyfikować trasy z największą liczbą incydentów oraz ich kosztem. Pomaga w podejmowaniu decyzji o unikaniu ryzykownych tras lub wprowadzeniu dodatkowych środków bezpieczeństwa.



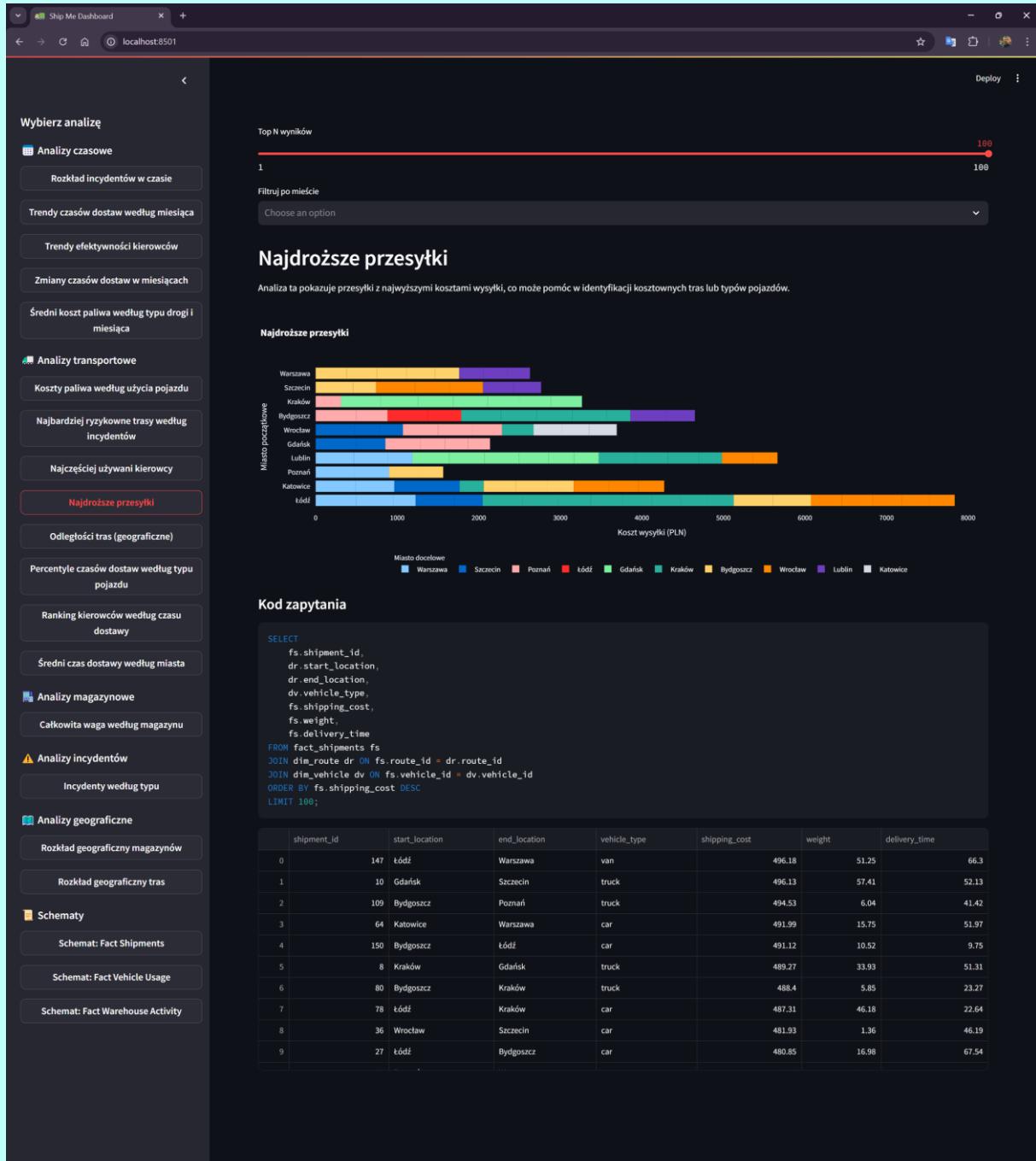
Analiza 8. Najczęściej używani kierowcy (Analizy transportowe)

Analiza ta pozwala zidentyfikować kierowców, którzy obsługują największą liczbę przesyłek. Pomaga w zrozumieniu obciążenia pracą i potencjalnego ryzyka przeciążenia.



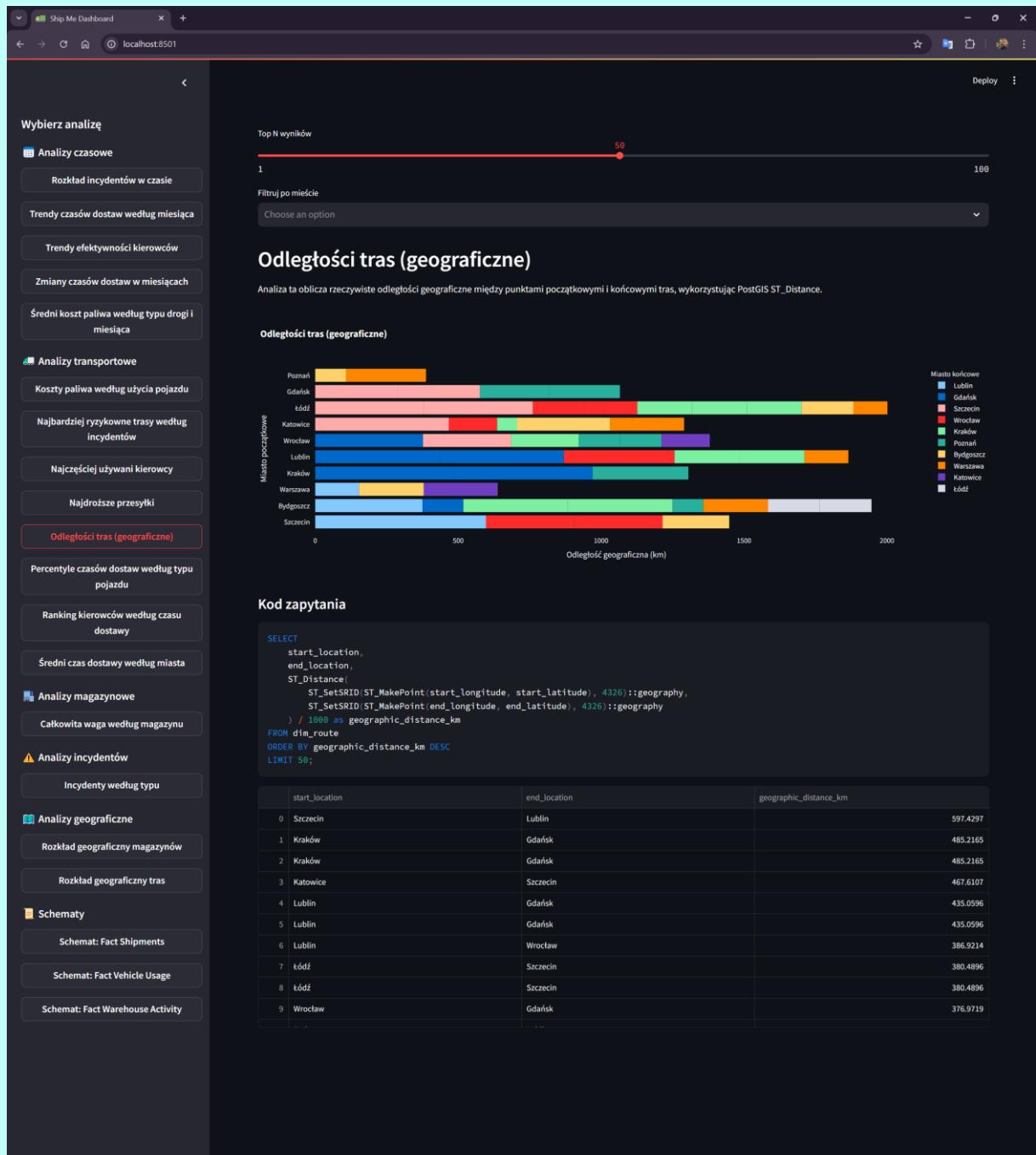
Analiza 9. Najdroższe przesyłki (Analizy transportowe)

Analiza ta pokazuje przesyłki z najwyższymi kosztami wysyłki, co może pomóc w identyfikacji kosztownych tras lub typów pojazdów.



Analiza 10. Odległości tras (geograficzne) (Analizy transportowe)

Analiza ta oblicza rzeczywiste odległości geograficzne między punktami początkowymi i końcowymi tras, wykorzystując PostGIS ST_Distance.



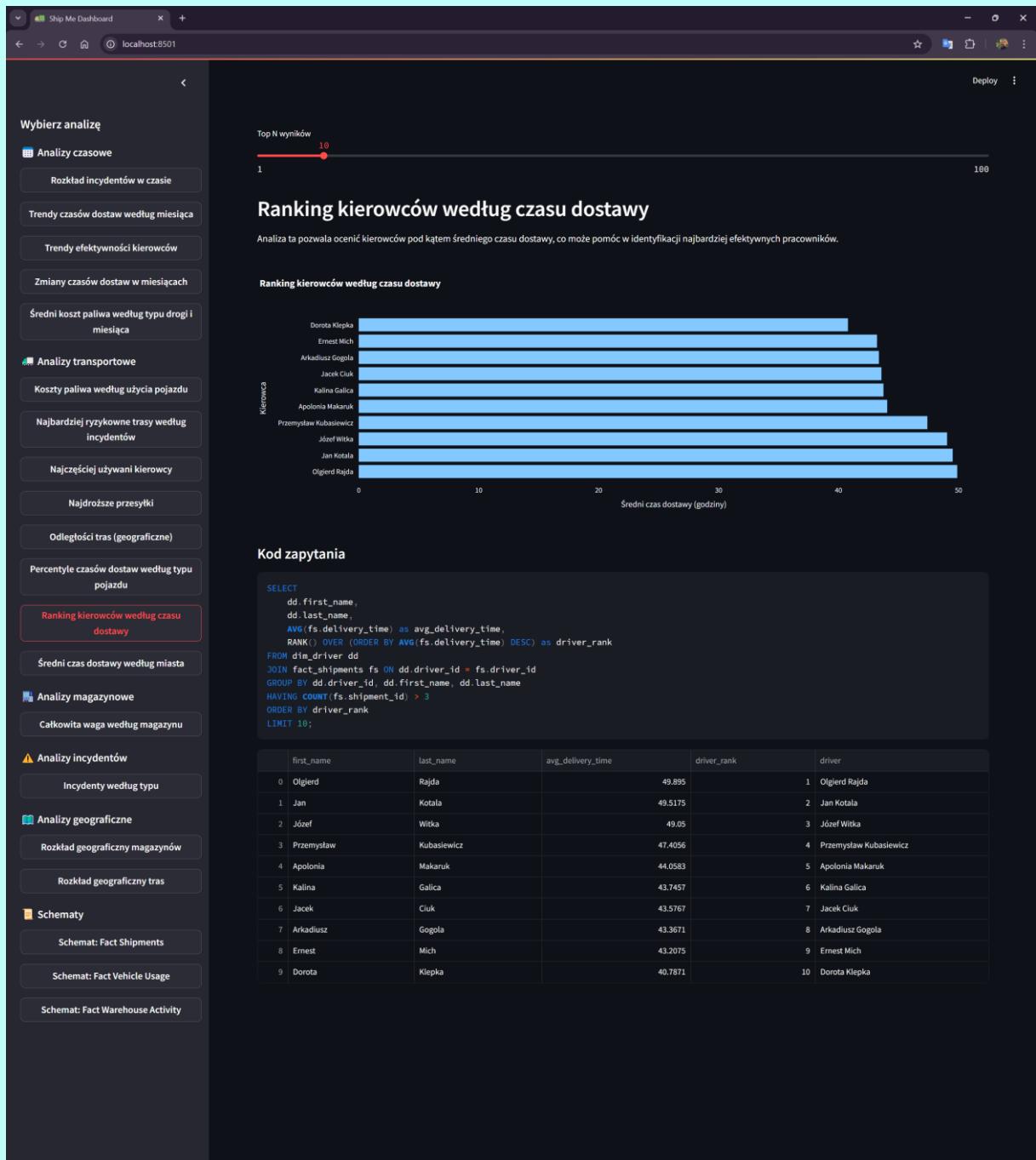
Analiza 11. Percentyle czasów dostaw według typu pojazdu (Analizy transportowe)

Analiza ta pozwala ocenić rozkład czasów dostaw dla różnych typów pojazdów, pokazując medianę oraz 90. percentyl czasów dostaw.



Analiza 12. Ranking kierowców według czasu dostawy (Analizy transportowe)

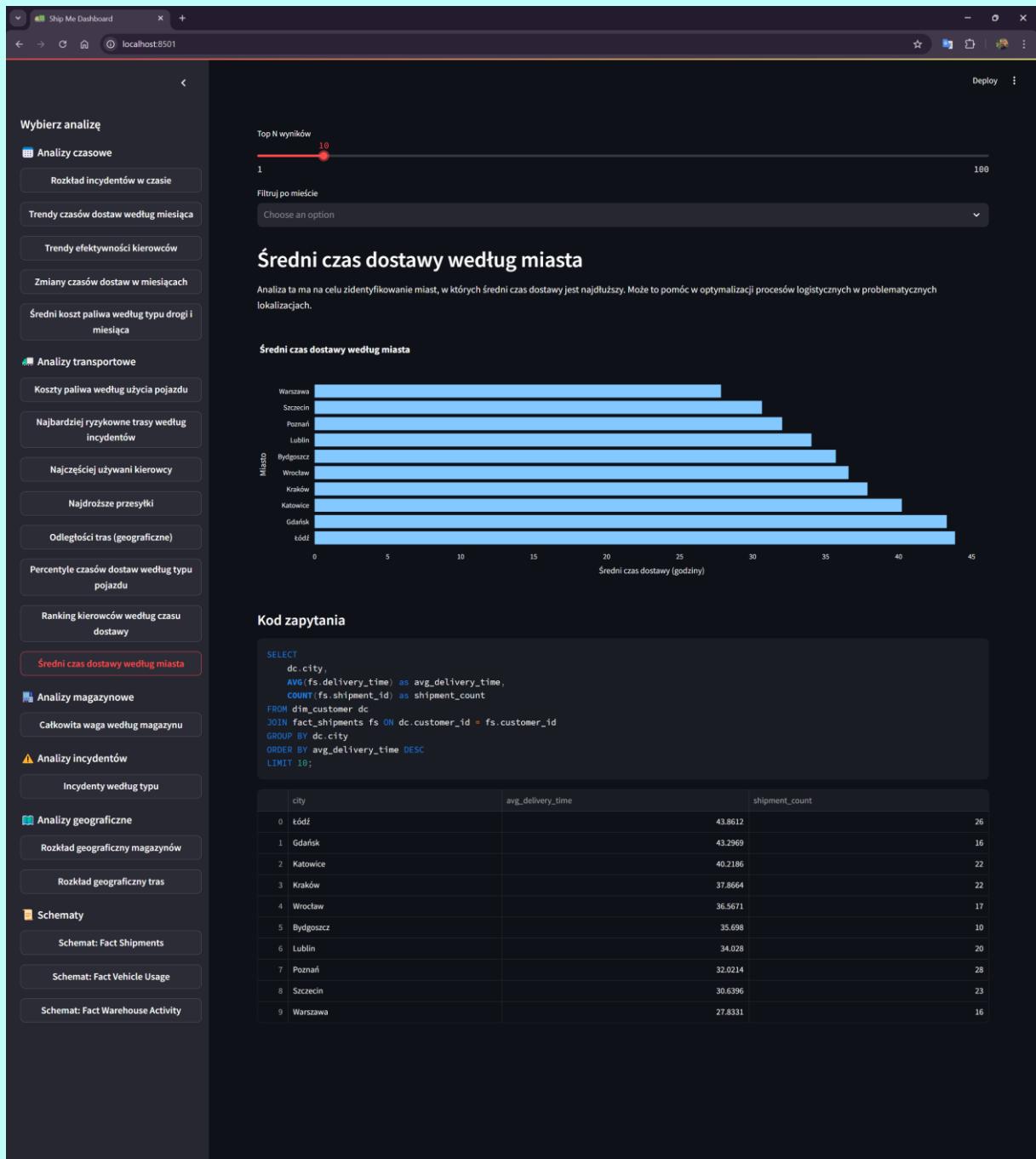
Analiza ta pozwala ocenić kierowców pod kątem średniego czasu dostawy, co może pomóc w identyfikacji najbardziej efektywnych pracowników.



Analiza 13. Średni czas dostawy według miasta (Analizy transportowe)

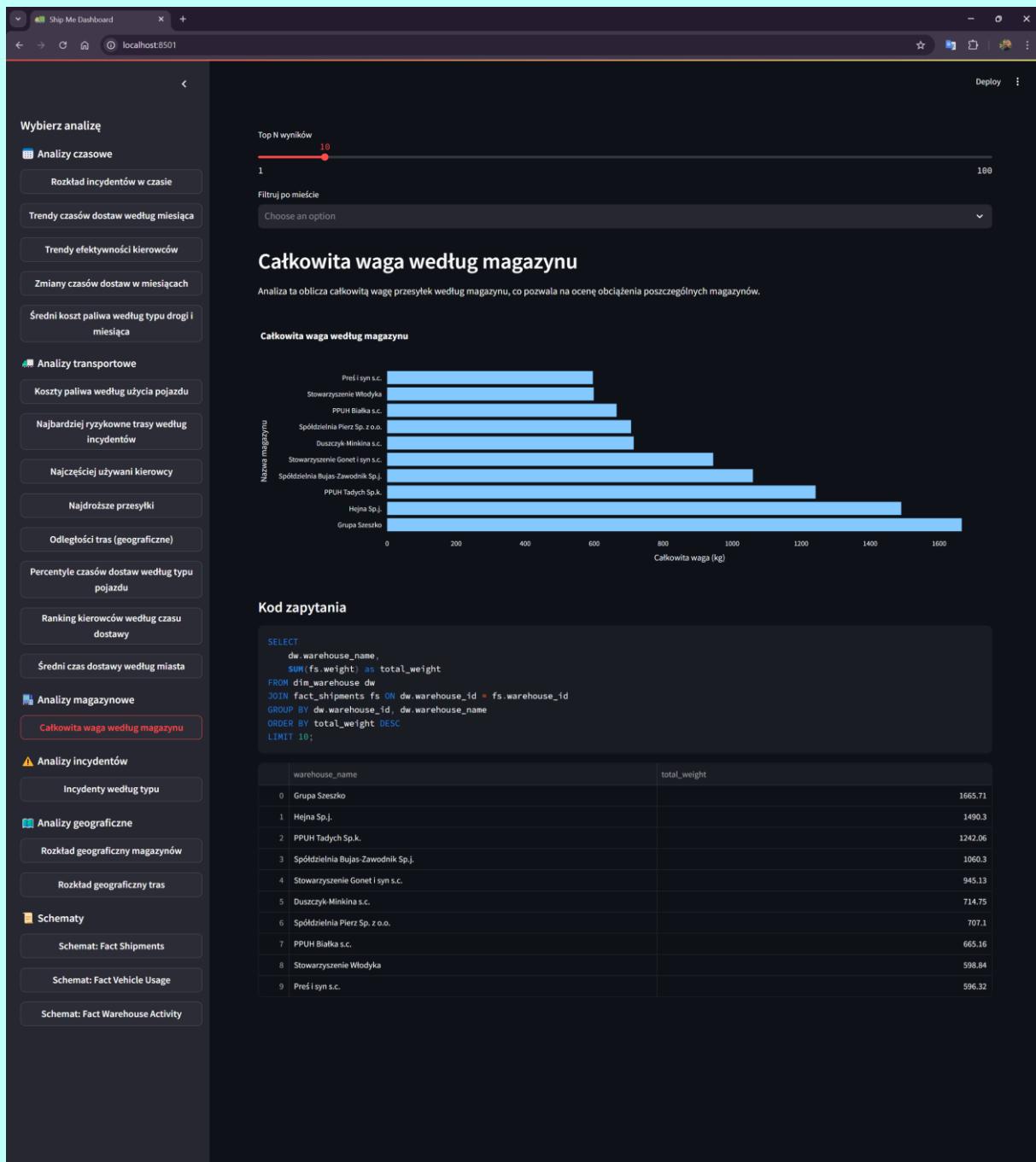
Analiza ta ma na celu zidentyfikowanie miast, w których średni czas dostawy jest najdłuższy.

Może to pomóc w optymalizacji procesów logistycznych w problematycznych lokalizacjach.



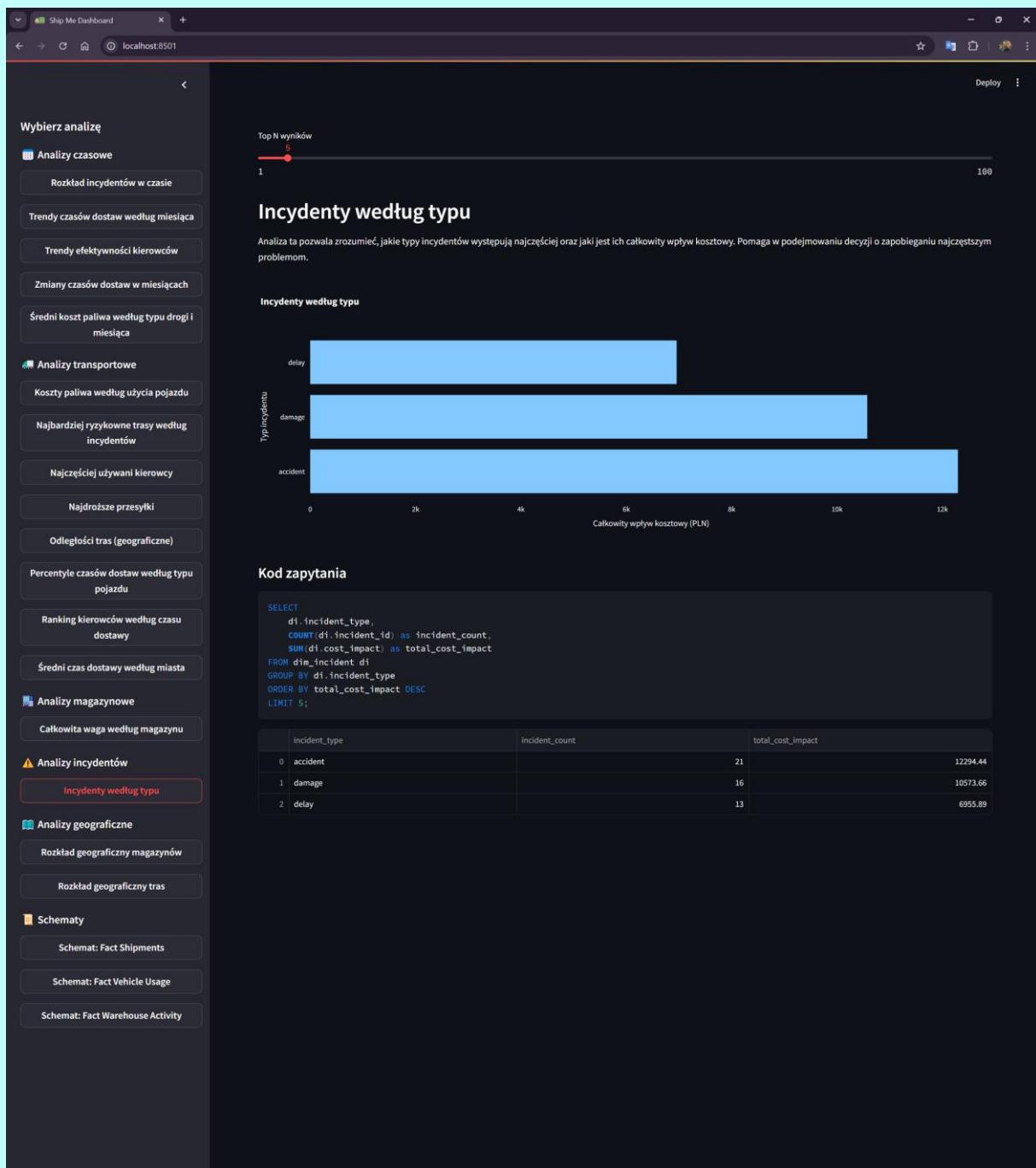
Analiza 14. Całkowita waga według magazynu (Analizy magazynowe)

Analiza ta oblicza całkowitą wagę przesyłek według magazynu, co pozwala na ocenę obciążenia poszczególnych magazynów.



Analiza 15. Incydenty według typu (⚠️ Analizy incydentów)

Analiza ta pozwala zrozumieć, jakie typy incydentów występują najczęściej oraz jaki jest ich całkowity wpływ kosztowy. Pomaga w podejmowaniu decyzji o zapobieganiu najczęstszym problemom.



Analiza 16. Rozkład geograficzny magazynów (Analizy geograficzne)

Analiza ta pokazuje lokalizację geograficzną magazynów na mapie, co pozwala na ocenę ich rozmieszczenia w Polsce.

Wybierz analizę

Analizy czasowe

- Rozkład incydentów w czasie
- Trendy czasów dostaw według miesiąca
- Trendy efektywności kierowców
- Zmiany czasów dostaw w miesiącach
- Średni koszt paliwa według typu drogi i miesiąca

Analizy transportowe

- Koszty paliwa według użycia pojazdu
- Najbardziej ryzykowne trasy według incydentów
- Najczęściej używany kierowcy
- Najdroższe przesyłki
- Odległość tras (geograficzne)
- Percentyle czasów dostaw według typu pojazdu
- Ranking kierowców według czasu dostawy
- Średni czas dostawy według miasta

Analizy magazynowe

- Całkowita waga według magazynu

Analytyczne

- Rozkład geograficzny magazynów**
- Rozkład geograficzny tras

Schematy

- Schemat: Fact Shipments
- Schemat: Fact Vehicle Usage
- Schemat: Fact Warehouse Activity

Filtruj po mieście

Choose an option

Rozkład geograficzny magazynów

Analiza ta pokazuje lokalizację geograficzną magazynów na mapie, co pozwala na ocenę ich rozmieszczenia w Polsce.

Rozkład geograficzny magazynów

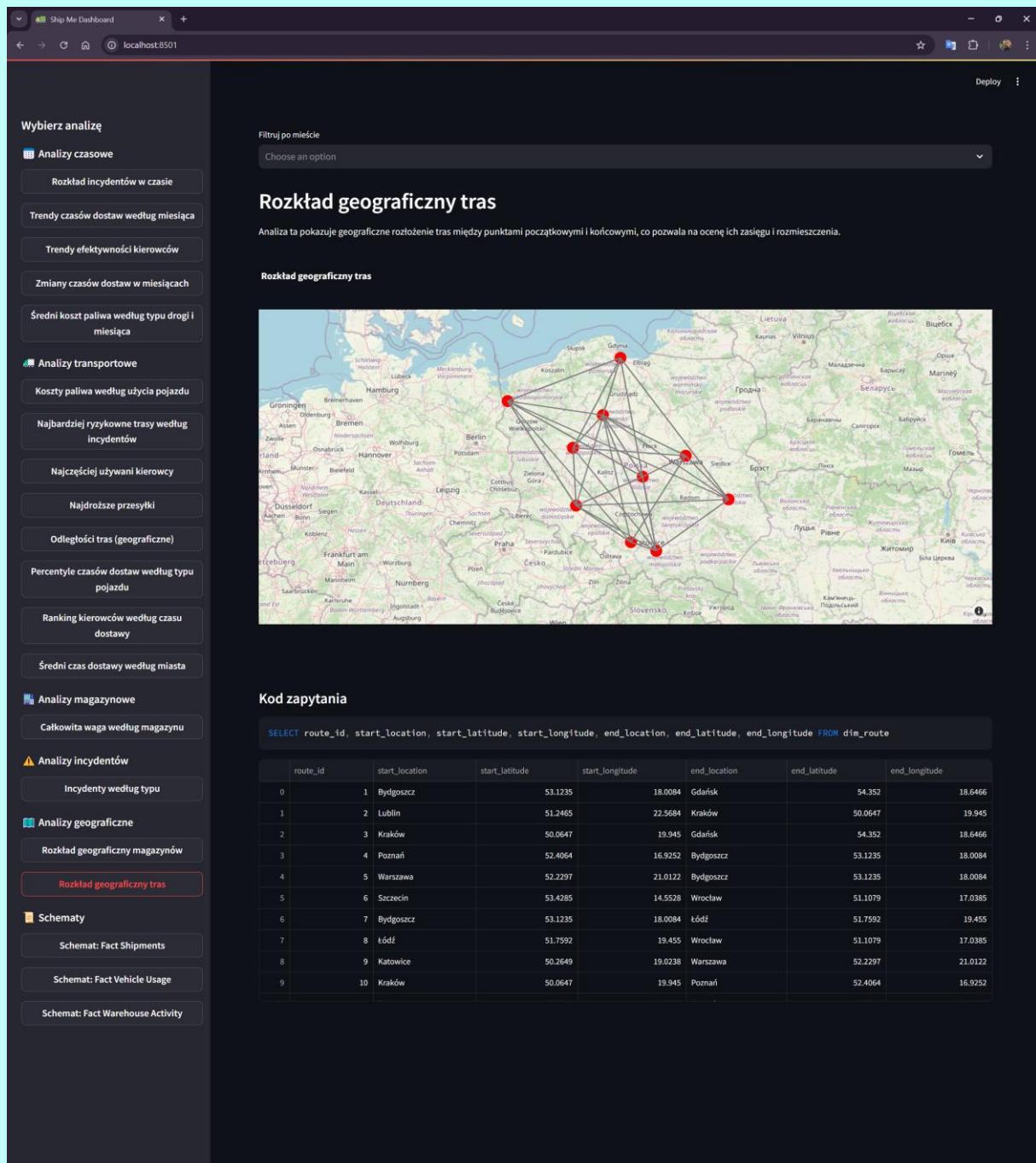
Kod zapytania

```
SELECT warehouse_name, city, latitude, longitude FROM dim_warehouse
```

warehouse_name	city	latitude	longitude
PPUH Tadys Sp.k.	Wrocław	51.1079	17.0385
Stowarzyszenie Włodyka	Wrocław	51.1079	17.0385
Prejsyn s.c.	Gdańsk	54.352	18.6466
Spółdzielnia Pierz Sp. z o.o.	Katowice	50.2649	19.0238
Stowarzyszenie Gonci syn s.c.	Kraków	50.0647	19.945
Hejna Sp.j.	Poznań	52.4064	16.9252
PPUH Biała s.c.	Lublin	51.2465	22.5684
Spółdzielnia Bujas-Zawodnik Sp.j.	Bydgoszcz	53.1235	18.0084
Grupa Szesko	Łódź	51.7592	19.455
Duszczyk-Minkina s.c.	Lublin	51.2465	22.5684

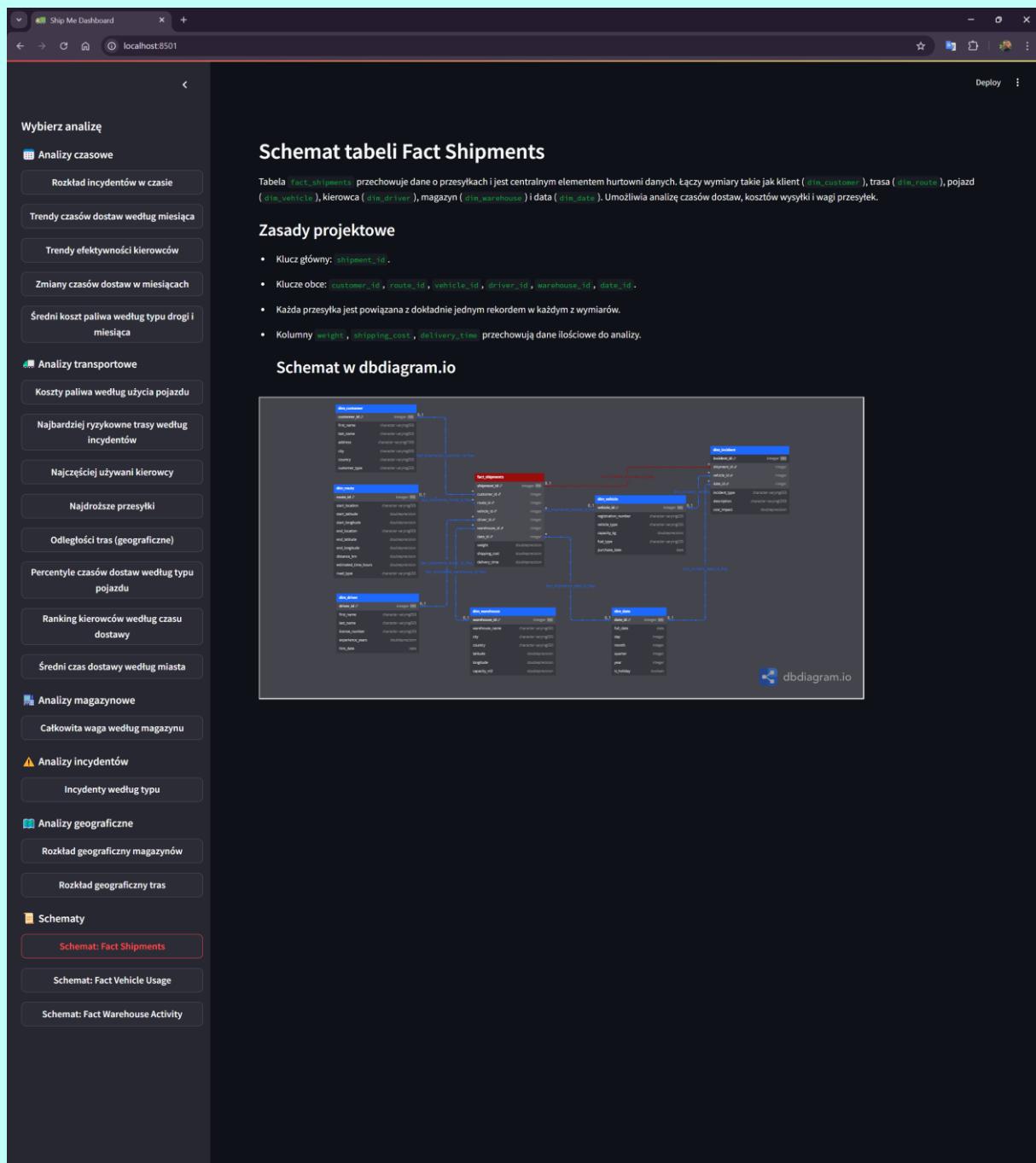
Analiza 17. Rozkład geograficzny tras (Analizy geograficzne)

Analiza ta pokazuje geograficzne rozłożenie tras między punktami początkowymi i końcowymi, co pozwala na ocenę ich zasięgu i rozmieszczenia.



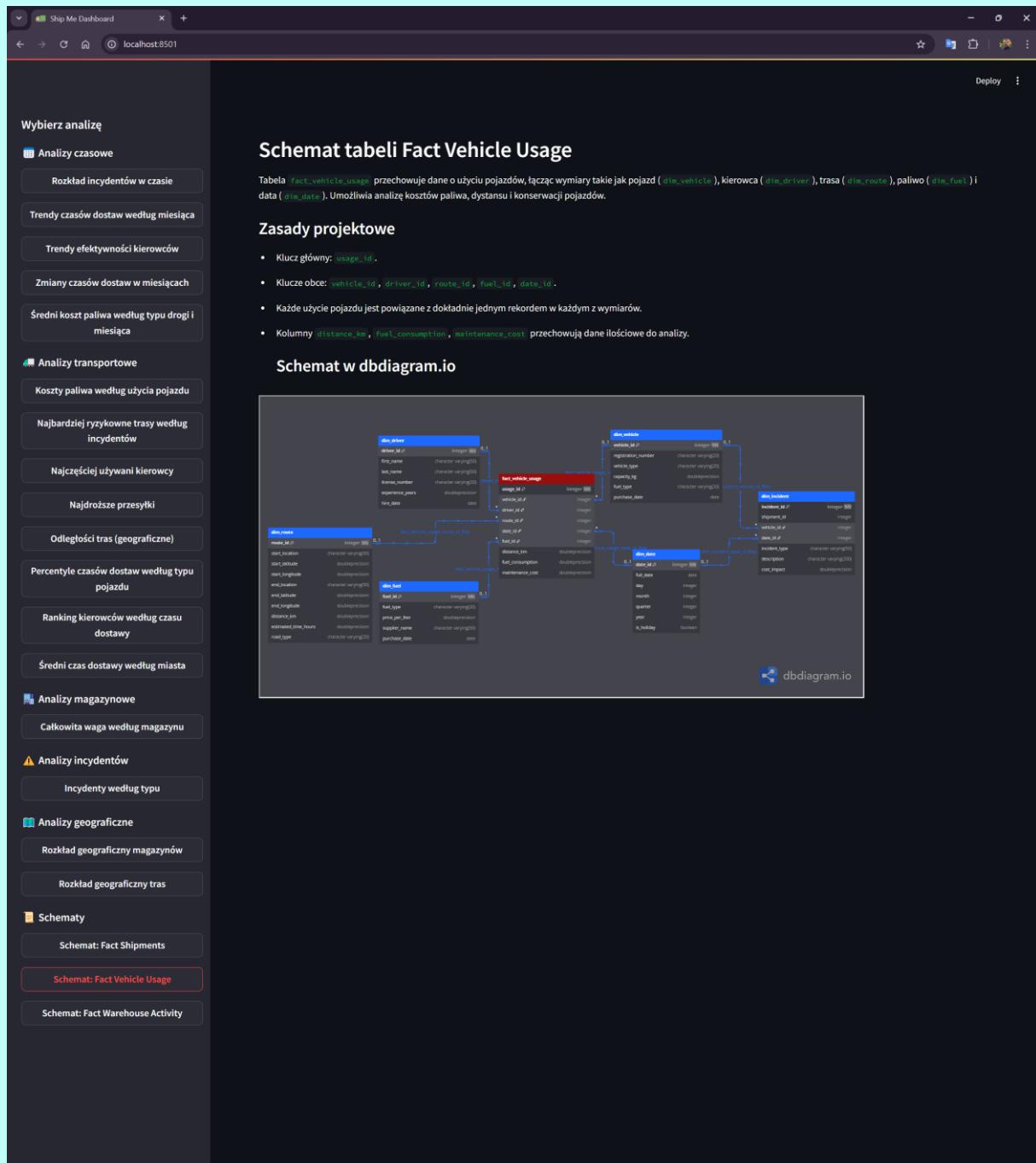
Schemat 1. Schemat tabeli Fact Shipments (Schematy)

Tabela **fact_shipments** przechowuje dane o przesyłkach i jest centralnym elementem hurtowni danych. Łączy wymiary takie jak klient (**dim_customer**), trasa (**dim_route**), pojazd (**dim_vehicle**), kierowca (**dim_driver**), magazyn (**dim_warehouse**) i data (**dim_date**). Umożliwia analizę czasów dostaw, kosztów wysyłki i wagi przesyłek.



Schemat 2. Schemat tabeli Fact Vehicle Usage (Schematy)

Tabela **fact_vehicle_usage** przechowuje dane o użyciu pojazdów, łącząc wymiary takie jak pojazd (**dim_vehicle**), kierowca (**dim_driver**), trasa (**dim_route**), paliwo (**dim_fuel**) i data (**dim_date**). Umożliwia analizę kosztów paliwa, dystansu i konserwacji pojazdów.



Schemat 3. Schemat tabeli Fact Warehouse Activity (☰ Schematy)

Tabela **fact_warehouse_activity** przechowuje dane o aktywnościach magazynowych, łącząc wymiary takie jak produkt (**dim_product**), magazyn (**dim_warehouse**) i data (**dim_date**). Umożliwia analizę ruchów magazynowych i czasów przechowywania.

Schemat tabeli Fact Warehouse Activity

Tabela `fact_warehouse_activity` przechowuje dane o aktywnościach magazynowych, łącząc wymiary takie jak produkt (`dim_product`), magazyn (`dim_warehouse`) i data (`dim_date`). Umożliwia analizę ruchów magazynowych i czasów przechowywania.

Zasady projektowe

- Klucz główny: `activity_id`.
- Klucze obce: `product_id`, `warehouse_id`, `date_id`.
- Każda aktywność jest powiązana z dokładnie jednym rekordem w każdym z wymiarów.
- Kolumny `stock_in`, `stock_out`, `storage_time` przechowują dane ilościowe do analizy.

Schemat w dbdiagram.io

```

    graph TD
        subgraph fact_warehouse_activity [Fact Warehouse Activity]
            direction LR
            FA((fact_warehouse_activity)) --- P1[dim_product]
            FA --- W1[dim_warehouse]
            FA --- D1[dim_date]
            FA --- I1[dim_incident]
            FA --- V1[dim_vehicle]
            P1 --> FA
            W1 --> FA
            D1 --> FA
            I1 --> FA
            V1 -.-> FA
        end
        subgraph dim_product [dim_product]
            direction TB
            product_id[product_id] --- product_name[product_name]
            product_name --- category[category]
            category --- weight_kg[weight_kg]
            weight_kg --- fragility[fragility]
        end
        subgraph dim_warehouse [dim_warehouse]
            direction TB
            warehouse_id[warehouse_id] --- warehouse_name[warehouse_name]
            warehouse_name --- city[city]
            city --- country[country]
            country --- latitude[latitude]
            latitude --- longitude[longitude]
            longitude --- capacity_m3[capacity_m3]
        end
        subgraph dim_date [dim_date]
            direction TB
            date_id[date_id] --- full_date[full_date]
            full_date --- day[day]
            day --- month[month]
            month --- quarter[quarter]
            quarter --- year[year]
            year --- is_holiday[is_holiday]
        end
        subgraph dim_incident [dim_incident]
            direction TB
            incident_id[incident_id] --- incident_type[incident_type]
            incident_type --- description[description]
            description --- cost_impact[cost_impact]
        end
        subgraph dim_vehicle [dim_vehicle]
            direction TB
            vehicle_id[vehicle_id] --- registration_number[registration_number]
            registration_number --- vehicle_type[vehicle_type]
            vehicle_type --- capacity_kg[capacity_kg]
            capacity_kg --- fuel_type[fuel_type]
            fuel_type --- purchase_date[purchase_date]
        end
    
```

Wnioski z pracy

Skuteczna integracja i analiza danych logistycznych

Projekt *ShipMe Data Warehouse* z powodzeniem zrealizował ideę hurtowni danych, integrując rozproszone dane logistyczne (przesyłki, pojazdy, trasy, magazyny, incydenty) w spójny model gwiaździsty.

Użycie tabel faktów (`fact_shipments`, `fact_vehicle_usage`, `fact_warehouse_activity`) oraz wymiarów (`dim_route`, `dim_vehicle`, itp.) umożliwiło efektywną analizę kluczowych aspektów działalności logistycznej, takich jak czasy dostaw, koszty paliwa czy obciążenie magazynów. Struktura hurtowni danych pozwoliła na szybkie generowanie raportów i wizualizacji w aplikacji Streamlit, co potwierdza jej użyteczność w podejmowaniu decyzji biznesowych.

Elastyczność i skalowalność projektu

Architektura oparta na SQLAlchemy i Alembic zapewniała elastyczność w zarządzaniu schematem bazy danych oraz łatwą migrację danych. Modułowa budowa kodu (np. oddzielne pliki w `src/pages/` i `src/analysis/`) umożliwia rozbudowę projektu o nowe analizy i funkcjonalności, takie jak predykcje ML. Hurtownia danych jest skalowalna – dodanie nowych tabel faktów czy wymiarów (np. dla analizy klientów lub pogody) wymaga jedynie aktualizacji schematu i zapytań, co czyni ją gotową na przyszłe potrzeby.

Wizualizacja danych jako kluczowy atut

Integracja z biblioteką Streamlit i Plotly pozwoliła na stworzenie interaktywnego interfejsu użytkownika, który prezentuje dane w przystępny sposób (wykresy słupkowe, liniowe, mapy). To pokazuje, że hurtownia danych nie tylko przechowuje informacje, ale również efektywnie je komunikuje użytkownikom końcowym.

Potencjał rozwoju hurtowni danych

Projekt ma duży potencjał do dalszego rozwoju. Możliwe kierunki obejmują:

- **rozbudowę modelu danych** - dodanie nowych wymiarów (np. pogoda, typy klientów) i faktów (np. koszty konserwacji pojazdów),
- **uczenie maszynowe** - wprowadzenie podstawowych modeli uczenia maszynowego (ML) oraz złożonych modeli (np. sieci neuronowe),
- **automatyzację** - implementacja ETL (Extract, Transform, Load) do regularnego aktualizowania danych z zewnętrznych źródeł,
- **interaktywność** - rozszerzenie aplikacji Streamlit o dashboardy z filtrami dynamicznymi i alertami w czasie rzeczywistym.

Podsumowanie

ShipMe Data Warehouse to solidna baza dla analizy procesów logistycznych, łącząca przechowywanie danych, ich wizualizację i predykcję. Choć napotkano trudności z niezbalansowanymi danymi w uczeniu maszynowym, projekt spełnia swoje założenia jako hurtownia danych, oferując solidne fundamenty do dalszej optymalizacji i rozbudowy. Dalsze prace powinny skupić się na wzbogaceniu danych i doskonaleniu modeli predykcyjnych, aby w pełni wykorzystać potencjał hurtowni w wsparciu decyzji operacyjnych.

Literatura

1. Agnieszka Chodkowska-Gyurics, *Hurtownie danych*, Wydawnictwo Naukowe PWN 2020.
2. Adam Pelikant, *Hurtownie danych. Od przetwarzania analitycznego do raportowania*. Wydanie II, Helion 2021.
3. Zdzisław Dybikowski, *PostgreSQL. Wydanie II*, Helion, 2012
4. Luca Ferrari, Enrico Pirozzi, *Learn PostgreSQL - Second Edition*, Packt Publishing, 2023
5. Jake VanderPlas, *Python Data Science. Niezbędne narzędzia do pracy z danymi. Wydanie II*, Helion, 2023
6. Tyler Richards, *Streamlit for Data Science - Second Edition*, Packt Publishing, 2023