

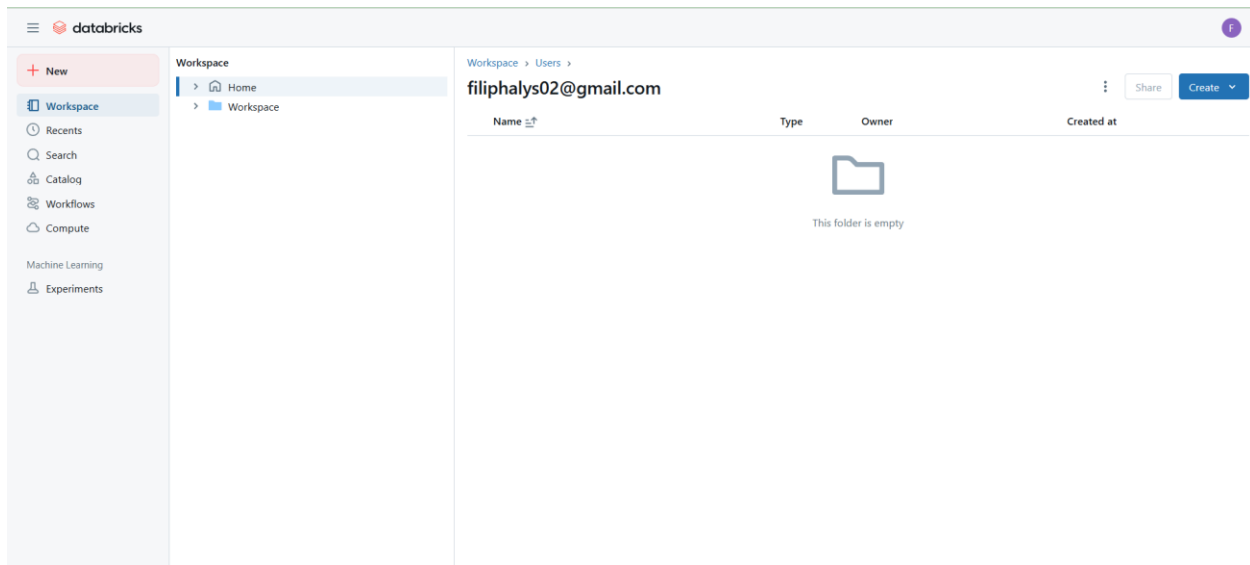
## Ćwiczenie 8

Celem biznesowym tego ćwiczenia jest Analiza, wizualizacja oraz wymodelowanie danych transakcyjnych klientów z wykorzystaniem technologii DataBricks. Ćwiczenie podzielono na kilka etapów:

1. Rejestracja i logowanie do Databricks
2. Wgranie danych (z CSV)
3. Załadowanie danych w notebooku (PySpark)
4. Oczyszczenie, przetworzenie danych (preprocessing)
5. Tworzenie tabel Delta Lake
6. Analiza danych przy pomocy języka SQL
7. Wizualizacja danych
8. Utworzenie modelu ML
9. Automatyzacja (job scheduling)
10. Udostępnianie danych

Poniżej zaprezentowano krok po kroku wykonanie ćwiczenia:

1. Najpierw utworzono konto i zalogowano się do DataBricks



Stworzono również własny Cluster do dalszej analizy:

State	Name	Runtime	Active memory	Active cores	Active DBU / h	Source	Creator	Notebooks
●	BI Cluster	12.2	15 GB	2 cores	1	UI	filiphalys02@gmail.com	-

- Następnie wgrano dane z 3 plików csv: klienci.csv, produkty.csv, transakcje.csv (przykład poniżej dla klientów, resztę plików załadowano analogicznie)

[Add data >](#)

## Create New Table

Data source


**Upload File** S3

DBFS Target Directory [?](#)

/FileStore/tables/ (optional)

Files uploaded to DBFS are accessible by everyone who has access to this workspace. [Learn more](#)

Files [?](#)

klienci.csv 

11.1 KB  
[Remove file](#)

✓ File uploaded to /FileStore/tables/klienci-4.csv

[?](#)

### Select a Cluster to Preview the Table

Choose a cluster with which you will read and preview the data.

Cluster [?](#)

BI Cluster

### Specify Table Attributes

Specify the Table Name, Database and Schema to add this to the data UI for other users to access

Table Name [?](#)

clients

Create in Database [?](#)

default

File Type [?](#)

CSV

Column Delimiter [?](#)

,

☒ First row is header [?](#)

☐ Infer schema [?](#)

☐ Multi-line [?](#)

Table Preview

customer_id	name	email	phone	address
STRING <input type="button" value="v"/>	STRING <input type="button" value="v"/>	STRING <input type="button" value="v"/>	STRING <input type="button" value="v"/>	STRING <input type="button" value="v"/>
CUST0000	April Johnson	ashleywilliams@hotmail.com	033-006-8702	9644 Zoe Cove, Petersonchester, RI 91118
CUST0001	Carrie Gallagher	jeremiahmendez@hotmail.com	187-288-7570	Unit 8955 Box 6927, DPO AP 61373
CUST0002	Angela Gray	annawilliams@johnston-flynn.com	+1-638-545-2275	6876 Ryan Lodge, Garychester, OI 17960
CUST0003	Jose Francis	ecobb@hotmail.com	+1-205-954-7837x3890	6660 Brown Park, North Christopher, NH 57602
CUST0004	Jessica Gross	nboyd@hotmail.com	(123)127-1029	24527 Roberts Plain Apt. 699, Lak Emily, WA 69627
CUST0005	Leah Herring	larsonlauren@yahoo.com	(857)782-5589	293 Holden Via Apt. 460, West Kendrafort, AL 51790

3. W kolejnych kroku załadowano dane przy pomocy Notebooka i PySaprk. Poniżej zaprezentowano wczytanie danych i wyświetlenie kilku rekordów z tabeli klientów:

The screenshot shows a Databricks notebook interface. On the left is a 'Catalog' sidebar with a search bar and a tree view containing 'hive\_metastore', 'default', 'clients', 'products', 'transactions', and 'samples'. The main area displays two code cells. The first cell, executed '1 minute ago (1s)', loads three tables: 'default.clients' into 'c', 'default.products' into 'p', and 'default.transactions' into 't'. The second cell, executed 'Just now (1s)', runs 'df.show(5)' to display the first five rows of the 'clients' table. Below the code, a table preview shows columns: customer\_id, name, email, phone, and address. The data rows are as follows:

customer_id	name	email	phone	address
CUST0000	April Johnson	ashleywilliams@ho...	033-006-8702	9644 Zoe Cove, Pe...
CUST0001	Carrie Gallagher	jeremiahmendez@ho...	187-288-7570	Unit 8955 Box 692...
CUST0002	Angela Gray	annawilliams@john...	+1-638-545-2275	6876 Ryan Lodge, ...
CUST0003	Jose Francis	ecobb@hotmail.com	+1-205-954-7837x3890	6660 Brown Park, ...
CUST0004	Jessica Gross	nboyd@hotmail.com	(123)127-1029	24527 Roberts Pla...

only showing top 5 rows

4. W kolejnym etapie wykonano prosty proces czyszczenia i przetwarzania danych. Skupiono się na detekcji wartości brakujących i usunięcia duplikatów. Operacje te wykonano w ponownie Pythonie (notebook).

The screenshot shows a Databricks notebook with a code cell executed '2 minutes ago (31s)'. The code performs the following steps: it calculates the number of duplicates for 'clients' (dup\_c), 'products' (dup\_p), and 'transactions' (dup\_t) by comparing the count of rows before and after dropping duplicates. It then prints these counts. Next, it drops the 'default.transactions' table and writes the deduplicated 'transactions' data back to the 'default.transactions' table using Delta format with 'overwrite' mode. Finally, it prints a message indicating that duplicates from the transactions table have been removed and the data is now in Delta format. The output shows that there were 0 duplicates for clients and products, and 10 duplicates for transactions.

```
dup_c = c.count() - c.dropDuplicates().count()
print(f"Liczba duplikatów klientów: {dup_c}")

dup_p = p.count() - p.dropDuplicates().count()
print(f"Liczba duplikatów produktów: {dup_p}")

dup_t = t.count() - t.dropDuplicates().count()
print(f"Liczba duplikatów transakcji: {dup_t}")

spark.sql("DROP TABLE IF EXISTS default.transactions")
t.dropDuplicates().write.format("delta").mode("overwrite").saveAsTable("default.transactions")
print("Usunięto duplikaty z transakcji i zapisano jako tabelę Delta.")
```

(22) Spark Jobs

Liczba duplikatów klientów: 0  
Liczba duplikatów produktów: 0  
Liczba duplikatów transakcji: 10  
Usunięto duplikaty z transakcji i zapisano jako tabelę Delta.

```
spark.sql("""
SELECT
    SUM(CASE WHEN
        transaction_id IS NULL
        OR customer_id IS NULL
        OR product_id IS NULL
        OR quantity IS NULL
        OR transaction_date IS NULL
        THEN 1 ELSE 0 END) AS missing_data_in_transactions
FROM default.transactions
""").show()
```

▶ (2) Spark Jobs

missing_data_in_transactions
0

W tym kroku wykonano również rozbiecie kolumny z datą transakcji w formacie YYYY-MM-DD, na trzy osobne kolumny przechowujące informację o odpowiednio: roku, miesiącu i dniu transakcji. Krok ten wykonano w celu ułatwienia grupowania danych po dacie w kolejnych punktach analizy.

```
t = t.withColumn("year", F.year("transaction_date").cast("int")) \
    .withColumn("month", F.month("transaction_date").cast("int")) \
    .withColumn("day", F.dayofmonth("transaction_date").cast("int"))

# Wyświetlamy wynik
t.show()
```

▶ (1) Spark Jobs

t: pyspark.sql.dataframe.DataFrame = [transaction\_id: string, customer\_id: string ... 6 more fields]

transaction_id	customer_id	product_id	quantity	transaction_date	year	month	day
TRX000000	CUST0075	PROD0040	5	2024-07-19	2024	7	19
TRX000001	CUST0008	PROD0045	5	2024-12-22	2024	12	22
TRX000002	CUST0047	PROD0027	5	2025-04-02	2025	4	2
TRX000003	CUST0044	PROD0049	1	2025-01-15	2025	1	15
TRX000004	CUST0029	PROD0036	3	2025-03-28	2025	3	28

5. W kolejnym etapie zapisano (nadpisano) tabele po operacjach z punktu 4 jako tabele Delta:

```
1 minute ago (20s) 8 Python
```

```
t.write.format("delta").mode("overwrite").option("mergeSchema", "true").saveAsTable("transactions")
spark.sql("DROP TABLE IF EXISTS default.products")
p.write.format("delta").mode("overwrite").option("mergeSchema", "true").saveAsTable("products")
spark.sql("DROP TABLE IF EXISTS default.customers")
c.write.format("delta").mode("overwrite").option("mergeSchema", "true").saveAsTable("customers")
```

► (20) Spark Jobs

6. Na podstawie tak przygotowanych danych wykonano kilka zapytań z wykorzystaniem języka SQL:

Najczęściej sprzedawane produkty (top 10):

```
Just now (2s) 1 SQL
```

```
%sql
SELECT p.product_name, COUNT(*) AS total_sales
FROM transactions t
JOIN products p ON t.product_id = p.product_id
GROUP BY p.product_name
ORDER BY total_sales DESC
```

► (3) Spark Jobs

► \_sqlId: pyspark.sql.dataframe.DataFrame = [product\_name: string, total\_sales: long]

	product_name	total_sales
1	Third	41
2	Run	29
3	Not	28
4	Consumer	28
5	Every	28
6	Today	27
7	Above	26
8	Second	26
9	Road	25
10	Worker	25

Klienci, którzy wydali najwięcej pieniędzy na zakupy (top 5):

```
Just now (3s) 2 SQL
```

```
%sql
SELECT
  c.customer_id,
  c.name,
  round(SUM(t.quantity * p.price), 2) AS total_spent
FROM transactions t
JOIN products p ON t.product_id = p.product_id
JOIN customers c ON t.customer_id = c.customer_id
GROUP BY c.customer_id, c.name
ORDER BY total_spent DESC
```

► (4) Spark Jobs

► \_sqlId: pyspark.sql.dataframe.DataFrame = [customer\_id: string, name: string ... 1 more field]

	customer_id	name	total_spent
1	CUST0096	Jackson Gray PhD	30594.06
2	CUST0029	Amanda Combs	26814.67
3	CUST0082	Christine Davis MD	23758.73
4	CUST0080	Miss Jessica Colon	23230.03
5	CUST0046	Charlotte Rose	23227.87

Ilość sprzedanych sztuk produktów, miesiąc po miesiącu:

Just now (2s)

3

SQL

```
%sql
SELECT
  year,
  month,
  SUM(quantity) AS total_products_sold
FROM transactions
GROUP BY year, month
ORDER BY year, month
```

(2) Spark Jobs

\_sqldf: pyspark.sql.dataframe.DataFrame = [year: integer, month: integer ... 1 more field]

Table

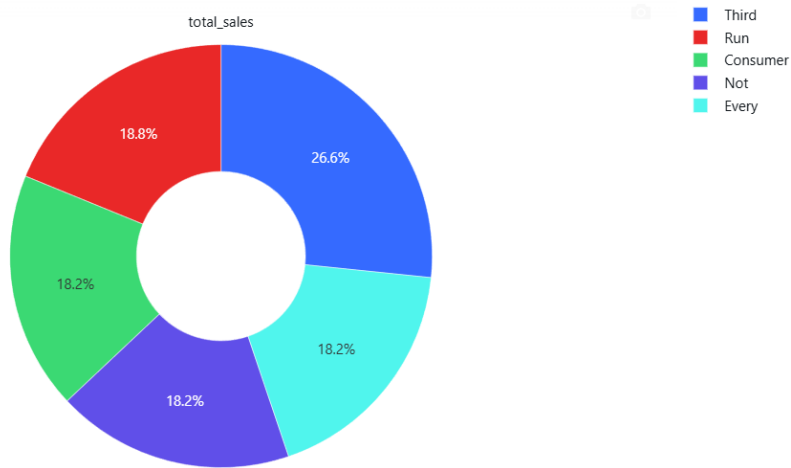
	year	month	total_products_sold
1	2024	5	205
2	2024	6	239
3	2024	7	213
4	2024	8	280
5	2024	9	269
6	2024	10	289
7	2024	11	232
8	2024	12	257
9	2025	1	209
10	2025	2	237

7. W celu wizualizacji danych skorzystano z wbudowanych funkcji w DataBricks.

Wykres słupkowy z ilością sprzedanych produktów (y) dla każdego miesiąca (x)

Month	total_products_sold
2024-10	289
2024-11	232
2024-12	257
2024-5	205
2024-6	239
2024-7	213
2024-8	280
2024-9	269
2025-1	209
2025-2	237
2025-3	213
2025-4	257
2025-5	45

Wartość sprzedaży (suma) dla 5 produktów o najwyższej wartości sprzedaży zamieszczona na wykresie typu pie-chart:



8. W następnym kroku utworzono bardzo prosty model ML przewidujący miesięczną ilość sprzedaży w kolejnych (przyszłych) miesiącach. Do uczenia modelu usunięto ostatni (aktualny, a więc niepełny w dane) miesiąc. Posłużono się modelem regresji liniowej z pakietu sklearn. Po utworzeniu modelu wygenerowano wartości dla 6 kolejnych, przyszłych miesięcy:

```
future_index = np.arange(len(pdf), len(pdf) + 6).reshape(-1, 1)
future_predictions = model.predict(future_index)

future_dates = pd.date_range(start=pdf['date'].max() + pd.offsets.MonthBegin(1), periods=6, freq='MS')

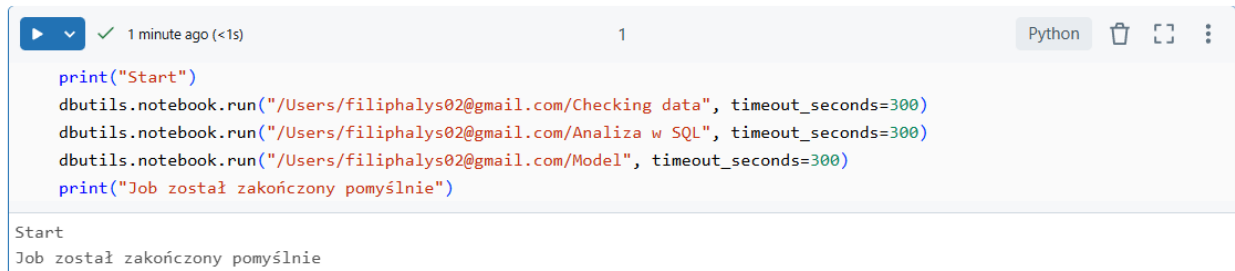
forecast_df = pd.DataFrame({
    'date': future_dates,
    'predicted_products_sold': future_predictions.astype(int)
})

print(forecast_df)
```

	date	predicted_products_sold
0	2025-05-01	244
1	2025-06-01	245
2	2025-07-01	245
3	2025-08-01	246
4	2025-09-01	246
5	2025-10-01	246

W przyszłych miesiącach wartość sprzedaży nieznacznie będzie wzrastać, choć wzrost ten najprawdopodobniej nie będzie zbyt duży.

9. W przedostatnim etapie ćwiczenia utworzono Job, do automatyzacji procesu ładowania danych, ich procesowania, czyszczenia, wizualizacji i modelowania. Stworzenie joba polegało na utworzeniu jednego notebook'a, który uruchamiałby wszystkie inne notebooki z odpowiednimi krokami pipelin'u.



```
print("Start")
dbutils.notebook.run("/Users/filiphalys02@gmail.com/Checking data", timeout_seconds=300)
dbutils.notebook.run("/Users/filiphalys02@gmail.com/Analiza w SQL", timeout_seconds=300)
dbutils.notebook.run("/Users/filiphalys02@gmail.com/Model", timeout_seconds=300)
print("Job został zakończony pomyślnie")
```

Start  
Job został zakończony pomyślnie

10. Finalnym etapem ćwiczenia była weryfikacja możliwości udostępnienia danych innym użytkownikom.

Można udostępniać dashboardsy lub eksportować dane do zewnętrznych narzędzi BI (Power BI, Tableau), a także udostępnić tabele innym użytkownikom workspace'a. Dostęp do tabeli można nadać użytkownikom DB za pomocą interfejsu UI, ale również chociażby z wykorzystaniem SQL:

*GRANT SELECT ON TABLE transactions TO `mail uzytkownika`*



## Ćwiczenie 9

Celem biznesowym tego ćwiczenia jest Analiza, wizualizacja oraz wymodelowanie danych transakcyjnych klientów z wykorzystaniem języka DAX i Power BI. Analizę przeprowadzono na podstawie danych ze sprzedaży samochodów (10.000 rekordów). Ćwiczenie podzielono na kilka etapów:

1. Import danych
2. Czyszczenie danych
3. Tworzenie przykładowych kalkulacji w DAX
4. Wizualizacja danych
5. Analiza trendów (język M)
6. Utworzenie raportu

Wykonanie:

W pierwszym kroku zaimportowano dane z pliku excel (XLSX) do środowiska w PowerBI. Poniżej zaprezentowano strukturę oraz kilka pierwszych rekordów danych:

Data	ID produktu	nazwa produktu	Selling Prize	sales quantity	customer id
45769	P1550	Chevrolet Malibu	13625,88	5	C88661
45119	P7573	Chevrolet Malibu	28717,71	5	C97589
45224	P5874	Chevrolet Malibu	13465,19	5	C47280
45344	P1148	Chevrolet Malibu	42654,36	5	C30469
45280	P8863	Chevrolet Malibu	14072,23	5	C74261
45466	P9084	Chevrolet Malibu	47152,26	5	C67852
45095	P1456	Chevrolet Malibu	44544,22	5	C58170
45662	P3841	Chevrolet Malibu	27667,2	5	C52960
45782	P6421	Chevrolet Malibu	49784,93	5	C40792

Podczas importu zweryfikowano, czy kolejne kolumny zapisują się w odpowiednich typach (okazało się, że kolumna z Datą zapisana jest niepoprawnie, więc ją konwertowano do typu date). Dokonano również kilku operacji czyszczących dane (usunięcie 3 rekordów z wartościami brakującymi).

Następnie utworzono kilka nowych kolumn. Wykorzystano język DAX:

- Całkowity przychód ze sprzedaży danej transakcji (ilość produktów \* cena):

1	Total Selling Value = 'Sheet1'[Selling Prize] * 'Sheet1'[sales quantity]
---	--

- Całkowity przychód ze sprzedaży (suma kolumny Total Selling Value)

```

1 Full Selling Value = CALCULATE(
2   SUM('Sheet1'[Total Selling Value]),
3   REMOVEFILTERS('Sheet1')
4 )

```

- Aby ułatwić grupowanie utworzono również kolumnę  
Year – z rokiem transakcji,  
Month – z miesiącem transakcji,  
YearMonth – z rokiem i miesiącem transakcji
- Średnią miesięczną sprzedaż z transakcji

```

1 Monthly Avg Per Transaction =
2 CALCULATE(
3   AVERAGE(Sheet1[Total Selling Value]),
4   ALLEXCEPT(Sheet1, Sheet1[YearMonth])
5 )

```

- Średnią roczną sprzedaż z transakcji

```

1 Annual Avg Per Transaction =
2 CALCULATE(
3   AVERAGE(Sheet1[Total Selling Value]),
4   ALLEXCEPT(Sheet1, Sheet1[Year])
5 )

```

- AOV – Średnią Wartość Zamówienia

```

1 AOV = AVERAGE(Sheet1[Total Selling Value])

```

- Średnią częstotliwość zakupów

```

1 Purchase Frequency =
2 CALCULATE(
3   COUNT(Sheet1[Total Selling Value]),
4   ALLEXCEPT(Sheet1, Sheet1[customer id], Sheet1[Year])
5 )

```

- Średni czas życia klienta – różnica między pierwszym a ostatnim zakupem

```

1 Customer Lifespan =
2 DATEDIFF(
3   MIN(Sheet1[YearMonth]),
4   MAX(Sheet1[YearMonth]),
5   MONTH
6 )

```

- Wartość życia klienta – CLTV

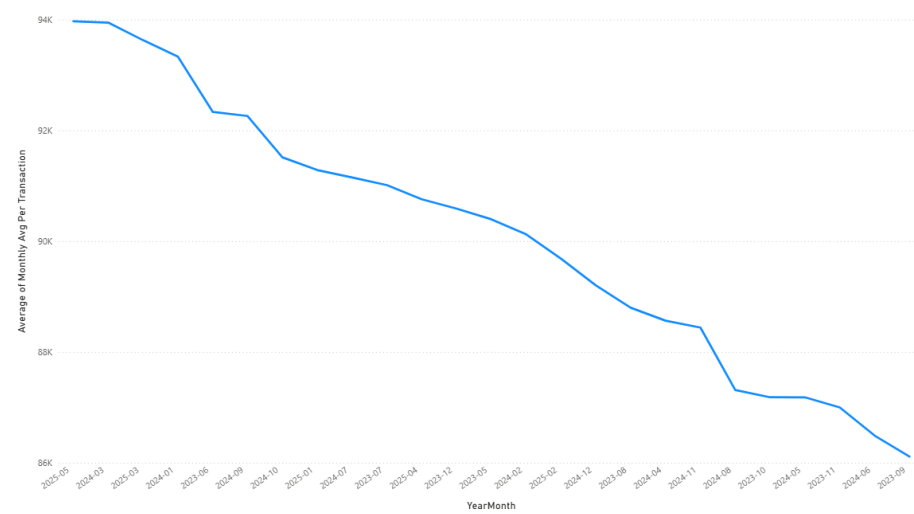
```

1 CLTV =
2 [AOV] *
3 [Purchase Frequency] *
4 [Customer Lifespan]

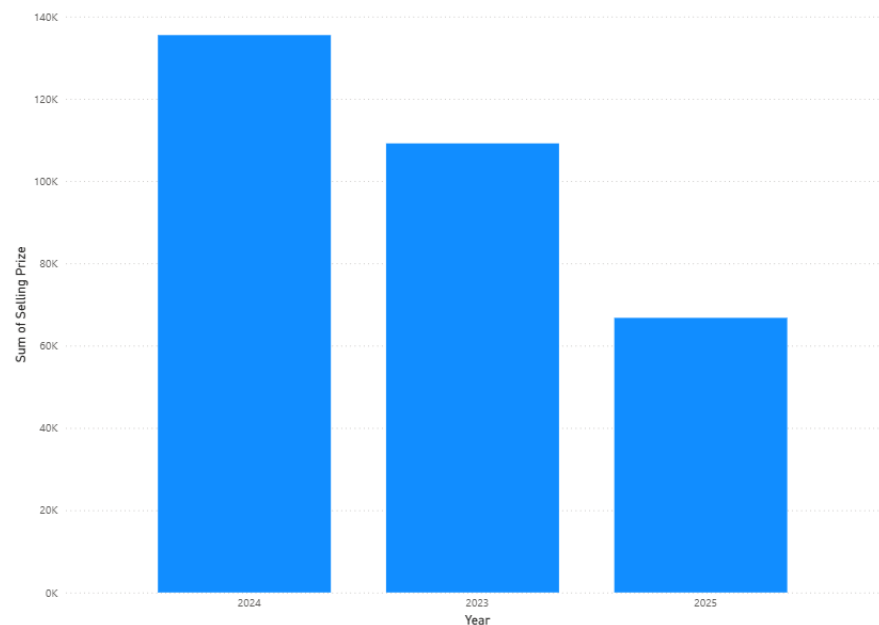
```

Następnie na podstawie utworzonych miar i KPI'ów stworzono kilka biznesowo wartościowych wykresów mających istotny wpływ na analizę sprzedaży samochodów.

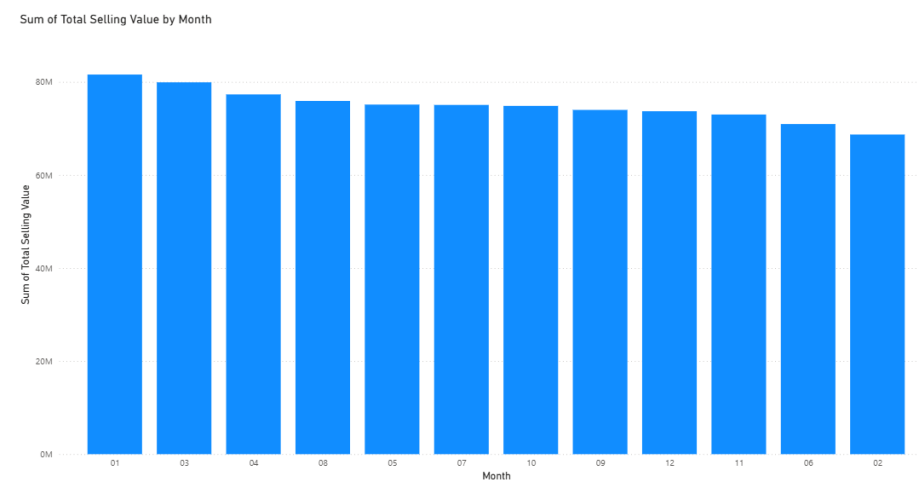
1. Wykres zmian średniej wartości w ramach jednej sprzedaży na przestrzeni miesięcy.



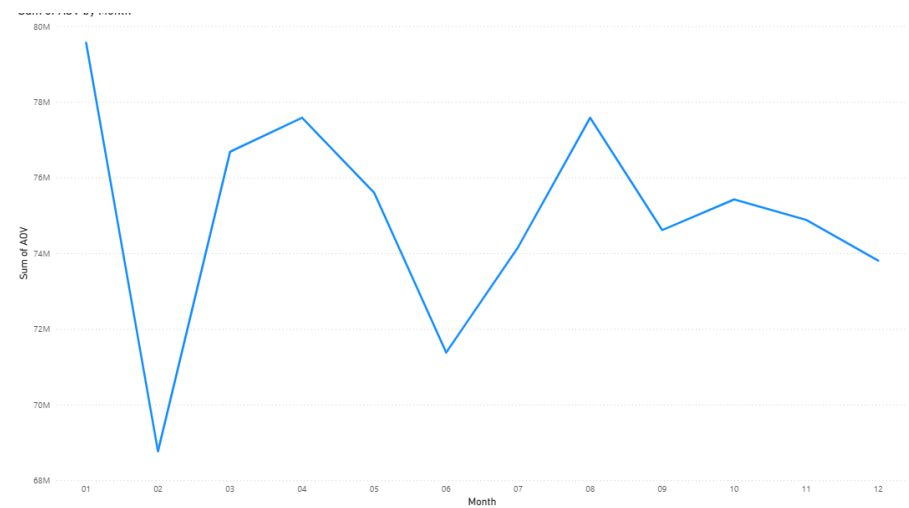
2. Wartość całkowitej sprzedaży dla danego roku



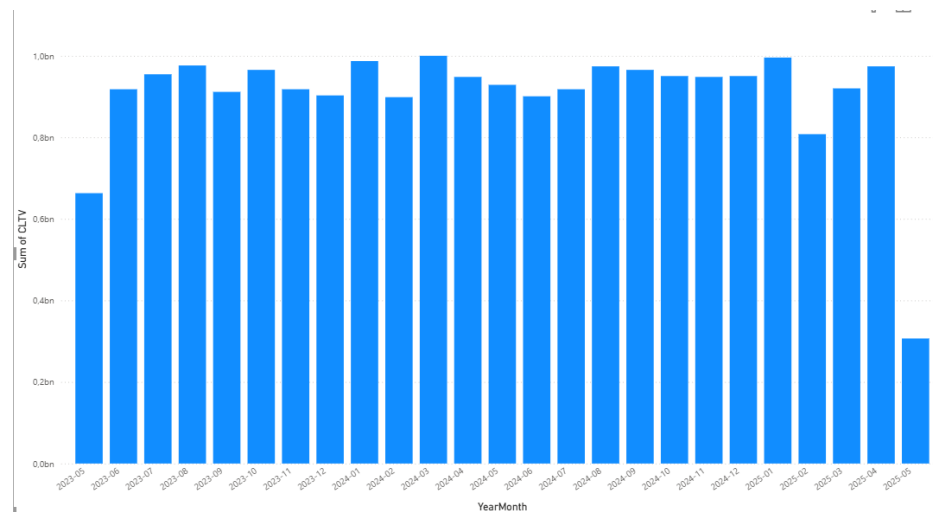
3. Wartość całkowitej sprzedaży dla danych miesięcy (bez uwzględnienia roku)



4. AOV dla danych miesięcy (bez uwzględnienia roku)



5. CLTV dla każdego miesiąca i roku



#### Podsumowanie i wnioski:

- Dane zostały zaimportowane z pliku zawierającego szczegóły transakcji, takie jak: data, ID produktu, cena sprzedaży (Selling Price), ilość (Sales Quantity), identyfikator klienta
- W Power Query dodano pomocnicze kolumny Year, Month oraz YearMonth, które umożliwiają analizę czasową
- Zastosowano język M do stworzenia kolumny Total Selling Value oraz do agregacji sprzedaży miesięcznej
- Na podstawie danych miesięcznych nie zaobserwowano wyraźnych wahań sprzedaży w czasie
- Obliczono średnią wartość zamówienia (AOV) – pokazuje, ile przeciętnie wydaje klient przy jednej transakcji
- Wyznaczono Customer Lifetime Value (CLTV) – ilustruje łączną wartość klienta w czasie i pomaga zidentyfikować klientów o największym potencjale
- Obliczono także średnie sprzedaże miesięczne i roczne per transakcja

#### Rekomendacje biznesowe:

- Skupić działania marketingowe w okresach niskiej sprzedaży (np. luty, czerwiec), by poprawić wyniki
- Analizować dalej klientów z najwyższym CLTV — mogą stanowić bazę do programów lojalnościowych
- Zastosować ML i nowoczesne narzędzia AI do przewidywania wartości sprzedaży w kolejnych miesiącach/latach na podstawie istniejących danych z transakcji