Sprawozdanie z Projektu:
Modern Data Lakehouse Analytics

Temat: Budowa odpornego potoku ETL oraz analiza segmentacji klientów (RFM) przy użyciu Azure i Databricks.
Technologie: Azure Data Lake Storage Gen2, Azure Databricks, PySpark, Unity Catalog, Delta Lake.

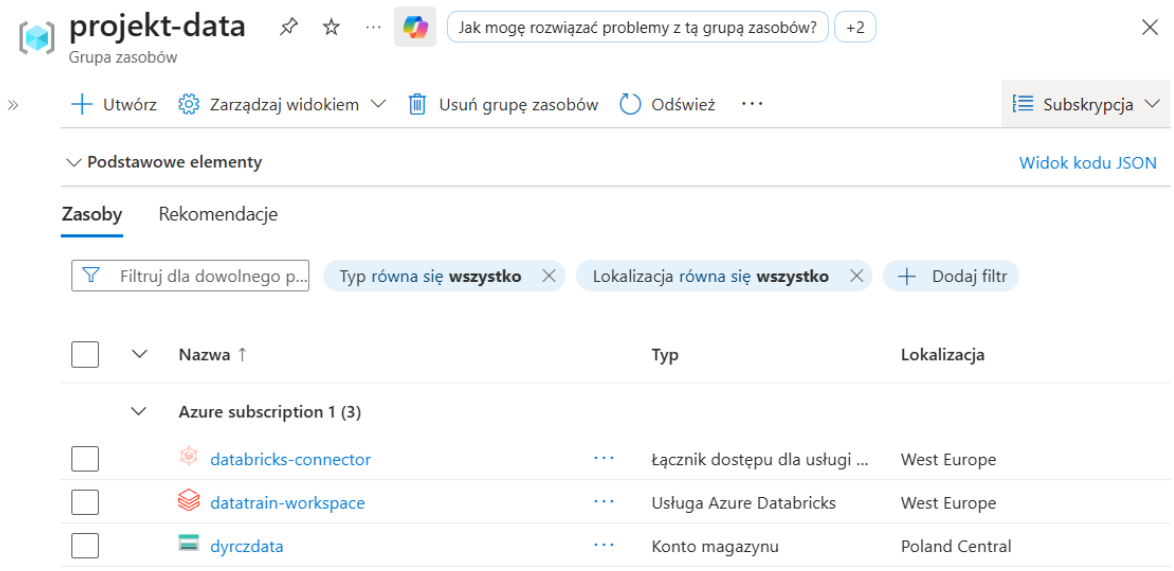
Wykonał: Przemysław Dyrcz

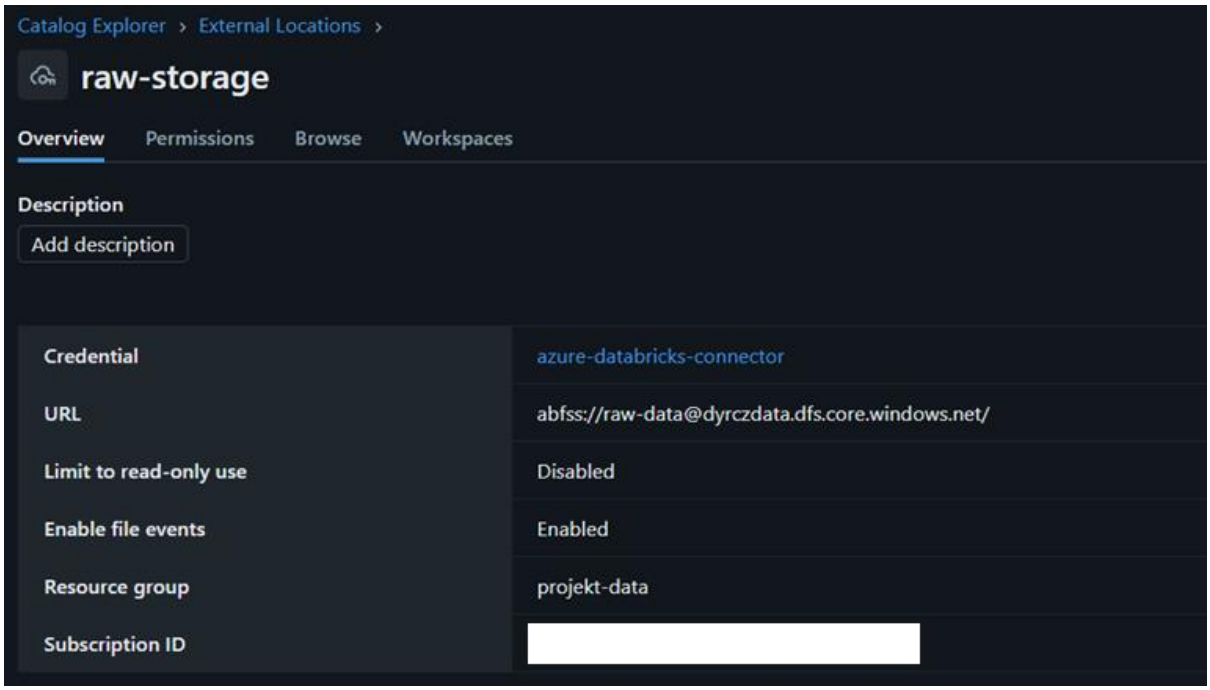
1. Wstęp i Architektura Rozwiązania

Celem projektu było stworzenie nowoczesnego środowiska analitycznego typu **Data Lakehouse**, integrującego chmurowe magazyny danych z zaawansowaną analityką Spark.

Kluczowe komponenty architektury:

- **Azure Data Lake Storage Gen2 (ADLS):** Magazyn danych surowych (Raw Data) w formacie CSV.
- **Unity Catalog:** System zarządzania danymi i uprawnieniami (Governance).
- **Azure Access Connector:** Zastosowanie tożsamości zarządzanej (Managed Identity) do bezpiecznego łączenia usług bez używania haseł i kluczy w kodzie.
- **Databricks Runtime:** Środowisko obliczeniowe do przetwarzania danych w skali Big Data.





2. Inżynieria Danych (ETL)

Krok 1: Bezpieczna Ingestia Danych

Dane zostały wczytane z kontenera Azure przy użyciu protokołu abfss. Wykorzystano mechanizm **External Location**, co pozwoliło na bezpieczny dostęp do plików.

- **Wyzwanie:** Rozróżnianie wielkości liter w nazwach plików (system Linuxowy w Databricks).
- **Rozwiązanie:** Standaryzacja ścieżek dostępu.

Krok 2: Czyszczenie, Standaryzacja (Silver Layer) i wstępna analiza

Proces czyszczenia objął najbardziej krytyczne aspekty jakości danych:

- **Obsługa błędów parsowania (Cast):** Zastosowano mechanizm try_cast (lub rzutowanie z obsługą błędów), aby uniknąć przerywania procesu przez błędne znaki w kolumnach numerycznych (np. tekst w polu ceny).
- **Walidacja dat:** Rozwiązano problem mieszanych formatów dat (M/d/yyyy oraz yyyy-MM-dd) przy użyciu funkcji coalesce i to_date, co pozwoliło na 100% poprawność osi czasu.
- **Standaryzacja:** Ujednolicono nazwy kolumn (małe litery, brak spacji) dla zapewnienia kompatybilności z SQL.

```
from pyspark.sql.functions import *

df_raw = spark.read.format("csv") \
    .option("header", "true") \
    .option("quote", "\"") \
    .option("escape", "\"") \
    .load("abfss://raw-data@dyrczdata.dfs.core.windows.net/Superstore.csv")

df_silver = df_raw.select(
    *[col(c).alias(c.replace(' ', '_').lower()) for c in df_raw.columns]
).withColumn(
    "order_date_clean",
    coalesce(
        to_date(col("order_date"), "yyyy-MM-dd"),
        to_date(col("order_date"), "M/d/yyyy")
    )
).withColumn(
    "sales", regexp_replace(col("sales"), "[^0-9.]", "").cast("double")
).withColumn(
    "profit", regexp_replace(col("profit"), "[^0-9.-]", "").cast("double")
)

df_silver = df_silver.filter(col("order_date_clean").isNotNull() & col("sales").isNotNull())

print(f"Dane oczyszczone. Pozostało rekordów: {df_silver.count()}")
display(df_silver.limit(5))
```

```
Dane oczyszczone. Pozostało rekordów: 9994
```

Table +

	row_id	order_id	order_date	ship_date	ship_mode	customer_id	customer_name	segment
1	1	CA-2017-152156	2017-11-08	2017-11-11	Second Class	CG-12520	Claire Gute	Consumer
2	2	CA-2017-152156	2017-11-08	2017-11-11	Second Class	CG-12520	Claire Gute	Consumer
3	3	CA-2017-138688	2017-06-12	2017-06-16	Second Class	DV-13045	Darrin Van Huff	Corporate
4	4	US-2016-108966	2016-10-11	2016-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer
5	5	US-2016-108966	2016-10-11	2016-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer

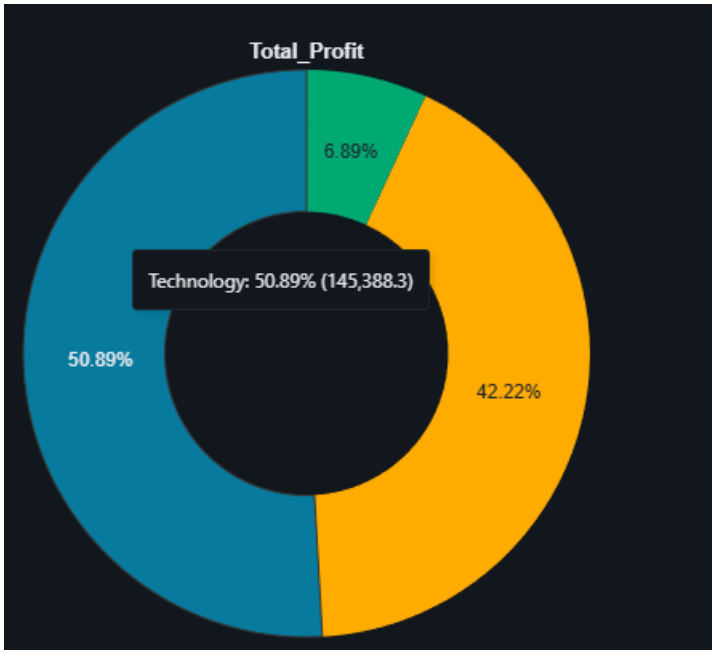
Wstępna analiza obejmowała raport zysków według kategorii.

```python
from pyspark.sql.functions import sum, round, col

df_report = df.groupBy("Category") \
    .agg(round(sum("Profit"), 2).alias("Total_Profit")) \
    .orderBy(col("Total_Profit").desc())

print("Raport zysków według kategorii:")
display(df_report)
```

Raport zysków według kategorii:

Table +

	Category	1.2 Total_
1	Technology	
2	Office Supplies	
3	Furniture	



3. Analiza Zaawansowana (Data Science)

Analiza Segmentacji RFM

Zaimplementowano algorytm segmentacji klientów oparty na trzech wymiarach:

1. **Recency (Aktualność):** Liczba dni od ostatniego zamówienia.
2. **Frequency (Częstotliwość):** Całkowita liczba zamówień klienta.
3. **Monetary (Wartość):** Sumaryczny przychód generowany przez klienta.

```python
from pyspark.sql import Window
from pyspark.sql.functions import *

current_date = df_silver.select(max("order_date_clean")).collect()[0][0]

rfm_base = df_silver.groupBy("customer_id", "customer_name").agg(
    datediff(lit(current_date), max("order_date_clean")).alias("recency"),
    countDistinct("order_id").alias("frequency"),
    round(sum("sales"), 2).alias("monetary")
)


def get_rfm_scores(df):
    w_r = Window.orderBy(col("recency").desc())
    w_f = Window.orderBy(col("frequency").asc())
    w_m = Window.orderBy(col("monetary").asc())

    return df.withColumn("r_score", ntile(4).over(w_r)) \
             .withColumn("f_score", ntile(4).over(w_f)) \
             .withColumn("m_score", ntile(4).over(w_m))

rfm_scored = get_rfm_scores(rfm_base)

rfm_final = rfm_scored.withColumn("total_rfm", col("r_score") + col("f_score") + col("m_score")) \
    .withColumn("segment",
        when(col("total_rfm") >= 11, "Mistrzowie (Champions)")
        .when(col("total_rfm") >= 9, "Lojalni Klienci")
        .when(col("total_rfm") >= 7, "Obiecujący")
        .when(col("total_rfm") >= 5, "Wymagający uwagi")
        .otherwise("Klienci traceni / Ryzykowni")
    )

print("Segmentacja RFM zakończona sukcesem!")
display(rfm_final.orderBy(col("total_rfm").desc()))
```

Table +

	customer_id	customer_name	recency	frequency	monetary	r_score	f_score	m_score	total_rfm
1	MM-17920	Michael Moore	7	11	3794.08	4	4	4	12
2	AT-10735	Annie Thurman	13	10	3831.86	4	4	4	12
3	CC-12220	Chris Cortes	20	12	3913.42	4	4	4	12
4	FP-14320	Frank Preis	23	8	4046.75	4	4	4	12
5	DM-13345	Denise Monton	22	8	4074.47	4	4	4	12
6	TC-21295	Toby Carlisle	27	8	4266.81	4	4	4	12
7	ND-18370	Natalie DeCherney	27	9	4326.14	4	4	4	12
8	GZ-14470	Gary Zandusky	7	9	4355.15	4	4	4	12
9	AI-10855	Arianne Irving	13	10	4375.79	4	4	4	12
10	TP-21130	Theone Pippenger	8	9	4454.06	4	4	4	12
11	ES-14080	Erin Smith	28	9	4657.92	4	4	4	12
12	RB-19465	Rick Bensley	9	12	4715.47	4	4	4	12
13	RA-19915	Russell Applegate	12	9	4793.54	4	4	4	12
14	DK-13225	Dean Katz	10	9	4802.39	4	4	4	12

Klienci zostali podzieleni na grupy (Mistrzowie, Lojalni, Ryzykowni) przy użyciu metod statystycznych (kwartyli) i funkcji okna PySpark.

Detekcja Anomalii

Wdrożono skrypty monitorujące jakość biznesową danych:

- **Anomalie logistyczne:** Wykrycie rekordów, w których data wysyłki poprzedzała datę zamówienia.

```
df_shipping = df_silver.withColumn(
    "ship_date_clean", coalesce(to_date(col("ship_date"), "yyyy-MM-dd"), to_date(col("ship_date"), "M/d/yyyy"))
).withColumn(
    "days_to_ship", datediff(col("ship_date_clean"), col("order_date_clean"))
)

anomalies_shipping = df_shipping.filter((col("days_to_ship") < 0) | (col("days_to_ship") > 15))

print("🕵️ Wykryto anomalie w datach wysyłki:")
display(anomalies_shipping.select("order_id", "order_date", "ship_date", "days_to_ship"))
```

```
Wykryto anomalie w datach wysyłki:
```

Table +

order_id	order_date	ship_date	days_to_ship

No rows returned

- **Anomalie statystyczne:** Identyfikacja transakcji o wartości przekraczającej 3 odchylenia standardowe (metoda Z-score).

```
stats = df_silver.select(mean("sales").alias("avg"), stddev("sales").alias("std")).collect()
avg_val, std_val = stats[0]["avg"], stats[0]["std"]

df_outliers = df_silver.filter(col("sales") > (avg_val + 3 * std_val))

print(f"Znaleziono {df_outliers.count()} transakcji o ekstremalnie wysokiej wartości:")
display(df_outliers.orderBy(col("sales").desc()))
```

```
Znaleziono 127 transakcji o ekstremalnie wysokiej wartości:
```

Table +

	row_id	order_id	order_date	ship_date	ship_mode	customer_id	customer_name
1	2698	CA-2015-145317	2015-03-18	2015-03-23	Standard Class	SM-20320	Sean Miller
2	6827	CA-2017-118689	2017-10-02	2017-10-09	Standard Class	TC-20980	Tamara Chand
3	8154	CA-2018-140151	2018-03-23	2018-03-25	First Class	RB-19360	Raymond Buch
4	2624	CA-2018-127180	2018-10-22	2018-10-24	First Class	TA-21385	Tom Ashbrook
5	4191	CA-2018-166709	2018-11-17	2018-11-22	Standard Class	HL-15040	Hunter Lopez
6	9040	CA-2017-117121	2017-12-17	2017-12-21	Standard Class	AB-10105	Adrian Barton
7	4099	CA-2015-116904	2015-09-23	2015-09-28	Standard Class	SC-20095	Sanjit Chand
8	4278	US-2017-107440	2017-04-16	2017-04-20	Standard Class	BS-11365	Bill Shonely
9	8489	CA-2017-158841	2017-02-02	2017-02-04	Second Class	SE-20110	Sanjit Engle

4. Wnioski Biznesowe (Insights)

Najważniejszym osiągnięciem analitycznym było zidentyfikowanie **"Toksycznych VIP-ów"**.

- **Definicja:** Klienci z segmentu "Mistrzowie" (wysoki obrót), którzy generują ujemny zysk dla firmy.

- **Przyczyna:** Analiza wykazała nadmierne stosowanie rabatów (średnio powyżej 40-50%) oraz wysokie koszty operacyjne w tej grupie.
- **Rekomendacja:** Wprowadzenie "Capu" (limitu) na rabaty dla wybranych grup produktowych w tym segmencie.

```
toxic_vips = rfm_final.join(···
)

toxic_vips_analysis = toxic_vips.filter(
    (col("segment").isin("Mistrzowie (Champions)", "Lojalni
    Klienci")) &
    (col("total_actual_profit") < 0)
).select(
    "customer_name",
    "segment",
    "monetary",
    "total_actual_profit",
    "avg_applied_discount",
    "total_orders_count"
).orderBy("total_actual_profit")

print("RAPORT: Klienci VIP generujący straty (Toksyczni
VIP-owie):")
display(toxic_vips_analysis)
```

Table | Visualization 1 | +

	customer_name	segment	monetary	total_actual_profit	avg_applied_discount	total_orders_count
1	Cindy Stewart	Lojalni Klienci	5690.05	-6626.39	0.2	9
2	Luke Foster	Mistrzowie (Champio...	3930.51	-3583.98	0.32	16
3	Sharelle Roach	Lojalni Klienci	3233.48	-3333.91	0.37	9
4	Henry Goldwyn	Mistrzowie (Champio...	3247.64	-2797.96	0.17	17
5	Sean Braxton	Lojalni Klienci	8057.89	-2082.75	0.24	17
6	Christine Phan	Lojalni Klienci	5888.28	-1850.3	0.21	15
7	Natalie Fritzler	Lojalni Klienci	8322.83	-1695.97	0.25	14
8	Tracy Blumstein	Lojalni Klienci	4737.49	-1603.05	0.27	20
9	Dan Campbell	Lojalni Klienci	3336.17	-1441.63	0.21	18
10	David Bremer	Lojalni Klienci	2973.09	-1421.77	0.14	14
11	Rose O'Brian	Lojalni Klienci	3815.48	-1262.57	0.29	12
12	Zuschuss Carroll	Mistrzowie (Champio...	8025.71	-1032.15	0.25	31
13	Victoria Pisteka	Lojalni Klienci	3360.53	-1018.78	0.17	14
14	Olvera Toch	Lojalni Klienci	3818.62	-925.12	0.16	10

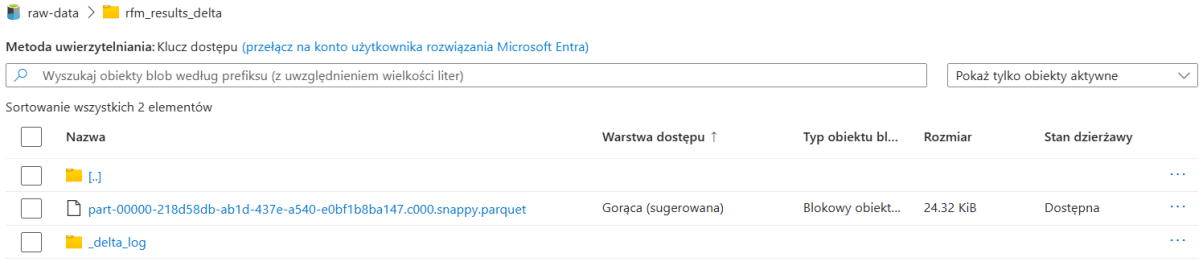
5. Trwałość Danych i Optymalizacja

Finalne wyniki zostały zapisane w formacie **Delta Lake**. Zapewnia to:

- **ACID Transactions:** Gwarancja spójności danych.
- **Time Travel:** Możliwość powrotu do poprzednich wersji danych.
- **Performance:** Szybsze odpytywanie danych przez narzędzia BI (np. Power BI).

```
output_path = "abfss://raw-data@dyrczdata.dfs.core.windows.net/rfm_results_delta"

rfm_final.write.format("delta").mode("overwrite").save(output_path)

print(f"Sukces! Wyniki zapisane pod ścieżką: {output_path}")
```

```
Sukces! Wyniki zapisane pod ścieżką: abfss://raw-data@dyrczdata.dfs.core.windows.net/rfm_results_delta
```

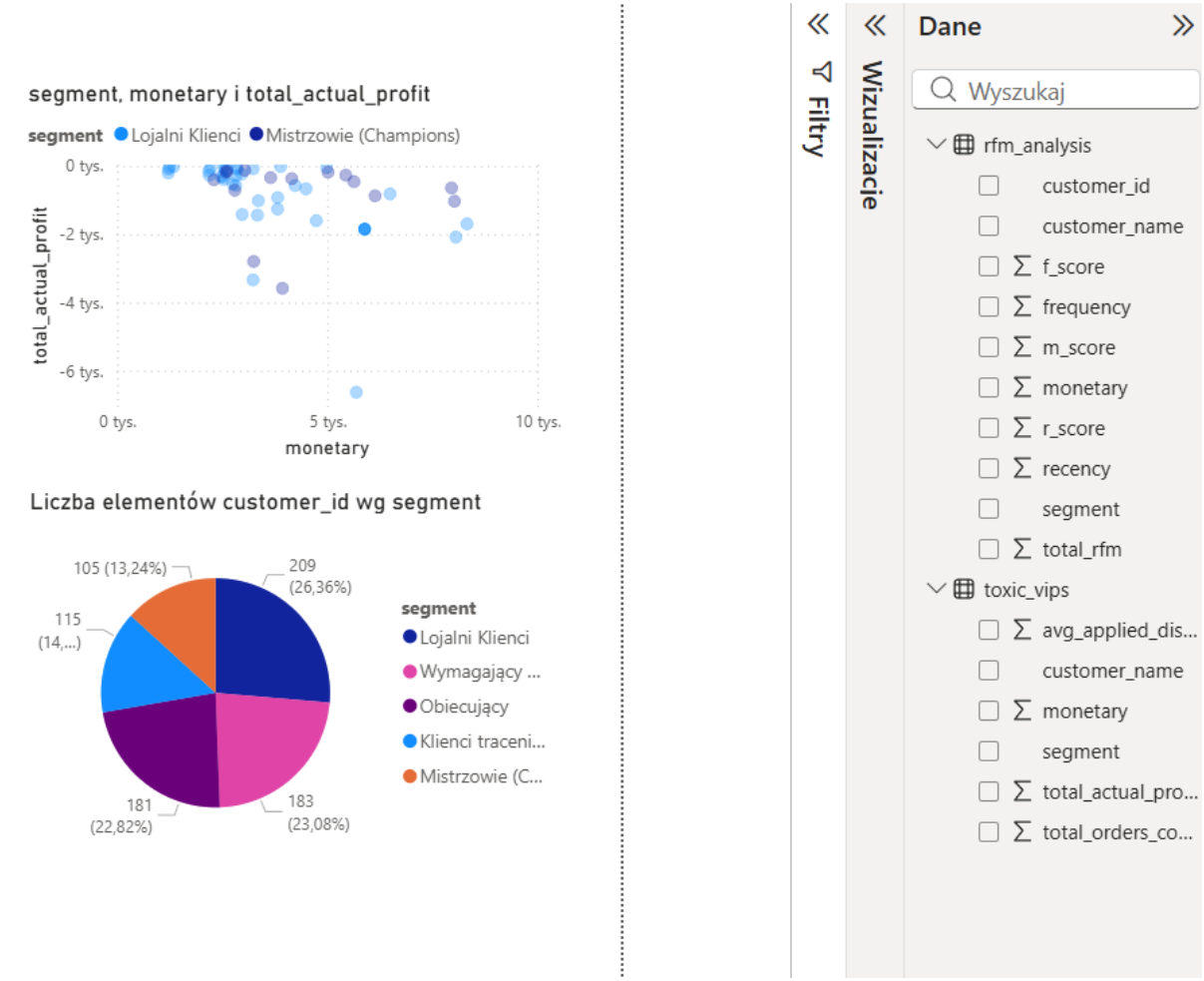


6. Integracja z Business Intelligence (Power BI)

W celu umożliwienia użytkownikom biznesowym interaktywnej analizy danych, wdrożono warstwę semantyczną w postaci tabel zewnętrznych Delta Lake, a następnie podłączono je do narzędzia **Microsoft Power BI**.

- **Model Danych:** Tabele rfm_analysis oraz toxic_vips zostały udostępnione dla warstwy raportowej.
- **Wizualizacja:** Stworzono dashboard menedżerski prezentujący strukturę segmentów klientów oraz macierz rentowności. Dzięki natywnej integracji Databricks z Power BI, dane są dostępne bez konieczności ich fizycznego kopiowania (DirectQuery/Import).

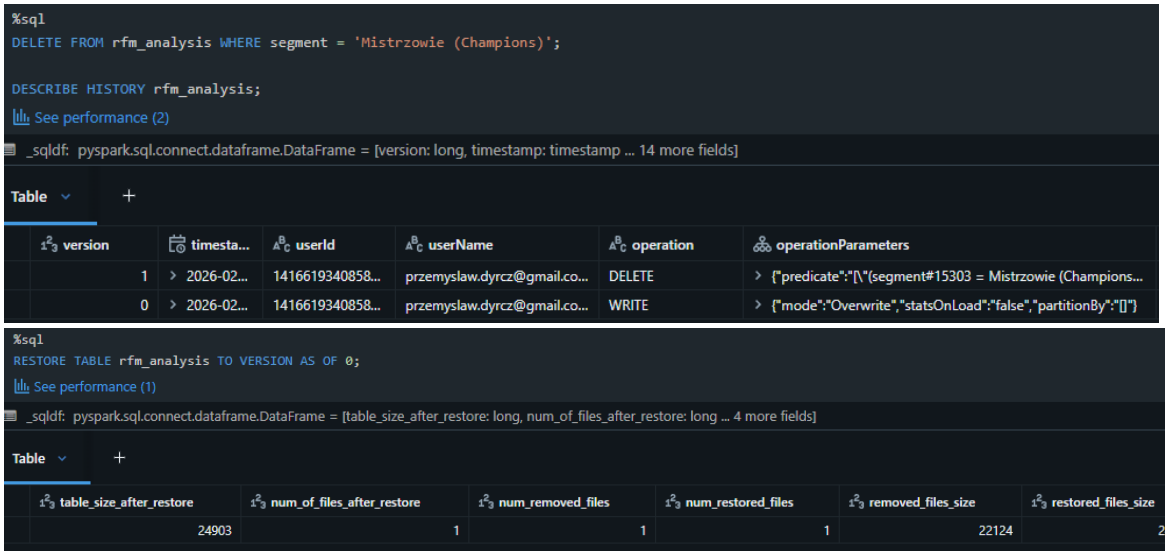
```
%sql
CREATE TABLE IF NOT EXISTS rfm_analysis
USING DELTA
LOCATION 'abfss://raw-data@dyrczdata.dfs.core.windows.net/rfm_results_delta';

CREATE TABLE IF NOT EXISTS toxic_vips
USING DELTA
LOCATION 'abfss://raw-data@dyrczdata.dfs.core.windows.net/toxic_vips_report';
```

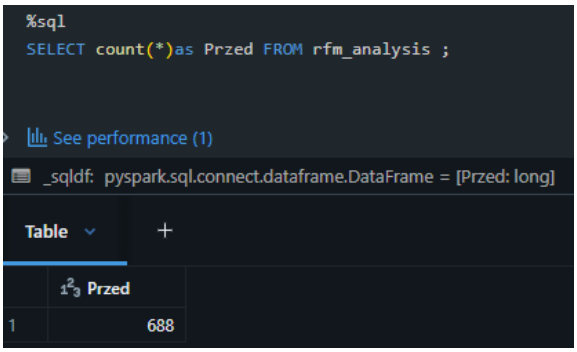


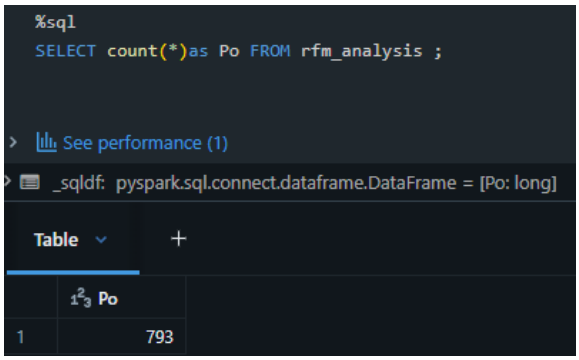
7. Zarządzanie Cyklem Życia Danych (Data Ops & Disaster Recovery)

Jako element strategii utrzymania danych, wykorzystano zaawansowane funkcje formatu **Delta Lake** do zapewnienia bezpieczeństwa i wydajności:

1. **Disaster Recovery (Time Travel):** Przeprowadzono symulację awarii krytycznej (przypadkowe usunięcie segmentu "Mistrzowie"). Wykorzystując dziennik transakcji Delta Log, przywrócono tabelę do stanu sprzed awarii w czasie poniżej 1 minuty za pomocą komendy RESTORE.







8. Podsumowanie Techniczne

Projekt zakończył się sukcesem, dostarczając w pełni funkcjonalną platformę analityczną typu **Lakehouse**. Osiągnięto wszystkie założone cele techniczne i biznesowe:

1. **Transformacja Danych:** Surowe, zanieczyszczone pliki sprzedażowe zostały przekształcone w "Złoty Standard" danych (Gold Layer), gotowy do raportowania. Rozwiązano krytyczne problemy jakości danych, takie jak niespójne formaty dat czy błędy typów.
2. **Wartość Biznesowa (ROI):** Zidentyfikowano kluczowy problem rentowności w segmencie "Mistrzowie" (tzw. Toksyczni VIP-owie). Dzięki integracji z **Power BI**, dział biznesowy uzyskał stały wgląd w te metryki bez konieczności angażowania IT.
3. **Bezpieczeństwo i Niezawodność:** Wdrożenie formatu **Delta Lake** zapewniło odporność systemu na awarie. Przeprowadzone testy Disaster Recovery (z użyciem Time Travel) udowodniły, że system jest w stanie odzyskać krytyczne dane w czasie poniżej 2 minut.

4. **Skalowalność:** Architektura oparta na Azure Data Lake Gen2 i Databricks jest gotowa na przetwarzanie wolumenów Big Data, a zastosowane optymalizacje (Z-Order) gwarantują wydajność zapytań przy rosnącej bazie danych.