EXPLORATORY DATA ANALYSIS ON CAR SELLING PRICE

PROJECT REPORT - BDM 1043 - Big Data Fundamentals

Submitted by:

JOBINA JOY – C0924759

ANAND CHANTHANANICKAL SAJEEVAN – C0928396

DEVI UNNI – C0928346

VISHNU CHANDRASEKHARAN: C0924468

JOEL GEORGE: C0927062

NICY PRINCE: C0924777

Submitted to

Prof. Quang Duong

Lambton College Mississauga Canada

Date: April 12, 2024

Table of Contents

INTRODUCTION

The "Vehicle Sales and Market Trends Dataset" presents a rich repository of information encompassing diverse aspects of vehicle sales, including manufacturer details, model specifications, transaction insights, market trends analysis, and condition-related data. This report encapsulates a comprehensive exploration of the dataset, emphasizing data cleaning, preprocessing, analysis, and visualization to derive meaningful insights into car selling prices and market dynamics.

The objective of this analysis is to uncover patterns, trends, and correlations within the dataset that can inform stakeholders in the automotive industry, market analysts, and car enthusiasts about market behaviors, popular car models, pricing variations, and geographical sales trends. Leveraging tools like pandas for data manipulation, DuckDB for SQL querying, and Matplotlib for visualization, this report aims to provide actionable insights that drive informed decision-making and facilitate further exploration of the dataset.

DATASET & SOURCE OF DATASET:

Link: https://www.kaggle.com/datasets/syedanwarafridi/vehicle-sales-data

The "Vehicle Sales and Market Trends Dataset" offers an extensive compilation of data concerning the sales of diverse vehicles. This dataset includes information such as the year, manufacturer, model, version, body style, transmission type, Vehicle Identification Number (VIN), registration state, condition rating, mileage, exterior and interior colors, seller details, Manheim Market Report (MMR) values, sale prices, and dates of sale.

DATASET KEY FEATURES:

Vehicle Details: This section offers specific information regarding each vehicle, covering its manufacturer, model, trim, and year of production.

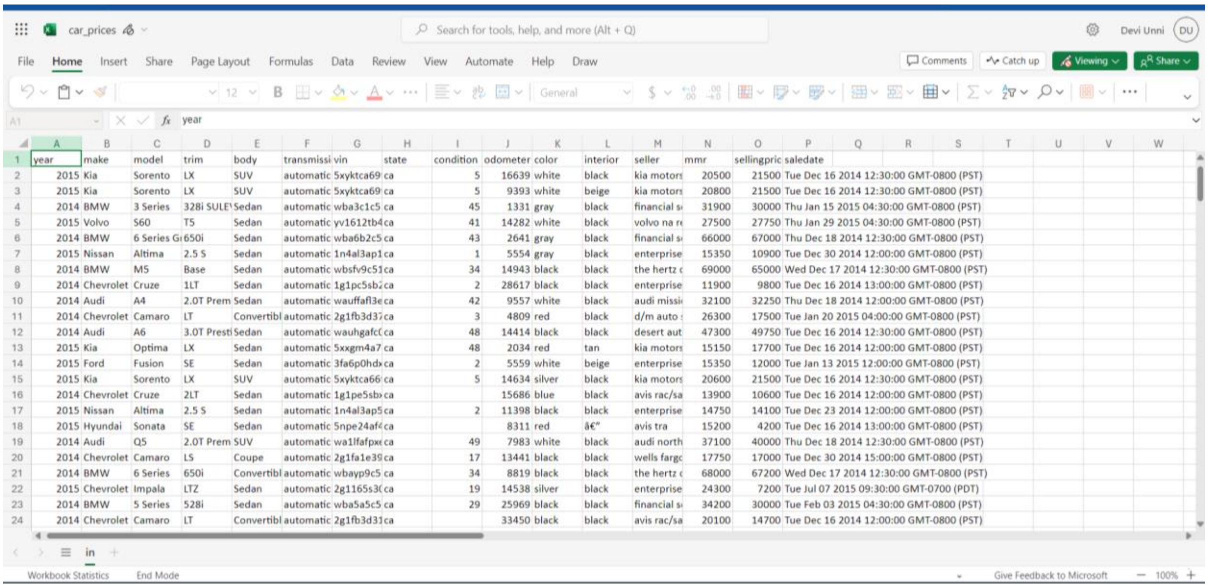
Transaction Insights: Here, you'll find details on sales transactions, including sale dates and selling prices.

Market Trends Analysis: Through Manheim Market Report (MMR) values, this dataset facilitates the estimation of each vehicle's market worth, aiding in the examination of market trends and fluctuations.

Condition and Mileage Data: Included are records on vehicle conditions and odometer readings, enabling the analysis of how these factors impact selling prices.

Dataset Format: Typically presented in a tabular format like CSV, with rows denoting individual vehicle sales transactions and columns representing various attributes associated with each transaction.

The "Vehicle Sales and Market Trends Dataset" offers a comprehensive repository of information related to the sales transactions of diverse vehicles. This dataset covers a wide array of details such as manufacturing year, make, model, trim, body style, transmission type, Vehicle Identification Number (VIN), registration state, condition rating, odometer reading, exterior and interior colors, seller details, Manheim Market Report (MMR) values, selling prices, and sale dates.



	year	make	model	trim	body	transmissi	vin	state	condition	odometer	color	interior	seller	mmr	sellingpric	saledate
2	2015	Kia	Sorento	LX	SUV	automatic	5xyktca69	ca	5	16639	white	black	kia motors	20500	21500	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
3	2015	Kia	Sorento	LX	SUV	automatic	5xyktca69	ca	5	9393	white	beige	kia motors	20800	21500	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
4	2014	BMW	3 Series	328i SULE	Sedan	automatic	wba3c1c5	ca	45	1331	gray	black	financial s	31900	30000	Thu Jan 15 2015 04:30:00 GMT-0800 (PST)
5	2015	Volvo	S60	T5	Sedan	automatic	yv1612tb4	ca	41	14282	white	black	volvo na r	27500	27750	Thu Jan 29 2015 04:30:00 GMT-0800 (PST)
6	2014	BMW	6 Series G	650i	Sedan	automatic	wba6b2c5	ca	43	2641	gray	black	financial s	66000	67000	Thu Dec 18 2014 12:30:00 GMT-0800 (PST)
7	2015	Nissan	Altima	2.5 S	Sedan	automatic	1n4al3ap1	ca	1	5554	gray	black	enterprise	15350	10900	Tue Dec 30 2014 12:00:00 GMT-0800 (PST)
8	2014	BMW	M5	Base	Sedan	automatic	wbsfv9c51	ca	34	14943	black	black	the hertz c	69000	65000	Wed Dec 17 2014 12:30:00 GMT-0800 (PST)
9	2014	Chevrolet	Cruze	1LT	Sedan	automatic	1g1pc5sb2	ca	2	28617	black	black	enterprise	11900	9800	Tue Dec 16 2014 13:00:00 GMT-0800 (PST)
10	2014	Audi	A4	2.0T Prem	Sedan	automatic	wauffafl3e	ca	42	9557	white	black	audi missi	32100	32250	Thu Dec 18 2014 12:00:00 GMT-0800 (PST)
11	2014	Chevrolet	Camaro	LT	Convertibl	automatic	2g1fb3d37	ca	3	4809	red	black	d/m auto	26300	17500	Tue Jan 20 2015 04:00:00 GMT-0800 (PST)
12	2014	Audi	A6	3.0T Presti	Sedan	automatic	wauhgafc(	ca	48	14414	black	black	desert aut	47300	49750	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
13	2015	Kia	Optima	LX	Sedan	automatic	5xxgm4a7	ca	48	2034	red	tan	kia motors	15150	17700	Tue Dec 16 2014 12:00:00 GMT-0800 (PST)
14	2015	Ford	Fusion	SE	Sedan	automatic	3fa6p0hd	ca	2	5559	white	beige	enterprise	15350	12000	Tue Jan 13 2015 12:00:00 GMT-0800 (PST)
15	2015	Kia	Sorento	LX	SUV	automatic	5xyktca66	ca	5	14634	silver	black	kia motors	20600	21500	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
16	2014	Chevrolet	Cruze	2LT	Sedan	automatic	1g1pe5sb	ca		15686	blue	black	avis rac/sa	13900	10600	Tue Dec 16 2014 12:00:00 GMT-0800 (PST)
17	2015	Nissan	Altima	2.5 S	Sedan	automatic	1n4al3ap5	ca	2	11398	black	black	enterprise	14750	14100	Tue Dec 23 2014 12:00:00 GMT-0800 (PST)
18	2015	Hyundai	Sonata	SE	Sedan	automatic	5npe24af4	ca		8311	red	â€"	avis tra	15200	4200	Tue Dec 16 2014 13:00:00 GMT-0800 (PST)
19	2014	Audi	Q5	2.0T Prem	SUV	automatic	wa1lfafpx	ca	49	7983	white	black	audi north	37100	40000	Thu Dec 18 2014 12:30:00 GMT-0800 (PST)
20	2014	Chevrolet	Camaro	LS	Coupe	automatic	2g1fa1e39	ca	17	13441	black	black	wells fargo	17750	17000	Tue Dec 30 2014 15:00:00 GMT-0800 (PST)
21	2014	BMW	6 Series	650i	Convertibl	automatic	wbayp9c5	ca	34	8819	black	black	the hertz c	68000	67200	Wed Dec 17 2014 12:30:00 GMT-0800 (PST)
22	2015	Chevrolet	Impala	LTZ	Sedan	automatic	2g1165s3(	ca	19	14538	silver	black	enterprise	24300	7200	Tue Jul 07 2015 09:30:00 GMT-0700 (PDT)
23	2014	BMW	5 Series	528i	Sedan	automatic	wba5a5c5	ca	29	25969	black	black	financial s	34200	30000	Tue Feb 03 2015 04:30:00 GMT-0800 (PST)
24	2014	Chevrolet	Camaro	LT	Convertibl	automatic	2g1fb3d31	ca		33450	black	black	avis rac/sa	20100	14700	Tue Dec 16 2014 12:00:00 GMT-0800 (PST)

DATA CLEANING & PREPROCESSING:

The dataset was obtained from a CSV file and required cleaning due to inconsistencies, missing values, and special characters in certain columns. The cleaning process involved using the Pandas library in Python to manipulate and transform the data.

DATA IMPORT:

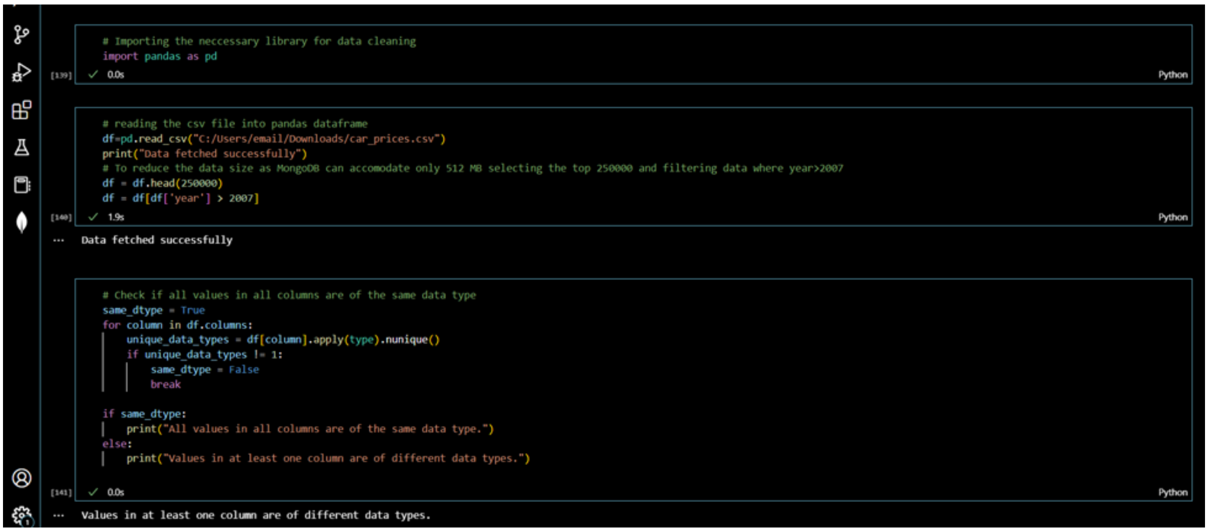
The Pandas library was imported to facilitate data manipulation. The CSV file containing the car prices data was read into a Pandas DataFrame.

DATA REDUCTION:

The dataset was reduced to the top 250,000 rows to fetch the relevant data and filtered to include only records with a year greater than 2007.

DATA TYPE VALIDATION:

A validation check was performed to ensure that all values in all columns were of the same data type. If inconsistencies were found, appropriate actions were taken.



```python
# Importing the neccessary library for data cleaning
import pandas as pd
```

```python
# reading the csv file into pandas dataframe
df=pd.read_csv("C:/Users/email/Downloads/car_prices.csv")
print("Data fetched successfully")
# To reduce the data size as MongoDB can accomodate only 512 MB selecting the top 250000 and filtering data where year>2007
df = df.head(250000)
df = df[df['year'] > 2007]
```

```
Data fetched successfully
```

```python
# Check if all values in all columns are of the same data type
same_dtype = True
for column in df.columns:
    unique_data_types = df[column].apply(type).nunique()
    if unique_data_types != 1:
        same_dtype = False
        break

if same_dtype:
    print("All values in all columns are of the same data type.")
else:
    print("Values in at least one column are of different data types.")
```

```
Values in at least one column are of different data types.
```

DATA CLEANING:

Several cleaning steps were executed:

The column name 'condition' was renamed to 'rating' for clarity.

Missing values in certain columns were filled with appropriate values such as 'NA' or the mode.

```python
# Checking for null values
print("Null values count before cleaning")
print(df.isna().sum())

# Fill missing values with 'NA' category
df['make'] = df['make'].fillna('NA')
df['model'] = df['model'].fillna('NA')
df['trim'] = df['trim'].fillna('NA')
df['color'] = df['color'].fillna('NA')
df['interior'] = df['interior'].fillna('NA')

# Fill missing values with mode or'0' --- mode means most frequently appearing value
df['body'] = df['body'].fillna(df['body'].mode()[0])
df['transmission'] = df['transmission'].fillna(df['transmission'].mode()[0])
df['condition'] = df['condition'].fillna(df['condition'].mode()[0])
df['odometer'] = df['odometer'].fillna(0)

# Convert the sellingprice column to integers
df['sellingprice'] = df['sellingprice'].round(0).astype(int)

# Remove null values
df.dropna(subset = ['vin'], inplace=True)
df.dropna(subset = ['saledate'],inplace=True)

print("Null values is replaced and droppped")

# Display the DataFrame to verify the changes
print("Null values count after cleaning")
print(df.isna().sum())
```
[143] ✓ 0.4s Python

Regular expression pattern is creating to match with special characters

```python
# Define a regular expression pattern to match special characters
pattern = r'[^a-zA-Z0-9\s]'  # Matches any character that is not a letter, digit, or whitespace

# Identify columns containing special characters
columns_with_special_chars = []
for column in df.columns:
    if any(df[column].astype(str).str.contains(pattern)):
        columns_with_special_chars.append(column)

# Print the columns containing special characters
print("Columns containing special characters:", columns_with_special_chars)
```
✓ 1.6s Python

```
Columns containing special characters: ['make', 'model', 'trim', 'body', 'color', 'interior', 'seller', 'saledate']
```

Special characters in specific columns ('color' and 'interior') were replaced with 'NA'.

```python
# Here we are cleaning the 'color' and 'interior' columns as special charcters are not acceptable for both.
# Create a boolean mask to identify rows containing special characters in the 'color' and 'interior' columns and replacing with 'NA'
# Define the regular expression pattern to match special characters
pattern = r'[^a-zA-Z0-9\s]'  # Matches any character that is not a letter, digit, or whitespace
mask = df['color'].str.contains(pattern)
df.loc[mask, 'color'] = 'NA'

mask = df['color'].str.contains(pattern)
df.loc[mask, 'color'] = 'NA'

print("Replaced special characters through boolean masking" )
```
✓ 0.1s Python

```
Replaced special characters through boolean masking
```

Null values before cleaning & after cleaning

```
Null values count before cleaning
year               0
make            1042
model           1084
trim            1065
body            1663
transmission   20931
vin                0
state              0
condition       3648
odometer           9
color            287
interior         287
seller             0
mmr                0
sellingprice       0
saledate           0
dtype: int64
Null values is replaced and droppped
```

```
Null values count after cleaning
year            0
make            0
model           0
trim            0
body            0
transmission    0
vin             0
state           0
condition       0
odometer        0
color           0
interior        0
seller          0
mmr             0
sellingprice    0
saledate        0
dtype: int64
```

Renaming the column condition to rating

```
# Rename column 'condition' to 'rating'
df.rename(columns={'condition': 'rating'}, inplace=True)
print("Renamed the column conditon to rating for better understanding")
[146]  ✓ 0.0s
...  Renamed the column conditon to rating for better understanding
```

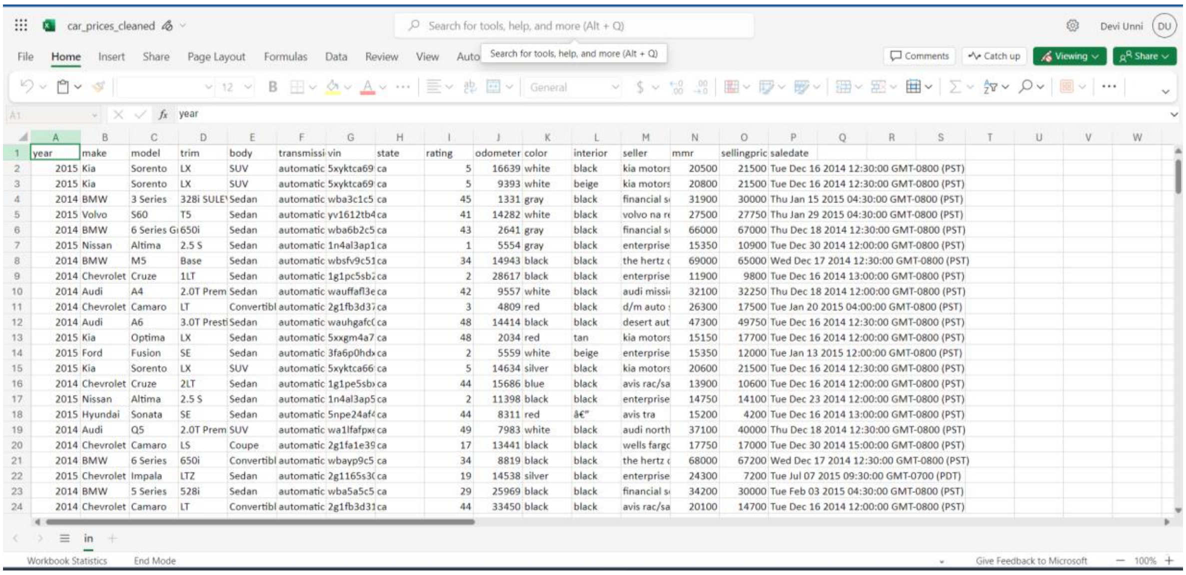
DATA EXPORT:

The cleaned dataset was saved to a new CSV file for further analysis or integration into other systems.

```
# Saving the result back to a csv file
df.to_csv("D:/PROJECTS/NOSQL/PandasOutputAfterDataCleaning/car_prices_cleaned.csv",index=False)
print("Successfullly saved cleaned data to D:/PROJECTS/NOSQL/PandasOutputAfterDataCleaning/car_prices_cleaned.csv")
✓ 1.3s
Successfullly saved cleaned data to D:/PROJECTS/NOSQL/PandasOutputAfterDataCleaning/car_prices_cleaned.csv
```

The data cleaning process successfully addressed various inconsistencies and missing values in the car prices dataset, ensuring its readiness for further analysis or use in other applications.

Data After Cleaning and preprocessing:



year	make	model	trim	body	transmissi	vin	state	rating	odometer	color	interior	seller	mmr	sellingpric	saledate
2015	Kia	Sorento	LX	SUV	automatic	5xyktca69	ca	5	16639	white	black	kia motors	20500	21500	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
2015	Kia	Sorento	LX	SUV	automatic	5xyktca69	ca	5	9393	white	beige	kia motors	20800	21500	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
2014	BMW	3 Series	328i SULE	Sedan	automatic	wba3c1c5	ca	45	1331	gray	black	financial s	31900	30000	Thu Jan 15 2015 04:30:00 GMT-0800 (PST)
2015	Volvo	S60	T5	Sedan	automatic	yv1612tb4	ca	41	14282	white	black	volvo na re	27500	27750	Thu Jan 29 2015 04:30:00 GMT-0800 (PST)
2014	BMW	6 Series G	650i	Sedan	automatic	wba6b2c5	ca	43	2641	gray	black	financial s	66000	67000	Thu Dec 18 2014 12:30:00 GMT-0800 (PST)
2015	Nissan	Altima	2.5 S	Sedan	automatic	1n4al3ap1	ca	1	5554	gray	black	enterprise	15350	10900	Tue Dec 30 2014 12:00:00 GMT-0800 (PST)
2014	BMW	M5	Base	Sedan	automatic	wbsfv9c51	ca	34	14943	black	black	the hertz	69000	65000	Wed Dec 17 2014 12:30:00 GMT-0800 (PST)
2014	Chevrolet	Cruze	1LT	Sedan	automatic	1g1pc5sb	ca	2	28617	black	black	enterprise	11900	9800	Tue Dec 16 2014 13:00:00 GMT-0800 (PST)
2014	Audi	A4	2.0T Prem	Sedan	automatic	wauffafl3e	ca	42	9557	white	black	audi missi	32100	32250	Thu Dec 18 2014 12:00:00 GMT-0800 (PST)
2014	Chevrolet	Camaro	LT	Convertibl	automatic	2g1fb3d37	ca	3	4809	red	black	d/m auto	26300	17500	Tue Jan 20 2015 04:00:00 GMT-0800 (PST)
2014	Audi	A6	3.0T Presti	Sedan	automatic	wauhgafc	ca	48	14414	black	black	desert aut	47300	49750	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
2015	Kia	Optima	LX	Sedan	automatic	5xxgm4a7	ca	48	2034	red	tan	kia motors	15150	17700	Tue Dec 16 2014 12:00:00 GMT-0800 (PST)
2015	Ford	Fusion	SE	Sedan	automatic	3fa6p0hd	ca	2	5559	white	beige	enterprise	15350	12000	Tue Jan 13 2015 12:00:00 GMT-0800 (PST)
2015	Kia	Sorento	LX	SUV	automatic	5xyktca66	ca	5	14634	silver	black	kia motors	20600	21500	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
2014	Chevrolet	Cruze	2LT	Sedan	automatic	1g1pe5sb	ca	44	15686	blue	black	avis rac/sa	13900	10600	Tue Dec 16 2014 12:00:00 GMT-0800 (PST)
2015	Nissan	Altima	2.5 S	Sedan	automatic	1n4al3ap5	ca	2	11398	black	black	enterprise	14750	14100	Tue Dec 23 2014 12:00:00 GMT-0800 (PST)
2015	Hyundai	Sonata	SE	Sedan	automatic	5npe24af	ca	44	8311	red	â€"	avis tra	15200	4200	Tue Dec 16 2014 13:00:00 GMT-0800 (PST)
2014	Audi	Q5	2.0T Prem	SUV	automatic	wa1lfafpx	ca	49	7983	white	black	audi north	37100	40000	Thu Dec 18 2014 12:30:00 GMT-0800 (PST)
2014	Chevrolet	Camaro	LS	Coupe	automatic	2g1fa1e39	ca	17	13441	black	black	wells farg	17750	17000	Tue Dec 30 2014 15:00:00 GMT-0800 (PST)
2014	BMW	6 Series	650i	Convertibl	automatic	wbayp9c5	ca	34	8819	black	black	the hertz	68000	67200	Wed Dec 17 2014 12:30:00 GMT-0800 (PST)
2015	Chevrolet	Impala	LTZ	Sedan	automatic	2g1165s3	ca	19	14538	silver	black	enterprise	24300	7200	Tue Jul 07 2015 09:30:00 GMT-0700 (PDT)
2014	BMW	5 Series	528i	Sedan	automatic	wba5a5c5	ca	29	25969	black	black	financial s	34200	30000	Tue Feb 03 2015 04:30:00 GMT-0800 (PST)
2014	Chevrolet	Camaro	LT	Convertibl	automatic	2g1fb3d31	ca	44	33450	black	black	avis rac/sa	20100	14700	Tue Dec 16 2014 12:00:00 GMT-0800 (PST)

DATA ANALYSIS & VISUALIZATION

Importing necessary libraries for data analysis and visualization.

DuckDB: Used for database operations and querying data.

pandas: Used for data manipulation, analysis, and integration with database results.

matplotlib.pyplot: Used for data visualization, providing tools to create various types of plots (e.g., line plots, bar charts, histograms).

```
import duckdb as duckdb1
import pandas as pd
import matplotlib.pyplot as plt
```

Integrating DuckDB (a SQL database) with pandas (for data handling) to facilitate data table creation and retrieval, bridging SQL database operations with Python data analysis capabilities

```
duckdb1.sql('CREATE TABLE carsales AS SELECT * FROM "C:/Users/vishn/Downloads/NOSQL/NOSQL/PandasOutputAfterDataCleaning/car_prices_cleaned.csv"')
[7]                                                                                                  Python

# Create a DuckDB connection
con = duckdb1.connect(database=':memory:', read_only=False)

# Create the carsales table from the CSV file
con.execute('CREATE TABLE carsales AS SELECT * FROM "C:/Users/vishn/Downloads/NOSQL/NOSQL/PandasOutputAfterDataCleaning/car_prices_cleaned.csv"')

# Fetch data from the carsales table into a DataFrame
df = con.execute('SELECT * FROM carsales').fetchdf()
[20]                                                                                                 Python
```

Querying the carsales table from the DuckDB database using the duckdb1.sql() method

```
[8]
duckdb1.sql('select * from carsales')
```

```
┌───────┬───────────────┬──────────────────────┬──────────────────────┬───┬───────┬──────────────┬──────────────────────┐
│ year  │     make      │        model         │         trim         │ … │  mmr  │ sellingprice │       saledate       │
│ int64 │    varchar    │       varchar        │       varchar        │   │ int64 │    int64     │       varchar        │
├───────┼───────────────┼──────────────────────┼──────────────────────┼───┼───────┼──────────────┼──────────────────────┤
│  2015 │ Kia           │ Sorento              │ LX                   │ … │ 20500 │        21500 │ Tue Dec 16 2014 12…  │
│  2015 │ Kia           │ Sorento              │ LX                   │ … │ 20800 │        21500 │ Tue Dec 16 2014 12…  │
│  2014 │ BMW           │ 3 Series             │ 328i SULEV           │ … │ 31900 │        30000 │ Thu Jan 15 2015 04…  │
│  2015 │ Volvo         │ S60                  │ T5                   │ … │ 27500 │        27750 │ Thu Jan 29 2015 04…  │
│  2014 │ BMW           │ 6 Series Gran Coupe  │ 650i                 │ … │ 66000 │        67000 │ Thu Dec 18 2014 12…  │
│  2015 │ Nissan        │ Altima               │ 2.5 S                │ … │ 15350 │        10900 │ Tue Dec 30 2014 12…  │
│  2014 │ BMW           │ M5                   │ Base                 │ … │ 69000 │        65000 │ Wed Dec 17 2014 12…  │
│  2014 │ Chevrolet     │ Cruze                │ 1LT                  │ … │ 11900 │         9800 │ Tue Dec 16 2014 13…  │
│  2014 │ Audi          │ A4                   │ 2.0T Premium Plus …  │ … │ 32100 │        32250 │ Thu Dec 18 2014 12…  │
│  2014 │ Chevrolet     │ Camaro               │ LT                   │ … │ 26300 │        17500 │ Tue Jan 20 2015 04…  │
│    ·  │      ·        │          ·           │ ·                    │ · │   ·   │          ·   │          ·           │
│    ·  │      ·        │          ·           │ ·                    │ · │   ·   │          ·   │          ·           │
│    ·  │      ·        │          ·           │ ·                    │ · │   ·   │          ·   │          ·           │
│  2010 │ Mazda         │ Mazda3               │ i SV                 │ … │  8900 │         5500 │ Tue Dec 23 2014 11…  │
│  2010 │ Mercedes-Benz │ C-Class              │ C300 Sport 4MATIC    │ … │ 17450 │        17200 │ Thu Dec 18 2014 10…  │
│  2010 │ Mazda         │ Mazda3               │ s Grand Touring      │ … │  9975 │        10400 │ Thu Dec 18 2014 09…  │
│  2010 │ Mazda         │ Mazda3               │ i SV                 │ … │  8075 │         7200 │ Thu Dec 18 2014 11…  │
│  2010 │ Mazda         │ Mazdaspeed3          │ Sport                │ … │ 12450 │        12200 │ Thu Dec 18 2014 08…  │
│  2010 │ Mazda         │ Mazda6               │ i Grand Touring      │ … │  8350 │         9100 │ Thu Dec 18 2014 11…  │
│  2010 │ Mazda         │ Mazda3               │ i Touring            │ … │  6625 │         6800 │ Fri Dec 19 2014 09…  │
│  2010 │ Mercedes-Benz │ E-Class              │ E350                 │ … │ 19400 │        19500 │ Fri Dec 19 2014 09…  │
│  2010 │ Mazda         │ Mazda6               │ i Sport              │ … │  8100 │         7700 │ Fri Dec 19 2014 09…  │
│  2010 │ Mazda         │ Mazda3               │ i SV                 │ … │  6975 │         5600 │ Thu Jan 15 2015 03…  │
├───────┴───────────────┴──────────────────────┴──────────────────────┴───┴───────┴──────────────┴──────────────────────┤
│ ? rows (>9999 rows, 20 shown)                                                                    16 columns (7 shown) │
└───────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┘
```

TOP-SELLING CAR MODEL

Defining an **SQL query to identify the top-selling car models** from the carsales table, grouping the data by model and counting the number of sales for each model. It then executes the query using the DuckDB connection (duckdb1) and fetches the results. Finally, it prints the top selling models along with their respective sales counts in a formatted table.

```python
# Define the SQL query to identify the top selling models
sql_query = """
SELECT model,
       COUNT(*) AS sales_count
FROM carsales
GROUP BY model
ORDER BY sales_count DESC
LIMIT 10
"""

# Execute the SQL query using the connection
result = duckdb1.execute(sql_query)

# Fetch the results
rows = result.fetchall()

# Print the results with headers
print("Top Selling Models")
print("------------------")
print("Model                Sales Count")
print("--------------------------------")
for row in rows:
    print(f"{row[0]:<20} {row[1]:<11}")
```

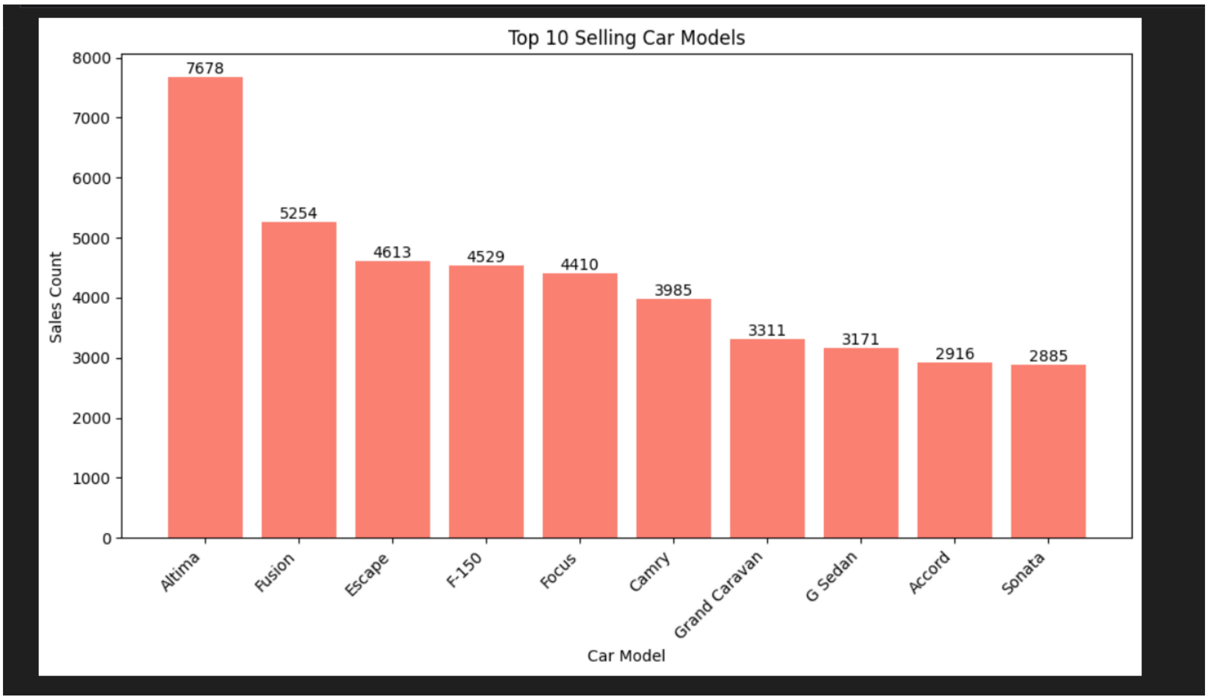
[9] Python

```
Top Selling Models
------------------
Model                    Sales Count
--------------------------------
Altima                   7678
Fusion                   5254
Escape                   4613
F-150                    4529
Focus                    4410
Camry                    3985
Grand Caravan            3311
G Sedan                  3171
Accord                   2916
Sonata                   2885
```

VISUALIZATION – TOP-SELLING CAR MODEL

Extracting model names and their corresponding sales counts from the previously fetched SQL query results. It then creates **a bar plot using Matplotlib to visualize the top 10 selling car models.** The bars in the plot are colored using a specified color ('salmon'), and each bar is annotated with its respective sales count value. Finally, the plot is displayed using plt.show().

```python
# Extract model names and sales counts
models = [row[0] for row in rows]
sales_counts = [row[1] for row in rows]

# Create a bar plot with a different color
plt.figure(figsize=(10, 6))
bars = plt.bar(models, sales_counts, color='salmon')  # Specify a different color here
plt.xlabel('Car Model')
plt.ylabel('Sales Count')
plt.title('Top 10 Selling Car Models')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()

# Annotate the bars with their values
for bar, count in zip(bars, sales_counts):
    plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() + 0.5, str(count), ha='center', va='bottom')

# Show the plot
plt.show()
```



AVERAGE SELLING PRICE BY CAR MAKE (ABOVE OVERALL AVERAGE)

Executing SQL queries on a carsales table using DuckDB:

1. Calculates the overall average selling price of all cars.
2. Retrieves average selling prices for car makes that exceed the overall average, and prints these results with their respective makes and prices.

```python
# The SQL query to calculate the overall average selling price
overall_avg_query = """
SELECT AVG(sellingprice) AS overall_avg_selling_price
FROM carsales
"""

# The overall average query using the connection
overall_avg_result = duckdb1.execute(overall_avg_query)
overall_avg_row = overall_avg_result.fetchone()
overall_avg_selling_price = overall_avg_row[0]

# The SQL query to calculate average selling price by car make
sql_query = f"""
SELECT make,
       AVG(sellingprice) AS avg_selling_price
FROM carsales
GROUP BY make
HAVING AVG(sellingprice) > {overall_avg_selling_price}
ORDER BY avg_selling_price DESC
"""

# Execute the SQL query using the connection
result = duckdb1.execute(sql_query)

# Fetch the results
rows = result.fetchall()

# Print the results
print("Average Selling Price by Car Make (Above Overall Average)")
print("----------------------------------------------------------")
for row in rows:
    print(f"{row[0]:<20} {row[1]:.2f}")
```
Python

```
Average Selling Price by Car Make (Above Overall Average)
--------------------------------------------------------
Rolls-Royce         154466.67
Ferrari             137750.00
Bentley             98984.38
Aston Martin        82833.33
Tesla               80750.00
airstream           71000.00
Maserati            57940.48
land rover          55333.33
Fisker              52750.00
porsche             50262.50
Porsche             47010.48
Land Rover          34799.80
bmw                 33324.23
Jaguar              31827.26
lincoln             28875.00
BMW                 27334.71
Mercedes-Benz       26716.82
Lexus               25627.59
Audi                25346.03
Ram                 24882.89
Infiniti            22448.26
Cadillac            21553.01
GMC                 20760.86
...
Jeep                18070.67
Subaru              17545.16
vw                  16821.05
Ford                16457.56
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

VISUALIZATION - AVERAGE SELLING PRICE BY CAR MAKE (ABOVE OVERALL AVERAGE)

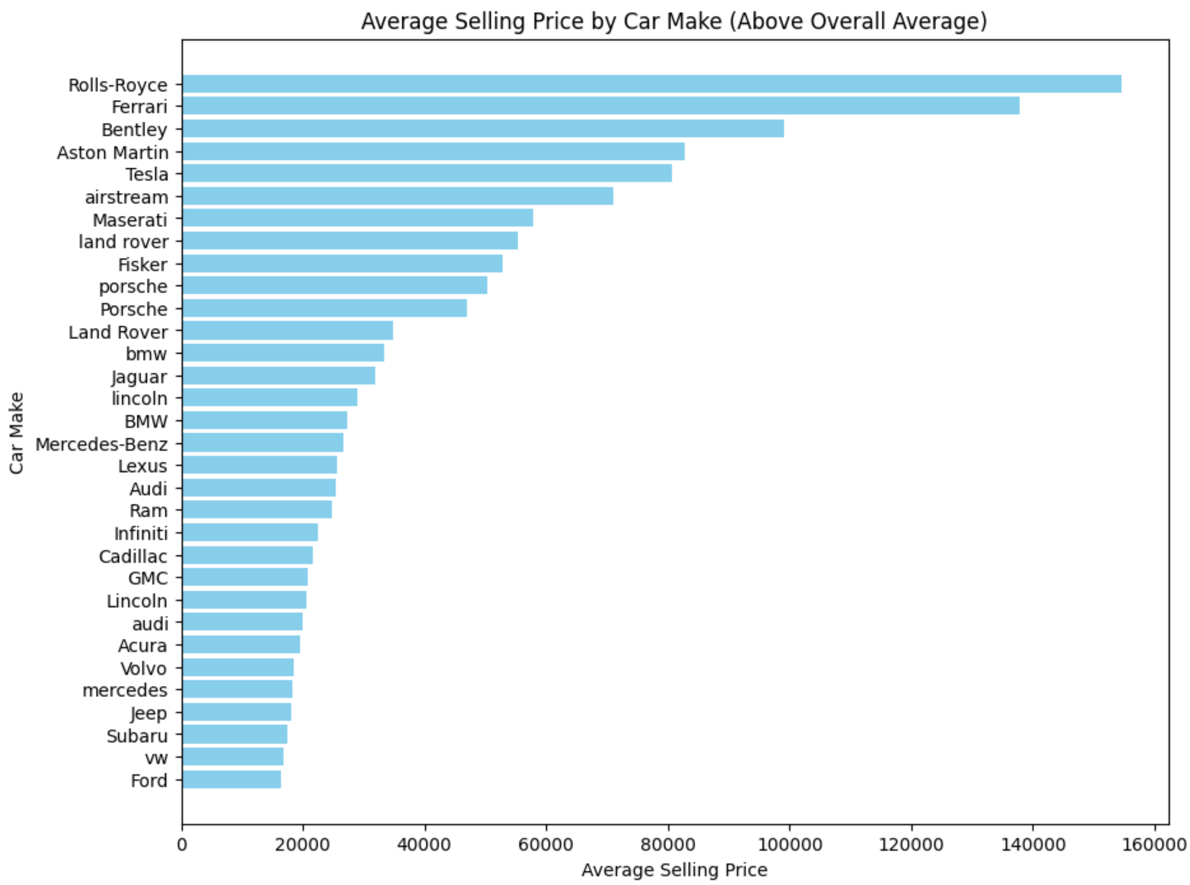
Extracting data to create a horizontal bar chart displaying average selling prices by car make:

1. Extracting `makes` (car makes) and `avg_prices` (corresponding average selling prices) from fetched rows (`rows`).

Creating a horizontal bar chart:

1. Using `plt.figure(figsize=(10, 8))` to specify the figure size.
2. Plotting the horizontal bars (`plt.barh`) where each bar represents a car make (`makes`) and its length represents the average selling price (`avg_prices`).
3. Setting labels (`xlabel`, `ylabel`) and a title (`title`) for the plot.
4. Inverting the y-axis (`plt.gca().invert_yaxis()`) to display the car make with the highest average selling price at the top.
5. Displaying the horizontal bar chart using `plt.show()` to visualize the comparison of average selling prices across different car makes.

```
makes = [row[0] for row in rows]
avg_prices = [row[1] for row in rows]

# Create the bar chart
plt.figure(figsize=(10, 8))
plt.barh(makes, avg_prices, color='skyblue')
plt.xlabel('Average Selling Price')
plt.ylabel('Car Make')
plt.title('Average Selling Price by Car Make (Above Overall Average)')
plt.gca().invert_yaxis()  # Invert y-axis to display the highest value at the top
plt.show()
```



TOP AND LOW RATED CAR MODEL YEAR WISE

Executing an SQL query using DuckDB:

1. The query identifies the top and low-rated car models for each year by ranking models based on their ratings.
2. Results are fetched and printed, displaying the year alongside the top-rated and low-rated car models for each year.

```python
# The SQL query to find the top and low-rated car model for each year
sql_query = """
WITH ranked_models AS (
    SELECT *,
           RANK() OVER(PARTITION BY year ORDER BY rating DESC) AS top_rank,
           RANK() OVER(PARTITION BY year ORDER BY rating ASC) AS low_rank
    FROM carsales
)
SELECT year,
       MAX(CASE WHEN top_rank = 1 THEN model ELSE NULL END) AS top_rated_model,
       MAX(CASE WHEN low_rank = 1 THEN model ELSE NULL END) AS low_rated_model
FROM ranked_models
GROUP BY year
"""

# Execute the SQL query using the connection
result = duckdb1.execute(sql_query)

# Fetch the results
rows = result.fetchall()

# Print the results
print("Top and Low Rated Car Model Year Wise")
print("--------------------------------------")
print("Year   Top Rated Model   Low Rated Model")
print("-----------------------------------------")
for row in rows:
    print(f"{row[0]}   {row[1]:<18} {row[2]}")
```

Python

```
Top and Low Rated Car Model Year Wise
--------------------------------------
Year   Top Rated Model   Low Rated Model
-----------------------------------------
2008   fortwo             xB
2009   Yukon              g3500
2010   Yukon XL           Yaris
2011   e150               galant
2012   Z4                 galant
2013   xD                 xB
2014   xB                 Yukon XL
2015   Z4                 capt
```

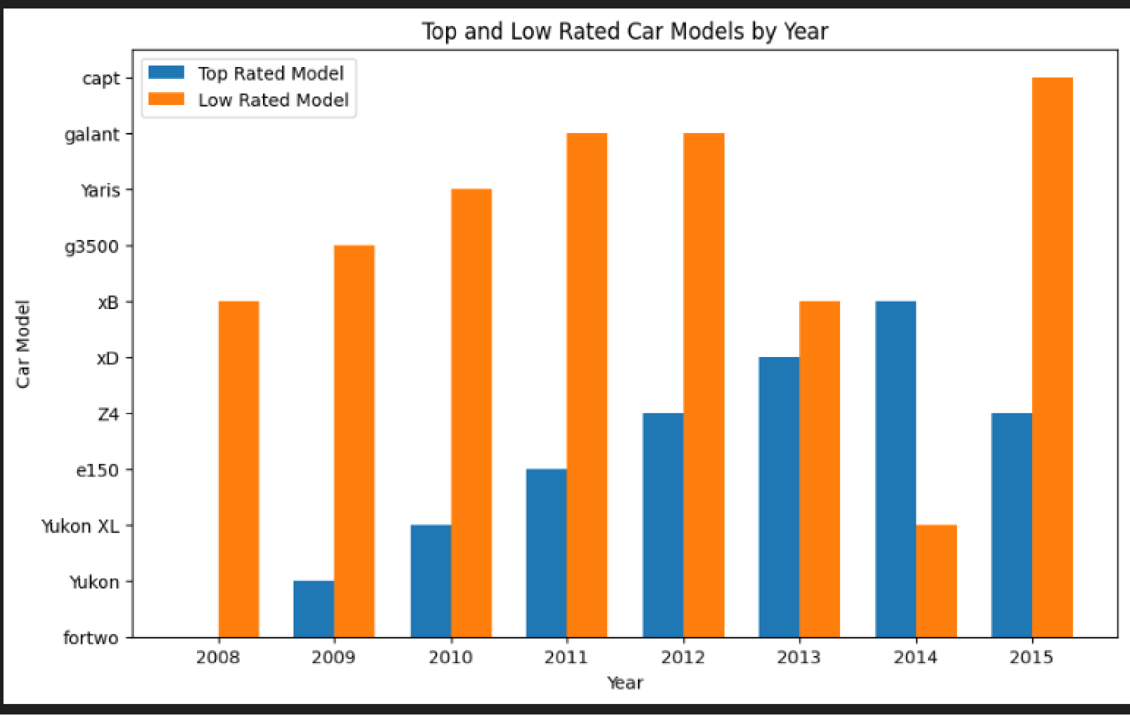
VISUALIZATION - TOP AND LOW RATED CAR MODEL YEAR WISE

Extracting data for creating a grouped bar chart showing top and low-rated car models by year:

1. Extracting `years`, `top_models`, and `low_models` from fetched rows (`rows`).
2. Creating a figure (`plt.figure(figsize=(10, 6))`) with specified dimensions.
3. Defining `bar_width` for setting the width of each bar and creating an index for x-axis positions.
4. Using `plt.bar` to plot bars for `top_models` and `low_models` side by side for each year.
   - Bars for `top_models` are plotted at `index` positions.
   - Bars for `low_models` are plotted shifted to the right (`[i + bar_width for i in index]`) to appear beside `top_models`.

5. Adding labels (`xlabel`, `ylabel`), title (`title`), and setting x-axis ticks (`plt.xticks`) with year labels.
6. Displaying a legend (`plt.legend()`) to differentiate between top-rated and low-rated models.
7. Showing the grouped bar chart using `plt.show()` to visualize the comparison of top and low-rated car models over the years.

```
# Extract data for plotting
years = [row[0] for row in rows]
top_models = [row[1] for row in rows]
low_models = [row[2] for row in rows]

# Create the grouped bar chart
plt.figure(figsize=(10, 6))
bar_width = 0.35
index = range(len(years))

plt.bar(index, top_models, bar_width, label='Top Rated Model')
plt.bar([i + bar_width for i in index], low_models, bar_width, label='Low Rated Model')

plt.xlabel('Year')
plt.ylabel('Car Model')
plt.title('Top and Low Rated Car Models by Year')
plt.xticks([i + bar_width / 2 for i in index], years)
plt.legend()
plt.show()
```



SALES OF TOP 5 MOST EXPENSIVE CAR MAKES OVER THE YEARS

Executing an SQL query to analyze sales data for the top 5 most expensive car makes:

1. Identifying the Top 5 Expensive Car Makes:

- The SQL query calculates the average selling price for each car make and selects the top 5 makes with the highest average selling prices.

- The result is printed to display the top 5 expensive car makes.

2. Analyzing Total Sales by Year for Top Expensive Makes:

- Another SQL query is defined to retrieve total sales data by year for the identified top expensive car makes.

- The query filters sales data to include only the top expensive makes (`make IN {tuple(top_expensive_makes)}`).

- Sales data is fetched and organized into a dictionary (`sales_data`) for plotting.

3. Plotting Total Sales Over Years for Each Make:

- A line graph is plotted to visualize the total sales trend over the years for each of the top 5 most expensive car makes.

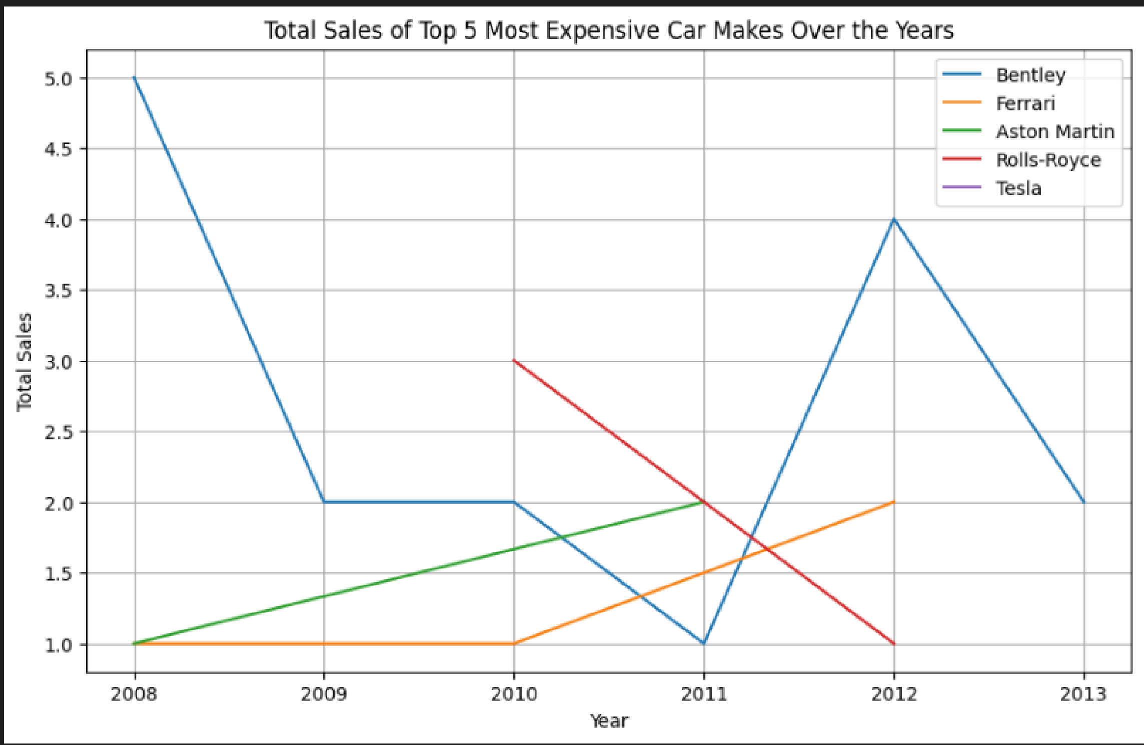
- Each line represents a car make (`make`) with years on the x-axis (`data['years']`) and total sales on the y-axis (`data['total_sales']`).

- Labels, title, legend, and grid are added to the plot for clarity.

This process allows for the analysis and visualization of sales performance over time for the top 5 most expensive car makes based on average selling prices.

```python
# SQL query to identify the 5 most expensive car makes
top_expensive_query = """
SELECT make
FROM (
    SELECT make, AVG(sellingprice) AS avg_price
    FROM carsales
    GROUP BY make
    ORDER BY avg_price DESC
    LIMIT 5
) AS top_expensive_makes
"""

# Execute the SQL query to get the 5 most expensive car makes
top_expensive_result = duckdb1.execute(top_expensive_query)
top_expensive_makes = [row[0] for row in top_expensive_result.fetchall()]
print("Top 5 expensive makes")
for i in top_expensive_makes:
    print(i)

# Define the SQL query to get total sales by year for the top expensive makes
sales_by_year_query = f"""
SELECT year, make, count(sellingprice) AS total_sales
FROM carsales
WHERE make IN {tuple(top_expensive_makes)}
GROUP BY year, make
ORDER BY year
"""

# Execute the SQL query to get total sales by year for the top expensive makes
sales_by_year_result = duckdb1.execute(sales_by_year_query)

# Fetch the results and organize them into a dictionary for plotting
sales_data = {}
for year, make, total_sales in sales_by_year_result.fetchall():
    if make not in sales_data:
        sales_data[make] = {'years': [], 'total_sales': []}
    sales_data[make]['years'].append(year)
    sales_data[make]['total_sales'].append(total_sales)

# Plot the line graph for each make
plt.figure(figsize=(10, 6))
for make, data in sales_data.items():
    plt.plot(data['years'], data['total_sales'], label=make)

plt.xlabel('Year')
plt.ylabel('Total Sales')
plt.title('Total Sales of Top 5 Most Expensive Car Makes Over the Years')
plt.legend()
plt.grid(True)
plt.show()
```

```
Top 5 expensive makes
Rolls-Royce
Ferrari
Bentley
Aston Martin
Tesla
```



TOP 10 ODOMETER READINGS FOR SUVs WITH AUTOMATIC TRANSMISSION

Executing an SQL query to retrieve the top 10 odometer readings for SUVs with automatic transmission:

1. The query selects all columns (`*`) from the `carsales` table where the body type is 'SUV' and the transmission type is 'automatic'.
2. Results are fetched (`rows = result.fetchall()`) and printed in a formatted table displaying car make, model, and odometer reading for each SUV.

```python
# SQL query to retrieve the top 10 odometer readings for SUVs with automatic transmission
sql_query = """
SELECT *
FROM carsales
WHERE body = 'SUV' AND transmission = 'automatic'
ORDER BY odometer DESC
LIMIT 10
"""

# Execute the SQL query using the connection
result = duckdb1.execute(sql_query)

# Fetch the results
rows = result.fetchall()

# Print the results
print("Top 10 Odometer Readings for SUVs with Automatic Transmission")
print("---------------------------------------------------------------")
print("Make         Model         Odometer Reading")
print("---------------------------------------------------------------")
for row in rows:
    print(f"{row[1]:<12} {row[2]:<12} {row[9]}")
```

Python

```
Top 10 Odometer Readings for SUVs with Automatic Transmission
---------------------------------------------------------------
Make         Model         Odometer Reading
---------------------------------------------------------------
Saturn       VUE          999999
Nissan       Rogue        999999
Ford         Escape       999999
Dodge        Journey      999999
Mazda        CX-7         999999
Chevrolet    Suburban     458184
Chevrolet    Suburban     450825
Chevrolet    Suburban     426418
Chevrolet    Suburban     393276
Chrysler     Aspen        366136
```

RELATIONSHIP BETWEEN SELLING PRICE AND ODOMETER READING

Executing an SQL query to retrieve selling prices and odometer readings from the `carsales` table using DuckDB:

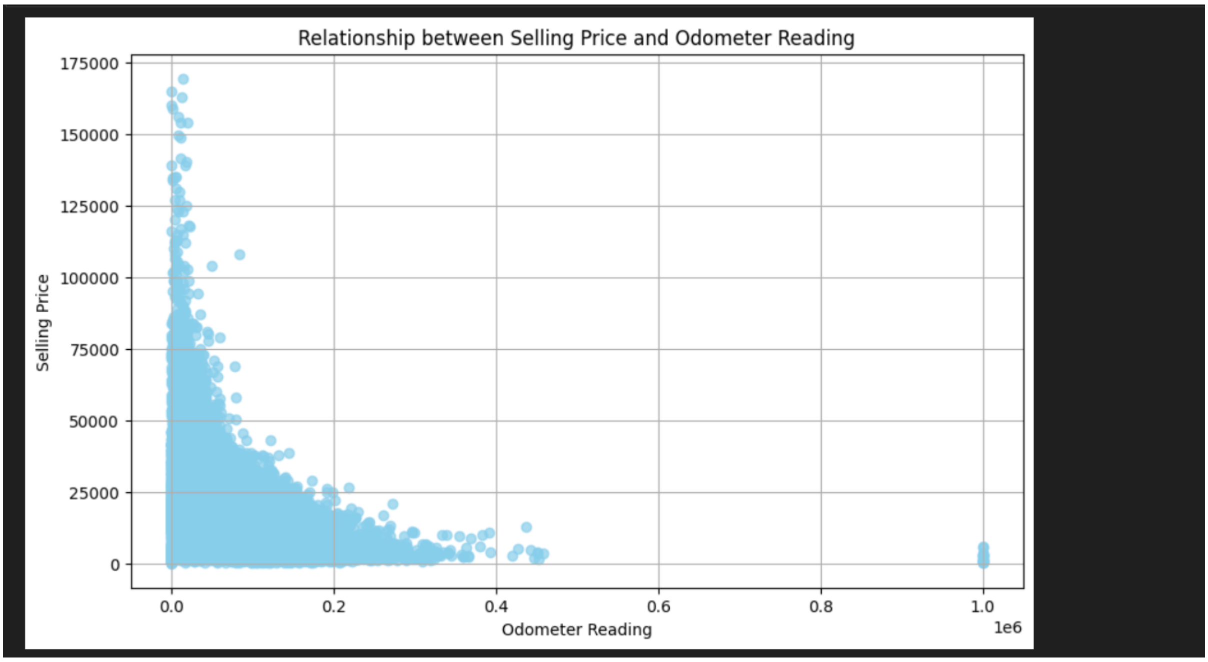
1. The SQL query selects `sellingprice` and `odometer` columns from the `carsales` table.
2. The query is executed using the DuckDB connection (`duckdb1.execute()`), and the results are fetched (`rows = result.fetchall()`).

3. The code then extracts selling prices (`selling_prices`) and odometer readings (`odometer_readings`) from the fetched rows.
4. Each row's selling price and odometer reading are printed using a loop (`for row in rows: print(f"Selling Price: {row[0]}, Odometer Reading: {row[1]}")`).
5. Finally, a scatter plot is created using Matplotlib (`plt.scatter()`) to visualize the relationship between selling price and odometer reading. The x-axis represents odometer readings (`odometer_readings`), the y-axis represents selling prices (`selling_prices`), and points are colored sky blue with transparency (`color='skyblue', alpha=0.7`). Axes labels, title, and grid lines are added for clarity, and the plot is displayed (`plt.show()`).

```python
# SQL query to retrieve selling price and odometer reading
sql_query = """
SELECT sellingprice, odometer
FROM carsales
"""

# Execute the SQL query using the connection
result = duckdb1.execute(sql_query)

# Fetch the results
rows = result.fetchall()

# Extract selling prices and odometer readings
selling_prices = [row[0] for row in rows]
odometer_readings = [row[1] for row in rows]

for row in rows:
    print(f"Selling Price: {row[0]}, Odometer Reading: {row[1]}")

# Plot a scatter plot of selling price vs. odometer reading
plt.figure(figsize=(10, 6))
plt.scatter(odometer_readings, selling_prices, color='skyblue', alpha=0.7)
plt.xlabel('Odometer Reading')
plt.ylabel('Selling Price')
plt.title('Relationship between Selling Price and Odometer Reading')
plt.grid(True)
plt.show()
```

Python

```
Selling Price: 21500, Odometer Reading: 16639
Selling Price: 21500, Odometer Reading: 9393
Selling Price: 30000, Odometer Reading: 1331
Selling Price: 27750, Odometer Reading: 14282
Selling Price: 67000, Odometer Reading: 2641
Selling Price: 10900, Odometer Reading: 5554
Selling Price: 65000, Odometer Reading: 14943
Selling Price: 9800, Odometer Reading: 28617
Selling Price: 32250, Odometer Reading: 9557
Selling Price: 17500, Odometer Reading: 4809
Selling Price: 49750, Odometer Reading: 14414
Selling Price: 17700, Odometer Reading: 2034
Selling Price: 12000, Odometer Reading: 5559
Selling Price: 21500, Odometer Reading: 14634
Selling Price: 10600, Odometer Reading: 15686
Selling Price: 14100, Odometer Reading: 11398
Selling Price: 4200, Odometer Reading: 8311
Selling Price: 40000, Odometer Reading: 7983
Selling Price: 17000, Odometer Reading: 13441
Selling Price: 67200, Odometer Reading: 8819
Selling Price: 7200, Odometer Reading: 14538
Selling Price: 30000, Odometer Reading: 25969
Selling Price: 14700, Odometer Reading: 33450
Selling Price: 23750, Odometer Reading: 5826
Selling Price: 65000, Odometer Reading: 10736
...
Selling Price: 23800, Odometer Reading: 32251
Selling Price: 15700, Odometer Reading: 30104
Selling Price: 20100, Odometer Reading: 23914
Selling Price: 18800, Odometer Reading: 55849
```

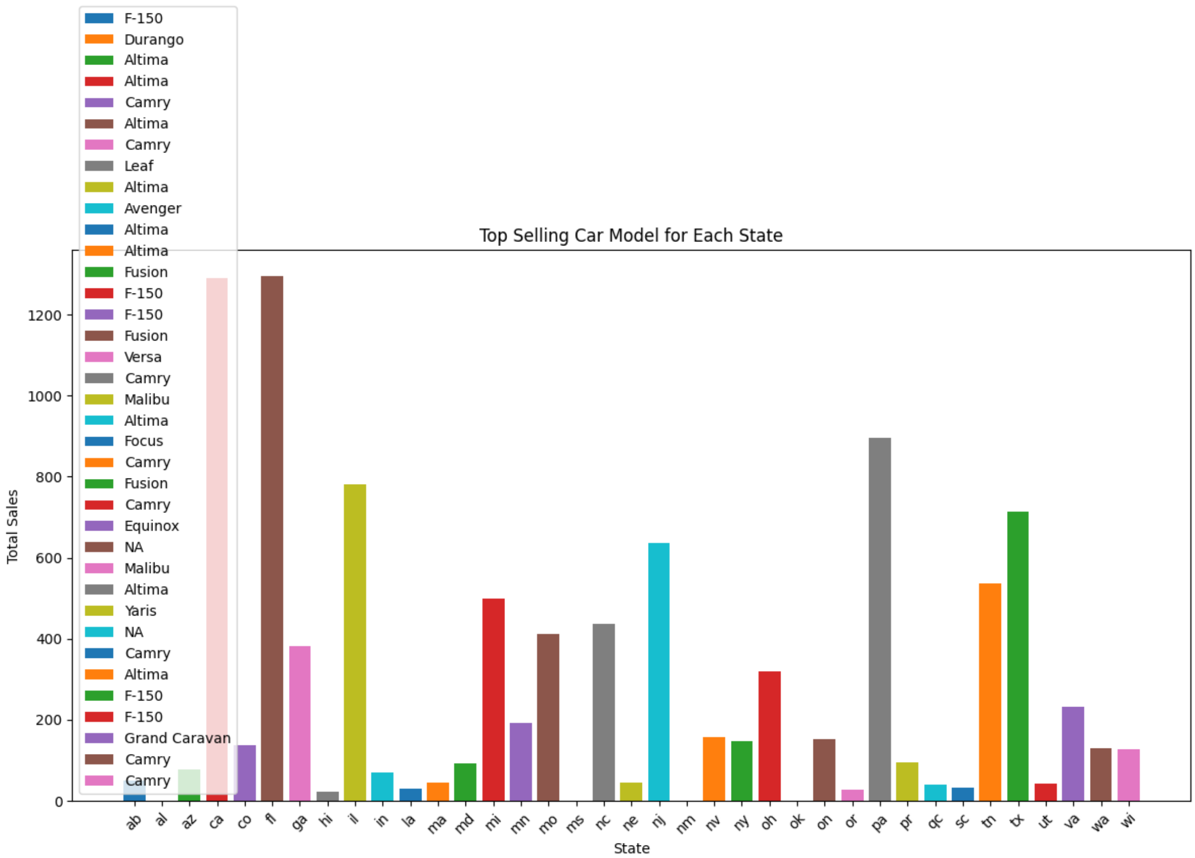


TOP SELLING CAR MODEL FOR EACH STATE

Defining an SQL query to retrieve and analyze top selling car models for each state from the `carsales` table using DuckDB:

1. The SQL query selects the state, car model, and total sales count for each model in each state, grouping the results by state and model, and ordering them by state and total sales count in descending order.
2. After executing the query and fetching the results, the code initializes a dictionary (`top_models_by_state`) to store the top selling car model for each state based on the total sales count.
3. The code iterates over the fetched results, populating the `top_models_by_state` dictionary with the top selling car model and its total sales count for each state.
4. Finally, a bar plot is generated using Matplotlib to visualize the top selling car model for each state, where each state is represented on the x-axis, and the total sales count is represented by the height of the bar. The car model for each state is shown using different bars, and the plot includes appropriate labels, title, and legend for clarity.

```python
# Define the SQL query to retrieve the top selling car model for each state
top_models_query = """
SELECT state, model, COUNT(*) AS total_sales
FROM carsales
GROUP BY state, model
ORDER BY state, total_sales DESC
"""

# Execute the SQL query
result = duckdb1.execute(top_models_query)

# Fetch the results
rows = result.fetchall()

# Initialize dictionaries to store top selling car model for each state
top_models_by_state = {}

# Iterate over the results and store the top selling car model for each state
for row in rows:
    state = row[0]
    model = row[1]
    total_sales = row[2]
    if state not in top_models_by_state:
        top_models_by_state[state] = (model, total_sales)

# Plot the top selling car model for each state
plt.figure(figsize=(12, 8))
for state, (model, total_sales) in top_models_by_state.items():
    plt.bar(state, total_sales, label=model)

plt.xlabel('State')
plt.ylabel('Total Sales')
plt.title('Top Selling Car Model for Each State')
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.show()
```

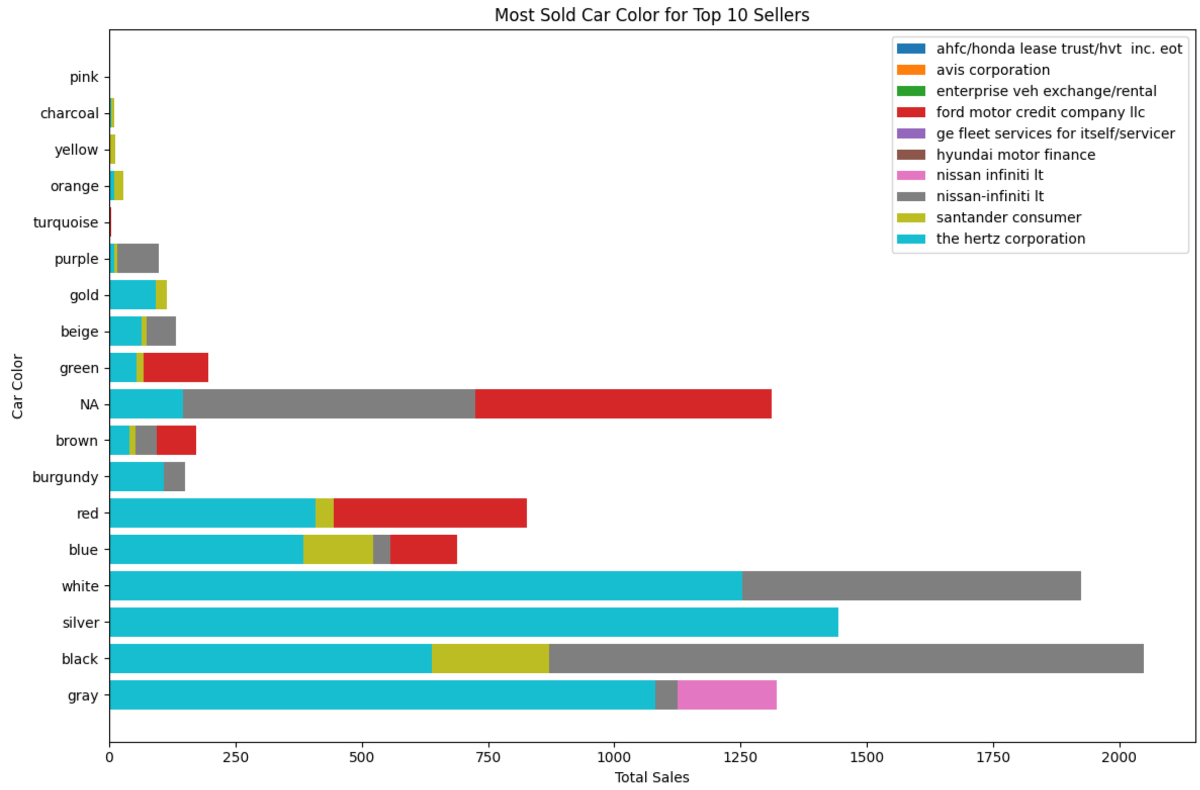


MOST SOLD CAR COLOR FOR TOP 10 SELLERS

Defining an SQL query to identify and analyze the most sold car color for each of the top 10 sellers from the `carsales` table using DuckDB:

1. The SQL query first identifies the top 10 sellers based on the total number of sales (`total_sales`), grouped by seller and ordered in descending order of sales count.
2. It then joins this list of top sellers (`top_sellers`) with the `carsales` table to retrieve the most sold car color (`most_sold_color`) and the corresponding total sales count for each seller.
3. After executing the SQL query and fetching the results, the code organizes the data into dictionaries (`seller_data`) for plotting. Each dictionary entry corresponds to a seller, containing lists of car colors (`colors`) and their corresponding total sales counts (`total_sales`).
4. Finally, a horizontal bar graph is generated using Matplotlib to visualize the most sold car color for each of the top 10 sellers. Each bar represents a seller, and the bar lengths indicate the total sales for different car colors. The plot includes appropriate labels, title, and legend for clarity.

```
# The SQL query to find the most sold car color for each of the top 10 sellers
query = """
WITH top_sellers AS (
    SELECT seller, COUNT(*) AS total_sales
    FROM carsales
    GROUP BY seller
    ORDER BY total_sales DESC
    LIMIT 10
)
SELECT ts.seller, cs.color AS most_sold_color, COUNT(*) AS total_sales
FROM carsales cs
JOIN top_sellers ts ON cs.seller = ts.seller
GROUP BY ts.seller, cs.color
ORDER BY ts.seller, total_sales DESC
"""
# Execute the SQL query
result = duckdb1.execute(query)
# Fetch the results
rows = result.fetchall()
# Organize the data into dictionaries for plotting
seller_data = {}
for row in rows:
    seller = row[0]
    color = row[1]
    total_sales = row[2]
    if seller no (variable) seller: Any
        seller_d                        [], 'total_sales': []}
    seller_data[seller]['colors'].append(color)
    seller_data[seller]['total_sales'].append(total_sales)
# Plot the bar graph for each seller
plt.figure(figsize=(12, 8))
for seller, data in seller_data.items():
    plt.barh(data['colors'], data['total_sales'], label=seller)
plt.xlabel('Total Sales')
plt.ylabel('Car Color')
plt.title('Most Sold Car Color for Top 10 Sellers')
plt.legend()
plt.tight_layout()
plt.show()
```

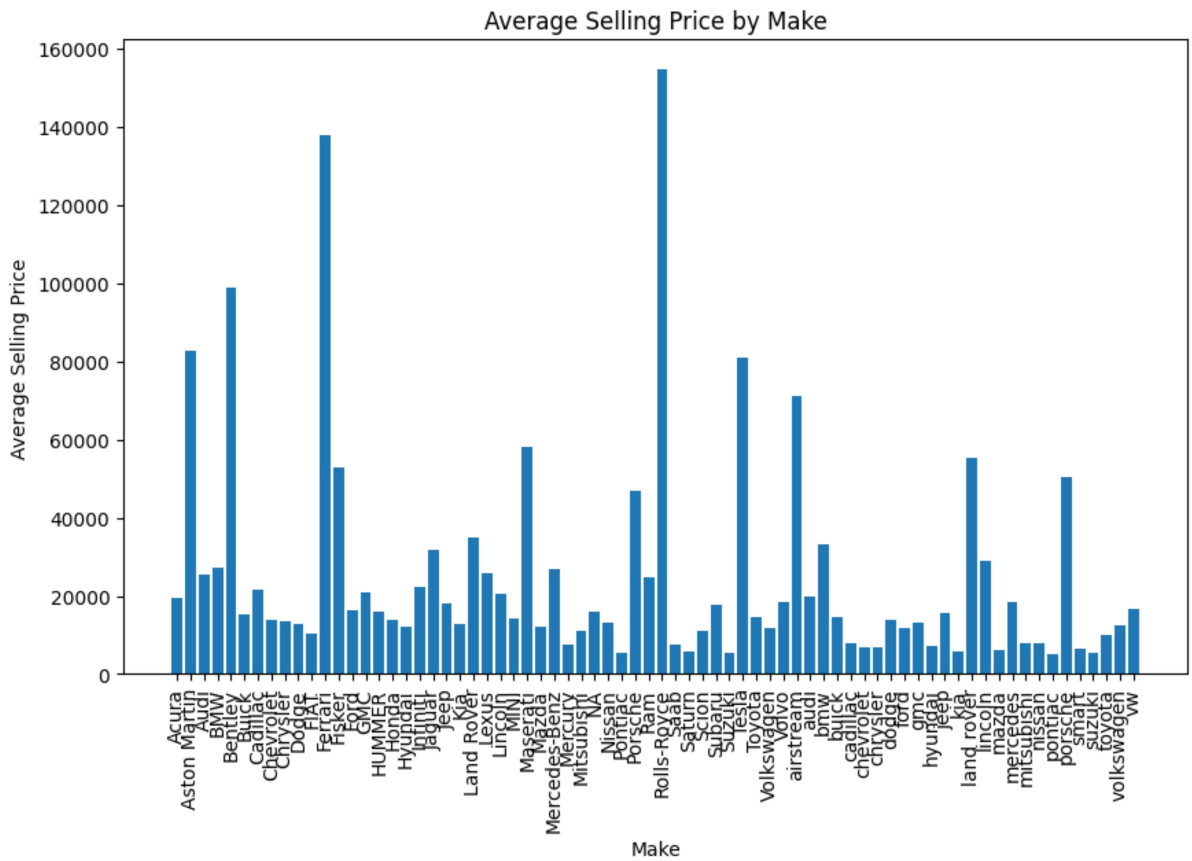


AVERAGE SELLING PRICE OF CARS BY MAKE

Creating a bar chart to display the average selling price of cars by make:

1. Calculating the mean selling price (`sellingprice`) for each car make (`make`) using `groupby` and `mean`, resulting in a DataFrame (`average_selling_price`).
2. Setting the figure size (`plt.figure(figsize=(10, 6))`) to control the plot dimensions.
3. Using `plt.bar` to create a bar chart, where each bar represents a car make (`average_selling_price['make']`) and its height corresponds to the average selling price (`average_selling_price['sellingprice']`).
4. Adding labels (`xlabel`, `ylabel`) and a title (`title`) to the plot for clarity.
5. Rotating the x-axis labels (`plt.xticks(rotation=90)`) to avoid overlap and improve readability.
6. Displaying the plot using `plt.show()` to visualize the average selling prices categorized by car make.

```
#Bar Chart of Average Selling Price by Make
average_selling_price = df.groupby('make')['sellingprice'].mean().reset_index()
plt.figure(figsize=(10, 6))
plt.bar(average_selling_price['make'], average_selling_price['sellingprice'])
plt.xlabel('Make')
plt.ylabel('Average Selling Price')
plt.title('Average Selling Price by Make')
plt.xticks(rotation=90)
plt.show()
```



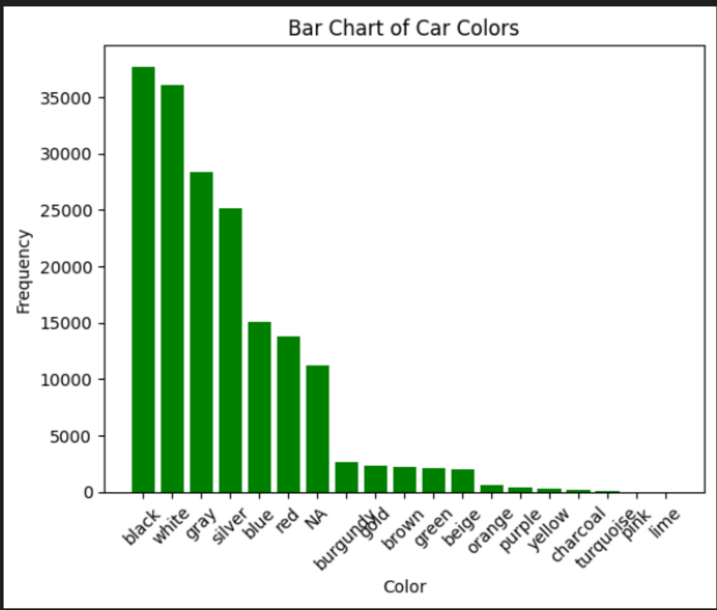
CAR COLORS BASED ON THEIR FREQUENCY

Plotting a bar chart of car colors based on their frequency:

1. Calculating the frequency of each car color (`color_counts`) using `value_counts()` on the 'color' column of the DataFrame (`df`).
2. Creating a bar chart (`plt.bar`) where the x-axis represents car colors (`color_counts.index`) and the bar heights represent their respective frequencies (`color_counts.values`), colored green (`color='green'`).
3. Setting labels (`xlabel`, `ylabel`), title (`title`), and rotating the x-axis labels (`xticks`) for better readability.

4. Displaying the plot using `plt.show()` to visualize the distribution of car colors by frequency.

```
#Bar Chart of Car Colors:
color_counts = df['color'].value_counts()
plt.bar(color_counts.index, color_counts.values, color='green')
plt.xlabel('Color')
plt.ylabel('Frequency')
plt.title('Bar Chart of Car Colors')
plt.xticks(rotation=45)
plt.show()
```

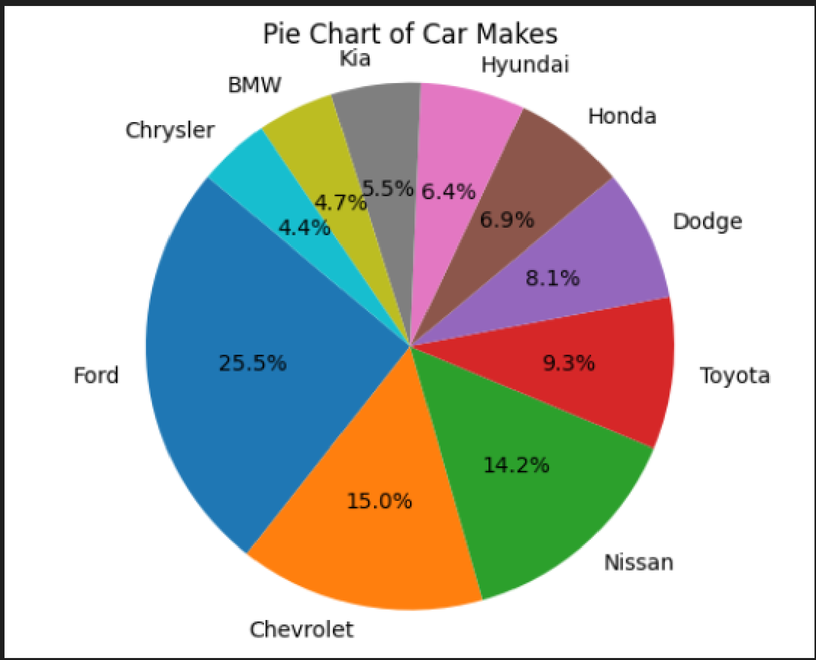


THE PROPORTION OF EACH CAR MAKE

Creating a pie chart to show the distribution of car makes:

1. Calculating the frequency of each car make (`make_counts`) from the DataFrame (`df`).
2. Using `plt.pie` to display the distribution with labels and percentages.
3. Setting the title and ensuring the chart appears circular (`plt.axis('equal')`).
4. Displaying the pie chart to visualize the proportion of each car make.

```python
#Pie Chart of Car Makes:
make_counts = df['make'].value_counts().head(10)
plt.pie(make_counts, labels=make_counts.index, autopct='%1.1f%%', startangle=140)
plt.title('Pie Chart of Car Makes')
plt.axis('equal')
plt.show()
```



CONCLUSION

In conclusion, the exploratory data analysis (EDA) conducted on the "Vehicle Sales and Market Trends Dataset" has yielded valuable insights into car selling prices and market trends. Through meticulous data cleaning and preprocessing, we ensured data quality and consistency, preparing the dataset for meaningful analysis. Utilizing SQL queries and data visualization techniques, we identified top-selling car models, analyzed sales trends, and visualized relationships between car attributes and pricing.

Key findings include the identification of top expensive car makes based on average selling prices, trends in total sales over years for these expensive makes, and insights into odometer readings for specific vehicle types. Additionally, analysis of car colors preferred by top sellers sheds light on consumer preferences.

This analysis underscores the importance of data-driven decision-making in understanding market dynamics and shaping business strategies within the automotive industry. Moving forward, further exploration of this dataset could involve predictive modeling to forecast sales trends or sentiment analysis to gauge customer preferences. This EDA serves as a foundational step towards unlocking deeper insights into the dynamics of vehicle sales and market trends.

THANK YOU!