BMW Global Sales Data Analysis using Apache PySpark Pipeline

1. Student Details:

- **Prajwal Tejamurthy - A00050899 (** Prajwal.tejamurthy2@mail.dcu.ie)
- **Shashank Ashok Gadavi - A00049544 (**Shashankashok.gadavi2@mail.dcu.ie)

2. Dataset Link:

- BMW Worldwide Sales Records (2010 to 2024):

  https://www.kaggle.com/datasets/ahmadrazakashif/bmw-worldwide-sales-records-20102024

- **Source Code:**

  https://gitlab.computing.dcu.ie/prajwal.tejamurthy2/2026-MCM-Cloud-BMW_Global_Sales_Data_Analysis

3. Video Demonstration:

The Video Walkthrough or demonstration can be found using the link below uploaded on DCU Google Drive.

https://drive.google.com/file/d/1tOc6abLEmEM_sgElCj9-6usO1hEMXnZT/view?usp=sharing

4. Introduction & Motivation:

The worldwide car industry is experiencing one of the largest changes in its history. Customer choices are moving from gasoline cars to hybrid and electric ones, luxury brands are rapidly growing in new markets and sales success is now influenced not just by engineering but also by data.

BMW, one of the most famous car brands in the world is a great example of how sales trends change in different areas, engine types and model ranges. When searching for a dataset that could effectively show a real data-engineering process, BMW's global sales data was the first to catch attention.

If we picture this scenario, A global BMW strategy team wants to find out:

- Which areas are becoming the best markets for SUVs?
- Are electric cars really becoming more popular each year?
- Which BMW models are not doing well and might need a new strategy?
- How does pricing change based on engine size, mileage and fuel type?

To find answers to these questions, they need more than just Excel spreadsheets or basic BI tools. They require a data pipeline that is scalable, repeatable and reliable. This led to the decision to analyze BMW Global Sales Data using PySpark.

This project showcases a full big-data analysis pipeline created with Apache PySpark to handle BMW's global sales data from 2010 to 2024. The reason for creating a Spark-based pipeline is due to the large size of the dataset (50,000 rows), the necessity for scalable data processing, complex cleaning methods and analytical changes.

Regarding the Pipeline:

The goal was to simulate what a data engineering team inside BMW (or any big manufacturer) would do:

- Gather sales data from different areas.
- Make sure all formats are the same and clean up any inconsistent formats.
- Check important metrics like engine size compared to fuel type.
- Detect anomalies in price, mileage and hybrid-electric cars.
- Create analytical models to show trends & produce business insights and dashboards.

Objectives:

The primary objectives of this study are:

- Use PySpark to process, clean and analyze a huge automotive dataset that spans several years, without using pandas, to show how data engineering works in real life.
- Find and correct missing values, inconsistent formats, unrealistic engine sizes, strange mileage numbers, and errors specific to electric or hybrid cars while keeping the data intact.
- Develop metrics like year-over-year sales growth, demand index, rolling 3-year averages, price deciles, mileage bands, and regional sales share to enhance further analysis.
- Conduct cohort analysis (to see how model age affects performance), estimate elasticity (how price affects demand), check for seasonality, and detect anomalies to mimic real OEM analytics processes.
- Visualize the best-selling models, yearly global sales trends, and price distribution patterns to show how BMW’s market changes from 2010 to 2024.

5. Related work

The Effects of Price Dispersion on Sales in the Automobile Industry: A Dynamic Panel Analysis

This research looks at a big collection of about 222,592 real prices from 59 different car models over 48 months in China to see how changes in prices (price dispersion) impact car sales. They calculate two ways to measure dispersion: the "percentage difference" (PD) between the highest and lowest prices and the coefficient of variation (CV) of the transaction prices. The main discovery is that a moderate amount of price variation is linked to higher sales meaning that having some differences in price seems to help increase the number of cars sold. The paper gives real evidence that connects pricing strategies (how much prices vary) with demand showing how sellers might use price differences to boost their sales.[1]

Analyzing price efficiency using machine learning generated price indices: The case of the Chilean used car market

This paper looks into the used-car market in Chile by analyzing a huge dataset of about 2.7 million used-car ads. They use machine learning, specifically a random forest method, to create synthetic "price indices" based on different models and brands, combining many individual price data points into organized indices. Their findings indicate that used-car prices adjust quickly and significantly, especially for newer and more expensive models, when the import prices of new cars fluctuate. The study shows how using machine-learning price indices and econometric/event-study methods can uncover the changing relationships between the supply of new cars, the value of used cars and how the market reacts. [2]

Demand for green and fossil fuel automobiles

This research looks at the demand for cars in the UK from 2008 to 2019. It compares regular cars (CVs: petrol/diesel) with alternative-fuel cars (AFVs: hybrid/electric). The study uses a special model that considers different consumer preferences and how prices affect choices. The researchers calculated how sensitive the demand is to price changes for both types of vehicles. They found that if the price of a car goes up by £1,000, the demand drops by about 3 to 5% for both CVs and AFVs. Here, the key point is that UK consumers don't automatically choose AFVs just because they are better for the environment when the prices are the same as CVs, the environmental benefits don't make people willing to pay more [3].

A. How our solution differs from these works?

- The first two papers look at how prices change or how well the used-car market works on a big scale (with lots of brands, many models, and wide markets). Our project zoomed in on just one brand (BMW) over several years (2010–2024), using detailed information for each sale (like model, region, fuel type, etc.). This lets us analyze things at the model and region level instead of just looking at the overall market.

- While earlier studies mainly use price and model-level data, our dataset has extra details like engine size, mileage, fuel type (including hybrid and electric), region, and transmission. This gives us more angles to look at. With this richer information, we can

do advanced analyses like checking quality, validating based on engine types, spotting outliers, looking at regional shares and understanding how different vehicle types (like hybrid versus petrol) affect things.

- The previous papers think that the transaction data is “clean and valid.” However, our project includes specific checks for data quality and business rules (like making sure engine sizes fit for EV/hybrid and checking for price oddities). This cleaning process is often overlooked in academic studies but is super important for real-world data.

- Our analysis is done entirely in PySpark (not using pandas), which makes it scalable, reproducible, and similar to big-data pipelines used in the industry. Many academic studies rely on summary statistics or machine learning with sampled or cleaned data. Our pipeline demonstrates full end-to-end ingestion → cleaning → feature engineering → analytics → visualization.

- Our project doesn’t only focus on price differences or demand elasticity; we also create cohorts, track rolling sales, develop demand indices, categorize prices into deciles, analyze by region, detect anomalies, handle electric and hybrid vehicles specially and create various visualizations.

6. Dataset Description:

The dataset for this project called "BMW Sales Data (2010–2024)" was obtained from Kaggle

A. Source of the Datasets:

This dataset gives a made-up but realistic look at BMW's sales activities over 15 years. It includes information about different regions, how various models are doing, types of fuel, pricing trends and vehicle details.

- It contains 50,000 individual sales records/rows of BMW cars.
- There are 11 different fields.
- It covers various car types: Petrol, Diesel, Hybrid and Electric.
- It includes multiple global regions: Europe, Asia, Africa, the Middle East, North America and South America.

Data Extraction process:

The dataset was downloaded directly as a CSV file from Kaggle and imported into the project environment.

The CSV was loaded using spark.read.csv () with schema inference, header recognition and multiline handling:

```python
from pyspark.sql import functions as F
from pyspark.sql import SparkSession

CSV_PATH = r"C:\Users\Admin\Downloads\BMW sales data (2010-2024).csv"

def load_csv(spark, path=CSV_PATH):
    df = spark.read.csv(path, header=True, inferSchema=True, multiLine=True, escape='"')
    return df
```

This enabled Spark to automatically detect column types (integer, double, string) and efficiently parallelize the ingestion step.

Once loaded into Spark, exploratory commands (printSchema, df.show(), count(), distinct(), null checks) were applied.

```python
def basic_inspect(df, sample_n=10, distinct_preview_n=8):
    print("\n--- Schema ---")
    df.printSchema()

    total = df.count()
    print(f"\n--- Total rows: {total} ---")

    print(f"\n--- Sample {sample_n} rows ---")
    df.show(sample_n, truncate=False)

    # Columns list
    cols = df.columns
    print("\n======= Columns (count: {}) =======".format(len(cols)))
    for c in cols:
        print("-", c)

    # Null counts per column
    exprs = [
        F.sum(F.when((F.col(c).isNull()) | (F.trim(F.col(c)) == ""), 1).otherwise(0)).alias(c)
        for c in cols
    ]
    nulls_row = df.select(exprs).collect()[0].asDict()
    print("\nNull / empty counts per column ")
    for k, v in nulls_row.items():
        print(f"{k}: {v}")

    # Distinct counts
    print(f"\n--- Distinct counts (first {distinct_preview_n} columns) ---")
    for c in cols[:distinct_preview_n]:
        cnt = df.select(F.countDistinct(F.col(c))).collect()[0][0]
        print(f"{c}: {cnt} distinct")
```

Then, the raw Spark Data Frame was then passed into the cleaning pipeline as part of the Integration into the Processing Pipeline.

7. Description of Data Processing Pipeline

The data processing stage is the main part of this project, where raw CSV files were converted into a clean, structured, and analysis-ready dataset. Since the datasets were large and came from different sources, the goal was to design a clear and systematic workflow using Apache Spark. The following steps describe the full data processing approach.

```
def standardize_column_names(df):
    new_names = [to_snake_case(c) for c in df.columns]
    for old, new in zip(df.columns, new_names):
        if old != new:
            df = df.withColumnRenamed(old, new)
    return df

def clean_string_columns(df, string_cols):
    for c in string_cols:
        # trim whitespace and collapse multiple spaces
        df = df.withColumn(c, F.when(F.col(c).isNull(), None).otherwise(F.trim(F.regexp_replace(F.col(c), r'\\s+', ' '))))
    return df

def normalize_categories(df):
    # Title-case common categorical columns for consistency
    for c in ["transmission", "fuel_type", "region", "color", "sales_classification", "model"]:
        if c in df.columns:
            df = df.withColumn(c, F.initcap(F.col(c)))
    # Standardize sales_classification to a small set
    if "sales_classification" in df.columns:
        df = df.withColumn("sales_classification",
                           F.when(F.col("sales_classification").isin("High","Low","Medium"),
                                  F.col("sales_classification")).otherwise(F.lit("Unknown")))
    return df

def enforce_types(df):
    # Cast numeric columns defensively
    if "year" in df.columns:
        df = df.withColumn("year", F.col("year").cast(T.IntegerType()))
    if "engine_size_l" in df.columns:
        df = df.withColumn("engine_size_l", F.col("engine_size_l").cast(T.DoubleType()))
    if "mileage_km" in df.columns:
        df = df.withColumn("mileage_km", F.col("mileage_km").cast(T.LongType()))
    if "price_usd" in df.columns:
        df = df.withColumn("price_usd", F.col("price_usd").cast(T.DoubleType()))
    if "sales_volume" in df.columns:
        df = df.withColumn("sales_volume", F.col("sales_volume").cast(T.IntegerType()))
```

```
root
 |-- model: string (nullable = true)
 |-- year: integer (nullable = true)
 |-- region: string (nullable = true)
 |-- color: string (nullable = true)
 |-- fuel_type: string (nullable = true)
 |-- transmission: string (nullable = true)
 |-- engine_size_l: double (nullable = true)
 |-- mileage_km: long (nullable = true)
 |-- price_usd: double (nullable = true)
 |-- sales_volume: integer (nullable = true)
 |-- sales_classification: string (nullable = true)
 |-- sale_year: integer (nullable = true)
 |-- sale_date: date (nullable = true)
 |-- flag_mileage_km_outlier: boolean (nullable = true)
 |-- flag_price_usd_outlier: boolean (nullable = true)
```

```
--- Total rows after cleaning: 50000 ---

--- Sample rows (10) ---
+--------+----+-------------+-----+----------+------------+-------------+----------+---------+------------+--------------------+---------+----------
+-----------------------+----------------------+
|model   |year|region       |color|fuel_type|transmission|engine_size_l|mileage_km|price_usd|sales_volume|sales_classification|sale_year|sale_date
|flag_mileage_km_outlier|flag_price_usd_outlier|
+--------+----+-------------+-----+----------+------------+-------------+----------+---------+------------+--------------------+---------+----------
+-----------------------+----------------------+
|I8      |2020|North America|Blue |Electric  |Manual      |4.3          |191305    |70057.0  |7327        |High                |2020     |2020-01-01
|false                  |false                 |
|M5      |2012|Europe       |Black|Petrol    |Automatic   |3.7          |124564    |119792.0 |3324        |Low                 |2012     |2012-01-01
|false                  |false                 |
|3 Series|2019|Africa       |White|Hybrid    |Automatic   |1.9          |149301    |44286.0  |7766        |High                |2019     |2019-01-01
|false                  |false                 |
|M3      |2018|North America|Grey |Diesel    |Automatic   |1.6          |120157    |117237.0 |8586        |High                |2018     |2018-01-01
|false                  |false                 |
|X5      |2022|Middle East  |Blue |Hybrid    |Manual      |3.3          |25709     |60203.0  |4029        |Low                 |2022     |2022-01-01
|false                  |false                 |
|I3      |2024|Europe       |Black|Hybrid    |Automatic   |3.8          |75550     |84588.0  |8537        |High                |2024     |2024-01-01
|false                  |false                 |
|7 Series|2012|Europe       |Black|Hybrid    |Automatic   |3.9          |131987    |40736.0  |9968        |High                |2012     |2012-01-01
|false                  |false                 |
|X6      |2024|Europe       |Grey |Diesel    |Manual      |2.1          |75841     |108867.0 |6839        |Low                 |2024     |2024-01-01
|false                  |false                 |
|X5      |2011|Asia         |Blue |Hybrid    |Manual      |4.1          |171452    |119451.0 |3086        |Low                 |2011     |2011-01-01
|false                  |false                 |
|5 Series|2021|North America|Blue |Petrol    |Automatic   |2.5          |151028    |31465.0  |8580        |High                |2021     |2021-01-01
|false                  |false                 |
+--------+----+-------------+-----+----------+------------+-------------+----------+---------+------------+--------------------+---------+----------
+-----------------------+----------------------+
only showing top 10 rows
```

Check for business-related issues and either mark or delete rows that have serious mistakes (like wrong year, price, mileage or missing model), while also keeping a record of what was done.

```
# Keep original raw copy for traceability
df = df.withColumnRenamed("price_usd", "price_usd_orig") \
       .withColumnRenamed("mileage_km", "mileage_km_orig") \
       .withColumnRenamed("engine_size_l", "engine_size_l_orig") \
       .withColumnRenamed("sales_volume", "sales_volume_orig") \
       .withColumnRenamed("year", "year_orig")

# Recreate cleaned column names (we'll fill these)
df = df.withColumn("price_usd", F.col("price_usd_orig").cast(T.DoubleType())) \
       .withColumn("mileage_km", F.col("mileage_km_orig").cast(T.LongType())) \
       .withColumn("engine_size_l", F.col("engine_size_l_orig").cast(T.DoubleType())) \
       .withColumn("sales_volume", F.col("sales_volume_orig").cast(T.IntegerType())) \
       .withColumn("year", F.col("year_orig").cast(T.IntegerType()))

# Trim string columns and normalize categories
for name, dtype in df.dtypes:
    if dtype == "string":
        df = df.withColumn(name, F.trim(F.regexp_replace(F.col(name), r'\s+', ' ')))
        if name in ("model","region","color","fuel_type","transmission","sales_classification"):
            df = df.withColumn(name, F.initcap(F.col(name)))

# create sale_date helper
df = df.withColumn("sale_year", F.col("year"))
df = df.withColumn("sale_date", F.to_date(F.concat_ws("-", F.col("sale_year"), F.lit("01"), F.lit("01")), "yyyy-MM-dd"))

# Initial validation flags (logical checks)
df = df.withColumn("flag_invalid_year", (F.col("year").isNull()) | (F.col("year") < YEAR_MIN) | (F.col("year") > YEAR_MAX))
df = df.withColumn("flag_invalid_price", (F.col("price_usd").isNull()) | (F.col("price_usd") < PRICE_MIN))
df = df.withColumn("flag_invalid_mileage", (F.col("mileage_km").isNull()) | (F.col("mileage_km") < 0))
df = df.withColumn("flag_invalid_engine", (F.col("engine_size_l").isNull()) | (F.col("engine_size_l") < ENGINE_MIN))
df = df.withColumn("flag_missing_model", F.col("model").isNull())
df = df.withColumn("flag_invalid_critical",
                   F.col("flag_missing_model") | F.col("flag_invalid_year") | F.col("flag_invalid_price") | F.col("sales_volume").isNull())
```

```
Rows before any cleaning: 50000
Rows after dropping critical invalids: 50000
Global medians used (price, mileage, engine): 75009.0 100382 3.2
IQR bounds (price): -16479.0 165169.0
IQR bounds (mileage): -103748.5 301519.5
IQR bounds (engine): -0.3999999999999999 6.799999999999999

Flag counts:
  invalid_critical: 0
  invalid_price: 0
  invalid_mileage: 0
  invalid_engine: 0
  imputed_mileage_model: 0
  imputed_mileage_global: 0
  imputed_engine_model: 0
  imputed_engine_global: 0
  outlier_price_iqr: 0
  outlier_mileage_iqr: 0
  outlier_engine_iqr: 0
```

8. Development of the pipeline:

Here, the pipeline was designed with a modular and scalable architecture, consisting of 3 phases:
A. Data Extraction
B. Data Processing
C. Data Visualization

A. Data Extraction Tool:

- The pipeline begins with a PySpark-based data ingestion component.
- PySpark's built-in .csv () reader is used because it supports distributed file loading, automatic schema inference, and efficient handling of large, multi-line datasets.
- The dataset contains 50,000 records across multiple attributes (model, region, fuel type, sales volume, mileage, price, etc.) and Spark's reader ensures that these fields are parsed in a structured way while retaining type integrity.

B. Data processing Tool:

After the BMW dataset is loaded, PySpark uses a series of operations that run at the same time by taking advantage of Spark SQL functions, Window functions and distributed aggregation tools to tidy up, check and improve the data for better analysis.

It implements schema correction, missing-value handling and text standardization using functions such as:

- With Column(), cast(), trim(), lower(),
- Conditional rules using when(), otherwise()

Cross-field consistency checks are performed usingF.when(), F.col(), F.abs(),

Derived metrics such as price per litre using arithmetic expressions,

- Statistical validation using:
- mean(), stddev(), approxQuantile(),

performs large-scale feature engineering using:

- groupBy().agg()
- Window functions (rowsBetween, lag, dense_rank)
- Aggregation functions including:
- sum(), avg(), count(), percentile_approx()

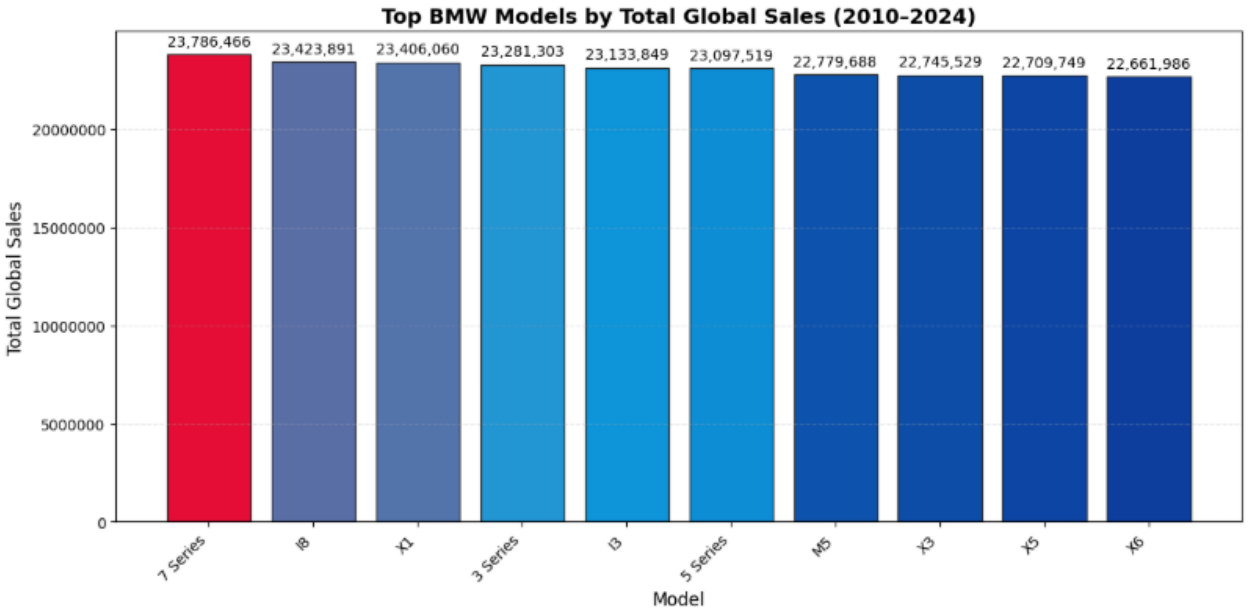
Advanced statistical tasks are implemented using:

- Cohort analysis (min year via F.min over window)

- Price anomaly detection using Z-score: F.abs(F.col("z_price")) > 3

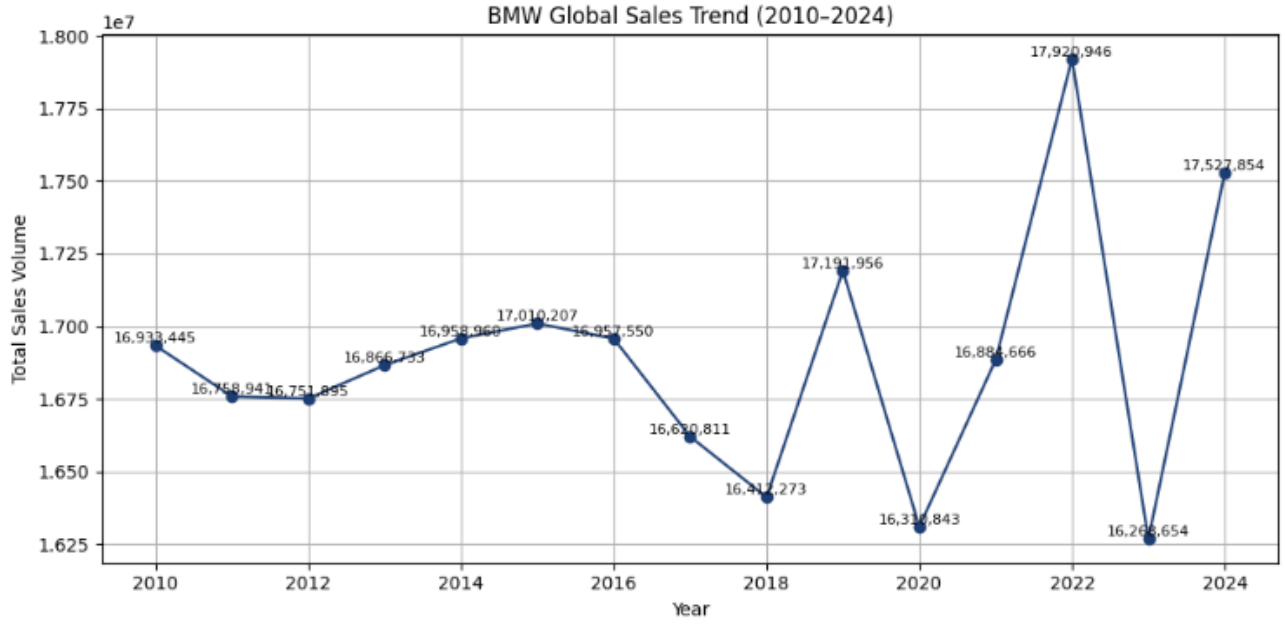
C. Data Visualization Tool:

Top BMW Models by Total Global Sales:



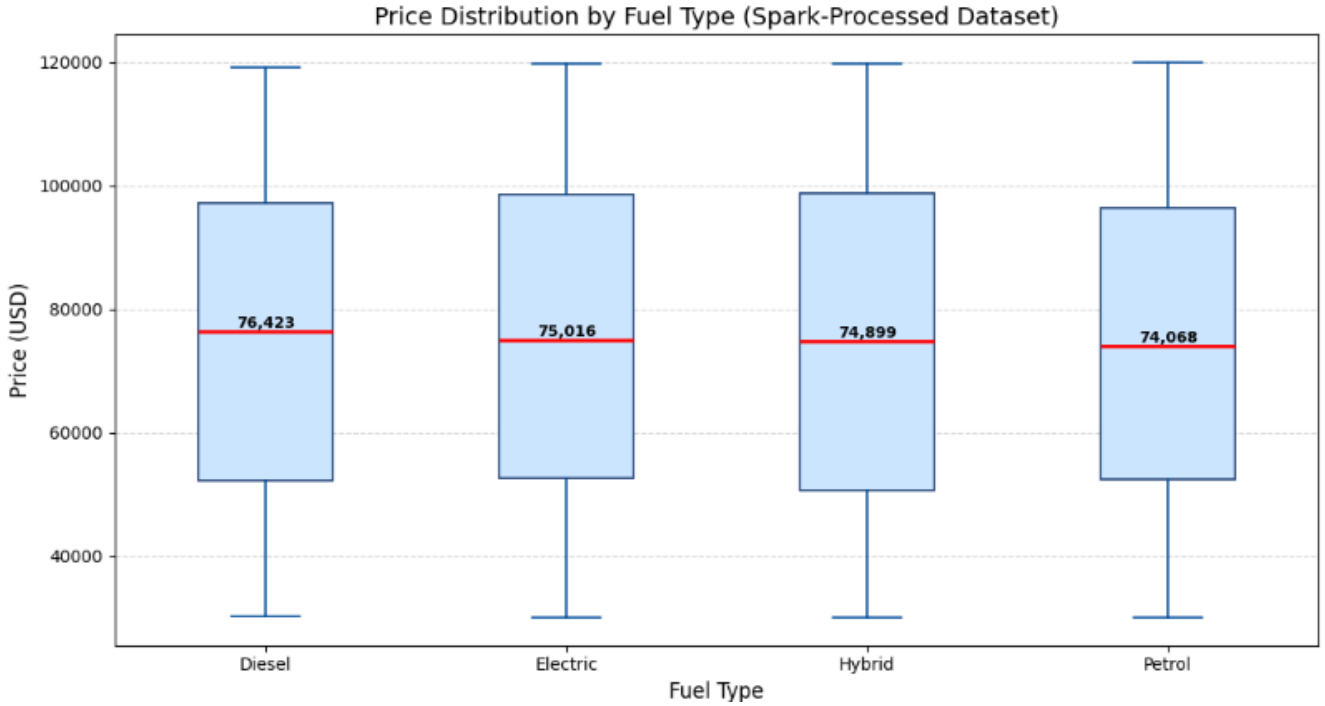
This above chart shows how BMW models are ranked according to their total sales worldwide from 2010 to 2024. It helps us see which models are the best sellers, what the long-term demand looks like and who the leaders in the market are.

BMW Global Sales Trend:



The above visualization points out Years of growth, Times of decline, Effects of market conditions and Trends for making predictions.

Price Distribution by Fuel Type:



This above visualization reveals the Price range differences, Outliers, Median and quartile insights and how fuel technology affects pricing.

9. Challenges and lessons learned

Throughout the whole process of building the pipeline, we encountered a number of technical difficulties. Tackling these problems allowed me to learn more about setting up data-processing workflows and to see how Spark works with actual datasets. The challenges we encountered are listed below:

- The source CSV had mixed case and spaces (like Model, Price_USD, Fuel_Type), which messed up the column references. We changed it to snake_case. This helps avoid AnalysisException because of name mismatches.
- Price_USD was loaded as an integer, but some calculations (like medians, z-scores, and price per liter) need float or double. We made sure to change the numeric columns to the right Spark types in a central enforce_types() step. This stops truncation or overflow and makes sure the aggregations are correct.
- We had to remove rows with missing models, invalid years, or sales_volume. So, we marked the invalid rows with boolean flags, then filtered them out, keeping the original columns (like price_usd_orig) and the flag columns so that every deletion can be checked.
- Electric vehicles (EVs) don't have engine displacement like regular cars, so using a simple price-per-liter doesn't make sense. Hybrids need more flexible rules. We added is_electric and is_hybrid flags and calculated a working engine field (engine_size_l_working) so that the rules and derived metrics are meaningful.

10. References

[1] Chiu, Y.-L., Du, J., & Wang, J.-N. (2022). The Effects of Price Dispersion on Sales in the Automobile Industry: A Dynamic Panel Analysis. SAGE Open, 12(3), 21582440221.
Available at: https://journals.sagepub.com/doi/full/10.1177/21582440221120647

[2] Lefort, F., & Díaz, F. (2025). Lefort, F., & Díaz, F. (2025). Analyzing price efficiency using machine learning generated price indices: The case of the Chilean used car market. Economic Modelling, 152, 107257.
Available at: https://www.sciencedirect.com/science/article/abs/pii/S0264999325002524

[3] Mandys, F., & Taneja, S. (2024). Demand for green and fossil fuel automobiles. Transportation Research Part A: Policy and Practice, 190, 104284.
Available at: https://kar.kent.ac.uk/107595/

[4] Kashif, A. R. (2025). BMW Worldwide Sales Records (2010–2024) [Data set]. Kaggle
Available at: https://www.kaggle.com/datasets/ahmadrazakashif/bmw-worldwide-sales-records-20102024

[5] The Apache Software Foundation. (2025). SQL Data Sources - CSV (Spark 4.0.1 Documentation) Apache Spark.
Available at: https://spark.apache.org/docs/latest/sql-data-sources-csv.html

[6] The Matplotlib development team. (2025). Using Matplotlib) [Documentation]. Matplotlib.
Available at: https://matplotlib.org/stable/contents.html