44517 - 03
Data Detectives

Driving Car Prices with Big Data

Divya Chelikani
Sai Sucharitha Konakanchi
Vinay Meenkeri
Swathi Beepeta

April 25, 2024

School of Computer Science and Information Systems
Northwest Missouri State University

1 Project idea

Our project is all about delving into the world of car prices data to understand what makes certain cars sell better than others. We're looking at everything from the year and make of the car to its color and condition. By using sophisticated data analysis techniques, we're aiming to uncover hidden patterns and trends that could explain why some cars are more popular than others. Our goal is to provide valuable insights that can help car dealerships, marketers, and researchers better understand the factors driving car sales in today's digital age. Ultimately, we want to contribute to the collective knowledge about what influences the success of car sales, helping businesses make smarter decisions and improve their performance in the market.

2 Project Description

Unraveling YouTube Channel Dynamics" is all about deeply understanding what makes YouTube channels successful. We want to look closely at things like the number of subscribers, how many people view the videos, the types of content being created, and where in the world the audience is located. To achieve this, we're using a powerful method called data science. It's like a detective tool for numbers. We're not just looking at the obvious stuff; we're trying to find hidden patterns, discover new trends, and figure out the most important factors that make a YouTube channel successful

3 Tools and Technologies

Our project will utilize a variety of data science tools to conduct a comprehensive analysis of Car prices.

3.1 Jupyter Notebook:

This versatile tool will serve as our primary environment for interactive documentation, code execution, and visualization. It provides a flexible platform for exploring and analyzing data.

3.2 Python Libraries:

Essential libraries such as Pandas, NumPy, and Matplotlib will be our go-to tools for data manipulation and numerical operations. These libraries significantly enhance our analysis capabilities by enabling efficient data processing.

3.3 PySpark:

For efficient handling of large datasets in our project, we'll use PySpark. To store and retrieve data effectively, we'll employ technologies like MySQL or

PostgreSQL. SQL will be utilized as the querying language for data manipulation.This approach ensures scalability and performance, enabling a comprehensive analysis of car prices.

3.4 Git:

Git is a version control system that helps teams manage code changes collaboratively. It tracks changes, allows branching for parallel development, facilitates collaboration, maintains version history, and syncs code between local and remote repositories. It's essential for effective code management and teamwork in software development.

4 Implementation steps

4.1 Downloading the Dataset from Kaggle:

- Download the Dataset from Kaggle
- Go to the Kaggle dataset webpage.
- Click on the "Download" button to save the dataset file (usually a .csv or .xlsx file) to your computer.

4.2 Data Cleaning:

- Open the dataset file in Excel.
- Remove any empty rows.
- Handle missing values by either filling them in or removing them.
- Adjust data types if needed (e.g., converting strings to numbers
- Save the changes to the dataset file.

4.3 Import Libraries and Load Dataset:

- Open your Python environment, such as Jupyter Notebook or Google Colab.
- Import Pandas by writing import pandas as pd in your Python script or notebook.
- Use pd.read csv() to load the dataset if it's a CSV file, or pd.read excel() if it's an Excel file.
- Import Matplotlib for plotting by writing import matplotlib.pyplot as plt. This allows you to create visualizations from your data.

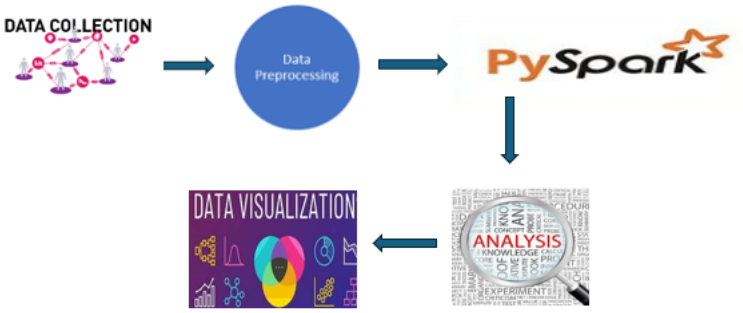

Figure 1: High-level architecture of the Car prices Analysis Project.

Architecture diagram & description

- **Data Import in Jupyter Notebook:** Import car prices dataset directly into Jupyter Notebook.
- **Data Processing:** Process and clean the imported data for analysis.
- **PySpark Operations:** Create a Framework and perform PySpark SQL operations directly on the dataset within Jupyter Notebook, allowing for flexible and efficient querying.
- **Data Analysis:** Conduct comprehensive analysis using Python Libraries, including Matplotlib for deep analysis.
- **Visualization:** Create interactive and insightful visualizations using tools like Tableau or Power BI.
- **Documentation:** Prepare a comprehensive and well-organized presentation of the entire process and outcomes using LaTeX.

5 Goals:

1. Identify car models with significant variations in monthly sales revenue.
2. The efficiency of each car in generating revenue based on its odometer reading.
3. Identify car models with high engagement relative to their sales.
4. The performance of different car categories based on their total sales in a year.
5. Calculate the average number of sales per car for the top three car manufacturers.

6. Identify the top-selling car models with the highest monthly earnings per car.

7. Identify dealerships that have experienced significant growth in sales over the past year

8. Identify car dealerships with the highest ratio of sales to inventory.

6 Results Summary

6.1 Identify car models with significant variations in monthly sales revenue.

The goal of this query is to give us an idea about calculating the average price variation for each car model by subtracting the MMR value from the selling price. It ensures data integrity by filtering out null values for selling price and MMR. The results are grouped by car model and sorted in descending order based on the calculated average price variation. This analysis provides insights into the pricing dynamics across different car models, highlighting models with significant price fluctuations.

The graph visually presents the top 15 car models with significant variations in monthly sales revenue. Each horizontal bar corresponds to a specific car model, with its length indicating the average price variation. This allows for easy identification of models experiencing notable fluctuations in sales revenue.

```python
import pandas as pd
```

```python
from pyspark.sql import SparkSession

# Create a SparkSession
spark = SparkSession.builder.getOrCreate()

# Load the data
Data = spark.read.format("csv").option("header", "true").load("cleaningdata.csv")

# Create a temporary view
Data.createOrReplaceTempView("carprices")

# Show the data
Data.show()
```

Figure 2: Screenshot

6.2 The efficiency of each car in generating revenue based on its odometer reading.

The goal of this Spark SQL query is to assess the efficiency of revenue generation for car videos based on views. It calculates revenue by subtracting the MMR value from the selling price and computes revenue per mile driven. The query ensures data integrity by filtering out null values for selling price, MMR, and odometer. Results are ordered in descending order based on revenue per mile, allowing for the identification of car videos that generate revenue most efficiently.

The scatter plot shows how efficiently each car model generates revenue based on its odometer reading. Each point represents a car model, with its vertical position indicating revenue per mile. It helps identify which models generate revenue more efficiently relative to their mileage.

6.3 Identify car models with high engagement relative to their sales.

The goal of this Spark SQL query is to identify car models with high engagement relative to their sales. It calculates the engagement per dollar metric by dividing the selling price by the Manheim Market Report (MMR) and multiplying by 100. The query filters out null values for selling price, MMR, and make, ensuring data integrity. Results are then ordered in descending order based on engagement per dollar, with the top 10 car models displayed. This analysis helps identify which car models attract more engagement from buyers compared to their sales price.

The histogram graph explains how revenue per mile is distributed across

different cars. Each bar represents a range of revenue per mile values, and its height indicates the frequency of cars within that range. Higher bars suggest more cars fall within that revenue per mile range. The table provides specific revenue per mile values for the top cars, sorted by highest revenue per mile.

6.4 The performance of different car categories based on their total sales in a year

The goal of this Spark SQL query is to assess the sales performance of various car categories by totaling their sales for each year. It calculates the total sales by model for each year and orders the results by year and total sales. This analysis aids in understanding the popularity and trends of different car categories over time.

Each bar represents a specific category, with its length indicating the total sales amount. The longer the bar, the higher the total sales for that category. The accompanying table provides details on the top-selling car models and their respective total sales.

6.5 Calculate the average number of sales per car for the top three car manufacturers

The goal of this Spark SQL query is to determine the average number of sales per car for the top three car manufacturers. It calculates the average sales per car by dividing the total selling price by the count of cars sold for each manufacturer. Results are ordered based on the total selling price of each manufacturer, with the top three manufacturers displayed. This analysis helps understand the sales efficiency of the top car manufacturers.

The graph used here is a pie chart, which displays the distribution of average sales per car for the top three car manufacturers. Each slice of the pie represents one manufacturer, and the size of each slice corresponds to the proportion of average sales per car attributed to that manufacturer. The accompanying table lists the top three manufacturers and their respective average sales per car.

6.6 Identify the top-selling car models with the highest monthly earnings per car

The goal of this Spark SQL query is to identify the top-selling car models with the highest monthly earnings per car. It calculates the total earnings and total cars sold for each car model, then determines the monthly earnings per car by dividing the total earnings by the total cars sold. Results are ordered based on the monthly earnings per car, with the top five car models displayed. This analysis helps identify the most profitable car models in terms of monthly earnings per car.

The graph used here is a vertical bar chart, representing the monthly earnings per car for the top-selling car models. Each bar corresponds to a specific car

model, with its height indicating the monthly earnings per car for that model. The accompanying table lists the top-selling car models along with their respective monthly earnings per car.

6.7 Identify dealerships that have experienced significant growth in sales over the past year

The goal of this Spark SQL query is to identify dealerships that have experienced significant growth in sales over the past year. It selects dealership names, counts the total sales for each dealership, and groups the data by dealership and sales year. The query filters sales data for the most recent year and the previous year. Results are ordered by dealership name and sales year to analyze sales performance over time. This analysis helps identify dealerships that have shown notable sales growth, aiding in strategic decision-making for business expansion.

The graph used here is a line chart, depicting the sales trend for dealerships over the past two years. Each line represents a dealership, with the x-axis showing the years and the y-axis representing the total sales. The accompanying table provides data for a single dealership, indicating its total sales for a specific year.

6.8 Identify car dealerships with the highest ratio of sales to inventory

The goal of this Spark SQL query is to identify car dealerships with the highest ratio of sales to inventory. It calculates the total inventory and total sales for each dealership, then computes the sales-to-inventory ratio. Results are ordered based on this ratio to determine which dealerships are most effectively converting their inventory into sales. This analysis aids in assessing the sales efficiency of different dealerships and optimizing inventory management strategies.

The graph used here is a pie chart, which visualizes the sales-to-inventory ratio for different car dealerships. Each slice of the pie represents a dealership, and the size of each slice corresponds to the proportion of sales-to-inventory ratio attributed to that dealership. The accompanying table provides details on various dealerships, including their total inventory, total sales, and the calculated sales-to-inventory ratio.

7 Conclusion

In conclusion, the "Driving Car Prices with Big Data" project aims to uncover patterns and trends in car pricing dynamics, addressing goals such as identifying influential factors in pricing, assessing market trends, and analyzing regional price variations. The outlined high-level architecture ensures a systematic approach from data acquisition to documentation, utilizing tools like Python and Git for efficient collaboration and version control. We delve into aspects such as vehicle specifications, market trends, and seller dynamics with a focus on data quality, speed, and security. This project not only sheds light on the determinants of car prices but also lays the groundwork for future research and applications in the automotive industry. Its iterative nature, involving data cleaning, processing, and comprehensive analysis, reflects a meticulous strategy for extracting valuable insights from large datasets.

8 References

Kaggle Dataset:
`https://www.kaggle.com/datasets/syedanwarafridi/vehicle-sales-data`

Github:
`https://github.com/Divyachelikani/Big-Data`

9 Results Images

```python
# 1. Identify car models with significant variations in monthly sales revenue with limit 15.
result_df = spark.sql('''
    SELECT
        model,
        AVG(sellingprice - mmr) AS average_price_variation
    FROM
        carprices
    WHERE
        sellingprice IS NOT NULL
        AND mmr IS NOT NULL
        AND (sellingprice + mmr) != 0
    GROUP BY
        model
    ORDER BY
        average_price_variation DESC
    LIMIT 15
''')

# Show the result of the SQL query
result_df.show()
```

```
+--------------+-----------------------+
|         model|average_price_variation|
+--------------+-----------------------+
|    interstate|                41500.0|
|         tahoe|                10000.0|
|        Rapide|                 8250.0|
|   CTS-V Wagon|                 8200.0|
|         siera|                 4525.0|
|       eurovan|                 3500.0|
|       equinox|                 3400.0|
|           DB9|     2883.3333333333335|
|      Marauder|                2843.75|
|ActiveHybrid 5|                 2825.0|
|ActiveHybrid 7|      2723.076923076923|
|           H3T|     2466.6666666666665|
|            H1|                 2350.0|
|      escalade|                 2300.0|
|        Savana|      2265.909090909091|
+--------------+-----------------------+
```

Figure 3: Screenshot

```python
# Extracting data from DataFrame columns
import matplotlib.pyplot as plt

models = [row['model'] for row in result_df.collect()]
price_variations = [row['average_price_variation'] for row in result_df.collect()]

# Plotting with improved readability
plt.figure(figsize=(6, 4))  # Adjust the figure size
plt.barh(models, price_variations, color='skyblue')
plt.xlabel('Average Price Variation')
plt.ylabel('Car Model')
plt.title('Top 15 Car Models with Significant Variations in Monthly Sales Revenue')
plt.xticks(fontsize=8)  # Adjust the font size of x-axis ticks
plt.yticks(fontsize=8)  # Adjust the font size of y-axis ticks
plt.tight_layout()
plt.show()
```

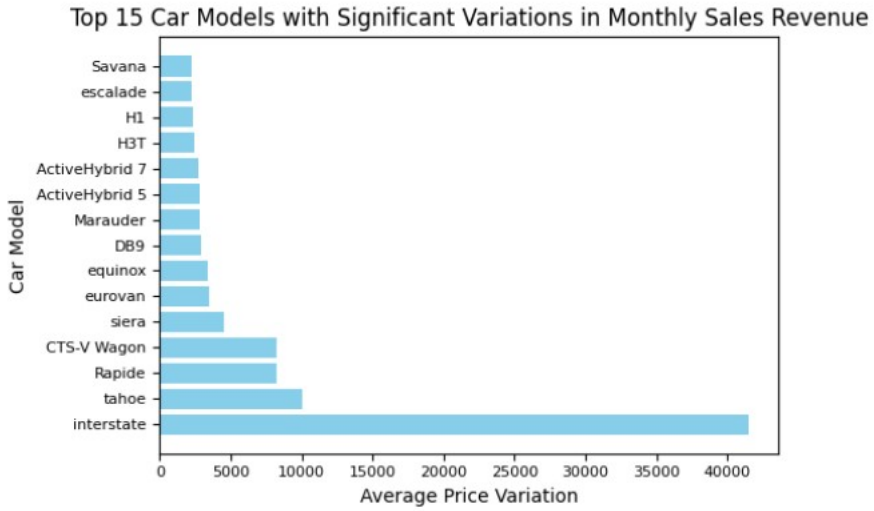

Figure 4: Top 15 Car models with significant variations in monthly sales revenue

```
# 2 The efficiency of each car in generating revenue based on its odometer reading.
result_df = spark.sql('''
     SELECT
        make,
        model,
        trim,
        sellingprice - mmr AS revenue,
        CASE WHEN odometer = 0 THEN NULL ELSE sellingprice / odometer END AS revenue_per_mile
    FROM
        carprices
    WHERE
        sellingprice IS NOT NULL
        AND mmr IS NOT NULL
        AND odometer IS NOT NULL
    ORDER BY
        revenue_per_mile DESC
            LIMIT 10

''')

# Show the result of the SQL query
result_df.show()
```

```
+-------------+----------------+-------------+--------+----------------+
|         make|           model|         trim| revenue|revenue_per_mile|
+-------------+----------------+-------------+--------+----------------+
|    Chevrolet|        Corvette|         Base|  5200.0|         37000.0|
|     Chrysler|Town and Country|    Touring-L|  8450.0|         36750.0|
|          BMW|              M5|         Base|-41000.0|         35000.0|
|        Mazda|         Tribute|            s| 26500.0|         35000.0|
|       Toyota|          Sienna|L 7-Passenger| 11000.0|         34500.0|
|          GMC|   Sierra 2500HD|          SLE| -9400.0|         30500.0|
|         Ford|            Edge|          SEL|  7200.0|         30000.0|
|Mercedes-Benz|        GL-Class| GL450 4MATIC|-22300.0|         29200.0|
|     Infiniti|         G Sedan|  G37 Journey| -1200.0|         23800.0|
|Mercedes-Benz|        GL-Class|        GL550|  1600.0|         23500.0|
+-------------+----------------+-------------+--------+----------------+
```

Figure 5: Screenshot

```
import matplotlib.pyplot as plt

# Extracting data from DataFrame columns
models = [row['model'] for row in result_df.collect()]
revenue_per_mile = [row['revenue_per_mile'] for row in result_df.collect()]

# Plotting as a scatter plot
plt.figure(figsize=(6, 8))
plt.scatter(models, revenue_per_mile, color='blue')
plt.xlabel('Car Model')
plt.ylabel('Revenue per Mile')
plt.title('Efficiency of Each Car in Generating Revenue Based on Odometer Reading')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

Figure 6: Screenshot

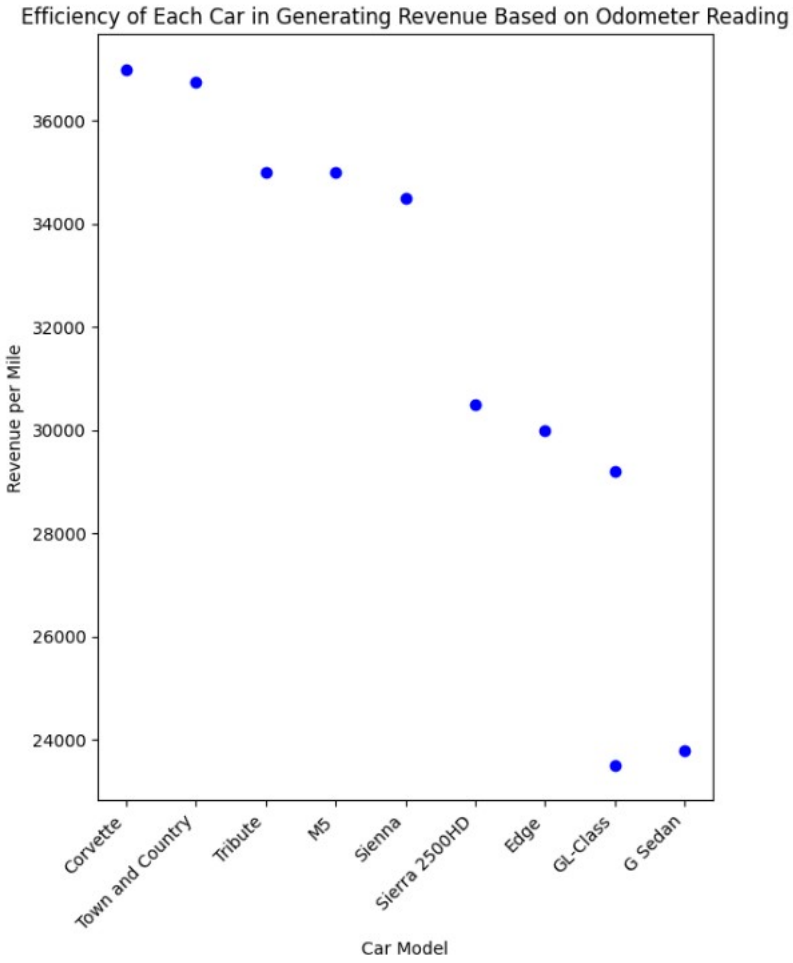

Figure 7: Efficiency of each car in generating revenue based on odometer reading

```
#3. Identify car models with high engagement relative to their sales.
result_df = spark.sql('''
     SELECT
        make,
        model,
        trim,
        CASE WHEN odometer = 0 THEN NULL ELSE sellingprice / odometer END AS revenue_per_mile
    FROM
        carprices
    WHERE
        sellingprice IS NOT NULL
        AND mmr IS NOT NULL
        AND odometer IS NOT NULL
    ORDER BY
        revenue_per_mile DESC
        limit 15
''')

# Show the result of the SQL query
result_df.show()
```

```
+-------------+----------------+-------------+----------------+
|         make|           model|         trim|revenue_per_mile|
+-------------+----------------+-------------+----------------+
|    Chevrolet|        Corvette|         Base|         37000.0|
|     Chrysler|Town and Country|    Touring-L|         36750.0|
|        Mazda|         Tribute|            s|         35000.0|
|          BMW|              M5|         Base|         35000.0|
|       Toyota|          Sienna|L 7-Passenger|         34500.0|
|          GMC|   Sierra 2500HD|          SLE|         30500.0|
|         Ford|            Edge|          SEL|         30000.0|
|Mercedes-Benz|        GL-Class| GL450 4MATIC|         29200.0|
|     Infiniti|         G Sedan|  G37 Journey|         23800.0|
|Mercedes-Benz|        GL-Class|        GL550|         23500.0|
|       Nissan|          Altima|        2.5 S|         23200.0|
|          GMC|        Yukon XL|     SLT 1500|         23000.0|
|         Ford|            Edge|          SEL|         21300.0|
|        Dodge| Ram Pickup 2500|          SLT|         21200.0|
|       Nissan|           Versa|       1.8 SL|         20500.0|
+-------------+----------------+-------------+----------------+
```

Figure 8: Screenshot

```python
import matplotlib.pyplot as plt

# Convert the result DataFrame to Pandas for easy plotting
result_pd = result_df.toPandas()

# Drop rows with NaN values in 'revenue_per_mile'
result_pd = result_pd.dropna(subset=['revenue_per_mile'])

# Plotting as a bar chart
plt.figure(figsize=(10, 6))
plt.hist(result_pd['revenue_per_mile'], bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Revenue per Mile')
plt.ylabel('Frequency')
plt.title('Distribution of Revenue per Mile for Different Cars')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

Figure 9: Screenshot

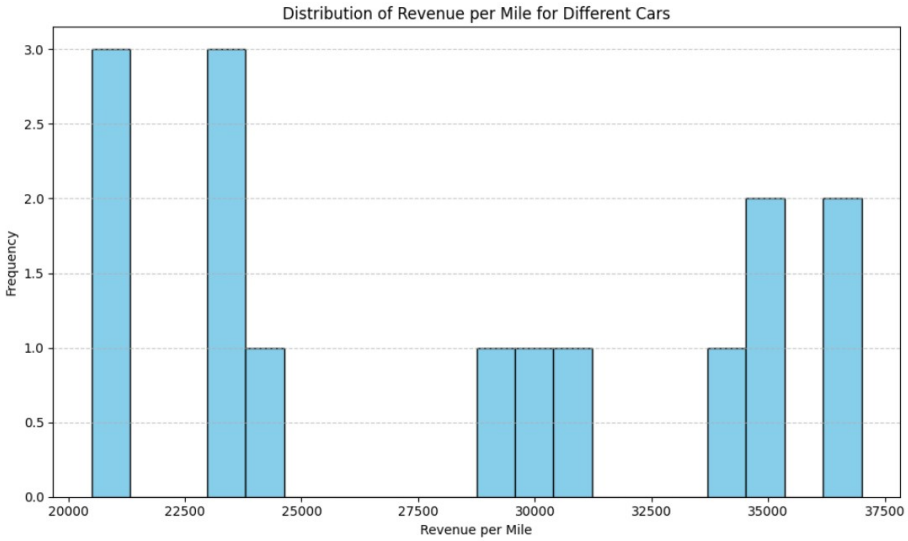

Figure 10: Distribution of revenue per mile for different cars

```
# 4. The performance of different car categories based on their total sales in a year
result_df = spark.sql('''
     SELECT
    year,
    model,
    SUM(sellingprice) AS total_sales
FROM
    carprices
GROUP BY
    year,
    model
ORDER BY
    year DESC, total_sales DESC

    limit =10;

''')

# Show the result of the SQL query
result_df.show()
```

```
+----+----------------+-----------+
|year|           model|total_sales|
+----+----------------+-----------+
|2015|         Sorento| 1.903775E7|
|2015|           Tahoe| 1.286915E7|
|2015|          Altima|  9768750.0|
|2015|        Suburban|  9752750.0|
|2015|        Explorer|  8800800.0|
|2015|             200|  6959350.0|
|2015|Silverado 2500HD|  6066750.0|
|2015|        3 Series|  5720250.0|
|2015|F-250 Super Duty|  5473350.0|
|2015|            CX-5|  5045250.0|
+----+----------------+-----------+
```

Figure 11: Screenshot

```python
import matplotlib.pyplot as plt

# Convert the result DataFrame to Pandas for easy plotting
result_pd = result_df.toPandas()

# Group the data by car category and calculate the total sales
total_sales_per_category = result_pd.groupby('model')['total_sales'].sum()

# Sorting the categories by total sales
total_sales_per_category = total_sales_per_category.sort_values(ascending=False)

# Plotting as a horizontal bar chart
plt.figure(figsize=(10, 8))
total_sales_per_category.plot(kind='barh', color='skyblue')
plt.xlabel('Total Sales')
plt.ylabel('Car Category')
plt.title('Total Sales of Different Car Categories')
plt.tight_layout()
plt.show()
```

Figure 12: Screenshot

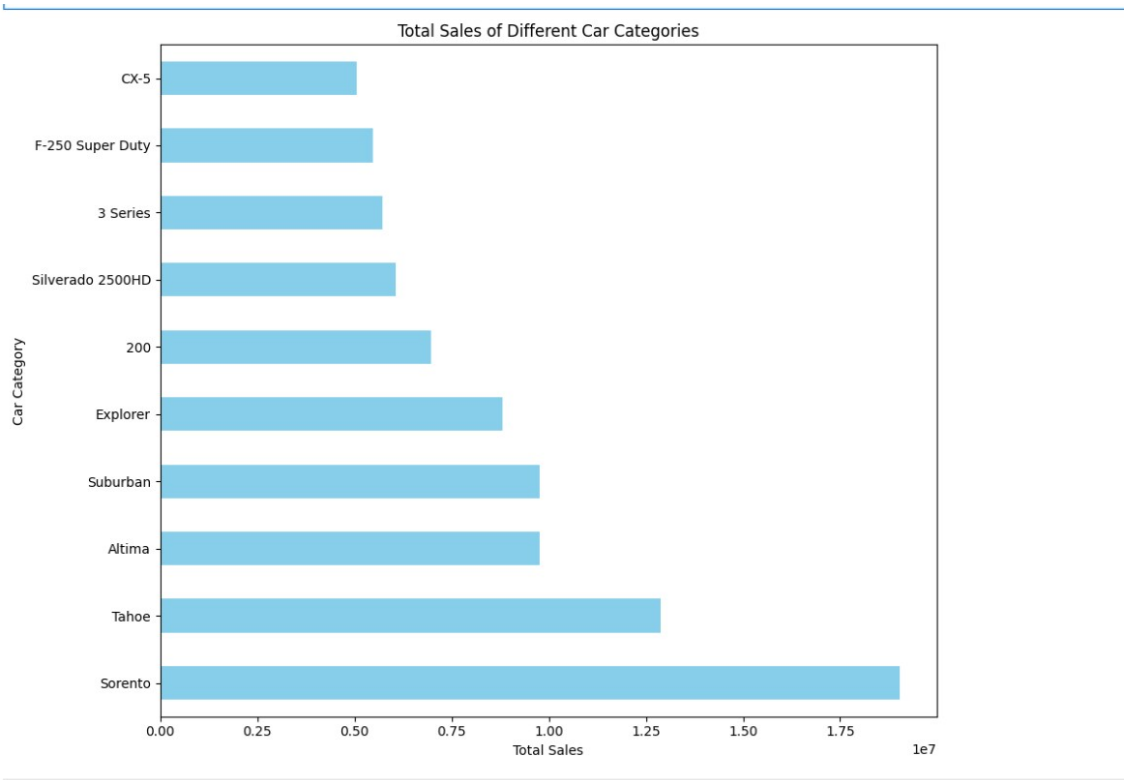

Figure 13: Screenshot

```
# 5 Calculate the average number of sales per car for the top three car manufacturers.
result_df = spark.sql('''
SELECT make AS manufacturer,
       SUM(sellingprice) / COUNT(*) AS average_sales_per_car
FROM carprices
GROUP BY make
ORDER BY SUM(sellingprice) DESC
LIMIT 3
''')

# Show the result of the SQL query
result_df.show()
```

```
+------------+--------------------+
|manufacturer|average_sales_per_car|
+------------+--------------------+
|        Ford|   13987.392903567474|
|   Chevrolet|   11977.397528116018|
|      Nissan|    11739.01596040485|
+------------+--------------------+
```

Figure 14: Screenshot

```
import matplotlib.pyplot as plt

# Convert the result DataFrame to Pandas for easy plotting
result_pd = result_df.toPandas()

# Sorting the data by average sales per car for consistency in plotting
result_pd.sort_values(by='average_sales_per_car', ascending=False, inplace=True)

# Selecting the top three manufacturers
top_three_manufacturers = result_pd.head(3)

# Plotting as a pie chart
plt.figure(figsize=(6, 6))
plt.pie(top_three_manufacturers['average_sales_per_car'], labels=top_three_manufacturers['manufacturer'], autopct='%1.1f%%', startangle=140)
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle
plt.title('Average Sales per Car for Top Three Car Manufacturers')
plt.tight_layout()
plt.show()
```

Figure 15: Screenshot

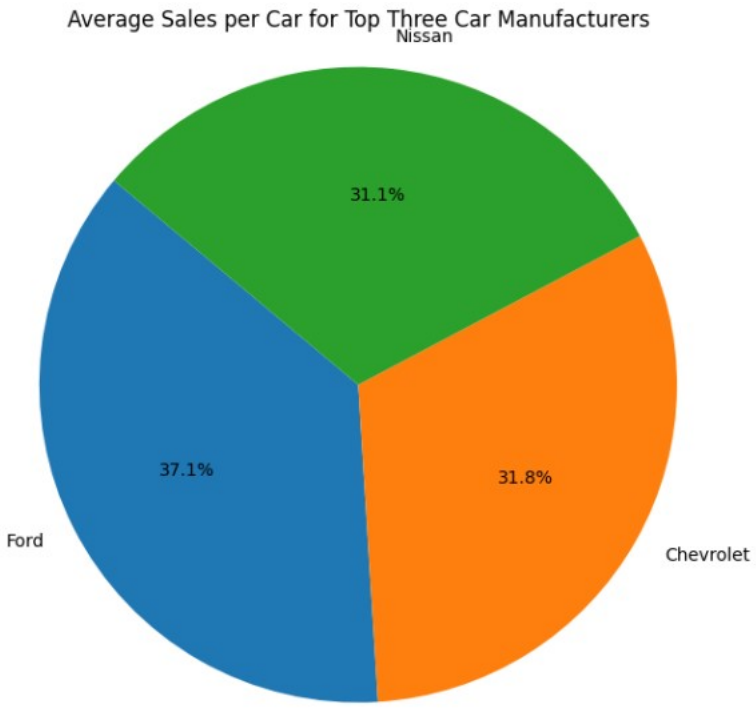

Figure 16: Screenshot

```
# 6 Identify the top-selling car models with the highest monthly earnings per car.
result_df = spark.sql('''
SELECT
    year, make, model,
    SUM(sellingprice) AS total_earnings,
    (SUM(sellingprice) / COUNT(*)) AS monthly_earnings_per_car
FROM carprices
GROUP BY year, make, model
ORDER BY monthly_earnings_per_car DESC
LIMIT 5
''')

# Show the result of the SQL query
result_df.show()
```

```
+----+-------------+----------+-------------+-----------------------+
|year|         make|     model|total_earnings|monthly_earnings_per_car|
+----+-------------+----------+-------------+-----------------------+
|2011|      Ferrari|458 Italia|      183000.0|               183000.0|
|2013|  Rolls-Royce|     Ghost|      171500.0|               171500.0|
|2012|  Rolls-Royce|     Ghost|      505500.0|               168500.0|
|2013|Mercedes-Benz|SLS AMG GT|      156500.0|               156500.0|
|2014|          BMW|        i8|     1388000.0|      154222.22222222222|
+----+-------------+----------+-------------+-----------------------+
```

Figure 17: Screenshot

```
:
import matplotlib.pyplot as plt

# Convert the result DataFrame to Pandas for easy plotting
result_pd = result_df.toPandas()

# Plotting as a vertical bar chart
plt.figure(figsize=(6, 4))
plt.bar(result_pd['make'] + ' ' + result_pd['model'], result_pd['monthly_earnings_per_car'], color='skyblue')
plt.xlabel('Car Model')
plt.ylabel('Monthly Earnings per Car')
plt.title('Top-Selling Car Models with Highest Monthly Earnings per Car')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better readability
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

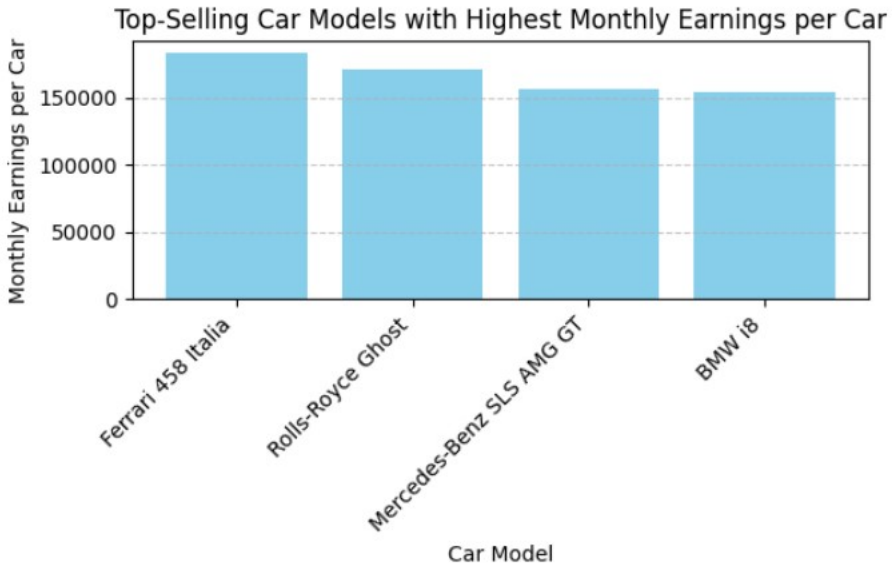

Figure 18: Screenshot

```
# 7. Identify dealerships that have experienced significant growth in sales over the past year
result_df = spark.sql('''
SELECT
    seller AS dealership,
    make,
    COUNT(*) AS total_sales,
    YEAR(saledate) AS sales_year
FROM carprices
WHERE YEAR(saledate) = (SELECT MAX(YEAR(saledate)) FROM carprices)
   OR YEAR(saledate) = (SELECT MAX(YEAR(saledate)) - 1 FROM carprices)
GROUP BY seller, make, YEAR(saledate)
ORDER BY dealership, sales_year DESC;
''')

# Show the result of the SQL query
result_df.show()
```

```
+----------+----------+-----------+----------+
|dealership|      make|total_sales|sales_year|
+----------+----------+-----------+----------+
|      gray|Volkswagen|          1|     16500|
+----------+----------+-----------+----------+
```

Figure 19: Screenshot

```python
import matplotlib.pyplot as plt

# Convert the result DataFrame to Pandas for easy plotting
result_pd = result_df.toPandas()

# Filter out dealerships with missing data for either year (to focus on those with sales in both years)
filtered_result = result_pd.groupby(['dealership', 'sales_year']).agg({'total_sales': 'sum'}).unstack()

# Drop any missing values
filtered_result = filtered_result.dropna()

# Convert sales years to string to avoid data type issues
filtered_result.columns = filtered_result.columns.map(lambda x: str(x))

# Plotting as a line chart
plt.figure(figsize=(8, 6))

for dealership in filtered_result.index:
    plt.plot(filtered_result.columns, filtered_result.loc[dealership], marker='o', label=dealership)

plt.xlabel('Year')
plt.ylabel('Total Sales')
plt.title('Sales Trend for Dealerships Over the Past Two Years')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.legend(title='Dealership', loc='upper left', bbox_to_anchor=(1, 1))
plt.tight_layout()
plt.show()
```

Figure 20: Screenshot

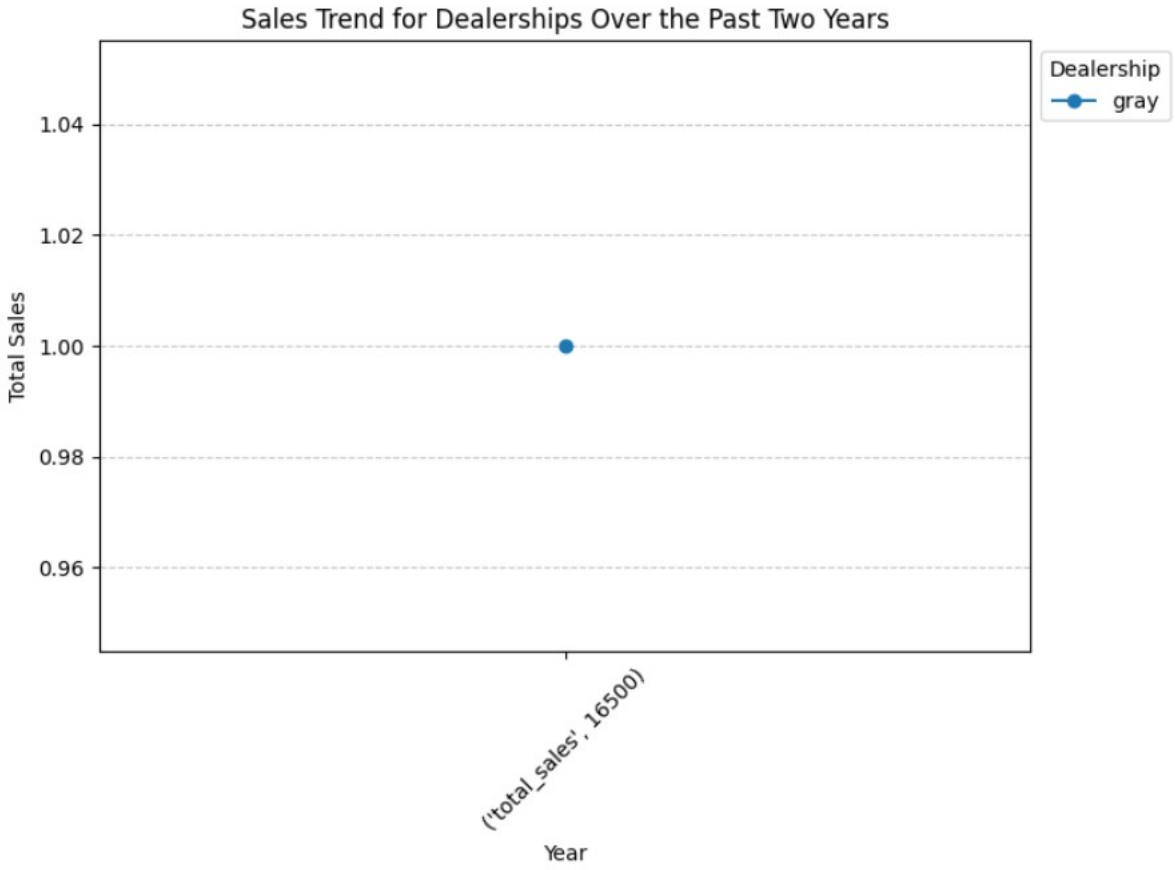

Figure 21: Screenshot

```
# 8. Identify car dealerships with the highest ratio of sales to inventory.
result_df = spark.sql('''
SELECT
    seller AS dealership,
    COUNT(*) AS total_inventory,
    COUNT(sellingprice) AS total_sales,
    ROUND(COUNT(sellingprice) * 1.0 / COUNT(*), 2) AS sales_to_inventory_ratio
FROM
    carprices
GROUP BY
    seller
ORDER BY
    sales_to_inventory_ratio DESC

    limit 15
''')

# Show the result of the SQL query
result_df.show()
```

```
+--------------------+---------------+-----------+------------------------+
|          dealership|total_inventory|total_sales|sales_to_inventory_ratio|
+--------------------+---------------+-----------+------------------------+
|balboa thrift & l...|             61|         61|                    1.00|
|california auto w...|            129|        129|                    1.00|
|repo remarketing/...|              6|          6|                    1.00|
|jaguar land rover...|             20|         20|                    1.00|
|   low gos used cars|             24|         24|                    1.00|
|montclair auto sl...|             59|         59|                    1.00|
|     pa distributors|             18|         18|                    1.00|
|autolenders liqui...|            388|        388|                    1.00|
|   bailey auto plaza|              6|          6|                    1.00|
|southern auto fin...|            157|        157|                    1.00|
|      rock chevrolet|              9|          9|                    1.00|
|premier toyota of...|             66|         66|                    1.00|
|        richmond bmw|             10|         10|                    1.00|
|jaguar land rover...|              4|          4|                    1.00|
|   loeber motors inc|              4|          4|                    1.00|
+--------------------+---------------+-----------+------------------------+
```

Figure 22: Screenshot

```python
import matplotlib.pyplot as plt

# Convert the result DataFrame to Pandas for easy plotting
result_pd = result_df.toPandas()

# Plotting as a pie chart
plt.figure(figsize=(8, 8))
plt.pie(result_pd['sales_to_inventory_ratio'], labels=result_pd['dealership'], autopct='%1.1f%%', startangle=140)
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle
plt.title('Sales to Inventory Ratio for Car Dealerships')
plt.tight_layout()
plt.show()
```

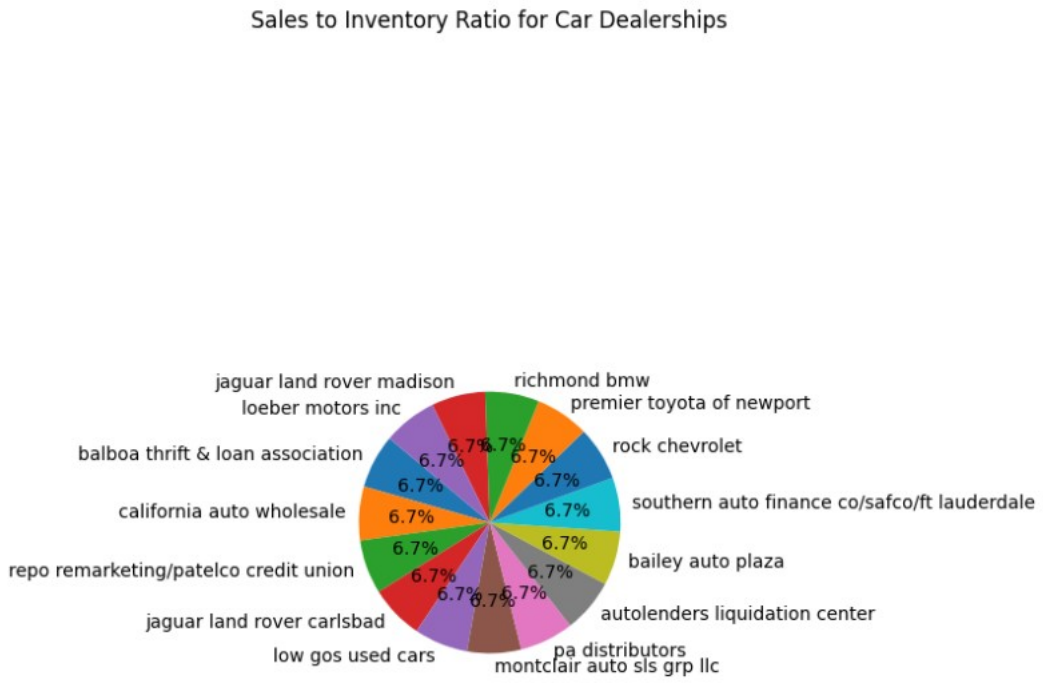

Figure 23: Screenshot