01-eda

July 27, 2024

1 Car Sales Project

[1]:
```
# Import packages
import pandas as pd
import pygwalker as pyg
import pandasql as psql
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import plotly.express as px
```

[3]:
```
# Import data
purchase_data = pd.read_csv("../data/lf_tech_test_purchase_data.csv",␣
 ↪index_col="customer_id")
vehicle_data = pd.read_csv("../data/lf_tech_test_vehicle_data.csv",␣
 ↪index_col="vehicle_id")
# Merge the DataFrames on 'vehicle_id'
merged_data = pd.merge(purchase_data, vehicle_data, on='vehicle_id')
```

[3]:
```
purchase_data
```

[3]:

customer_id	customer_uuid	purchase_date	rrp_discount	\
1	90acb38b-f9cf-47bd-9023-91cf727e450f	2023-01-31	0.00	
2	5a8eb750-75f3-4666-9114-f0aa2861e413	2023-12-30	0.00	
3	e62b93cd-bcf2-4ec1-aeb1-a752ab480061	2022-05-10	0.00	
4	e980700f-5a02-4e01-b052-c17190eaf4ce	2023-07-18	0.00	
5	c5504810-12ea-43c5-ae71-77ca8fc1926f	2023-02-04	0.00	
…	…	…	…	
1999996	b97120a3-43c2-463e-b6ad-4ee7805c6aa8	2024-02-02	0.00	
1999997	dcb756ba-f9c1-42fc-b9a5-4817a8124fc1	2021-10-29	0.00	
1999998	29829ee0-8532-4914-8f5d-0136344f162e	2020-06-06	0.09	
1999999	3311ec84-d037-4289-9a00-5cf327f572dc	2022-12-28	0.00	
2000000	f8c51c6e-5a29-49df-a517-b7f586081dab	2020-12-23	0.00	

customer_id	vehicle_id	city	lat	lng	approx_population

```
1                   366      Bristol  51.4536 -2.5975             707412
2                   923         Bath  51.3800 -2.3600              94782
3                   717         Bath  51.3800 -2.3600              94782
4                   586    Liverpool  53.4094 -2.9785             513441
5                   155  Southampton  50.9025 -1.4042             855569
…                   …           …       …       …                   …
1999996             443   Manchester  53.4790 -2.2452             547627
1999997             873         Bath  51.3800 -2.3600              94782
1999998             669         Bath  51.3800 -2.3600              94782
1999999             330      Bristol  51.4536 -2.5975             707412
2000000             678         Bath  51.3800 -2.3600              94782

[2000000 rows x 8 columns]
```

[4]:
```
purchase_data.columns
```

[4]:
```
Index(['customer_uuid', 'purchase_date', 'rrp_discount', 'vehicle_id', 'city',
       'lat', 'lng', 'approx_population'],
      dtype='object')
```

[21]:
```
walker = pyg.walk(
    purchase_data,
    spec="./pygwalker-spec.json",
    kernel_computation=True,
)
```

```
Box(children=(HTML(value='\n<div id="ifr-pyg-00061e39ede014f16vzdJE1b7UlKwXH4"␣
↪style="height: auto">\n    <hea…

<IPython.core.display.HTML object>
```

[26]:
```
walker.display_chart("Car Sales Map")
```

```
<IPython.core.display.HTML object>
```

[38]:
```
vehicle_data
```

[38]:
```
                                                  name  \
vehicle_id
0                              2024 Jeep Wagoneer Series II
1                           2024 Jeep Grand Cherokee Laredo
2                                 2024 GMC Yukon XL Denali
3                               2023 Dodge Durango Pursuit
4                                    2024 RAM 3500 Laramie
…                                                      …
973         2024 Mercedes-Benz Sprinter 2500 Standard Roof
974                  2024 Dodge Hornet Hornet R/T Plus Eawd
975                               2024 Jeep Wagoneer Base
976                  2024 Nissan Murano SV Intelligent AWD
```

```
977                        2024 Chevrolet Silverado 2500 WT

                                                    description           make  \
vehicle_id
0           \n      \n        Heated Leather Seats, Nav Sy…           Jeep
1           Al West is committed to offering every custome…           Jeep
2                                                         NaN            GMC
3           White Knuckle Clearcoat 2023 Dodge Durango Pur…          Dodge
4           \n      \n        2024 Ram 3500 Laramie Billet…            RAM
…                                                         …            …
973         2024 Mercedes-Benz Sprinter 2500 Cargo 144 WB …  Mercedes-Benz
974         Dealer Comments +++ Price Ends 5/31/2024 +++ A…          Dodge
975         \n      \n        The ALL New Friendship CDJR …           Jeep
976         \n      \n        CVT with Xtronic, AWD.At Tod…         Nissan
977         01u 2024 Chevrolet Silverado 2500HD Work Truck…      Chevrolet

                     model type  year    price  \
vehicle_id
0                  Wagoneer  New  2024  74600.0
1            Grand Cherokee  New  2024  50170.0
2                  Yukon XL  New  2024  96410.0
3                   Durango  New  2023  46835.0
4                      3500  New  2024  81663.0
…                     …  …   …      …
973           Sprinter 2500  New  2024  59037.0
974                  Hornet  New  2024  49720.0
975                Wagoneer  New  2024  69085.0
976                  Murano  New  2024  43495.0
977         Silverado 2500  New  2024  48995.0

                                                      engine  cylinders  \
vehicle_id
0                                    24V GDI DOHC Twin Turbo        6.0
1                                                        OHV        6.0
2           6.2L V-8 gasoline direct injection, variable v…        8.0
3                                              16V MPFI OHV        8.0
4                                   24V DDI OHV Turbo Diesel        6.0
…                                                         …        …
973                                16V DDI DOHC Turbo Diesel        4.0
974         4 gasoline direct injection, DOHC, Multiair va…        4.0
975                                  24V GDI DOHC Twin Turbo        6.0
976         6 DOHC, variable valve control, regular unlead…        6.0
977         8 gasoline direct injection, variable valve co…        8.0

                fuel  mileage                          transmission  \
vehicle_id
0           Gasoline     10.0                     8-Speed Automatic
```

```
1           Gasoline      1.0                    8-Speed Automatic
2           Gasoline      0.0                            Automatic
3           Gasoline     32.0                    8-Speed Automatic
4             Diesel     10.0                    6-Speed Automatic
…               …       …                              …
973           Diesel     10.0                    9-Speed Automatic
974         Gasoline      0.0  6-Spd Aisin F21-250 PHEV Auto Trans
975         Gasoline     20.0                    8-Speed Automatic
976         Gasoline      6.0                            Automatic
977         Gasoline     31.0                            Automatic

                            trim          body  doors  \
vehicle_id
0                      Series II           SUV    4.0
1                         Laredo           SUV    4.0
2                         Denali           SUV    4.0
3                        Pursuit           SUV    4.0
4                        Laramie  Pickup Truck    4.0
…                            …           …    …
973                Standard Roof     Cargo Van    3.0
974         Hornet R/T Plus Eawd           SUV    4.0
975                         Base           SUV    4.0
976           SV Intelligent AWD           SUV    4.0
977                           WT  Pickup Truck    4.0

                     exterior_color    interior_color        drivetrain
vehicle_id
0                             White      Global Black  Four-wheel Drive
1                          Metallic      Global Black  Four-wheel Drive
2                      Summit White  Teak/Light Shale  Four-wheel Drive
3           White Knuckle Clearcoat             Black   All-wheel Drive
4                            Silver             Black  Four-wheel Drive
…                               …                 …              …
973                    Arctic White             Black  Rear-wheel Drive
974                   Acapulco Gold             Black   All-wheel Drive
975                   Diamond Black             Black  Four-wheel Drive
976             Pearl White Tricoat          Graphite   All-wheel Drive
977                Wheatland Yellow         Jet Black  Rear-wheel Drive

[978 rows x 18 columns]
```

[11]:
```
vehicle_data.columns
```

[11]:
```
Index(['vehicle_id', 'name', 'description', 'make', 'model', 'type', 'year',
       'price', 'engine', 'cylinders', 'fuel', 'mileage', 'transmission',
       'trim', 'body', 'doors', 'exterior_color', 'interior_color',
       'drivetrain'],
```

```
      dtype='object')
```

[13]:
```
vehicle_data['year'].max()
```

[13]:
```
np.int64(2025)
```

[14]:
```
vehicle_data['year'].min()
```

[14]:
```
np.int64(2023)
```

[23]:
```
pyg.walk(vehicle_data)
```

```
Box(children=(HTML(value='<div id="ifr-pyg-00061e0e6b9de207S5htoJ6b0jIx4qLs"␣
 ↪style="height: auto">\n    <head>…

<IPython.core.display.HTML object>
```

[23]:
```
<pygwalker.api.pygwalker.PygWalker at 0x7bb52f4d4d00>
```

2 Questions

1. What is the most popular car **make** and **colour**?
2. What was the most **expensive** Mazda car sold?
3. How many **"green"** cars were sold in each **city** during the **period** 2023-10-13 to 2024-02-02?
4. What is the average **price** paid for a car in each **city**?
5. What are the top 5 most **popular** cars in each **city** for all vehicles sold in **2024**?
6. Is there any relationship between **price** and **discount** given?
7. What is the **total unit sales** and **total revenue** for each **month** for each **city**?
8. A key stakeholder has asked how to make the **most amount of revenue** from selling cars, what advice would you give them to best achieve this?
9. (Optional) Are there any other interesting observations in the data?

3 1. What is the most popular car make and colour?

For this problem, I need to:

- Join the tables on `vehicle_id`
- Use count aggregation to find highest count
- Order in descending order
- Use columns `make` and `exterior_color` (not `interior_color`)

I tried using MySQL queries, but wanted to double check the results, so I then used `pandas` - much quicker to execut - which both returned the same result.

So, I will use the MySQL queries from now on, which are easier to formulate than pandas.

ANSWER: The most popular car make is **RAM** and most popular colour is **Bright White Clearcoat**.

[17]:
```
# Define the MySQL query
query = """
SELECT make, exterior_color, COUNT(*) as count
FROM purchase_data p
JOIN vehicle_data v
ON p.vehicle_id = v.vehicle_id
GROUP BY make, exterior_color
ORDER BY count DESC
LIMIT 1
"""

# Execute the SQL query
most_popular = psql.sqldf(query, locals())

# Print the result
print("Most popular car make and color:")
print(most_popular)
```

```
Most popular car make and color:
  make          exterior_color   count
0  RAM  Bright White Clearcoat  111104
```

[31]:
```
# Define the query for the most popular car make and color combination
query_combination = """
SELECT make, exterior_color, COUNT(*) as count
FROM purchase_data p
JOIN vehicle_data v
ON p.vehicle_id = v.vehicle_id
GROUP BY make, exterior_color
ORDER BY count DESC
LIMIT 1
"""

# Execute the SQL query
most_popular_combination = psql.sqldf(query_combination, locals())

# Print the result
print("Most popular car make and color combination:")
print(most_popular_combination)
```

```
Most popular car make and color combination:
  make          exterior_color   count
0  RAM  Bright White Clearcoat  111104
```

[32]:
```
# Define the query for the most popular car make (ignoring color)
query_make = """
SELECT make, COUNT(*) as count
FROM purchase_data p
```

```
JOIN vehicle_data v
ON p.vehicle_id = v.vehicle_id
GROUP BY make
ORDER BY count DESC
LIMIT 1
"""

# Execute the SQL query
most_popular_make = psql.sqldf(query_make, locals())

# Print the result
print("Most popular car make:")
print(most_popular_make)
```

```
Most popular car make:
   make   count
0  Jeep  446898
```

[33]:
```
# Define the query for the most popular car color (ignoring make)
query_color = """
SELECT exterior_color, COUNT(*) as count
FROM purchase_data p
JOIN vehicle_data v
ON p.vehicle_id = v.vehicle_id
GROUP BY exterior_color
ORDER BY count DESC
LIMIT 1
"""

# Execute the SQL query
most_popular_color = psql.sqldf(query_color, locals())

# Print the result
print("Most popular car color:")
print(most_popular_color)
```

```
Most popular car color:
           exterior_color   count
0  Bright White Clearcoat  204610
```

[29]:
```
# Merging the two DataFrames on 'vehicle_id'
merged_data = pd.merge(purchase_data, vehicle_data, on='vehicle_id')

# Grouping by 'make' and 'external_color' and counting the occurrences
grouped_data = merged_data.groupby(['make', 'exterior_color']).size().
 ↪reset_index(name='count')

# Finding the most popular car make and color
```

```
most_popular = grouped_data.loc[grouped_data['count'].idxmax()]

print(f"The most popular car make was {most_popular['make']} with color␣
 ↪{most_popular['exterior_color']} (count {most_popular['count']})")
#print(f"Make: {most_popular['make']}, Color: {most_popular['exterior_color']},␣
 ↪Count: {most_popular['count']}")
```

```
The most popular car make was RAM with color Bright White Clearcoat (count
111104)
```

[30]:
```
# Merging the two DataFrames on 'vehicle_id'
merged_data = pd.merge(purchase_data, vehicle_data, on='vehicle_id')

# Grouping by 'make' and 'exterior_color' and counting the occurrences
grouped_data = merged_data.groupby(['make', 'exterior_color']).size().
 ↪reset_index(name='count')

# Finding the most popular car make and color combination
most_popular_combination = grouped_data.loc[grouped_data['count'].idxmax()]
print(f"The most popular car make and color combination was␣
 ↪{most_popular_combination['make']} with color␣
 ↪{most_popular_combination['exterior_color']} (count␣
 ↪{most_popular_combination['count']})")

# Finding the most popular car make (ignoring color)
grouped_make = merged_data.groupby('make').size().reset_index(name='count')
most_popular_make = grouped_make.loc[grouped_make['count'].idxmax()]
print(f"The most popular car make was {most_popular_make['make']} (count␣
 ↪{most_popular_make['count']})")

# Finding the most popular car color (ignoring make)
grouped_color = merged_data.groupby('exterior_color').size().
 ↪reset_index(name='count')
most_popular_color = grouped_color.loc[grouped_color['count'].idxmax()]
print(f"The most popular car color was {most_popular_color['exterior_color']}␣
 ↪(count {most_popular_color['count']})")
```

```
The most popular car make and color combination was RAM with color Bright White
Clearcoat (count 111104)
The most popular car make was Jeep (count 446898)
The most popular car color was Bright White Clearcoat (count 204610)
```

4 2. What was the most expensive Mazda car sold?

I was not sure what information to provide for the car. I started with all columns (`SELECT *`) from the `purchase_data`, and then narrowed down to more specific columns.

I used `ORDER BY` rather than `MAX()` to return the row information, not just the price.

ANSWER: The most expensive Mazda car sold was a 2024 **Mazda CX-90 PHEV Base** in **Jet Black Mica** for 60,200.0, bought in Southampton

[19]:
```
# Define the MySQL query
query = """
SELECT p.customer_id, p.purchase_date, p.city, p.vehicle_id,
    v.name, v.description, v.model, v.year, v.cylinders, v.fuel, v.
 ↪transmission, v.body, v.doors, v.exterior_color, v.drivetrain, v.price
FROM purchase_data p
JOIN vehicle_data v
ON p.vehicle_id = v.vehicle_id
WHERE make = "Mazda"
ORDER BY v.price DESC
LIMIT 1
"""

# Execute the SQL query
most_expensive = psql.sqldf(query, locals())

# Print the result
print("Most expensive Mazda car sold:")
print(most_expensive)
```

```
Most expensive Mazda car sold:
   customer_id purchase_date         city  vehicle_id  \
0           67    2023-10-22  Southampton         140

                         name  \
0  2024 Mazda CX-90 PHEV Base

                                        description       model  year  \
0  Since 1960, Wheeler Chevrolet Cadillac Mazda h…  CX-90 PHEV  2024

   cylinders      fuel transmission body  doors  exterior_color  \
0        4.0  Gasoline    Automatic  SUV    4.0  Jet Black Mica

        drivetrain    price
0  All-wheel Drive  60200.0
```

[20]:
```
most_expensive
```

[20]:
```
    customer_id purchase_date         city  vehicle_id  \
0           67    2023-10-22  Southampton         140

                         name  \
0  2024 Mazda CX-90 PHEV Base

                                        description       model  year  \
```

```
0  Since 1960, Wheeler Chevrolet Cadillac Mazda h…  CX-90 PHEV  2024

   cylinders      fuel transmission body  doors  exterior_color  \
0        4.0  Gasoline    Automatic  SUV    4.0  Jet Black Mica

        drivetrain    price
0  All-wheel Drive  60200.0
```

[25]:
```
car_name = most_expensive['name'].iloc[0]
car_price = most_expensive['price'].iloc[0]  # or .values[0] or .item()
print(f"The most expensive Mazda car sold was a {car_name} priced at␣
 ↪${car_price}.")
```

```
The most expensive Mazda car sold was a 2024 Mazda CX-90 PHEV Base priced at
$60200.0.
```

5 3. How many "green" cars were sold in each city during the period 2023-10-13 to 2024-02-02?

At first, I used HAVING to filter the data range, after grouping by city, but the DataFrame returned empty.

I changed the query to include the date filtering within the WHERE clause.

ANSWER:

During this period, the most green cars were sold in Nottingham.

[37]:
```
# Define the MySQL query
query = """
SELECT p.city, COUNT(*) as count
FROM purchase_data p
JOIN vehicle_data v
ON p.vehicle_id = v.vehicle_id
WHERE v.exterior_color = "Green"
AND p.purchase_date BETWEEN "2023-10-13" AND "2024-02-02"
GROUP BY p.city
ORDER BY count DESC
"""

# Execute the SQL query
green_cars = psql.sqldf(query, locals())

# Print the result
print("Number of green cars sold in each city from 2023-10-13 to 2024-02-02: ")
print(green_cars)
```

```
Number of green cars sold in each city from 2023-10-13 to 2024-02-02:
          city  count
```

```
0   Nottingham    287
1  Southampton    272
2      Bristol    167
3    Liverpool    128
4   Manchester     77
```

6 4. What is the average price paid for a car in each city?

[39]:
```
# Define the MySQL query
query = """
SELECT city, AVG(price) AS average_price
FROM purchase_data p
JOIN vehicle_data v
ON p.vehicle_id = v.vehicle_id
GROUP BY p.city
ORDER BY average_price DESC
"""

# Execute the SQL query
average_price_by_city = psql.sqldf(query, locals())

# Print the result
print("Average price for cars in each city: ")
print(average_price_by_city)
```

```
Average price for cars in each city:
          city  average_price
0   Nottingham   63240.205154
1   Manchester   61281.542211
2         Bath   61264.517471
3      Bristol   60959.203213
4   Birmingham   58456.433227
5    Liverpool   57768.446794
6  Southampton   57110.310149
```

7 5. What are the top 5 most popular cars in each city for all vehicles sold in 2024?

This was a bit tricky, because:

- I need to group by city and by car name, and then count the occurrences.
- I need to rank the cars within each city and select the top 5.

These are a bit tricky to do with `pandasql` and SQL, so I will use pandas instead, which makes the logic easier to see.

- Dates are given in YYYY-MM-DD format, not YYYY format. To get around this with SQL, I would use LIKE, but with pandas, I convert the `purchase_date` column to DateTime, and

use the `dt.year` (datetime) method to filter for the year.

[41]:
```
# Merge DataFrames on vehicle_id; this is like SQL JOIN
merged_data = pd.merge(purchase_data, vehicle_data, on='vehicle_id')

# Filter for sales in 2024
merged_data['purchase_date'] = pd.to_datetime(merged_data['purchase_date'])
filtered_data = merged_data[merged_data['purchase_date'].dt.year == 2024]

# Group by city and car name, then count the occurrences; similar to GROUP BY␣
 ↪in SQL
grouped_data = filtered_data.groupby(['city', 'name']).size().
 ↪reset_index(name='count')

# Sort by city and then by count in descending order, and rank within each city
grouped_data = grouped_data.sort_values(['city', 'count'], ascending=[True,␣
 ↪False])
grouped_data['rank'] = grouped_data.groupby('city')['count'].
 ↪rank(method='first', ascending=False)

# Filter to get the top 5 cars per city
top_5_cars_per_city = grouped_data[grouped_data['rank'] <= 5]

# Display the results
print("Top 5 most popular cars in each city for vehicles sold in 2024: ")
print(top_5_cars_per_city)
```

```
Top 5 most popular cars in each city for vehicles sold in 2024:
            city                                    name  count  rank
112         Bath          2024 Jeep Grand Cherokee 4xe Base   1578   1.0
171         Bath                   2024 RAM 3500 Tradesman   1396   2.0
169         Bath                     2024 RAM 3500 Laramie   1157   3.0
117         Bath         2024 Jeep Grand Cherokee L Limited   1129   4.0
168         Bath                    2024 RAM 3500 Big Horn   1053   5.0
261   Birmingham                   2024 RAM 3500 Tradesman   1186   1.0
241   Birmingham                   2024 Jeep Wagoneer Base    579   2.0
223   Birmingham                  2024 GMC Yukon XL Denali    513   3.0
260   Birmingham                     2024 RAM 3500 Laramie    485   4.0
190   Birmingham                2023 Dodge Durango Pursuit    371   5.0
322      Bristol           2024 Jeep Grand Cherokee Limited    550   1.0
272      Bristol                2023 Dodge Durango Pursuit    517   2.0
339      Bristol     2024 Mercedes-Benz EQS 450 Base 4MATIC    513   3.0
281      Bristol                           2024 BMW i5 M60    490   4.0
292      Bristol                2024 Dodge Hornet R/T Plus    393   5.0
415    Liverpool                2024 Mazda CX-90 PHEV Base    876   1.0
411    Liverpool             2024 Jeep Wrangler 4xe Rubicon    752   2.0
422    Liverpool                   2024 RAM 3500 Tradesman    543   3.0
354    Liverpool                2023 Dodge Durango Pursuit    496   4.0
```

```
423    Liverpool                2024 RAM ProMaster 3500 Base    440   5.0
512   Manchester                     2024 RAM 2500 Tradesman    518   1.0
448   Manchester           2024 Chevrolet Silverado 1500 LT    348   2.0
471   Manchester                2024 Honda Prologue TOURING    333   3.0
435   Manchester                 2023 Dodge Durango Pursuit    316   4.0
465   Manchester           2024 Ford Mustang Mach-E Premium    305   5.0
521   Nottingham                 2023 Dodge Durango Pursuit    965   1.0
579   Nottingham              2024 Mazda CX-90 PHEV Premium    521   2.0
527   Nottingham                2024 Audi Q8 e-tron Premium    455   3.0
570   Nottingham               2024 Jeep Wagoneer Series II    426   4.0
586   Nottingham                      2024 RAM 2500 Laramie    404   5.0
668  Southampton                2024 Mazda CX-90 PHEV Base    542   1.0
601  Southampton                 2023 Dodge Durango Pursuit    477   2.0
678  Southampton  2024 RAM 3500 Laramie Crew Cab 4x4 8' Box    461   3.0
634  Southampton                   2024 Hyundai IONIQ 5 SEL    355   4.0
670  Southampton              2024 Mazda CX-90 PHEV Premium    330   5.0
```

8 6. Is there any relationship between price and discount given?

I used `matplotlib` but then changed to `plotly` to make the points easier to identify interactively.

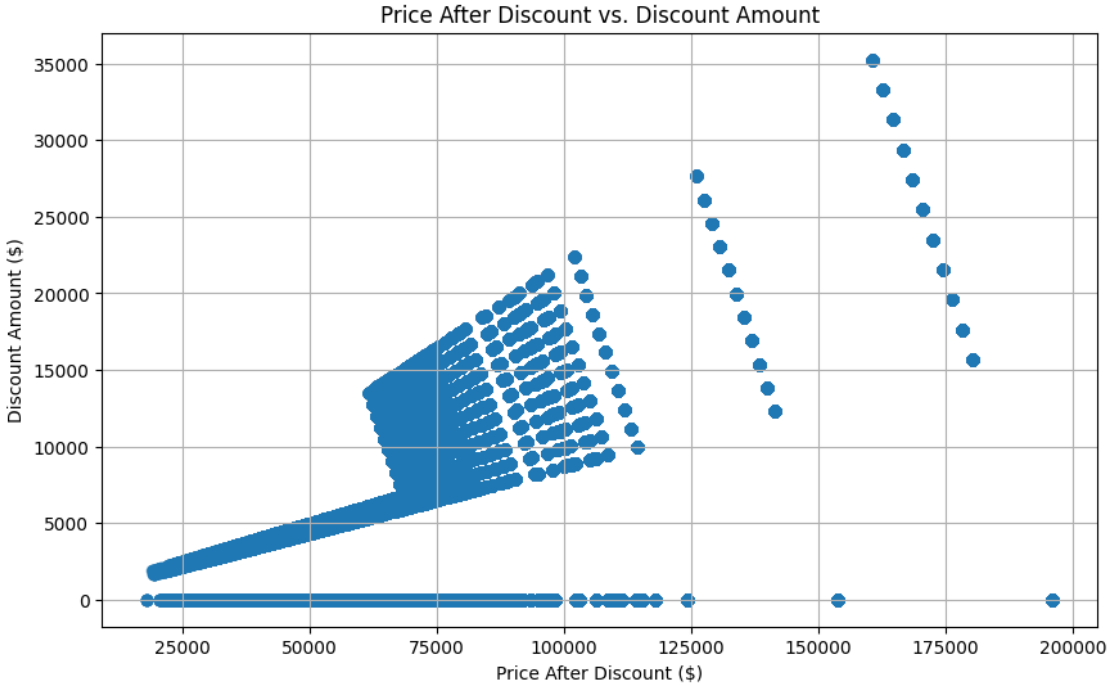
Overall, the relationship between price and discount given appears positive and linear, so as price increases, the discount increases.

There seems to be 12 relatively distinct lines, which might be worth further investigating.

[44]:
```python
# Merge the DataFrames on 'vehicle_id' to get both price and rrp_discount in a␣
 ↪single DataFrame
merged_data = pd.merge(purchase_data, vehicle_data, on='vehicle_id')

# Calculate the discount amount
merged_data['discount_amount'] = merged_data['price'] *␣
 ↪merged_data['rrp_discount']

# Calculate the final price after discount
merged_data['price_after_discount'] = merged_data['price'] -␣
 ↪merged_data['discount_amount']

# Calculate correlation between price and discount amount
correlation = merged_data['discount_amount'].
 ↪corr(merged_data['price_after_discount'])
print(f"Correlation between price after discount and discount amount:␣
 ↪{correlation}")

# Plotting price_after_discount vs. discount_amount
plt.figure(figsize=(10, 6))
plt.scatter(merged_data['price_after_discount'],␣
 ↪merged_data['discount_amount'], alpha=0.6)
```

```
plt.title('Price After Discount vs. Discount Amount')
plt.xlabel('Price After Discount ($)')
plt.ylabel('Discount Amount ($)')
plt.grid(True)
plt.show()
```

```
Correlation between price after discount and discount amount: 0.3797009116985185
```



[3]:
```
# Merge the DataFrames on 'vehicle_id' to get both price and rrp_discount in a␣
 ↪single DataFrame
merged_data = pd.merge(purchase_data, vehicle_data, on='vehicle_id')

# Calculate the discount amount
merged_data['discount_amount'] = merged_data['price'] *␣
 ↪merged_data['rrp_discount']

# Calculate the final price after discount
merged_data['price_after_discount'] = merged_data['price'] -␣
 ↪merged_data['discount_amount']

# Create the scatter plot using Plotly Express
fig = px.scatter(
    merged_data,
    x='price_after_discount',
```

```
    y='discount_amount',
    title='Price After Discount vs. Discount Amount',
    labels={'price_after_discount': 'Price After Discount ($)',␣
 ↪'discount_amount': 'Discount Amount ($)'},
    opacity=0.7,
    size_max=10,
    template='plotly_dark'  # Set dark mode background
)

# Size of points
fig.update_traces(marker=dict(size=8))

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

[5]:
```
# Merge the DataFrames on 'vehicle_id' to get both price and rrp_discount in a␣
 ↪single DataFrame
merged_data = pd.merge(purchase_data, vehicle_data, on='vehicle_id')

# Calculate the discount amount
merged_data['discount_amount'] = merged_data['price'] *␣
 ↪merged_data['rrp_discount']

# Calculate the final price after discount
merged_data['price_after_discount'] = merged_data['price'] -␣
 ↪merged_data['discount_amount']

# Set the style of the visualization
sns.set(style="darkgrid")

# Create the scatter plot using Seaborn
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=merged_data,
    x='price_after_discount',
    y='discount_amount',
    hue='city',  # Color points by city
    palette='viridis',  # Set the color palette
    s=100,  # Size of points
    alpha=0.7
)

# Set plot title and labels
plt.title('Price After Discount vs. Discount Amount')
plt.xlabel('Price After Discount ($)')
plt.ylabel('Discount Amount ($)')
```

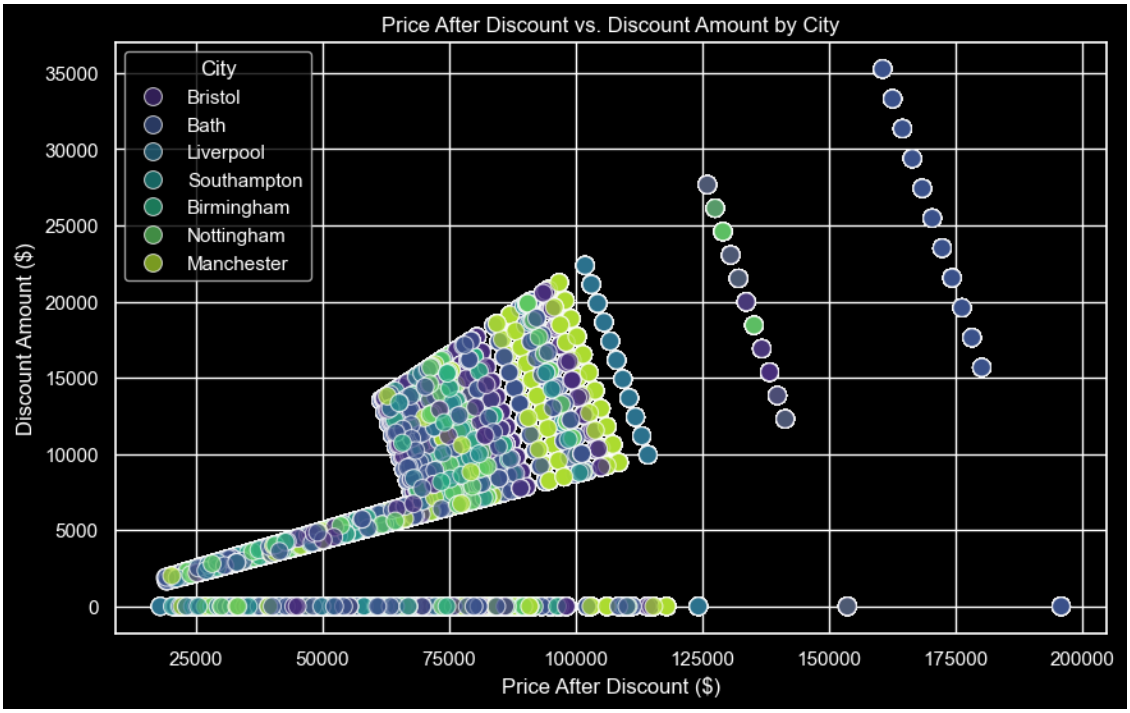
```
# Show the plot
plt.legend(title='City')
plt.show()
```

```
/home/solaris/miniconda3/envs/car-sales/lib/python3.10/site-
packages/IPython/core/pylabtools.py:170: UserWarning: Creating legend with
loc="best" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```

[9]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Merge the DataFrames on 'vehicle_id'
merged_data = pd.merge(purchase_data, vehicle_data, on='vehicle_id')

# Calculate the discount amount
merged_data['discount_amount'] = merged_data['price'] *␣
 ↪merged_data['rrp_discount']

# Calculate the final price after discount
merged_data['price_after_discount'] = merged_data['price'] -␣
 ↪merged_data['discount_amount']
```

```python
# Set dark mode
sns.set_style("darkgrid")
plt.style.use("dark_background")  # inverts colors to dark theme

# Plot: Price after discount vs. Discount amount colored by City
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=merged_data,
    x='price_after_discount',
    y='discount_amount',
    hue='city',  # Color points by city
    palette='viridis',
    s=100,  # Size of points
    alpha=0.7
)

# Set plot title and labels
plt.title('Price After Discount vs. Discount Amount by City')
plt.xlabel('Price After Discount ($)')
plt.ylabel('Discount Amount ($)')
plt.legend(title='City')
plt.show()
```

```
/home/solaris/miniconda3/envs/car-sales/lib/python3.10/site-
packages/IPython/core/pylabtools.py:170: UserWarning: Creating legend with
loc="best" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```



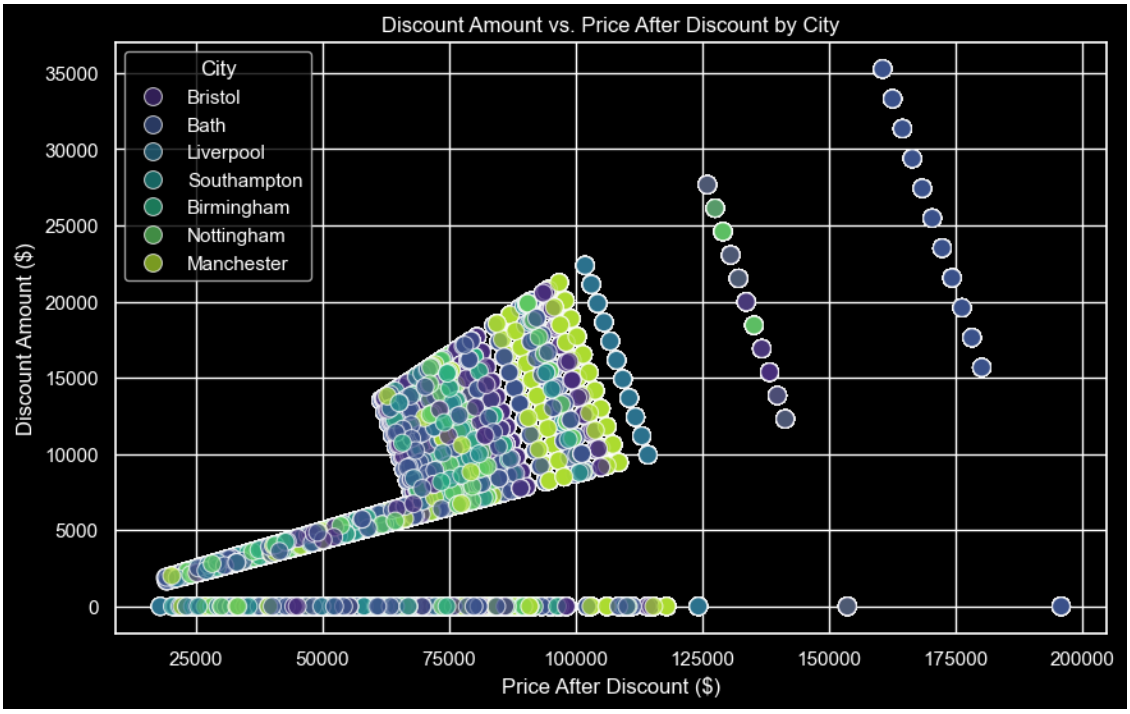
[12]:
```
# Merge and calculate
merged_data = pd.merge(purchase_data, vehicle_data, on='vehicle_id')
merged_data['discount_amount'] = merged_data['price'] *␣
 ↪merged_data['rrp_discount']

# Set dark mode
sns.set_style("darkgrid")
plt.style.use("dark_background")  # inverts colors to dark theme

# Plot discount amount vs. price before discount, colored by city
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=merged_data,
    x='price',  # Original price before discount
    y='discount_amount',
    hue='city',  # Color points by city
    palette='viridis',
    s=100,  # Size of points
    alpha=0.7
)

# Set plot title and labels
plt.title('Discount Amount vs. Price Before Discount by City')
plt.xlabel('Price Before Discount ($)')
plt.ylabel('Discount Amount ($)')
plt.legend(title='City')
plt.show()
```

```
/home/solaris/miniconda3/envs/car-sales/lib/python3.10/site-
packages/IPython/core/pylabtools.py:170: UserWarning: Creating legend with
loc="best" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```

[13]:
```
# Merge and calculate
merged_data = pd.merge(purchase_data, vehicle_data, on='vehicle_id')
merged_data['discount_amount'] = merged_data['price'] *␣
 ↪merged_data['rrp_discount']
merged_data['price_after_discount'] = merged_data['price'] -␣
 ↪merged_data['discount_amount']

# Set dark mode
sns.set_style("darkgrid")
plt.style.use("dark_background")  # Inverts colors to dark theme

# Plot price after discount vs. discount amount, colored by city
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=merged_data,
    x='price_after_discount',  # Price after discount
    y='discount_amount',       # Discount amount
    hue='city',                # Color points by city
    palette='viridis',
    s=100,                     # Size of points
    alpha=0.7
)

# Set plot title and labels
```

```
plt.title('Discount Amount vs. Price After Discount by City')
plt.xlabel('Price After Discount ($)')
plt.ylabel('Discount Amount ($)')
plt.legend(title='City')
plt.show()
```

```
/home/solaris/miniconda3/envs/car-sales/lib/python3.10/site-
packages/IPython/core/pylabtools.py:170: UserWarning: Creating legend with
loc="best" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```



[17]:
```
# Convert relevant non-numeric columns to categorical
categorical_columns_purchase = [
    'customer_uuid', 'purchase_date', 'city'
]

categorical_columns_vehicle = [
    'name', 'description', 'make', 'model', 'type', 'engine', 'fuel',␣
 ↪'transmission', 'trim', 'body', 'exterior_color', 'interior_color',␣
 ↪'drivetrain'
]

# Convert these columns to categorical types
for col in categorical_columns_purchase:
    purchase_data[col] = purchase_data[col].astype('category')
```

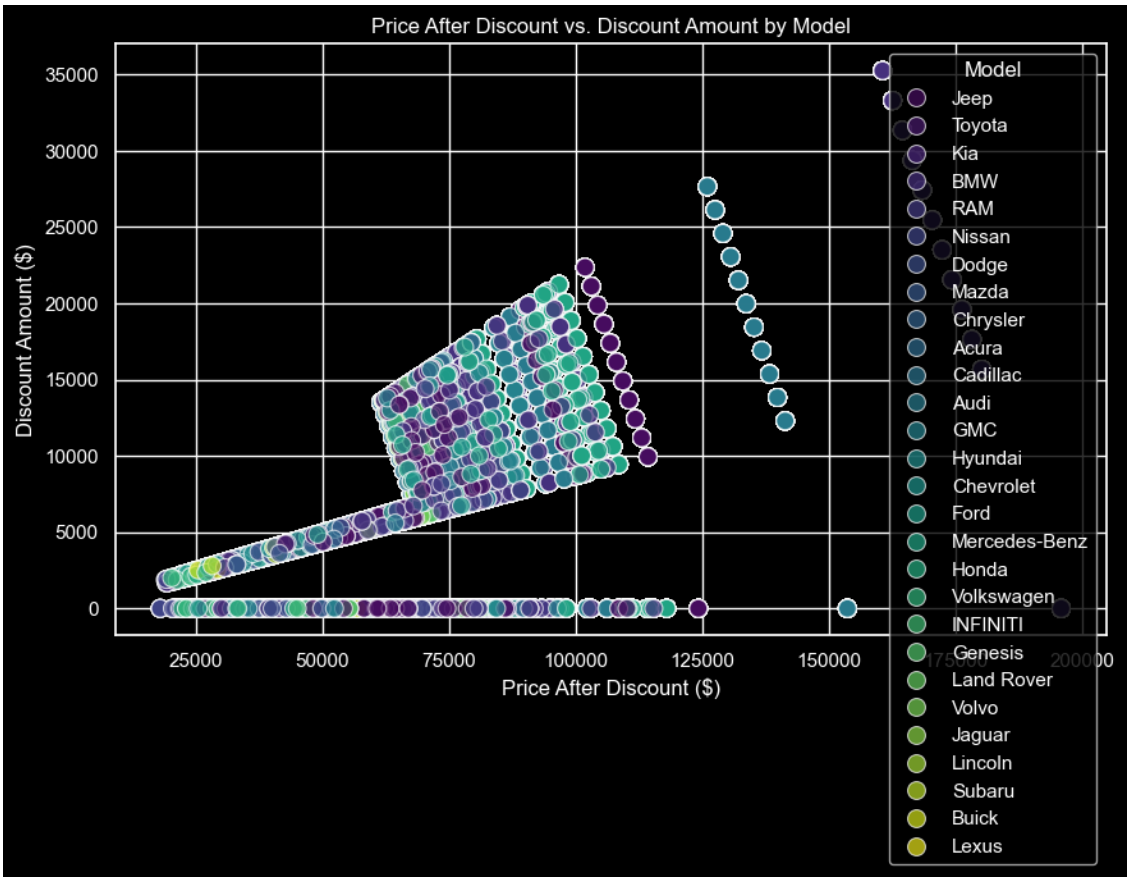
```
for col in categorical_columns_vehicle:
    vehicle_data[col] = vehicle_data[col].astype('category')

# Check unique levels
def check_unique_levels(df, categorical_columns):
    print(f"Checking number of unique levels for categorical columns:")
    for col in categorical_columns:
        unique_levels = df[col].nunique()
        print(f"Column '{col}' has {unique_levels} unique levels.")

print("Categorical variables in purchase_data:")
check_unique_levels(purchase_data, categorical_columns_purchase)

print("\nCategorical variables in vehicle_data:")
check_unique_levels(vehicle_data, categorical_columns_vehicle)
```

```
Categorical variables in purchase_data:
Checking number of unique levels for categorical columns:
Column 'customer_uuid' has 2000000 unique levels.
Column 'purchase_date' has 1552 unique levels.
Column 'city' has 7 unique levels.

Categorical variables in vehicle_data:
Checking number of unique levels for categorical columns:
Column 'name' has 353 unique levels.
Column 'description' has 739 unique levels.
Column 'make' has 28 unique levels.
Column 'model' has 150 unique levels.
Column 'type' has 1 unique levels.
Column 'engine' has 99 unique levels.
Column 'fuel' has 7 unique levels.
Column 'transmission' has 38 unique levels.
Column 'trim' has 197 unique levels.
Column 'body' has 8 unique levels.
Column 'exterior_color' has 261 unique levels.
Column 'interior_color' has 90 unique levels.
Column 'drivetrain' has 4 unique levels.
```

[11]:
```
# Plot: Price after discount vs. Discount amount colored by Make
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=merged_data,
    x='price_after_discount',
    y='discount_amount',
    hue='make',  # Color points by make
    palette='viridis',  # Colorblind-friendly palette
```

```
    s=100,  # Size of points
    alpha=0.7
)

# Set plot title and labels
plt.title('Price After Discount vs. Discount Amount by Make')
plt.xlabel('Price After Discount ($)')
plt.ylabel('Discount Amount ($)')
plt.legend(title='Model')
plt.show()
```

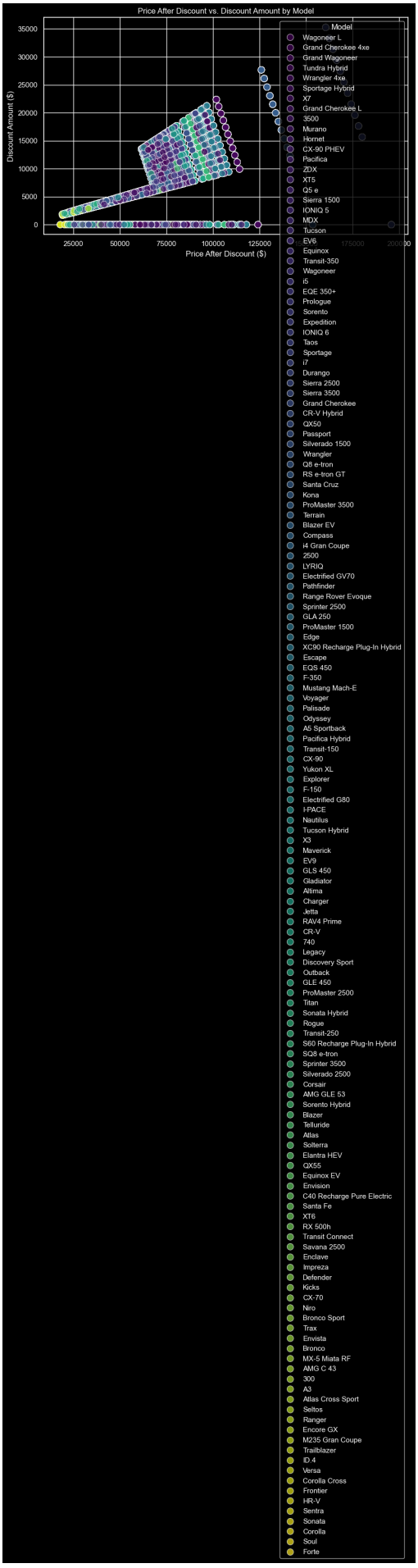
```
/home/solaris/miniconda3/envs/car-sales/lib/python3.10/site-
packages/IPython/core/pylabtools.py:170: UserWarning: Creating legend with
loc="best" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```



[15]:
```
# Plot: Price after discount vs. Discount amount colored by Model
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=merged_data,
```

```
    x='price_after_discount',
    y='discount_amount',
    hue='model',  # Color points by model
    palette='viridis',  # Colorblind-friendly palette
    s=100,  # Size of points
    alpha=0.7
)

# Set plot title and labels
plt.title('Price After Discount vs. Discount Amount by Model')
plt.xlabel('Price After Discount ($)')
plt.ylabel('Discount Amount ($)')
plt.legend(title='Model')
plt.show()
```

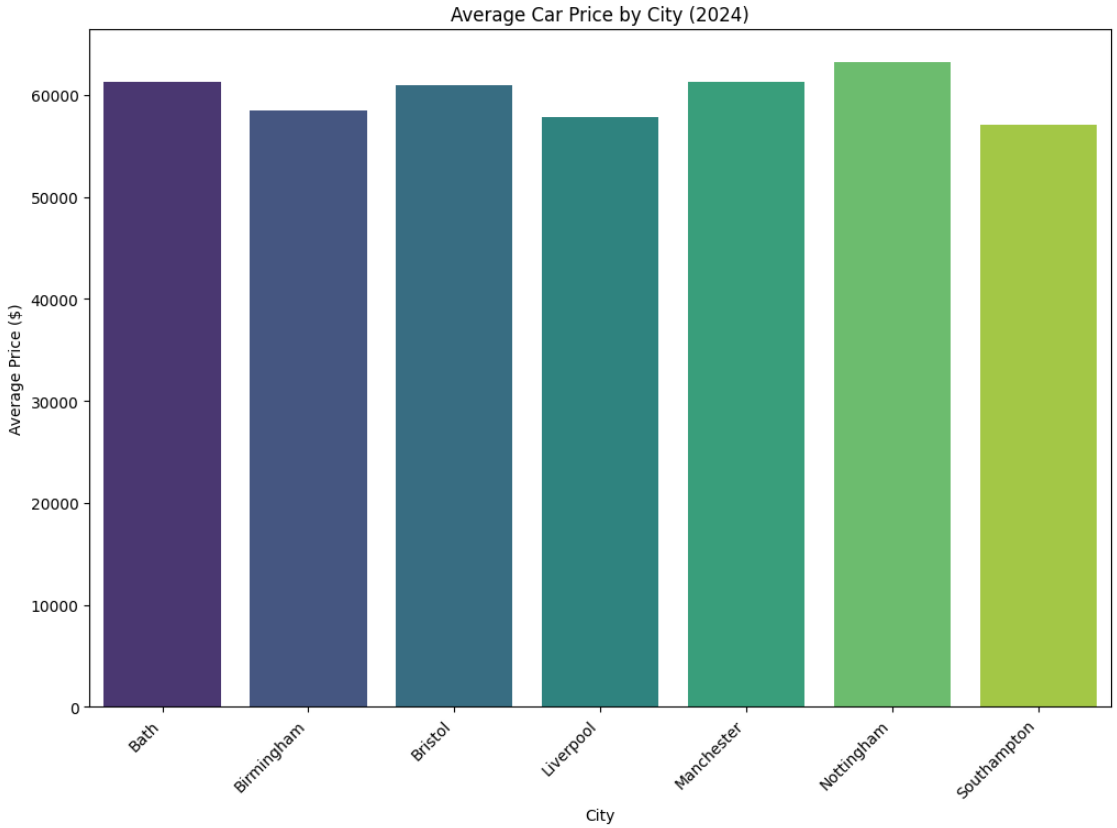
```
/home/solaris/miniconda3/envs/car-sales/lib/python3.10/site-
packages/IPython/core/pylabtools.py:170: UserWarning: Creating legend with
loc="best" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```


Price After Discount vs. Discount Amount by Model
Discount Amount ($)
Price After Discount ($)
35000
30000
25000
20000
15000
10000
5000
0
25000
50000
75000
100000
125000
175000
Model
Wagoneer L
Grand Cherokee 4xe
Grand Wagoneer
Tundra Hybrid
Wrangler 4xe
Sportage Hybrid
X7
Grand Cherokee L
3500
Murano
Hornet
CX-90 PHEV
Pacifica
ZDX
XT5
Q5 e
Sierra 1500
IONIQ 5
MDX
Tucson
EV6
Equinox
Transit-350
Wagoneer
i5
EQE 350+
Prologue
Sorento
Expedition
IONIQ 6
Taos
Sportage
i7
Durango
Sierra 2500
Sierra 3500
Grand Cherokee
CR-V Hybrid
QX50
Passport
Silverado 1500
Wrangler
Q8 e-tron
RS e-tron GT
Santa Cruz
Kona
ProMaster 3500
Terrain
Blazer EV
Compass
i4 Gran Coupe
2500
LYRIQ
Electrified GV70
Pathfinder
Range Rover Evoque
Sprinter 2500
GLA 250
ProMaster 1500
Edge
XC90 Recharge Plug-In Hybrid
Escape
EQS 450
F-350
Mustang Mach-E
Voyager
Palisade
Odyssey
A5 Sportback
Pacifica Hybrid
Transit-150
CX-90
Yukon XL
Explorer
F-150
Electrified G80
I-PACE
Nautilus
Tucson Hybrid
X3
Maverick
EV9
GLS 450
Gladiator
Altima
Charger
Jetta
RAV4 Prime
CR-V
740
Legacy
Discovery Sport
Outback
GLE 450
ProMaster 2500
Titan
Sonata Hybrid
Rogue
Transit-250
S60 Recharge Plug-In Hybrid
SQ8 e-tron
Sprinter 3500
Silverado 2500
Corsair
AMG GLE 53
Sorento Hybrid
Blazer
Telluride
Atlas
Solterra
Elantra HEV
QX55
Equinox EV
Envision
C40 Recharge Pure Electric
Santa Fe
XT6
RX 500h
Transit Connect
Savana 2500
Enclave
Impreza
Defender
Kicks
CX-70
Niro
Bronco Sport
Trax
Envista
Bronco
MX-5 Miata RF
AMG C 43
300
A3
Atlas Cross Sport
Seltos
Ranger
Encore GX
M235 Gran Coupe
Trailblazer
ID.4
Versa
Corolla Cross
Frontier
HR-V
Sentra
Sonata
Corolla
Soul
Forte

[5]:
```
# Group by city and calculate the average price
average_price_by_city = merged_data.groupby('city')['price'].mean().
 ↪reset_index(name='average_price')

# Print the results
print("Average price of cars by city in 2024:")
for index, row in average_price_by_city.iterrows():
    print(f"City: {row['city']}, Average Price: ${row['average_price']:.2f}")

# Plot the data
plt.figure(figsize=(12, 8))
sns.barplot(
    data=average_price_by_city,
    x='city',
    y='average_price',
    palette='viridis'  # Colorblind-friendly palette
)

# Set plot title and labels
plt.title('Average Car Price by City (2024)')
plt.xlabel('City')
plt.ylabel('Average Price ($)')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better␣
 ↪readability

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

```
Average price of cars by city in 2024:
City: Bath, Average Price: $61264.52
City: Birmingham, Average Price: $58456.43
City: Bristol, Average Price: $60959.20
City: Liverpool, Average Price: $57768.45
City: Manchester, Average Price: $61281.54
City: Nottingham, Average Price: $63240.21
City: Southampton, Average Price: $57110.31
```

```
/tmp/ipykernel_135880/3408398468.py:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(
```



9 7. What is the total unit sales and total revenue for each month for each city?

I want to do this with SQL, however, the DATE_FORMAT() function does not work with `pandasql`.

So, I will create a new DataFrame with the month extracted, and use this in my SQL query.

The table is quite difficult to read and interpret, so I decide to plot it.

The plots show a difference, which might be statistically significant, between unit sales and revenues between Bath, and other cities, with Bath having higher sales/revenues.

[5]:
```
# Convert purchase_date to datetime
purchase_data['purchase_date'] = pd.to_datetime(purchase_data['purchase_date'])

# Create a new column for month (formatted as 'YYYY-MM')
purchase_data['purchase_month'] = purchase_data['purchase_date'].dt.
  ↪to_period('M').astype(str)

# Define the MySQL query
```

```
query = """
SELECT
    p.purchase_month,
    p.city,
    COUNT(*) AS total_unit_sales,
    SUM(v.price) AS total_revenue
FROM purchase_data p
JOIN vehicle_data v
ON p.vehicle_id = v.vehicle_id
GROUP BY p.purchase_month, p.city
ORDER BY p.purchase_month, p.city;
"""

# Execute the SQL query
result = psql.sqldf(query, locals())

# Print the result
print("Total unit sales and total revenue for each month and city:")
print(result)
```

```
Total unit sales and total revenue for each month and city:
    purchase_month         city  total_unit_sales  total_revenue
0          2020-01         Bath             13572    831286176.0
1          2020-01   Birmingham              4459    261352924.0
2          2020-01      Bristol              4453    268963876.0
3          2020-01    Liverpool              4337    251318958.0
4          2020-01   Manchester              4470    272697744.0
..             …          …                 …            …
352        2024-03      Bristol              4386    267236929.0
353        2024-03    Liverpool              4433    255333570.0
354        2024-03   Manchester              4389    269096281.0
355        2024-03   Nottingham              4422    279895767.0
356        2024-03  Southampton              4427    253623104.0

[357 rows x 4 columns]
```

[8]:
```
# Convert purchase_date to datetime
purchase_data['purchase_date'] = pd.to_datetime(purchase_data['purchase_date'])

# Create a new column for month (formatted as 'YYYY-MM')
purchase_data['purchase_month'] = purchase_data['purchase_date'].dt.
  ↪to_period('M').astype(str)

# Merge with vehicle_data to include price information
merged_data = pd.merge(purchase_data, vehicle_data, on='vehicle_id')

# Aggregate data: Total unit sales and total revenue by month and city
```

```
monthly_summary = merged_data.groupby(['purchase_month', 'city']).agg(
    total_unit_sales=('vehicle_id', 'count'),
    total_revenue=('price', 'sum')
).reset_index()

# Convert purchase_month to datetime for plotting
monthly_summary['purchase_month'] = pd.
  ↪to_datetime(monthly_summary['purchase_month'], format='%Y-%m')

# Plot total unit sales
fig1 = px.line(
    monthly_summary,
    x='purchase_month',
    y='total_unit_sales',
    color='city',
    markers=True,
    title='Total Unit Sales Over Time by City',
    labels={'purchase_month': 'Month', 'total_unit_sales': 'Total Unit Sales'}
)

# Experiment with different ways to show cities:
# Use different line styles for cities in total unit sales plot
fig1.update_traces(
    line=dict(width=2, dash='solid')  # Solid lines for unit sales
)
fig1.update_layout(
    legend_title='City',
    template='plotly_dark'
)

# Plot total revenue
fig2 = px.line(
    monthly_summary,
    x='purchase_month',
    y='total_revenue',
    color='city',
    markers=True,
    title='Total Revenue Over Time by City',
    labels={'purchase_month': 'Month', 'total_revenue': 'Total Revenue'}
)

# Experiment with different ways to show cities:
# Use different marker shapes for cities in total revenue plot
fig2.update_traces(
    mode='markers+lines',
    marker=dict(size=10, symbol='circle')  # Circular markers for revenue
)
```

```
fig2.update_layout(
    legend_title='City',
    template='plotly_dark'
)

# Show the plots
fig1.show()
fig2.show()
```

[9]:
```
# Convert purchase_date to datetime
purchase_data['purchase_date'] = pd.to_datetime(purchase_data['purchase_date'])

# Create a new column for month (formatted as 'YYYY-MM')
purchase_data['purchase_month'] = purchase_data['purchase_date'].dt.
 ↪to_period('M').astype(str)

# Merge with vehicle_data to include price information
merged_data = pd.merge(purchase_data, vehicle_data, on='vehicle_id')

# Aggregate data: Total unit sales and total revenue by month and city
monthly_summary = merged_data.groupby(['purchase_month', 'city']).agg(
    total_unit_sales=('vehicle_id', 'count'),
    total_revenue=('price', 'sum')
).reset_index()

# Convert purchase_month to datetime for plotting
monthly_summary['purchase_month'] = pd.
 ↪to_datetime(monthly_summary['purchase_month'], format='%Y-%m')

# Plot total unit sales with log scale
fig1 = px.line(
    monthly_summary,
    x='purchase_month',
    y='total_unit_sales',
    color='city',
    markers=True,
    title='Total Unit Sales Over Time by City',
    labels={'purchase_month': 'Month', 'total_unit_sales': 'Total Unit Sales'}
)

# Apply log scale to the y-axis
fig1.update_layout(
    yaxis_type='log',
    yaxis_title='Total Unit Sales',
    legend_title='City',
    template='plotly_dark'
)
```

```
# Plot total revenue with log scale
fig2 = px.line(
    monthly_summary,
    x='purchase_month',
    y='total_revenue',
    color='city',
    markers=True,
    title='Total Revenue Over Time by City',
    labels={'purchase_month': 'Month', 'total_revenue': 'Total Revenue'}
)

# Apply log scale to the y-axis
fig2.update_layout(
    yaxis_type='log',
    yaxis_title='Total Revenue',
    legend_title='City',
    template='plotly_dark'
)

# Show the plots
fig1.show()
fig2.show()
```

10 8. A key stakeholder has asked how to make the most amount of revenue from selling cars, what advice would you give them to best achieve this?

When looking at revenue by vehicle, I started by incorrectly grouping by `vehicle_id`, when I should have grouped by `make` and `model`.

1. Relocate to Bath, based on previous plots.
2. Start selling brands such as Jeep, Dodge, RAM, Mazda, Chevrolet.

[11]:
```
# Define MySQL query
query = """
SELECT v.name, v.vehicle_id, SUM(v.price) AS total_revenue
FROM purchase_data p
JOIN vehicle_data v
ON p.vehicle_id = v.vehicle_id
GROUP BY v.vehicle_id
ORDER BY total_revenue DESC
LIMIT 10
"""

# Execute the SQL query using pandasql
```

```
best_revenue_vehicles = psql.sqldf(query, locals())

# Print the result
print("Top 10 vehicles providing the best revenue:")
print(best_revenue_vehicles)
```

```
Top 10 vehicles providing the best revenue:
                                    name  vehicle_id  total_revenue
0                         2024 BMW i7 M70         862    812768355.0
1          2024 Audi RS e-tron GT quattro         319    661481760.0
2     2024 Jeep Grand Wagoneer Series III         598    520483250.0
3  2024 Mercedes-Benz EQS 450 Base 4MATIC         475    484682380.0
4                        2024 BMW X7 M60i         441    482675545.0
5  2024 Mercedes-Benz EQS 450 Base 4MATIC         367    482370000.0
6  2024 Mercedes-Benz EQS 450 Base 4MATIC         407    474791440.0
7                    2024 BMW i7 eDrive50         289    464863005.0
8  2024 Mercedes-Benz EQS 450 Base 4MATIC         708    459108115.0
9  2024 Mercedes-Benz EQS 450 Base 4MATIC         250    454385820.0
```

[27]:
```
# Convert purchase_date to datetime
purchase_data['purchase_date'] = pd.to_datetime(purchase_data['purchase_date'])

# Create a new column for month (formatted as 'YYYY-MM')
purchase_data['purchase_month'] = purchase_data['purchase_date'].dt.
 ↪to_period('M').astype(str)

# Merge with vehicle_data to include price information
merged_data = pd.merge(purchase_data, vehicle_data, on='vehicle_id')

# Define the SQL query
query = """
    SELECT
        purchase_month,
        city,
        COUNT(vehicle_id) AS total_unit_sales,
        SUM(price) AS total_revenue,
        AVG(price) AS avg_price_per_unit
    FROM merged_data
    GROUP BY purchase_month, city
"""

# Execute the query using pandasql
result = psql.sqldf(query, locals())

# Convert purchase_month to datetime for plotting
result['purchase_month'] = pd.to_datetime(result['purchase_month'],␣
 ↪format='%Y-%m')
```

```
# Plot total unit sales with log scale
fig1 = px.line(
    result,
    x='purchase_month',
    y='total_unit_sales',
    color='city',
    markers=True,
    title='Total Unit Sales Over Time by City',
    labels={'purchase_month': 'Month', 'total_unit_sales': 'Total Unit Sales'}
)

# Apply log scale to the y-axis
fig1.update_layout(
    yaxis_type='log',
    yaxis_title='Total Unit Sales',
    legend_title='City',
    template='plotly_dark'
)

# Plot total revenue with log scale
fig2 = px.line(
    result,
    x='purchase_month',
    y='total_revenue',
    color='city',
    markers=True,
    title='Total Revenue Over Time by City',
    labels={'purchase_month': 'Month', 'total_revenue': 'Total Revenue'}
)

# Apply log scale to the y-axis
fig2.update_layout(
    yaxis_type='log',
    yaxis_title='Total Revenue',
    legend_title='City',
    template='plotly_dark'
)

# Plot average price per unit over time
fig3 = px.line(
    result,
    x='purchase_month',
    y='avg_price_per_unit',
    color='city',
    markers=True,
    title='Average Price Per Unit Over Time by City',
```

```
    labels={'purchase_month': 'Month', 'avg_price_per_unit': 'Average Price Per␣
 ↪Unit'},
    template='plotly_dark'
)

# Show the plots
fig1.show()
fig2.show()
fig3.show()
```

[29]:
```
# Convert purchase_date to datetime
purchase_data['purchase_date'] = pd.to_datetime(purchase_data['purchase_date'])

# Create a new column for month (formatted as 'YYYY-MM')
purchase_data['purchase_month'] = purchase_data['purchase_date'].dt.
 ↪to_period('M').astype(str)

# Merge with vehicle_data to include price information
merged_data = pd.merge(purchase_data, vehicle_data, on='vehicle_id')

# Define the SQL query to aggregate total revenue by approx_population
query = """
    SELECT
        approx_population,
        SUM(price) AS total_revenue
    FROM merged_data
    GROUP BY approx_population
"""

# Execute the query using pandasql
revenue_by_population = psql.sqldf(query, locals())

# Plot total revenue by approx_population
fig = px.scatter(
    revenue_by_population,
    x='approx_population',
    y='total_revenue',
    title='Total Revenue by Approximate Population',
    labels={'approx_population': 'Approximate Population', 'total_revenue':␣
 ↪'Total Revenue'},
    trendline='ols'  # Add a trendline to see the general trend
)

# Update layout for better readability
fig.update_layout(
    xaxis_title='Approximate Population',
    yaxis_title='Total Revenue',
```

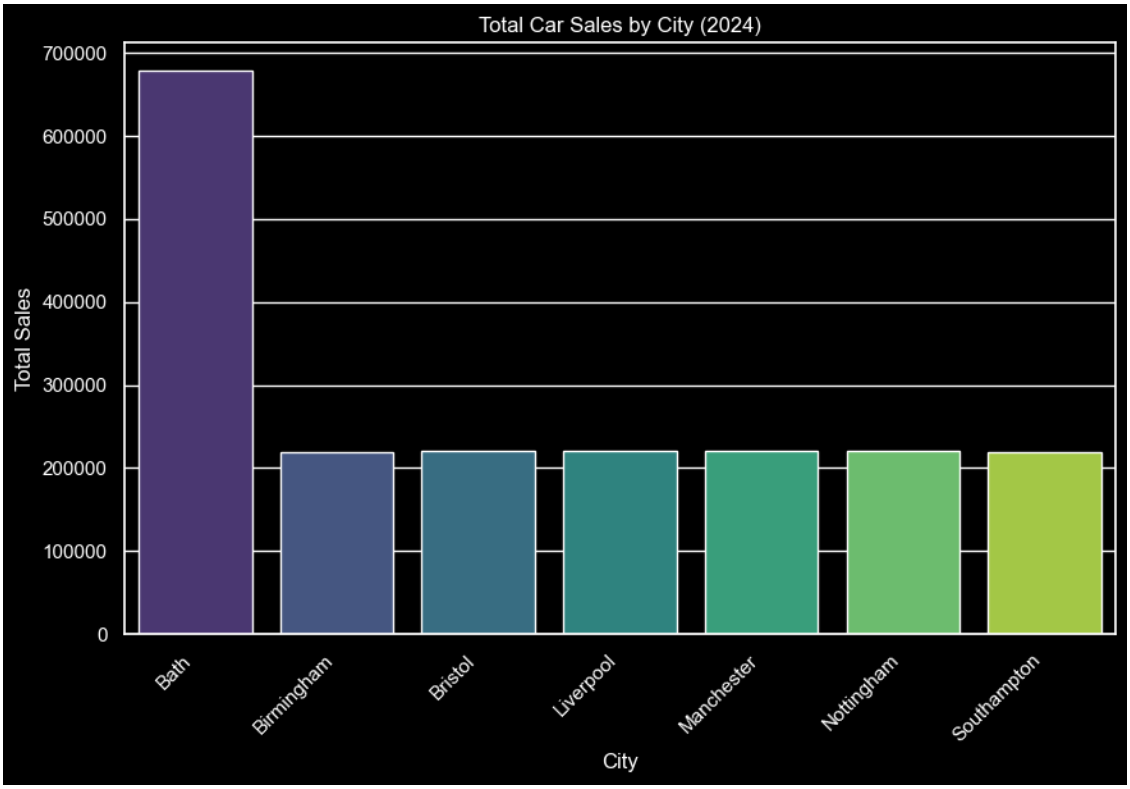
```
    template='plotly_dark'
)

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

[30]:
```
# Convert purchase_date to datetime
purchase_data['purchase_date'] = pd.to_datetime(purchase_data['purchase_date'])

# Create a new column for month (formatted as 'YYYY-MM')
purchase_data['purchase_month'] = purchase_data['purchase_date'].dt.
 ↪to_period('M').astype(str)

# Merge with vehicle_data to include price information
merged_data = pd.merge(purchase_data, vehicle_data, on='vehicle_id')

# Define the SQL query to aggregate total revenue by approx_population and city
query = """
    SELECT
        approx_population,
        city,
        SUM(price) AS total_revenue
    FROM merged_data
    GROUP BY approx_population, city
"""

# Execute the query using pandasql
revenue_by_population_city = psql.sqldf(query, locals())

# Plot total revenue by approx_population with color for city
fig = px.scatter(
    revenue_by_population_city,
    x='approx_population',
    y='total_revenue',
    color='city',  # Add color for different cities
    title='Total Revenue by Approximate Population and City',
    labels={'approx_population': 'Approximate Population', 'total_revenue':␣
 ↪'Total Revenue'},
    trendline='ols'  # Add a trendline to see the general trend
)

# Update layout for better readability
fig.update_layout(
    xaxis_title='Approximate Population',
    yaxis_title='Total Revenue',
    template='plotly_dark'
)
```

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

[29]:
```
# Group by city and count the number of sales
sales_by_city = merged_data.groupby('city').size().
 ↪reset_index(name='total_sales')

# Set dark mode for the plot
sns.set_style("darkgrid")
plt.style.use("dark_background")

# Plot total car sales by city
plt.figure(figsize=(10, 6))
sns.barplot(
    data=sales_by_city,
    x='city',
    y='total_sales',
    palette='viridis'  # Colorblind-friendly palette
)

# Set plot title and labels
plt.title('Total Car Sales by City (2024)')
plt.xlabel('City')
plt.ylabel('Total Sales')
plt.xticks(rotation=45, ha='right')  # Rotate city labels for better readability
plt.show()
```

```
/tmp/ipykernel_48525/1984652000.py:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(
```

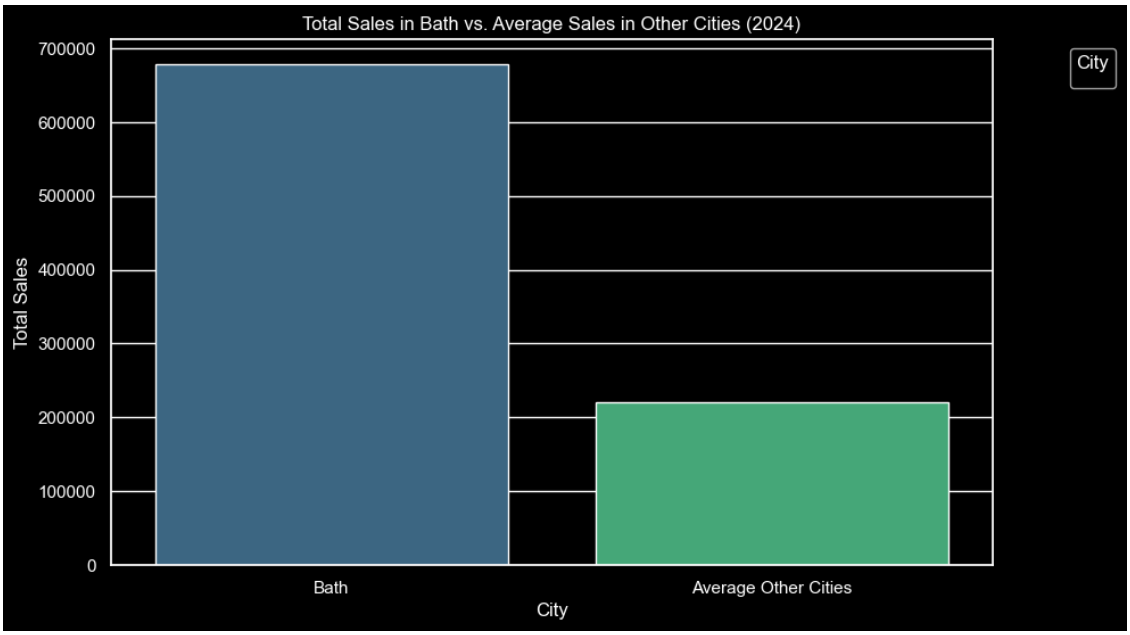


[31]:
```
# Find sales for Bath
bath_sales = sales_by_city[sales_by_city['city'] == 'Bath']['total_sales'].
 ↪values

# Calculate average sales for other cities
average_sales_other_cities = sales_by_city[sales_by_city['city'] !=␣
 ↪'Bath']['total_sales'].mean()

# Compute the difference
difference = bath_sales - average_sales_other_cities

# Display the results
print(f"Total sales in Bath: {bath_sales[0]}")
print(f"Average sales in other cities: {average_sales_other_cities:.2f}")
print(f"Difference: {difference[0]:.2f}")

# Prepare data for plotting
plot_data = pd.DataFrame({
    'City': ['Bath', 'Average Other Cities'],
    'Total Sales': [bath_sales[0], average_sales_other_cities]
})
```

```python
# Set dark mode for the plot
sns.set_style("darkgrid")
plt.style.use("dark_background")

# Plot the data
plt.figure(figsize=(10, 6))
bar_plot = sns.barplot(
    data=plot_data,
    x='City',
    y='Total Sales',
    palette='viridis'  # Colorblind-friendly palette
)

# Set plot title and labels
plt.title('Total Sales in Bath vs. Average Sales in Other Cities (2024)')
plt.ylabel('Total Sales')
plt.xlabel('City')

# Add custom legend manually
handles, labels = bar_plot.get_legend_handles_labels()
plt.legend(handles, labels, title='City', loc='upper right', bbox_to_anchor=(1.
 ↪15, 1))

plt.show()
```

```
Total sales in Bath: 678034
Average sales in other cities: 220327.67
Difference: 457706.33
```

```
/tmp/ipykernel_48525/3847254333.py:29: UserWarning: No artists with labels found
to put in legend.  Note that artists whose label start with an underscore are
ignored when legend() is called with no argument.
  plt.legend(title='City', loc='upper right', bbox_to_anchor=(1.15, 1))
```



[31]:
```
# Define SQL query to get total revenue by vehicle make and model
query = """
SELECT v.make, v.model, SUM(v.price) AS total_revenue
FROM purchase_data p
JOIN vehicle_data v
ON p.vehicle_id = v.vehicle_id
GROUP BY v.make, v.model
ORDER BY total_revenue DESC
LIMIT 10
"""

# Execute the SQL query using pandasql
best_revenue_vehicles = psql.sqldf(query, locals())

# Add a rank column starting from 1
best_revenue_vehicles['rank'] = range(1, len(best_revenue_vehicles) + 1)

# Print the result with ranking
print("Top 10 vehicles providing the best revenue:")
print(best_revenue_vehicles)

# Plot total revenue by vehicle make with revenue on x-axis and make on y-axis
fig = px.bar(
    best_revenue_vehicles,
    x='total_revenue',
    y='make',  # Show make on the y-axis
```

```
    orientation='h',  # Horizontal bar plot
    title='Top 10 Vehicle Makes by Total Revenue',
    labels={'total_revenue': 'Total Revenue', 'make': 'Vehicle Make'},
    text='total_revenue'  # Display revenue values on bars
)

# Update layout for better readability
fig.update_layout(
    xaxis_title='Total Revenue',
    yaxis_title='Vehicle Make',
    template='plotly_dark',
    xaxis_tickprefix='$',  # Show currency prefix
    yaxis_categoryorder='total ascending'  # Sort bars in ascending order of␣
 ↪revenue
)

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

```
Top 10 vehicles providing the best revenue:
        make                model  total_revenue  rank
0        RAM                 3500   1.290089e+10     1
1       Jeep  Grand Cherokee 4xe   5.456865e+09     2
2      Mazda           CX-90 PHEV   5.285632e+09     3
3       Jeep             Wagoneer   4.337264e+09     4
4      Dodge               Hornet   4.112463e+09     5
5      Dodge              Durango   3.859909e+09     6
6  Chevrolet      Silverado 1500   3.806809e+09     7
7       Jeep    Grand Cherokee L   3.643796e+09     8
8       Jeep           Wagoneer L   3.643266e+09     9
9       Jeep        Wrangler 4xe   3.424804e+09    10
```

[32]:
```
import pandas as pd
import pandasql as psql
import plotly.express as px

# Define SQL query to get total revenue by vehicle make and model
query = """
SELECT v.make, v.model, SUM(v.price) AS total_revenue
FROM purchase_data p
JOIN vehicle_data v
ON p.vehicle_id = v.vehicle_id
GROUP BY v.make, v.model
ORDER BY total_revenue DESC
LIMIT 10
"""
```

```
# Execute the SQL query using pandasql
best_revenue_vehicles = psql.sqldf(query, locals())

# Add a rank column starting from 1
best_revenue_vehicles['rank'] = range(1, len(best_revenue_vehicles) + 1)

# Print the result with ranking
print("Top 10 vehicles providing the best revenue:")
print(best_revenue_vehicles)

# Plot total revenue by vehicle make with revenue on x-axis and make on y-axis
fig = px.bar(
    best_revenue_vehicles,
    x='total_revenue',
    y='make',  # Show make on the y-axis
    orientation='h',  # Horizontal bar plot
    title='Top 10 Vehicle Makes by Total Revenue',
    labels={'total_revenue': 'Total Revenue', 'make': 'Vehicle Make'},
    text='total_revenue'  # Display revenue values on bars
)

# Update layout for better readability
fig.update_layout(
    xaxis_title='Total Revenue',
    yaxis_title='Vehicle Make',
    template='plotly_dark',
    xaxis_tickprefix='$',  # Show currency prefix
    yaxis_categoryorder='total ascending'  # Sort bars in ascending order of␣
 ↪revenue
)

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

```
Top 10 vehicles providing the best revenue:
        make                model  total_revenue  rank
0        RAM                 3500   1.290089e+10     1
1       Jeep  Grand Cherokee 4xe   5.456865e+09     2
2      Mazda           CX-90 PHEV   5.285632e+09     3
3       Jeep             Wagoneer   4.337264e+09     4
4      Dodge               Hornet   4.112463e+09     5
5      Dodge              Durango   3.859909e+09     6
6  Chevrolet      Silverado 1500   3.806809e+09     7
7       Jeep    Grand Cherokee L   3.643796e+09     8
8       Jeep           Wagoneer L   3.643266e+09     9
9       Jeep         Wrangler 4xe   3.424804e+09    10
```

[34]:
```
# Define the SQL query to get the total revenue by make and model
query = """
SELECT v.make, v.model, SUM(v.price) AS total_revenue
FROM purchase_data p
JOIN vehicle_data v ON p.vehicle_id = v.vehicle_id
GROUP BY v.make, v.model
ORDER BY total_revenue DESC
LIMIT 10
"""

# Execute the SQL query using pandasql
best_revenue_vehicles = psql.sqldf(query, locals())

# Add a rank column starting from 1
best_revenue_vehicles['rank'] = range(1, len(best_revenue_vehicles) + 1)

# Plot total revenue by vehicle make with models as stacked bars
fig = px.bar(
    best_revenue_vehicles,
    x='total_revenue',
    y='make',  # Show make on the y-axis
    color='model',  # Stack by model
    orientation='h',  # Horizontal bar plot
    title='Top 10 Vehicle Makes by Total Revenue with Models Stacked',
    labels={'total_revenue': 'Total Revenue', 'make': 'Vehicle Make', 'model':␣
 ↪'Vehicle Model'},
    text='total_revenue',  # Display revenue values on bars
    color_discrete_sequence=px.colors.qualitative.Plotly  # Optional: Customize␣
 ↪color sequence
)

# Update layout for better readability
fig.update_layout(
    xaxis_title='Total Revenue',
    yaxis_title='Vehicle Make',
    template='plotly_dark',
    xaxis_tickprefix='$',  # Show currency prefix
    yaxis_categoryorder='total ascending',  # Sort bars in ascending order of␣
 ↪revenue
    barmode='stack'  # Stack bars
)

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

[38]:
```
# Define the SQL query to get the total revenue by make and model
query = """
```

```
SELECT v.make, v.model, SUM(v.price) AS total_revenue
FROM purchase_data p
JOIN vehicle_data v ON p.vehicle_id = v.vehicle_id
GROUP BY v.make, v.model
ORDER BY total_revenue DESC
LIMIT 10
"""

# Execute the SQL query using pandasql
best_revenue_vehicles = psql.sqldf(query, locals())

# Add a rank column starting from 1
best_revenue_vehicles['rank'] = range(1, len(best_revenue_vehicles) + 1)

# Plot total revenue by vehicle make with models as text on stacked bars
fig = px.bar(
    best_revenue_vehicles,
    x='total_revenue',
    y='make',  # Show make on the y-axis
    color='make',  # Color by make
    orientation='h',  # Horizontal bar plot
    title='Top 10 Vehicle Makes and Models by Total Revenue',
    labels={'total_revenue': 'Total Revenue', 'make': 'Vehicle Make', 'model':␣
 ↪'Vehicle Model'},
    text='model',  # Display model names as text on bars
    color_discrete_sequence=px.colors.qualitative.Plotly  # Optional: Customize␣
 ↪color sequence
)

# Update layout for better readability
fig.update_layout(
    xaxis_title='Total Revenue',
    yaxis_title='Vehicle Make',
    template='plotly_dark',
    xaxis_tickprefix='$',  # Show currency prefix
    yaxis_categoryorder='total ascending',  # Sort bars in ascending order of␣
 ↪revenue
    barmode='stack'  # Stack bars
)

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

[40]:
```
# Define SQL query to get total revenue by vehicle make and model
query = """
SELECT v.make, v.model, SUM(v.price) AS total_revenue
FROM purchase_data p
```

```
JOIN vehicle_data v
ON p.vehicle_id = v.vehicle_id
GROUP BY v.make, v.model
ORDER BY total_revenue DESC
LIMIT 10
"""

# Execute the SQL query using pandasql
best_revenue_vehicles = psql.sqldf(query, locals())

# Add a rank column starting from 1
best_revenue_vehicles['rank'] = range(1, len(best_revenue_vehicles) + 1)

# Reorder columns to have 'rank' as the first column
best_revenue_vehicles = best_revenue_vehicles[['rank', 'make', 'model',␣
 ↪'total_revenue']]

# Print the result with ranking, without displaying the index
print("Top 10 vehicles providing the best revenue:")
print(best_revenue_vehicles.to_string(index=False))
```

```
Top 10 vehicles providing the best revenue:
 rank      make              model  total_revenue
    1       RAM               3500   1.290089e+10
    2      Jeep Grand Cherokee 4xe   5.456865e+09
    3     Mazda         CX-90 PHEV   5.285632e+09
    4      Jeep           Wagoneer   4.337264e+09
    5     Dodge             Hornet   4.112463e+09
    6     Dodge            Durango   3.859909e+09
    7 Chevrolet     Silverado 1500   3.806809e+09
    8      Jeep   Grand Cherokee L   3.643796e+09
    9      Jeep         Wagoneer L   3.643266e+09
   10      Jeep       Wrangler 4xe   3.424804e+09
```

[3]:
```
# Define SQL query to get total revenue by vehicle make
query = """
SELECT v.make, SUM(v.price) AS total_revenue
FROM purchase_data p
JOIN vehicle_data v ON p.vehicle_id = v.vehicle_id
GROUP BY v.make
ORDER BY total_revenue DESC
LIMIT 10
"""

# Execute the SQL query using pandasql
revenue_by_make = psql.sqldf(query, locals())
```

```
# Add a rank column starting from 1
revenue_by_make['rank'] = range(1, len(revenue_by_make) + 1)

# Print the result with ranking
print("Top 10 vehicle makes by total revenue:")
print(revenue_by_make.to_string(index=False))
```

```
Top 10 vehicle makes by total revenue:
         make  total_revenue  rank
         Jeep   2.737888e+10     1
          RAM   1.836506e+10     2
         Ford   9.675410e+09     3
        Dodge   8.751811e+09     4
Mercedes-Benz   7.862464e+09     5
      Hyundai   6.779339e+09     6
          BMW   6.424036e+09     7
        Mazda   5.749266e+09     8
    Chevrolet   4.978919e+09     9
         Audi   4.406617e+09    10
```

[4]:
```
# Plot total revenue by vehicle make
fig = px.bar(
    revenue_by_make,
    x='total_revenue',
    y='make',  # Show make on the y-axis
    color='make',  # Color by make
    orientation='h',  # Horizontal bar plot
    title='Top 10 Vehicle Makes by Total Revenue',
    labels={'total_revenue': 'Total Revenue', 'make': 'Vehicle Make'},
    color_discrete_sequence=px.colors.qualitative.Plotly  # Optional: Customize␣
 ↪color sequence
)

# Update layout for better readability
fig.update_layout(
    xaxis_title='Total Revenue',
    yaxis_title='Vehicle Make',
    template='plotly_dark',
    xaxis_tickprefix='$',  # Show currency prefix
    yaxis_categoryorder='total ascending'  # Sort bars in ascending order of␣
 ↪revenue
)

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

[5]:
```
# Define SQL query to get total revenue by vehicle model
query = """
SELECT v.make, v.model, SUM(v.price) AS total_revenue
FROM purchase_data p
JOIN vehicle_data v ON p.vehicle_id = v.vehicle_id
GROUP BY v.make, v.model
ORDER BY total_revenue DESC
LIMIT 10
"""

# Execute the SQL query using pandasql
best_revenue_models = psql.sqldf(query, locals())

# Add a rank column starting from 1
best_revenue_models['rank'] = range(1, len(best_revenue_models) + 1)

# Reorder columns to have 'rank' as the first column
best_revenue_models = best_revenue_models[['rank', 'make', 'model',␣
 ↪'total_revenue']]

# Print the result with ranking, without displaying the index
print("Top 10 vehicle models providing the best revenue:")
print(best_revenue_models.to_string(index=False))
```

```
Top 10 vehicle models providing the best revenue:
 rank      make              model  total_revenue
    1       RAM               3500   1.290089e+10
    2      Jeep Grand Cherokee 4xe   5.456865e+09
    3     Mazda         CX-90 PHEV   5.285632e+09
    4      Jeep           Wagoneer   4.337264e+09
    5     Dodge             Hornet   4.112463e+09
    6     Dodge            Durango   3.859909e+09
    7 Chevrolet     Silverado 1500   3.806809e+09
    8      Jeep   Grand Cherokee L   3.643796e+09
    9      Jeep         Wagoneer L   3.643266e+09
   10      Jeep       Wrangler 4xe   3.424804e+09
```

[6]:
```
# Plot total revenue by vehicle model
fig = px.bar(
    best_revenue_models,
    x='total_revenue',
    y='model',  # Show model on the y-axis
    color='make',  # Color by make
    orientation='h',  # Horizontal bar plot
    title='Top 10 Vehicle Models by Total Revenue',
    labels={'total_revenue': 'Total Revenue', 'model': 'Vehicle Model'},
    text='make',  # Display make names as text on bars
```

```
    color_discrete_sequence=px.colors.qualitative.Plotly  # Optional: Customize␣
 ↪color sequence
)

# Update layout for better readability
fig.update_layout(
    xaxis_title='Total Revenue',
    yaxis_title='Vehicle Model',
    template='plotly_dark',
    xaxis_tickprefix='$',  # Show currency prefix
    yaxis_categoryorder='total descending'  # Sort bars in descending order of␣
 ↪revenue
)

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

[7]:
```
# Plot total revenue by vehicle model
fig = px.bar(
    best_revenue_models,
    x='total_revenue',
    y='model',  # Show model on the y-axis
    color='make',  # Color by make
    orientation='h',  # Horizontal bar plot
    title='Top 10 Vehicle Models by Total Revenue',
    labels={'total_revenue': 'Total Revenue', 'model': 'Vehicle Model'},
    text='make',  # Display make names as text on bars
    color_discrete_sequence=px.colors.qualitative.Plotly  # Optional: Customize␣
 ↪color sequence
)

# Update layout to reverse y-axis
fig.update_layout(
    xaxis_title='Total Revenue',
    yaxis_title='Vehicle Model',
    template='plotly_dark',
    xaxis_tickprefix='$',  # Show currency prefix
    yaxis_categoryorder='total ascending',  # Sort bars in ascending order of␣
 ↪revenue
    yaxis_categoryarray=best_revenue_models['model'][::-1]  # Reverse the␣
 ↪y-axis categories
)

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

[8]:
```
# Define SQL query to get total revenue by vehicle make and model
query = """
SELECT v.make, v.model, SUM(v.price) AS total_revenue
FROM purchase_data p
JOIN vehicle_data v ON p.vehicle_id = v.vehicle_id
GROUP BY v.make, v.model
ORDER BY total_revenue DESC
"""

# Execute the SQL query using pandasql
revenue_data = psql.sqldf(query, locals())

# Print the data to check its structure
print(revenue_data.head())
```

```
    make               model  total_revenue
0    RAM                3500   1.290089e+10
1   Jeep  Grand Cherokee 4xe   5.456865e+09
2  Mazda          CX-90 PHEV   5.285632e+09
3   Jeep            Wagoneer   4.337264e+09
4  Dodge              Hornet   4.112463e+09
```

[10]:
```
# Create a line chart with Plotly Express
fig = px.line(
    revenue_data,
    x='model',  # Use model as the x-axis
    y='total_revenue',  # Use total revenue as the y-axis
    color='make',  # Color lines by make
    markers=True,  # Show markers on the lines
    title='Total Revenue by Vehicle Make and Model',
    labels={'total_revenue': 'Total Revenue', 'model': 'Vehicle Model'},
    line_shape='linear'  # Use linear lines
)

# Update marker size and other properties
fig.update_traces(marker=dict(size=10))  # Set marker size

# Update layout for better readability
fig.update_layout(
    xaxis_title='Vehicle Model',
    yaxis_title='Total Revenue',
    template='plotly_dark',
    xaxis_tickangle=-45,  # Rotate x-axis labels for better readability
    xaxis_title_standoff=25,  # Space between x-axis title and tick labels
    yaxis_tickprefix='$',  # Show currency prefix
    legend_title='Vehicle Make'  # Title for the legend
)
```

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

[11]:
```
import pandas as pd
import plotly.express as px

# Define SQL query to get total revenue by vehicle make and model
query = """
SELECT v.make, v.model, SUM(v.price) AS total_revenue
FROM purchase_data p
JOIN vehicle_data v ON p.vehicle_id = v.vehicle_id
GROUP BY v.make, v.model
ORDER BY total_revenue DESC
"""

# Execute the SQL query using pandasql
revenue_data = psql.sqldf(query, locals())

# Create a line chart with Plotly Express
fig = px.line(
    revenue_data,
    x='model',  # Use model as the x-axis values
    y='total_revenue',  # Use total revenue as the y-axis
    color='make',  # Color lines by make
    markers=True,  # Show markers on the lines
    title='Total Revenue by Vehicle Make and Model',
    labels={'total_revenue': 'Total Revenue', 'model': 'Vehicle Model'},
    line_shape='linear'  # Use linear lines
)

# Create a mapping from model to make
model_to_make = revenue_data[['model', 'make']].drop_duplicates().
 ↪sort_values(by='model')
model_to_make = model_to_make.set_index('model')['make'].to_dict()

# Update x-axis tick labels with make names
fig.update_layout(
    xaxis=dict(
        tickmode='array',
        tickvals=list(model_to_make.keys()),  # Use model names as tick values
        ticktext=[model_to_make[model] for model in model_to_make.keys()]  #␣
 ↪Use make names as tick labels
    ),
    xaxis_title='Vehicle Make',
    yaxis_title='Total Revenue',
    template='plotly_dark',
    xaxis_tickangle=-45,  # Rotate x-axis labels for better readability
```

```
    yaxis_tickprefix='$',  # Show currency prefix
    legend_title='Vehicle Make'  # Title for the legend
)

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

[12]:
```
# Define SQL query to get total revenue by vehicle make and model
query = """
SELECT v.make, v.model, SUM(v.price) AS total_revenue
FROM purchase_data p
JOIN vehicle_data v ON p.vehicle_id = v.vehicle_id
GROUP BY v.make, v.model
ORDER BY total_revenue DESC
"""

# Execute the SQL query using pandasql
revenue_data = psql.sqldf(query, locals())

# Create a line chart with Plotly Express
fig = px.line(
    revenue_data,
    x='model',  # Use model as the x-axis values
    y='total_revenue',  # Use total revenue as the y-axis
    color='make',  # Color lines by make
    markers=True,  # Show markers on the lines
    title='Total Revenue by Vehicle Make and Model',
    labels={'total_revenue': 'Total Revenue', 'model': 'Vehicle Model'},
    line_shape='linear'  # Use linear lines
)

# Custom tick labels for x-axis
# Map each model to its respective make
model_to_make = revenue_data[['model', 'make']].drop_duplicates().
 ↪sort_values(by='model')

# Create a custom label for each tick
tick_labels = []
tick_positions = []
for model in model_to_make['model'].unique():
    make = model_to_make.loc[model_to_make['model'] == model, 'make'].values[0]
    tick_positions.append(model)
    tick_labels.append(make)

# Update x-axis tick labels to only show the make once
fig.update_layout(
    xaxis=dict(
```

```
        tickmode='array',
        tickvals=tick_positions,  # Use model names as tick values
        ticktext=tick_labels  # Use make names as tick labels
    ),
    xaxis_title='Vehicle Make',
    yaxis_title='Total Revenue',
    template='plotly_dark',
    xaxis_tickangle=-45,  # Rotate x-axis labels for better readability
    yaxis_tickprefix='$',  # Show currency prefix
    legend_title='Vehicle Make'  # Title for the legend
)

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

[13]:
```
import pandas as pd
import plotly.express as px

# Define SQL query to get total revenue by vehicle make and model
query = """
SELECT v.make, v.model, SUM(v.price) AS total_revenue
FROM purchase_data p
JOIN vehicle_data v ON p.vehicle_id = v.vehicle_id
GROUP BY v.make, v.model
ORDER BY total_revenue DESC
"""

# Execute the SQL query using pandasql
revenue_data = psql.sqldf(query, locals())

# Create a line chart with Plotly Express
fig = px.line(
    revenue_data,
    x='model',  # Use model as the x-axis values
    y='total_revenue',  # Use total revenue as the y-axis
    color='make',  # Color lines by make
    markers=True,  # Show markers on the lines
    title='Total Revenue by Vehicle Make and Model',
    labels={'total_revenue': 'Total Revenue', 'model': 'Vehicle Model'},
    line_shape='linear'  # Use linear lines
)

# Extract unique makes and their positions
unique_makes = revenue_data[['model', 'make']].drop_duplicates().
 ↪sort_values(by='model')

# Create labels and positions for x-axis
```

```
tick_labels = unique_makes['make'].unique()  # Unique make names
tick_positions = unique_makes['model'].unique()  # Unique model names

# Display labels only at intervals or for unique makes
label_interval = max(1, len(tick_positions) // 10)  # Adjust interval based on␣
 ↪the number of labels
visible_ticks = tick_positions[::label_interval]

# Update x-axis tick labels to show only some of the makes
fig.update_layout(
    xaxis=dict(
        tickmode='array',
        tickvals=visible_ticks,  # Set tick positions
        ticktext=[unique_makes.loc[unique_makes['model'] == tick, 'make'].
 ↪values[0] for tick in visible_ticks]  # Set labels
    ),
    xaxis_title='Vehicle Make',
    yaxis_title='Total Revenue',
    template='plotly_dark',
    xaxis_tickangle=-45,  # Rotate x-axis labels for better readability
    yaxis_tickprefix='$',  # Show currency prefix
    legend_title='Vehicle Make'  # Title for the legend
)

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

[14]:
```
import pandas as pd
import plotly.express as px

# Define SQL query to get total revenue by vehicle make and model
query = """
SELECT v.make, v.model, SUM(v.price) AS total_revenue
FROM purchase_data p
JOIN vehicle_data v ON p.vehicle_id = v.vehicle_id
GROUP BY v.make, v.model
ORDER BY total_revenue DESC
"""

# Execute the SQL query using pandasql
revenue_data = psql.sqldf(query, locals())

# Create a line chart with Plotly Express
fig = px.line(
    revenue_data,
    x='model',  # Use model as the x-axis values
    y='total_revenue',  # Use total revenue as the y-axis
```

```
    color='make',  # Color lines by make
    markers=True,  # Show markers on the lines
    title='Total Revenue by Vehicle Make and Model',
    labels={'total_revenue': 'Total Revenue', 'model': 'Vehicle Model'},
    line_shape='linear'  # Use linear lines
)

# Get unique makes and their representative models
unique_makes = revenue_data[['make', 'model']].drop_duplicates()
representative_models = unique_makes.groupby('make').first().reset_index()

# Create tick labels and positions
tick_labels = representative_models['make'].tolist()
tick_positions = representative_models['model'].tolist()

# Update x-axis to show only one label per make
fig.update_layout(
    xaxis=dict(
        tickmode='array',
        tickvals=tick_positions,  # Set tick positions
        ticktext=tick_labels  # Set labels
    ),
    xaxis_title='Vehicle Make',
    yaxis_title='Total Revenue',
    template='plotly_dark',
    xaxis_tickangle=-45,  # Rotate x-axis labels for better readability
    yaxis_tickprefix='$',  # Show currency prefix
    legend_title='Vehicle Make'  # Title for the legend
)

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

[22]:
```
import pandas as pd
import plotly.express as px

# Define SQL query to get total revenue by vehicle make and model
query = """
SELECT v.make, v.model, SUM(v.price) AS total_revenue
FROM purchase_data p
JOIN vehicle_data v ON p.vehicle_id = v.vehicle_id
GROUP BY v.make, v.model
ORDER BY total_revenue DESC
"""

# Execute the SQL query using pandasql
revenue_data = psql.sqldf(query, locals())
```

```
# Create a line chart with Plotly Express
fig = px.line(
    revenue_data,
    x='total_revenue',  # Use total revenue as the x-axis values
    y='model',  # Use model as the y-axis
    color='make',  # Color lines by make
    markers=True,  # Show markers on the lines
    title='Total Revenue by Vehicle Make and Model',
    labels={'total_revenue': 'Total Revenue', 'model': 'Vehicle Model'},
    line_shape='linear'  # Use linear lines
)

# Get unique makes and their representative models
unique_makes = revenue_data[['make', 'model']].drop_duplicates()
representative_models = unique_makes.groupby('make').first().reset_index()

# Create tick labels and positions
tick_labels = representative_models['make'].tolist()
tick_positions = representative_models['model'].tolist()

# Update y-axis to show only one label per make
fig.update_layout(
    yaxis=dict(
        tickmode='array',
        tickvals=tick_positions,  # Set tick positions
        ticktext=tick_labels  # Set labels
    ),
    xaxis_title='Total Revenue',
    yaxis_title='Vehicle Model',
    template='plotly_dark',
    xaxis_tickprefix='$',  # Show currency prefix
    yaxis_tickangle=-45,
    legend_title='Vehicle Make'  # Title for the legend
)

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

[19]:
```
import pandas as pd
import plotly.express as px

# Define SQL query to get total revenue by vehicle make and model
query = """
SELECT v.make, v.model, SUM(v.price) AS total_revenue
FROM purchase_data p
JOIN vehicle_data v ON p.vehicle_id = v.vehicle_id
```

```
GROUP BY v.make, v.model
ORDER BY total_revenue DESC
"""

# Execute the SQL query using pandasql
revenue_data = psql.sqldf(query, locals())

# Create a line chart with Plotly Express
fig = px.line(
    revenue_data,
    x='total_revenue',  # Use total revenue as the x-axis values
    y='model',  # Use model as the y-axis
    color='make',  # Color lines by make
    markers=True,  # Show markers on the lines
    title='Total Revenue by Vehicle Make and Model',
    labels={'total_revenue': 'Total Revenue', 'model': 'Vehicle Model'},
    line_shape='linear'  # Use linear lines
)

# Get unique makes and their representative models
unique_makes = revenue_data[['make', 'model']].drop_duplicates()
representative_models = unique_makes.groupby('make').first().reset_index()

# Create tick labels and positions
tick_labels = representative_models['make'].tolist()
tick_positions = representative_models['model'].tolist()

# Space out the labels on the y-axis by selecting every nth label or␣
 ↪customizing as needed
spacing_factor = 2  # Adjust this number to increase/decrease spacing
spaced_tick_positions = tick_positions[::spacing_factor]  # Select every nth␣
 ↪position
spaced_tick_labels = tick_labels[::spacing_factor]  # Select every nth label

# Update layout to adjust y-axis labels, spacing, and reverse the order
fig.update_layout(
    yaxis=dict(
        tickmode='array',
        tickvals=spaced_tick_positions,  # Set spaced tick positions
        ticktext=spaced_tick_labels,  # Set spaced labels
        tickangle=0,  # Make y-axis labels horizontal
        autorange='reversed'  # Reverse y-axis so greater revenue is at the top
    ),
    xaxis_title='Total Revenue',
    yaxis_title='',  # Remove y-axis title
    template='plotly_dark',
    xaxis_tickprefix='$',  # Show currency prefix
```

```
    legend_title='Vehicle Make'  # Title for the legend
)

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

[20]:
```
import pandas as pd
import plotly.express as px

# Define SQL query to get total revenue by vehicle make and model
query = """
SELECT v.make, v.model, SUM(v.price) AS total_revenue
FROM purchase_data p
JOIN vehicle_data v ON p.vehicle_id = v.vehicle_id
GROUP BY v.make, v.model
ORDER BY total_revenue DESC
"""

# Execute the SQL query using pandasql
revenue_data = psql.sqldf(query, locals())

# Create a line chart with Plotly Express
fig = px.line(
    revenue_data,
    x='total_revenue',  # Use total revenue as the x-axis values
    y='model',  # Use model as the y-axis
    color='make',  # Color lines by make
    markers=True,  # Show markers on the lines
    title='Total Revenue by Vehicle Make and Model',
    labels={'total_revenue': 'Total Revenue', 'model': 'Vehicle Model'},
    line_shape='linear'  # Use linear lines
)

# Get unique makes and their representative models
unique_makes = revenue_data[['make', 'model']].drop_duplicates()
representative_models = unique_makes.groupby('make').first().reset_index()

# Create tick labels and positions
tick_labels = representative_models['make'].tolist()
tick_positions = representative_models['model'].tolist()

# Space out the labels on the y-axis by selecting every nth label or␣
 ↪customizing as needed
spacing_factor = 2  # Adjust this number to increase/decrease spacing
spaced_tick_positions = tick_positions[::spacing_factor]  # Select every nth␣
 ↪position
spaced_tick_labels = tick_labels[::spacing_factor]  # Select every nth label
```

```
# Calculate the range for the y-axis
y_max = revenue_data['model'].nunique()  # Get number of unique models for␣
 ↪y-axis range
y_min = -1  # Add padding at the bottom

# Update layout to adjust y-axis labels, spacing, and reverse the order
fig.update_layout(
    yaxis=dict(
        tickmode='array',
        tickvals=spaced_tick_positions,  # Set spaced tick positions
        ticktext=spaced_tick_labels,  # Set spaced labels
        tickangle=0,  # Make y-axis labels horizontal
        autorange='reversed',  # Reverse y-axis so greater revenue is at the top
        range=[y_min, y_max],  # Add space at the top and bottom of the y-axis
        fixedrange=False  # Ensure axis does not auto-adjust beyond the␣
 ↪specified range
    ),
    xaxis_title='Total Revenue',
    yaxis_title='',  # Remove y-axis title
    template='plotly_dark',
    xaxis_tickprefix='$',  # Show currency prefix
    legend_title='Vehicle Make'  # Title for the legend
)

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

[21]:
```
import pandas as pd
import plotly.express as px

# Define SQL query to get total revenue by vehicle make and model
query = """
SELECT v.make, v.model, SUM(v.price) AS total_revenue
FROM purchase_data p
JOIN vehicle_data v ON p.vehicle_id = v.vehicle_id
GROUP BY v.make, v.model
ORDER BY total_revenue DESC
"""

# Execute the SQL query using pandasql
revenue_data = psql.sqldf(query, locals())

# Create a line chart with Plotly Express
fig = px.line(
    revenue_data,
    x='total_revenue',  # Use total revenue as the x-axis values
```

```
    y='model',  # Use model as the y-axis
    color='make',  # Color lines by make
    markers=True,  # Show markers on the lines
    title='Total Revenue by Vehicle Make and Model',
    labels={'total_revenue': 'Total Revenue', 'model': 'Vehicle Model'},
    line_shape='linear'  # Use linear lines
)

# Get unique makes and their representative models
unique_makes = revenue_data[['make', 'model']].drop_duplicates()
representative_models = unique_makes.groupby('make').first().reset_index()

# Create tick labels and positions
tick_labels = representative_models['make'].tolist()
tick_positions = representative_models['model'].tolist()

# Increase spacing between y-axis labels and points
spacing_factor = 3  # Adjust this number to increase/decrease spacing
spaced_tick_positions = tick_positions[::spacing_factor]  # Select every nth␣
 ↪position
spaced_tick_labels = tick_labels[::spacing_factor]  # Select every nth label

# Calculate the range for the y-axis with extra padding
num_models = len(tick_positions)
padding = 2  # Padding to add at the top and bottom
y_max = num_models + padding
y_min = -padding

# Update layout to adjust y-axis labels, spacing, and reverse the order
fig.update_layout(
    yaxis=dict(
        tickmode='array',
        tickvals=spaced_tick_positions,  # Set spaced tick positions
        ticktext=spaced_tick_labels,  # Set spaced labels
        tickangle=0,  # Make y-axis labels horizontal
        autorange='reversed',  # Reverse y-axis so greater revenue is at the top
        range=[y_min, y_max],  # Add extra space at the top and bottom of the␣
 ↪y-axis
        fixedrange=False  # Ensure axis does not auto-adjust beyond the␣
 ↪specified range
    ),
    xaxis_title='Total Revenue',
    yaxis_title='',  # Remove y-axis title
    template='plotly_dark',
    xaxis_tickprefix='$',  # Show currency prefix
    legend_title='Vehicle Make'  # Title for the legend
)
```

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

[24]:
```
# Define SQL query to get total revenue by vehicle make and model
query = """
SELECT v.make, v.model, SUM(v.price) AS total_revenue
FROM purchase_data p
JOIN vehicle_data v ON p.vehicle_id = v.vehicle_id
GROUP BY v.make, v.model
ORDER BY total_revenue DESC
"""

# Execute the SQL query using pandasql
revenue_data = psql.sqldf(query, locals())

# Create a line chart with Plotly Express
fig = px.line(
    revenue_data,
    x='total_revenue',  # Use total revenue as the x-axis values
    y='model',  # Use model as the y-axis
    color='make',  # Color lines by make
    markers=True,  # Show markers on the lines
    title='Total Revenue by Vehicle Make and Model',
    labels={'total_revenue': 'Total Revenue', 'model': 'Vehicle Model'},
    line_shape='linear'  # Use linear lines
)

# Update y-axis to hide labels and reverse the order
fig.update_layout(
    yaxis=dict(
        showticklabels=False,  # Hide y-axis tick labels
        autorange='reversed'  # Reverse the y-axis order
    ),
    xaxis_title='Total Revenue',
    yaxis_title='Vehicle Model',
    template='plotly_dark',
    xaxis_tickprefix='$',  # Show currency prefix
    yaxis_tickangle=-45,
    legend_title='Vehicle Make'  # Title for the legend
)

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