MiniProject

April 19, 2025

1 SC1015 Mini Project

1.1 Problem Statement:

Analysing Used Car Sales Data to understand what attributes affect the Selling Price of Used Cars most strongly. (1) We aim to construct a neural network to predict the prices of Used Cars (2) We will propose which attributes should be considered by a prospective seller looking to list their car for sale. This will assist sellers with setting a reasonable price for their car and improve likelihood of successful sales, maximising profit earned from the sale where possible (3) We could analyse the most value for money option depending on the mileage, type of car and sale price of the car

1.2 Dataset Used: https://www.kaggle.com/datasets/sandeep1080/used-car-sales/data

1.3 Members

Name: Kee Chong Wei | Matriculation Number: U2320846D Name: Jiang Zong Zhe | Matriculation Number: U2322460F Name: Chew Jin Cheng | Matriculation Number: U2321537J

Workflow:

(1) Identify null values and drop/impute them if necessary If null values account for more than 50% of samples for any category, consider dropping column entirely due to lack of meaningful data Else, consider data imputation, replacing null values with suitable values (median, mean)

(2) Analyse data types and dataset description to distinguish categorical and numerical variables

(3) *Numerical Data Analysis* (3.1) Create Boxplots for each variable to identify if any outliers present (3.2) Plot correlation matrix to identify which variables correlate strongest with "Sold Price" (3.3) Perform Linear Regression to get a model which can predict an ideal Sold Price based on numerical variables with the highest correlation

(4) *Categorical Data Analysis* (4.1) Identify which values related with Sold Price (multiple box plots of diff type of data in a certain category) (4.2) Utilise Random Forest to get a model which can predict an ideal Sold Price based on the most relevant categorical variables

(5) Beyond the Content of the Course

[1]:
```
import math
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
import numpy as np
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import Input
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler,OneHotEncoder
from sklearn.metrics import mean_squared_error, r2_score
from scipy.stats import pearsonr
from statsmodels.nonparametric.smoothers_lowess import lowess

tf.get_logger().setLevel('ERROR')
```

[2]:
```
data = pd.read_csv("used_car_sales.csv")
```

[3]:
```
data.head
```

[3]:
```
<bound method NDFrame.head of           ID Distributor Name        Location  Car
Name Manufacturer Name  \
0     02KE17          Carmudi      California  Fortuner            Toyota
1     EPMPC8        Carousell    Philadelphia     Creta           Hyundai
2     SQKXAP          Carsome  North Carolina   Scorpio          Mahindra
3     PWP2QK          Trivett  North Carolina     Plato             Prazo
4     FNDDKM            Zupps        Portland     Dzire            Maruti
…      …              …             …        …                …
9995  ZHLCSG              APE           Texas     Yodha              Tata
9996  2BJE0Y          Carsome        Portland   Scorpio          Mahindra
9997  40VJ83            Trust  North Carolina    Seltos               Kia
9998  M2ECXT          Carsome         Detroit     Swift            Maruti
9999  28W445              Olx        Portland     Swift            Maruti

         Car Type  Color    Gearbox  Number of Seats  Number of Doors  …  \
0             SUV   Gray  Automatic                8                5  …
1       Hatchback   Blue  Automatic                5                5  …
2             SUV   Gray  Automatic                5                5  …
3     Convertible   Gray  Automatic                2                2  …
4           Sedan    Red  Automatic                5                5  …
…             …    …        …              …              …  …
9995        Truck   Blue     Manual                3                2  …
9996          SUV  Black  Automatic                5                5  …
9997    Hatchback  Black  Automatic                5                5  …
9998        Sedan  Black  Automatic                5                4  …
```

```
9999        Sedan  White  Automatic                5                 4  …

     Purchased Date  Car Sale Status   Sold Date  Purchased Price-$  \
0        2022-10-26          Un Sold  1970-01-01               8296
1        2017-08-25             Sold  2021-03-03               5659
2        2018-06-13          Un Sold  1970-01-01               8430
3        2023-05-14             Sold  2024-04-02               6919
4        2022-08-24          Un Sold  1970-01-01               6864
…               …                …          …                 …
9995     2023-12-29             Sold  2024-03-23               6102
9996     2019-06-13          Un Sold  1970-01-01               8108
9997     2020-02-17          Un Sold  1970-01-01               5945
9998     2018-05-03          Un Sold  1970-01-01               6893
9999     2024-05-18          Un Sold  1970-01-01               6771

      Sold Price-$ Margin-% Sales Agent Name Sales Rating  Sales Commission-$  \
0                0        0           Pranav            1                   0
1             4770      -16           Vihaan            5                   0
2                0        0           Aarush            4                   0
3             7942       15          Anushka            1                 205
4                0        0            Pavan            3                   0
…              …      …              …          …                 …
9995          5041      -17          Supriya            3                   0
9996             0        0           Aarush            4                   0
9997             0        0           Pranav            4                   0
9998             0        0           Swathi            2                   0
9999             0        0           Vihaan            5                   0

       Feedback
0       Average
1          Good
2          Good
3       Average
4          Poor
…        …
9995  Excellent
9996  Excellent
9997       Poor
9998    Average
9999       Poor

[10000 rows x 25 columns]>
```

2 (1) Identify null values and drop/impute them if necessary

[4]:
```
null_counts = data.isnull().sum()

print(null_counts)
```

```
ID                     0
Distributor Name       0
Location               0
Car Name               0
Manufacturer Name      0
Car Type               0
Color                  0
Gearbox                0
Number of Seats        0
Number of Doors        0
Energy                 0
Manufactured Year      0
Price-$                0
Mileage-KM             0
Engine Power-HP        0
Purchased Date         0
Car Sale Status        0
Sold Date              0
Purchased Price-$      0
Sold Price-$           0
Margin-%               0
Sales Agent Name       0
Sales Rating           0
Sales Commission-$     0
Feedback               0
dtype: int64
```

3 No Null values present in any column, hence no need for dropping any data or data imputation.

4 (2) Analyse data types and dataset description to distinguish categorical and numerical variables

[5]:
```
data.dtypes
```

[5]:
```
ID                     object
Distributor Name       object
Location               object
Car Name               object
Manufacturer Name      object
```

```
Car Type                object
Color                   object
Gearbox                 object
Number of Seats          int64
Number of Doors          int64
Energy                  object
Manufactured Year        int64
Price-$                  int64
Mileage-KM               int64
Engine Power-HP          int64
Purchased Date          object
Car Sale Status         object
Sold Date               object
Purchased Price-$        int64
Sold Price-$             int64
Margin-%                 int64
Sales Agent Name        object
Sales Rating             int64
Sales Commission-$       int64
Feedback                object
dtype: object
```

[6]:
```
categorical_data_labels = ['ID','Distributor Name','Location','Car␣
 ↪Name','Manufacturer Name','Car␣
 ↪Type','Color','Gearbox','Energy','Manufactured Year',
                   'Purchased Date','Car Sale Status','Sold Date','Sales Agent␣
 ↪Name','Sales Rating','Feedback']
numerical_data_labels = ['Number of Seats','Number of␣
 ↪Doors','Price-$','Mileage-KM','Engine Power-HP','Purchased Price-$','Sold␣
 ↪Price-$','Margin-%']
```

In variable **'Car Sale Status'**, it can be seen that some of the samples describe cars that have not yet been sold. We should only use data for cars that have managed to be sold as inputs to our model. The unsold car data can provide info on boundary values of price or other variables which might cause a car to be undesirable and hence not sold.

[7]:
```
sold_cars = data[data['Car Sale Status'] == 'Sold']
unsold_cars = data[data['Car Sale Status'] == 'Unsold']
```

5 (3) Numerical Data Analysis

[8]:
```
sold_cars_numerical_data = data[data['Car Sale Status'] ==␣
 ↪'Sold'][numerical_data_labels]
```

[9]:
```
sold_cars_numerical_data.head
```

[9]:

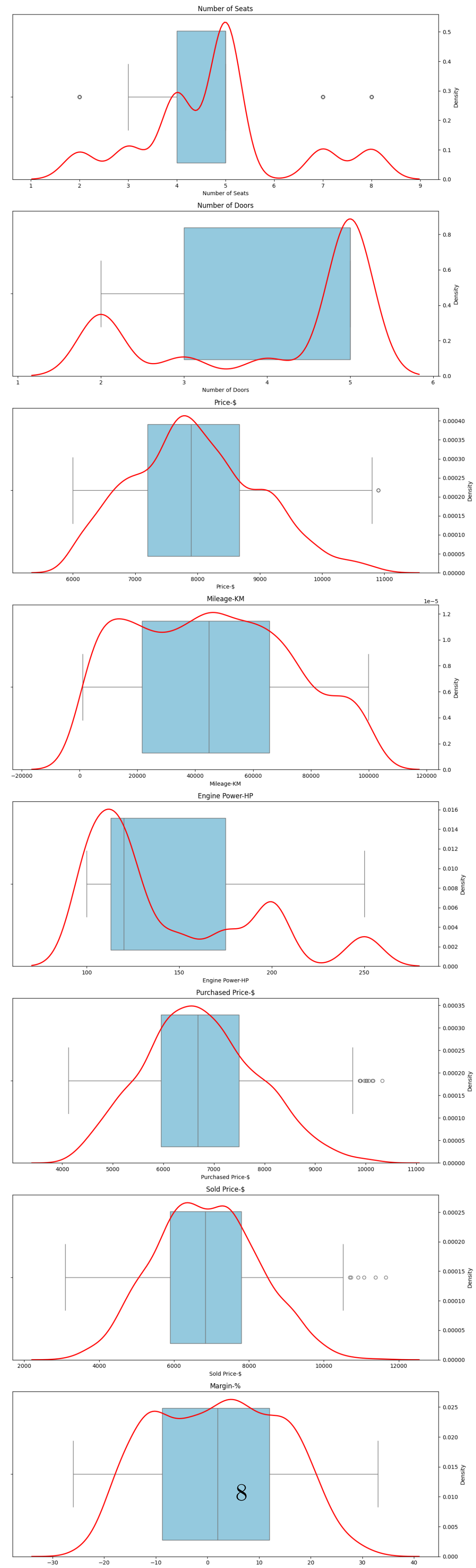
```
<bound method NDFrame.head of       Number of Seats  Number of Doors  Price-$
Mileage-KM  Engine Power-HP  \
1                  5                5     7600       34637              113
3                  2                2     8800       46250              250
5                  8                5     9300       38716              200
14                 5                5     9300       95835              120
17                 5                5     6900       66534              113
…               …              …      …         …             …
9961               5                4     7300       37448              115
9981               4                5     6800       60579              103
9983               5                5     7600       68001              150
9993               5                5     7800       96006              120
9995               3                2     7100       55333              100

      Purchased Price-$  Sold Price-$  Margin-%
1                  5659          4770       -16
3                  6919          7942        15
5                  8533          9929        16
14                 8557          8896         4
17                 6382          5418       -15
…                 …            …        …
9961               6544          6255        -4
9981               5962          6110         2
9983               6751          7361         9
9993               7033          6857        -3
9995               6102          5041       -17

[2166 rows x 8 columns]>
```

6 (3.1) Create Boxplots for each variable to identify if any outliers present

[10]:

```python
# find if there are outliers in any numerical data

plt.figure(figsize=(12, 5 * len(sold_cars_numerical_data.columns)))  # Adjust␣
↪height dynamically

for i, column in enumerate(sold_cars_numerical_data.columns, 1):
    ax_box = plt.subplot(len(sold_cars_numerical_data.columns), 1, i)

    sns.boxplot(x=sold_cars_numerical_data[column], ax=ax_box, color="skyblue")

    ax_kde = ax_box.twinx()
    sns.kdeplot(sold_cars_numerical_data[column], ax=ax_kde, color="red",␣
↪linewidth=2)
```

```
    ax_box.set_title(column)

plt.tight_layout()
plt.show()
```



Number of Seats
Density
Number of Seats
Number of Doors
Density
Number of Doors
Price-$
Density
Price-$
Mileage-KM
1e−5
Density
Mileage-KM
Engine Power-HP
Density
Engine Power-HP
Purchased Price-$
Density
Purchased Price-$
Sold Price-$
Density
Sold Price-$
Margin-%
Density
Margin-%

Outliers present in following:

Number of Seats Price-$ Purchased Price-$ Sold Price-$

Given nature of dataset, outliers could simply be special models of car or a rare sale where the distributor or customer overpaid for the car

Explore further by creating Boolean Masks for all variables with outliers and identify if any datapoints are outliers in all of the variables, then decide if outliers should be kept.

[11]:
```
sold_cars_numerical_data['Number of Seats'].describe()
```

[11]:
```
count    2166.000000
mean        4.775623
std         1.515273
min         2.000000
25%         4.000000
50%         5.000000
75%         5.000000
max         8.000000
Name: Number of Seats, dtype: float64
```

[12]:
```
seats_IQR = sold_cars_numerical_data['Number of Seats'].quantile(0.75) -␣
 ↪sold_cars_numerical_data['Number of Seats'].quantile(0.25)
outliers_number_of_seats = ((sold_cars_numerical_data['Number of Seats'] <␣
 ↪(sold_cars_numerical_data['Number of Seats'].quantile(0.25)-1.
 ↪5*(seats_IQR))) | (sold_cars_numerical_data['Number of Seats'] >␣
 ↪(sold_cars_numerical_data['Number of Seats'].quantile(0.75)+1.
 ↪5*(seats_IQR))))
```

[13]:
```
sold_cars_numerical_data['Price-$'].describe()
```

[13]:
```
count     2166.000000
mean      7972.853186
std       1018.585662
min       6000.000000
25%       7200.000000
50%       7900.000000
75%       8675.000000
max      10900.000000
Name: Price-$, dtype: float64
```

[14]:
```
price_IQR = sold_cars_numerical_data['Price-$'].quantile(0.75) -␣
 ↪sold_cars_numerical_data['Price-$'].quantile(0.25)
```

```
outliers_price = ((sold_cars_numerical_data['Price-$'] <␣
↪(sold_cars_numerical_data['Price-$'].quantile(0.25)-1.5*(price_IQR))) |␣
↪(sold_cars_numerical_data['Price-$'] > (sold_cars_numerical_data['Price-$'].
↪quantile(0.75)+1.5*(price_IQR))))
```

[15]:
```
sold_cars_numerical_data['Purchased Price-$'].describe()
```

[15]:
```
count     2166.000000
mean      6740.786704
std       1123.729436
min       4125.000000
25%       5954.750000
50%       6678.500000
75%       7494.500000
max      10332.000000
Name: Purchased Price-$, dtype: float64
```

[16]:
```
purchased_price_IQR = sold_cars_numerical_data['Purchased Price-$'].quantile(0.
↪75) - sold_cars_numerical_data['Purchased Price-$'].quantile(0.25)
outliers_purchased_price = ((sold_cars_numerical_data['Purchased Price-$'] <␣
↪(sold_cars_numerical_data['Purchased Price-$'].quantile(0.25)-1.
↪5*(purchased_price_IQR))) | (sold_cars_numerical_data['Purchased Price-$'] >␣
↪(sold_cars_numerical_data['Purchased Price-$'].quantile(0.75)+1.
↪5*(purchased_price_IQR))))
```

[17]:
```
sold_cars_numerical_data['Sold Price-$'].describe()
```

[17]:
```
count     2166.000000
mean      6868.772392
std       1371.740344
min       3091.000000
25%       5893.000000
50%       6839.500000
75%       7802.250000
max      11657.000000
Name: Sold Price-$, dtype: float64
```

[18]:
```
sold_price_IQR = sold_cars_numerical_data['Sold Price-$'].quantile(0.75) -␣
↪sold_cars_numerical_data['Sold Price-$'].quantile(0.25)
outliers_sold_price = ((sold_cars_numerical_data['Sold Price-$'] <␣
↪(sold_cars_numerical_data['Sold Price-$'].quantile(0.25)-1.
↪5*(sold_price_IQR))) | (sold_cars_numerical_data['Sold Price-$'] >␣
↪(sold_cars_numerical_data['Sold Price-$'].quantile(0.75)+1.
↪5*(sold_price_IQR))))
```

[19]:
```
outliers_all = outliers_number_of_seats & outliers_price &␣
↪outliers_purchased_price & outliers_sold_price
```

```
outlier_rows = sold_cars_numerical_data[outliers_all]

print(outlier_rows)
```

```
      Number of Seats  Number of Doors  Price-$  Mileage-KM  Engine Power-HP  \
6383                2                2    10900       70934              250

      Purchased Price-$  Sold Price-$  Margin-%
6383              10332         11657        13
```

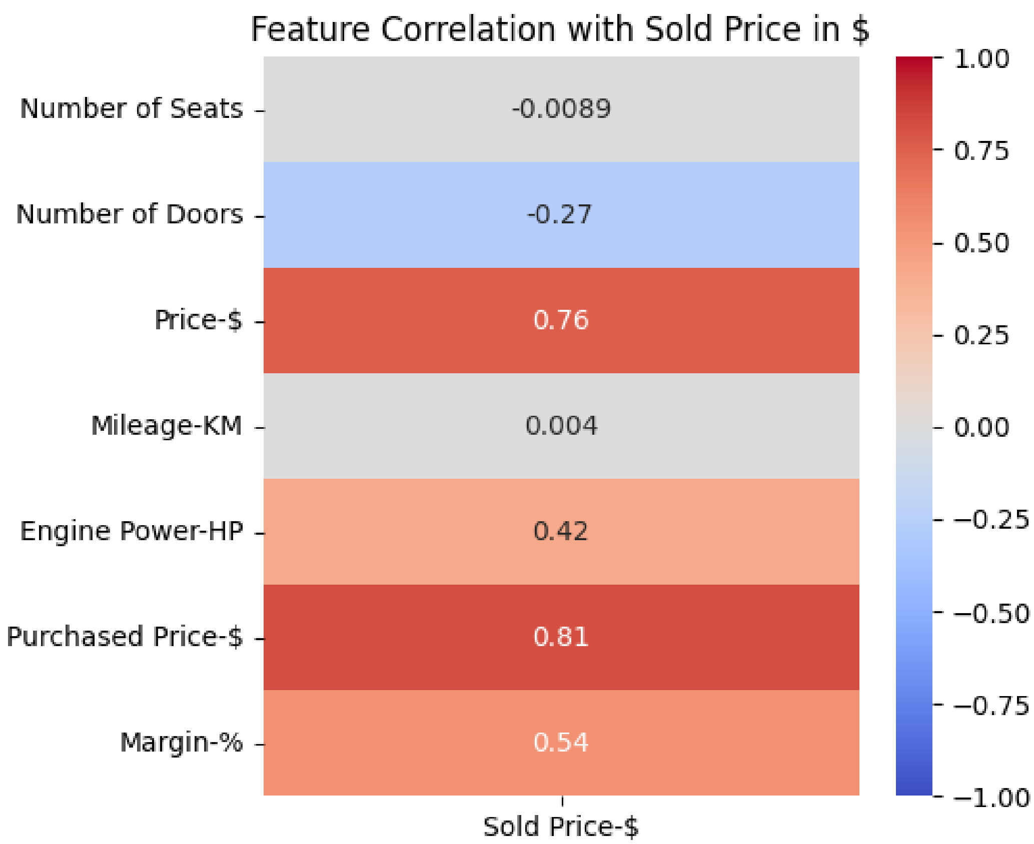
Only one data entry is an outlier in all attributes. Consider removing this datapoint as it might affect training of model

[20]:
```
# Remove the outlier from the DataFrame
sold_cars_numerical_data = sold_cars_numerical_data[~outliers_all]

sold_cars_numerical_data
```

[20]:
```
      Number of Seats  Number of Doors  Price-$  Mileage-KM  Engine Power-HP  \
1                   5                5     7600       34637              113
3                   2                2     8800       46250              250
5                   8                5     9300       38716              200
14                  5                5     9300       95835              120
17                  5                5     6900       66534              113
…                 …                …      …          …                …
9961                5                4     7300       37448              115
9981                4                5     6800       60579              103
9983                5                5     7600       68001              150
9993                5                5     7800       96006              120
9995                3                2     7100       55333              100

      Purchased Price-$  Sold Price-$  Margin-%
1                  5659          4770       -16
3                  6919          7942        15
5                  8533          9929        16
14                 8557          8896         4
17                 6382          5418       -15
…                   …             …         …
9961               6544          6255        -4
9981               5962          6110         2
9983               6751          7361         9
9993               7033          6857        -3
9995               6102          5041       -17

[2165 rows x 8 columns]
```

7 (3.2) Plot correlation matrix to identify which variables correlate strongest with "Sold Price"

[21]:
```
sold_cars_numerical_data_scaled = pd.DataFrame(StandardScaler().
    ↪fit_transform(sold_cars_numerical_data), columns=sold_cars_numerical_data.
    ↪columns)
corr_matrix = sold_cars_numerical_data_scaled.corr()
target_corr = corr_matrix[['Sold Price-$']].drop(index='Sold Price-$')
plt.figure(figsize=(5, 5))
sns.heatmap(target_corr, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title("Feature Correlation with Sold Price in $")
plt.show()
```



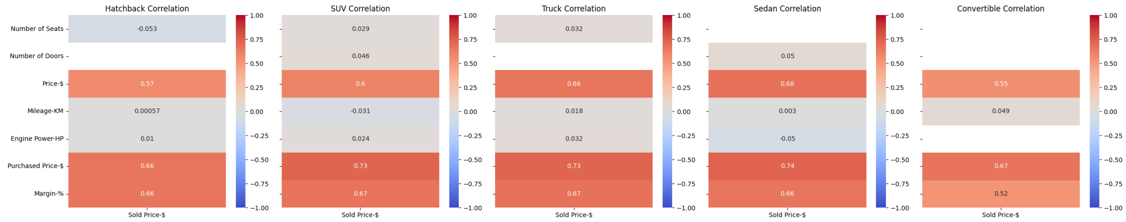
Mileage is low correlation, this doesn't really make sense because a car with low mileage should be priced higher than a car with high mileage (hypothesis)

This may be because multiple cars are being compared across different types. Consider exploring correlation by Car Type

[22]:
```
sold_cars['Car Type'].value_counts()
hatchback_data = sold_cars[sold_cars['Car Type'] == 'Hatchback']
SUV_data = sold_cars[sold_cars['Car Type'] == 'SUV']
truck_data = sold_cars[sold_cars['Car Type'] == 'Truck']
sedan_data = sold_cars[sold_cars['Car Type'] == 'Sedan']
convertible_data = sold_cars[sold_cars['Car Type'] == 'Convertible']
```

[23]:
```
hatchback_numerical_data = hatchback_data[numerical_data_labels]
SUV_numerical_data = SUV_data[numerical_data_labels]
truck_numerical_data = truck_data[numerical_data_labels]
sedan_numerical_data = sedan_data[numerical_data_labels]
convertible_numerical_data = convertible_data[numerical_data_labels]
```

[24]:
```
car_types = ['Hatchback', 'SUV', 'Truck', 'Sedan', 'Convertible']
car_data = [hatchback_data, SUV_data, truck_data, sedan_data, convertible_data]

fig, axes = plt.subplots(1, len(car_types), figsize=(5 * len(car_types), 5),␣
 ↪sharey=True)

for i, (car_type, data) in enumerate(zip(car_types, car_data)):
    numerical_data = data[numerical_data_labels]
    scaled_data = pd.DataFrame(StandardScaler().fit_transform(numerical_data),␣
 ↪columns=numerical_data.columns)
    corr_matrix = scaled_data.corr()


    target_corr = corr_matrix[['Sold Price-$']].drop(index='Sold Price-$')


    sns.heatmap(target_corr, annot=True, cmap='coolwarm', vmin=-1, vmax=1,␣
 ↪ax=axes[i])
    axes[i].set_title(f"{car_type} Correlation")

plt.tight_layout()
plt.show()
```



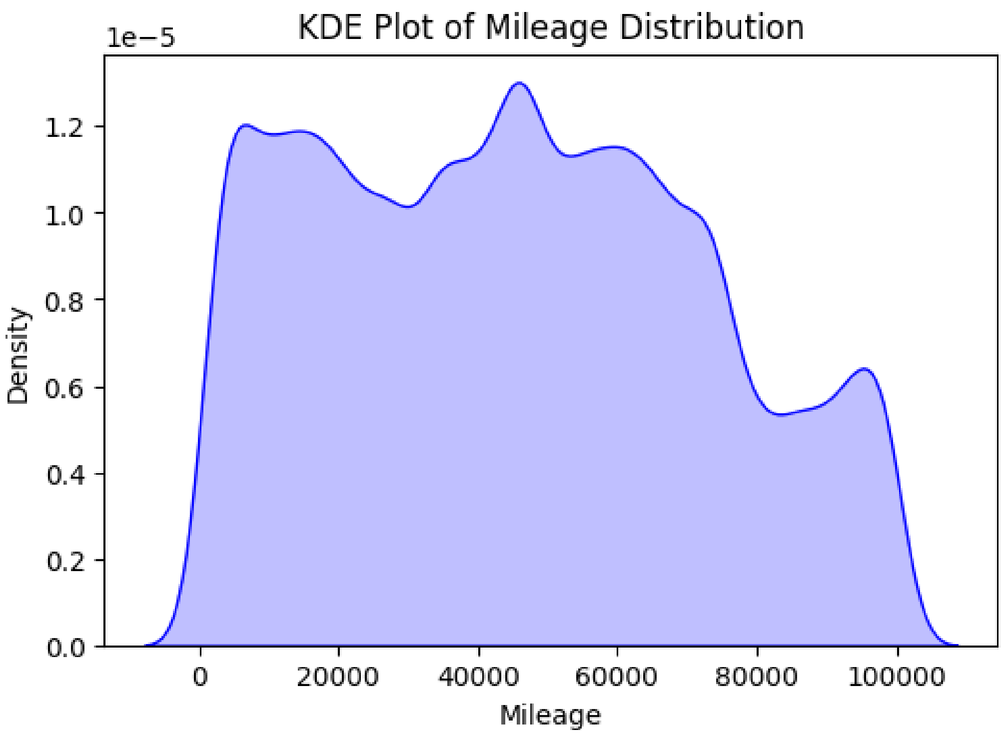
The negligible correlation between mileage and sold price, though counterintuitive, could be due to two possible reasons that require further exploration.

(1) Limited range of mileage values in the dataset If most cars have similar mileage, there wouldn't be enough variation to establish a strong relationship with price, making it difficult to observe a clear trend.

(2) Presence of sudden price fluctuations at certain mileage thresholds
Selling prices may not decrease smoothly with mileage; instead, there could be sudden drops or spikes at specific points (e.g., after surpassing a major milestone like 100,000 km). These discontinuities can weaken the overall correlation, even if mileage does influence price in a more complex, nonlinear manner.

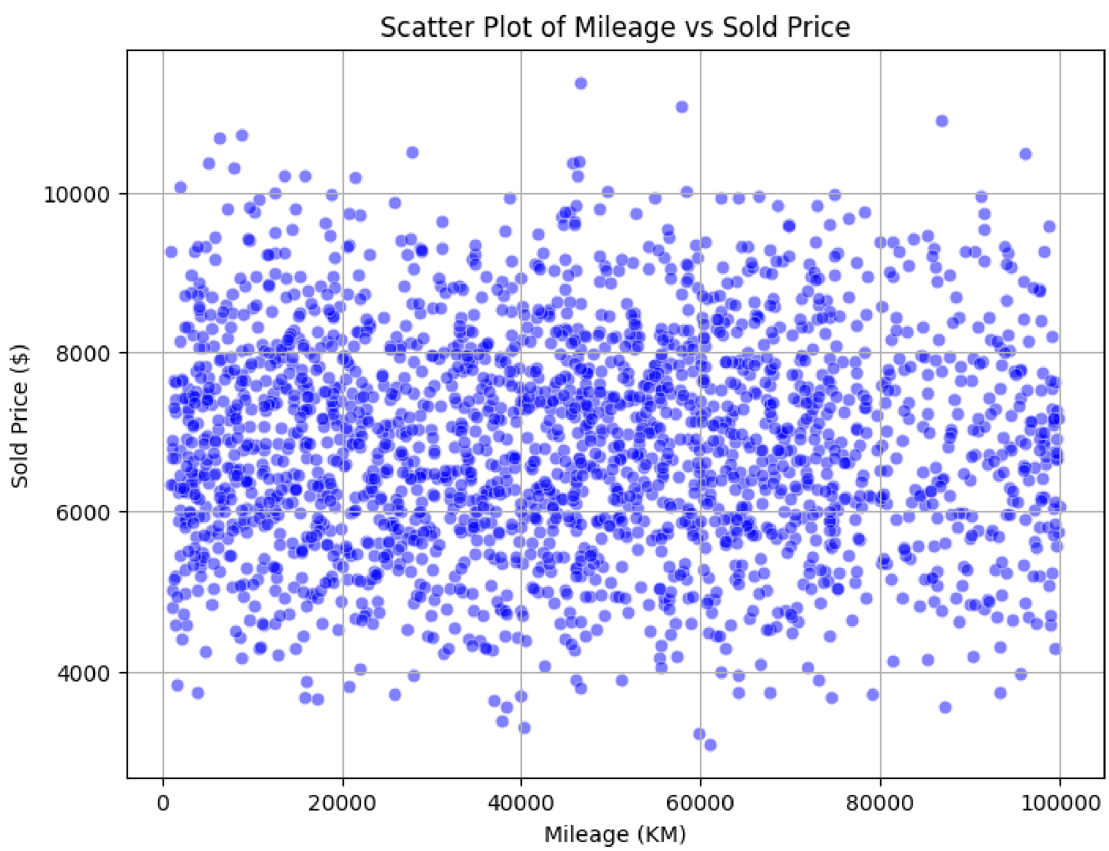
To investigate (1): consider a KDE plot

[25]:
```
#KDE of Mileage
plt.figure(figsize=(6, 4))
sns.kdeplot(sold_cars_numerical_data['Mileage-KM'], fill=True, color='blue',␣
  ↪bw_adjust=0.5)
plt.xlabel("Mileage")
plt.ylabel("Density")
plt.title("KDE Plot of Mileage Distribution")
plt.show()
```



Based on the KDE plot, the mileage values cover a wide range (from 0 to over 100,000 km). This suggests that mileage is not limited to a narrow range, meaning that point (1) does not hold as the cause for the low correlation with selling price.

[26]:
```
#Scatter Plot

plt.figure(figsize=(8, 6))
sns.scatterplot(x=sold_cars_numerical_data['Mileage-KM'],
                y=sold_cars_numerical_data['Sold Price-$'],
                alpha=0.5, color='blue')

plt.xlabel("Mileage (KM)")
plt.ylabel("Sold Price ($)")
plt.title("Scatter Plot of Mileage vs Sold Price")
plt.grid(True)

plt.show()
```



If there were sudden spikes at threshold values, we would expect to see tight clusters of points past a certain mileage value in the scatter plot. However, the points are evenly distributed with no clear linear relationship between mileage and sold price.

Hence, point (2) doesn't account for the poor correlation between mileage and sold price too.

It seems mileage truly does not affect the Sold Price according to this dataset. The variables with the strongest correlation seem to be:

- Price (According to the dataset description, it represents the Listed Price of the car in USD. We assume that means the market retail price of the car)
- Purchased Price (amount the distributor purchased the car for)
- Margin (profit made on the sale)
- Engine Power (in horsepower)

8 (3.3) Perform Linear Regression to get a model which can predict an ideal Sold Price based on numerical variables with the highest correlation

Machine Learning Technique: Linear Regression

Given that margin will not be a variable known by the seller (since the car has not yet been sold), our Linear Regression model will be based on:

- Purchased Price
- Engine Power

We have opted to utilise Purchased Price without Price to avoid Multicollinearity in the Linear Regression as Purchased Price and Price are likely dependent variables. Additionally, for a seller, it would be more important to price the car in consideration of the amount they paid for the car than the retail price the car is going for on the market.

[27]:
```
X = sold_cars_numerical_data[['Engine Power-HP', 'Purchased Price-$']]
y = sold_cars_numerical_data['Sold Price-$']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

rmse = math.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"Root Mean Squared Error: {rmse:.2f}")
print(f"R² Score: {r2:.2f}")
```

```
Root Mean Squared Error: 799.46
R² Score: 0.66
```

[28]:
```
sample_data =  pd.DataFrame([[320, 28000]], columns=['Engine Power-HP',␣
 ↪'Purchased Price-$'])

sample_pred = model.predict(sample_data)
```

```
print(f"Predicted Sold Price: ${sample_pred[0]:,.2f}")
```

```
Predicted Sold Price: $27,849.69
```

[29]:
```
sample_data =  pd.DataFrame([[320, 26000]], columns=['Engine Power-HP',␣
↪'Purchased Price-$'])

sample_pred = model.predict(sample_data)

print(f"Predicted Sold Price: ${sample_pred[0]:,.2f}")
```

```
Predicted Sold Price: $25,889.00
```

9 (4) Categorical Data Analysis

[30]:
```
categorical_data_labels = ['ID','Distributor Name','Location','Car␣
↪Name','Manufacturer Name','Car␣
↪Type','Color','Gearbox','Energy','Manufactured Year',
                    'Purchased Date','Car Sale Status','Sold Date','Sales Agent␣
↪Name','Sales Rating','Feedback','Sold Price-$']
sold_cars_categorical_data = sold_cars[categorical_data_labels]
```

A cursory analysis of the data indicates certain variables can be omitted as they are **not relevant** to qualities of the car that can be controlled by a prospective seller.

These include: - ID - Distributor Name - Sales Agent Name - Sales Rating - Car Sale Status - Feedback

Since Car Names are strongly related to Manufacturer Name in the sense that it wouldn't make much sense for a seller to sell their car with only the Car Model name without the Manufacturer Name, it would be simpler to combine the 2 columns into one variable "Car Name-Manufacturer Name"

[31]:
```
sold_cars_categorical_data = sold_cars_categorical_data.drop(columns =␣
↪['ID','Distributor Name','Sales Agent Name','Sales Rating','Car Sale␣
↪Status','Feedback'])
```

[32]:
```
sold_cars_categorical_data['Car Name-Manufacturer Name'] = (
    sold_cars_categorical_data['Car Name'] + ' - ' +␣
↪sold_cars_categorical_data['Manufacturer Name']
)
sold_cars_categorical_data = sold_cars_categorical_data.drop(columns = ['Car␣
↪Name','Manufacturer Name'])
```

Since Sold and Purchased Dates are recorded to the precision of the day, it would be easier to analyse the dates by grouping them by year

[33]:

```
sold_cars_categorical_data['Sold Date'] = pd.
↪to_datetime(sold_cars_categorical_data['Sold Date'])
sold_cars_categorical_data['Purchased Date'] = pd.
↪to_datetime(sold_cars_categorical_data['Purchased Date'])
sold_cars_categorical_data['Year Sold'] = sold_cars_categorical_data['Sold␣
↪Date'].dt.year
sold_cars_categorical_data['Year Purchased'] =␣
↪sold_cars_categorical_data['Purchased Date'].dt.year
```

[34]:

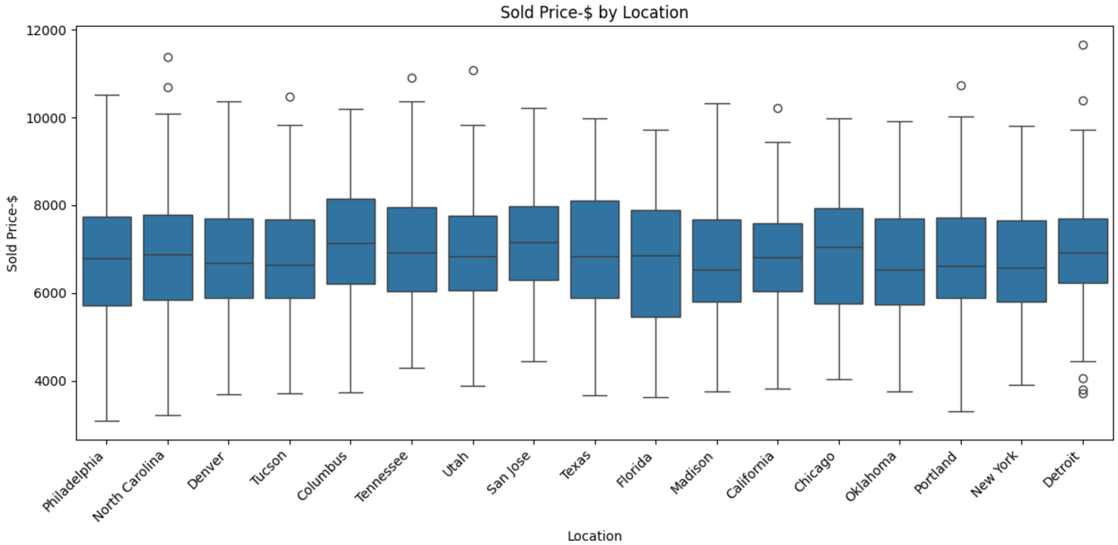
```
sold_cars_categorical_data['Car Name-Manufacturer Name'].value_counts()
```

[34]:

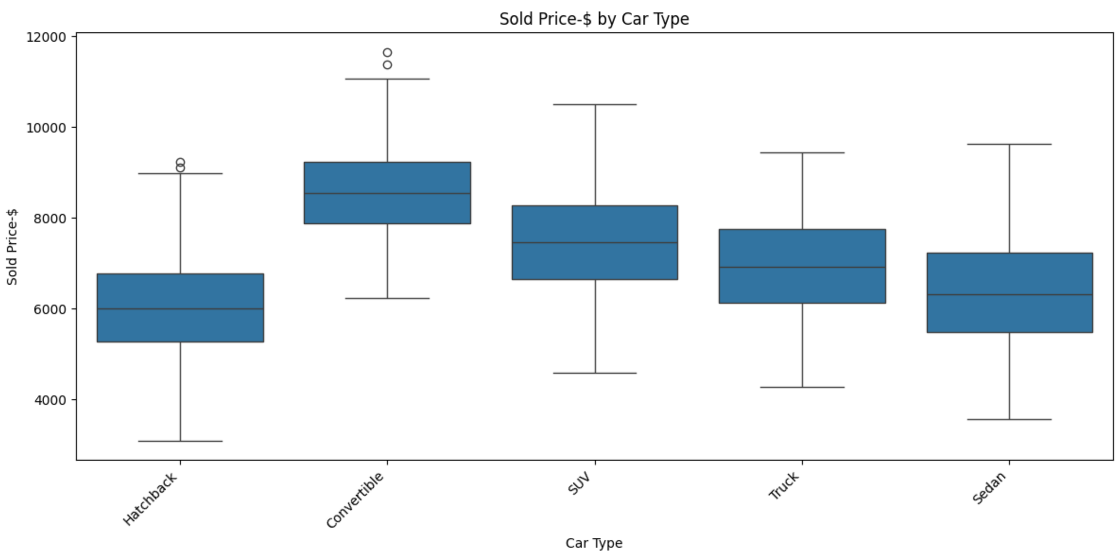
```
Car Name-Manufacturer Name
Yodha - Tata          194
Creta - Hyundai       184
Kags - Renault        181
Fortuner - Toyota     179
i20 - Hyundai         174
Etriga - Maruti       170
Hilux - Toyota        167
Plato - Prazo         162
Thar - Mahindra       161
Dzire - Maruti        154
Seltos - Kia          151
Swift - Maruti        149
Scorpio - Mahindra    140
Name: count, dtype: int64
```

[35]:

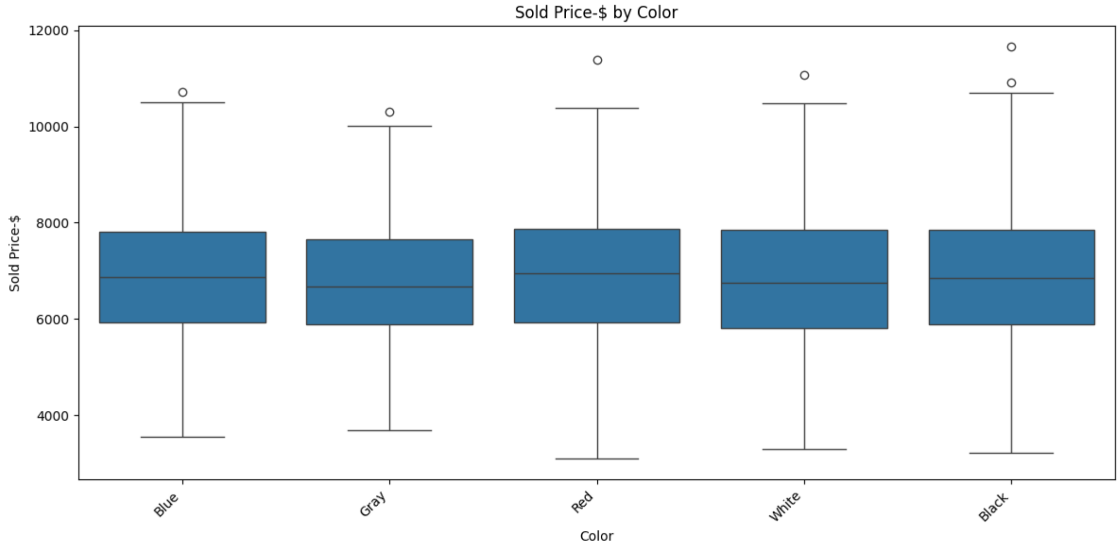
```
# Define the target variable
target = 'Sold Price-$'
columns = sold_cars_categorical_data.columns
skipped_columns = ['Sold Price-$','Sold Date','Purchased Date','Year Sold']
# Loop through each categorical variable and plot boxplot
for col in columns:
    if col not in skipped_columns:
        plt.figure(figsize=(12, 6))
        sns.boxplot(x=sold_cars_categorical_data[col],␣
↪y=sold_cars_categorical_data[target])
        plt.title(f'{target} by {col}')
        plt.xticks(rotation=45, ha='right')
        plt.tight_layout()
        plt.show()
```

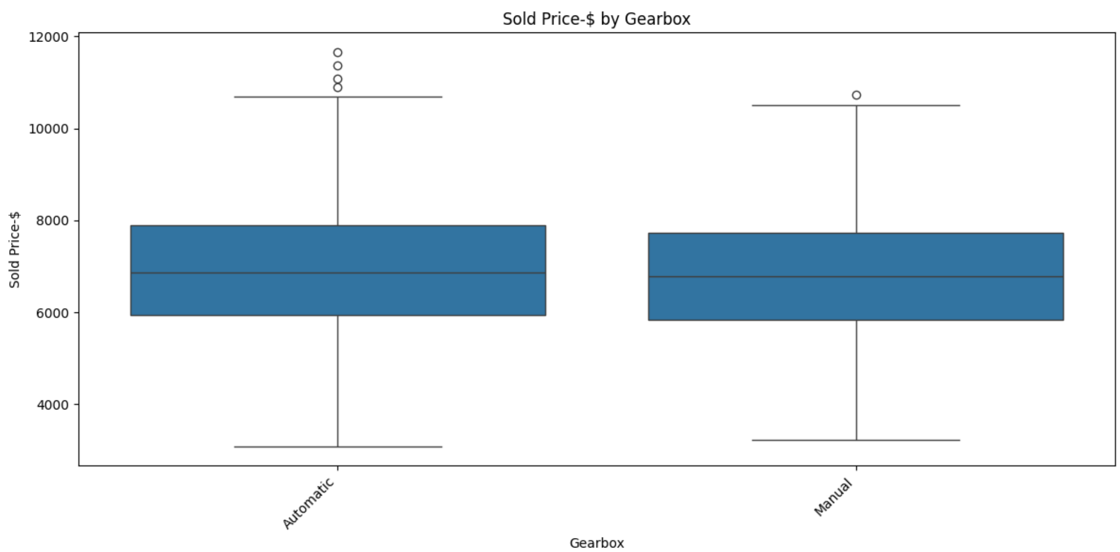


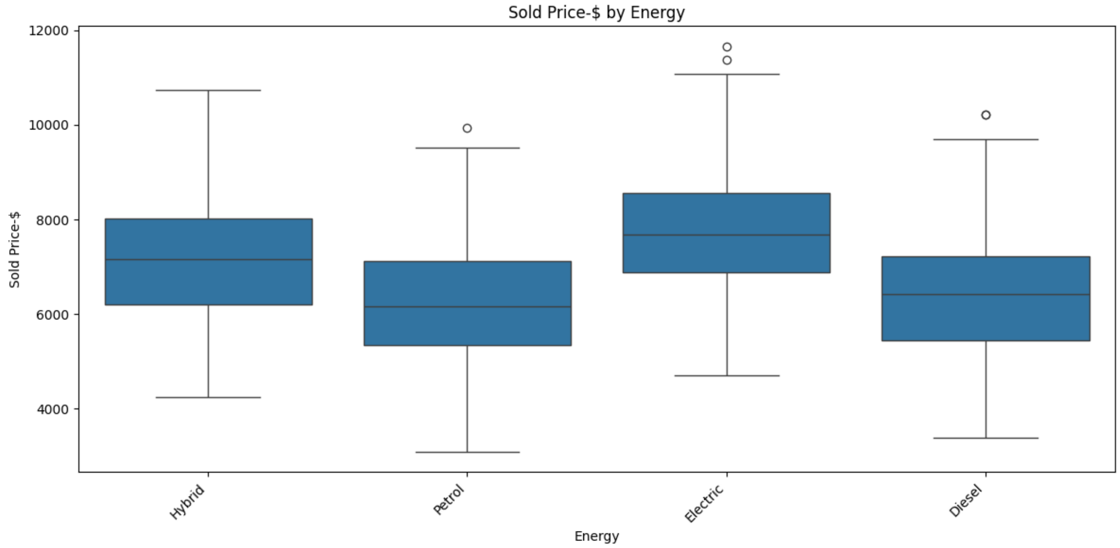
Sold Price-$ by Location
Sold Price-$
12000
10000
8000
6000
4000
Philadelphia
North Carolina
Denver
Tucson
Columbus
Tennessee
Utah
San Jose
Texas
Florida
Madison
California
Chicago
Oklahoma
Portland
New York
Detroit
Location

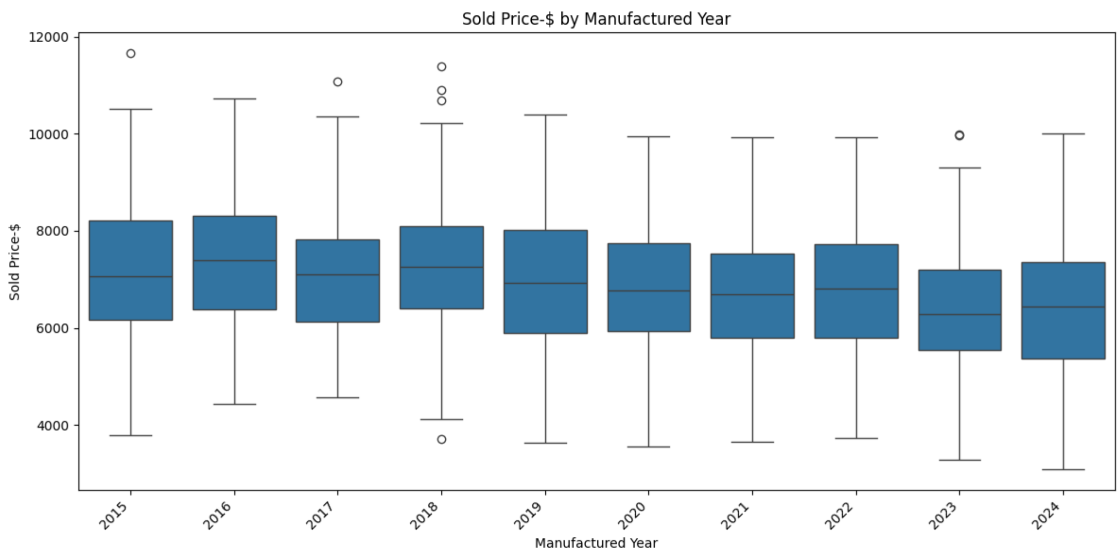


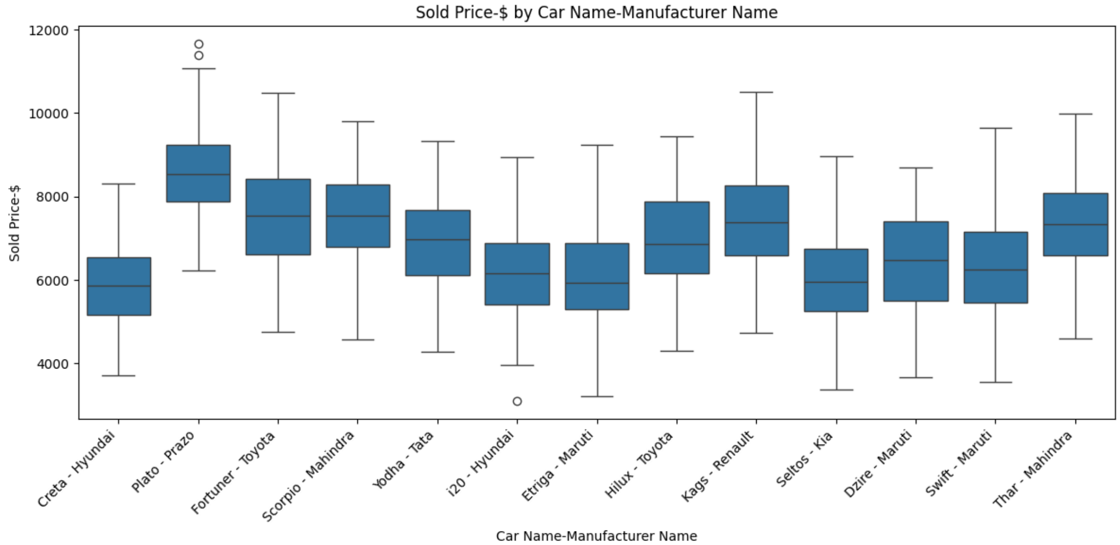
Sold Price-$ by Car Type
Sold Price-$
12000
10000
8000
6000
4000
Hatchback
Convertible
SUV
Truck
Sedan
Car Type

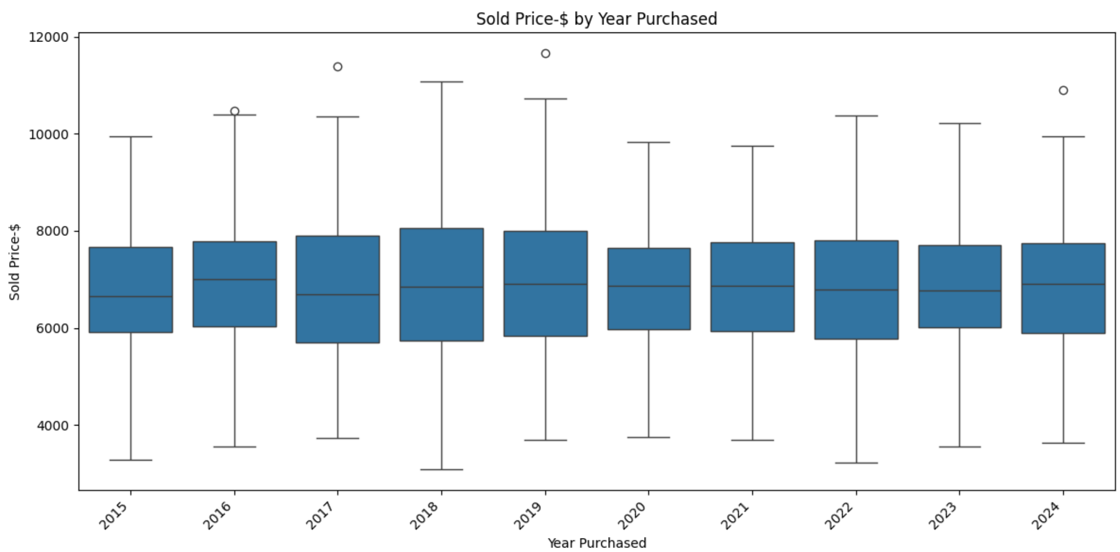

Sold Price-$ by Color
Sold Price-$
12000
10000
8000
6000
4000
Blue
Gray
Red
White
Black
Color


Sold Price-$ by Gearbox
Sold Price-$
12000
10000
8000
6000
4000
Automatic
Manual
Gearbox


Sold Price-$ by Energy
12000
10000
8000
6000
4000
Sold Price-$
Hybrid
Petrol
Electric
Diesel
Energy


Sold Price-$ by Manufactured Year
12000
10000
8000
6000
4000
Sold Price-$
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
Manufactured Year





By examining the differences in the boxplot for each category in a categorical variable, we can derive if the categorical variable affects Sold Price.

It can be observed that the following variables have a significant effect on Sold Price: - Car Type - Energy - Car Name-Manufacturer Name

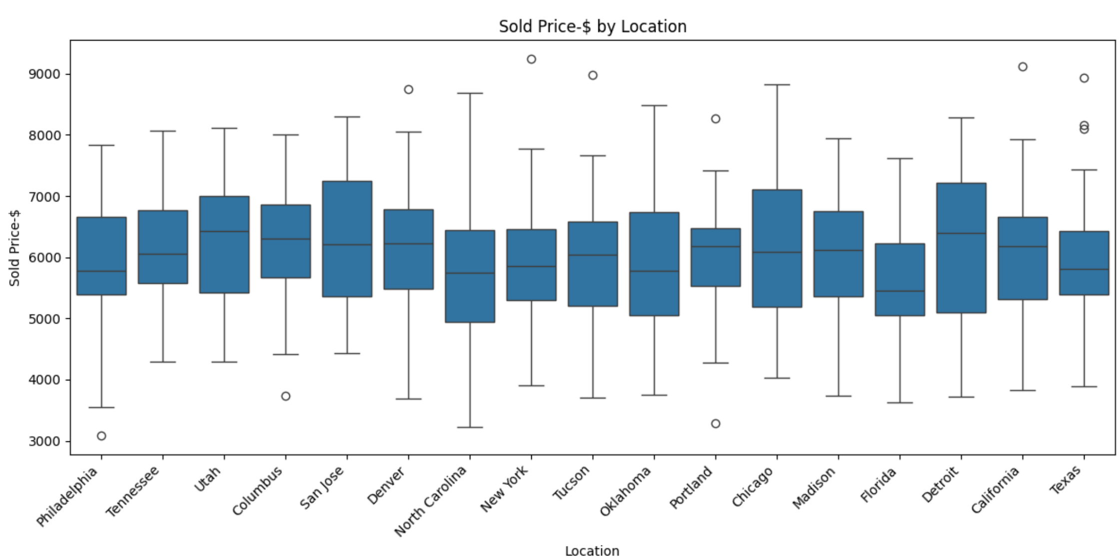
However, this may be due to the fact that different car types (that have different manufacturers, price distributions etc.) are analysed together, producing inconsequential results.

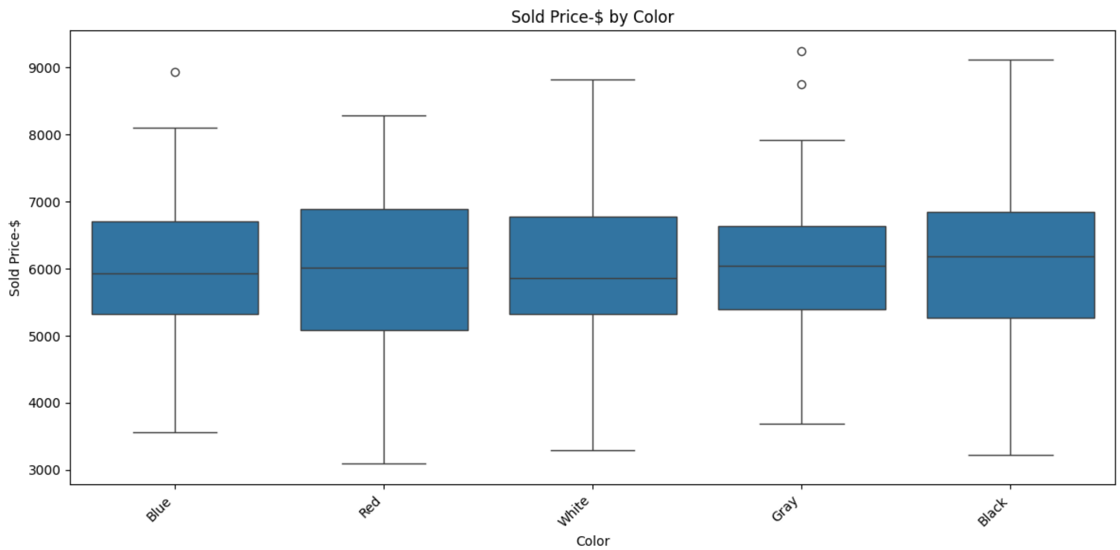
In line with our goal to recommend a price for potential sellers of cars, we will perform an analysis for each Car Type to determine the effect of the categorical variables on Sold Price.

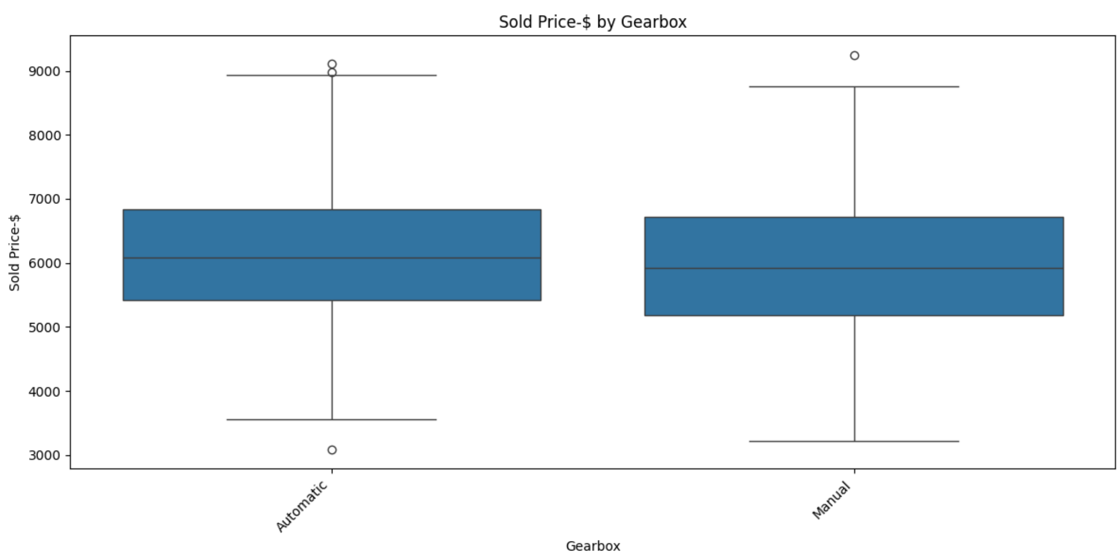
```
[36]: hatchback_categorical_data = sold_cars_categorical_data[sold_cars_categorical_data['Car Type'] == 'Hatchback']
convertible_categorical_data = sold_cars_categorical_data[sold_cars_categorical_data['Car Type'] == 'Convertible']
SUV_categorical_data = sold_cars_categorical_data[sold_cars_categorical_data['Car Type'] == 'SUV']
truck_categorical_data = sold_cars_categorical_data[sold_cars_categorical_data['Car Type'] == 'Truck']
sedan_categorical_data = sold_cars_categorical_data[sold_cars_categorical_data['Car Type'] == 'Sedan']
```

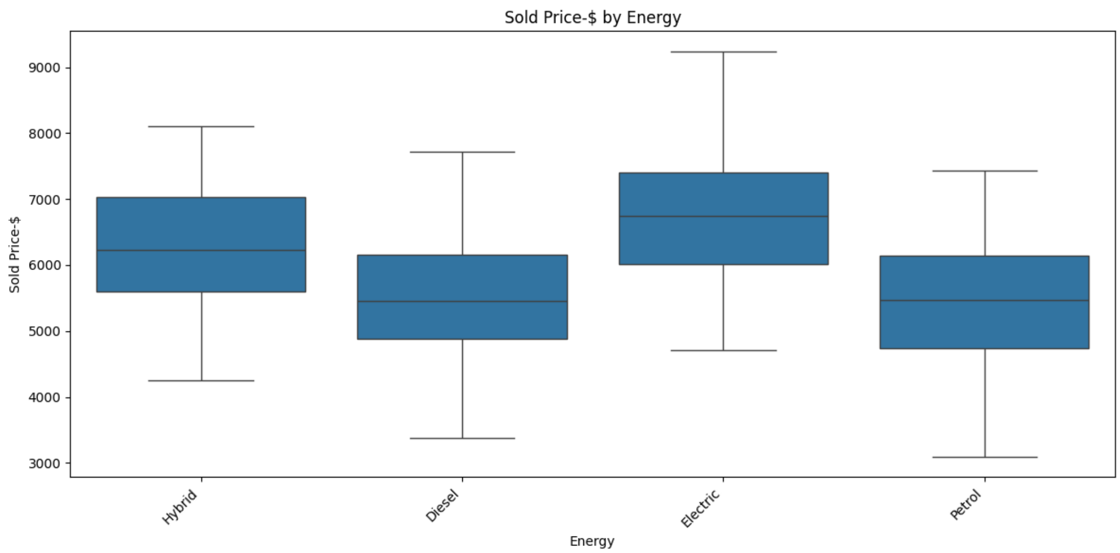
10 Hatchback Analysis

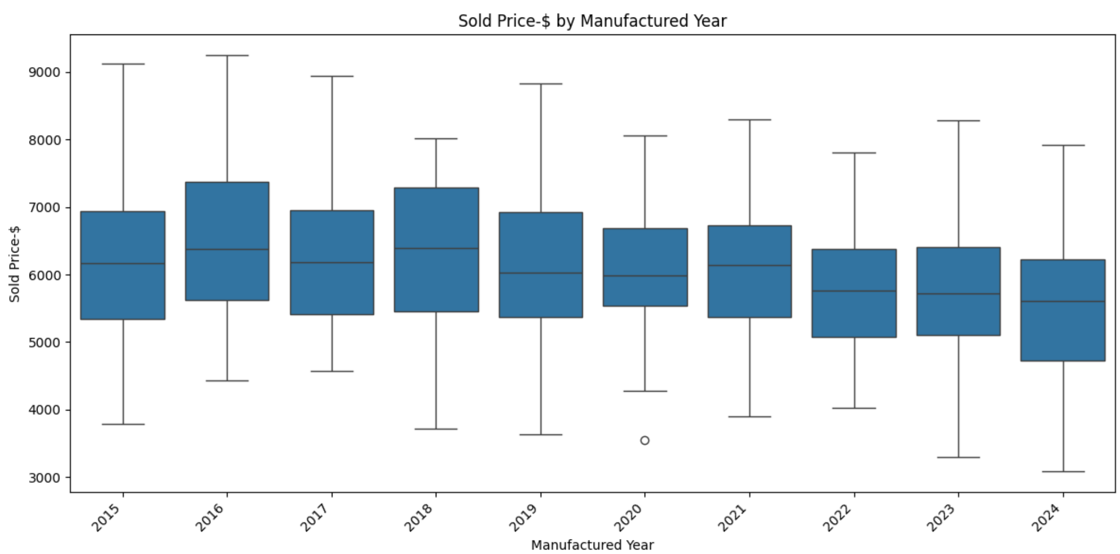
```
[37]: # Define the target variable
target = 'Sold Price-$'
columns = hatchback_categorical_data.columns
skipped_columns = ['Sold Price-$','Sold Date','Purchased Date','Car Type','Year Sold']
# Loop through each categorical variable and plot boxplot
for col in columns:
    if col not in skipped_columns:
        plt.figure(figsize=(12, 6))
        sns.boxplot(x=hatchback_categorical_data[col], y=hatchback_categorical_data[target])
        plt.title(f'{target} by {col}')
        plt.xticks(rotation=45, ha='right')
        plt.tight_layout()
        plt.show()
```

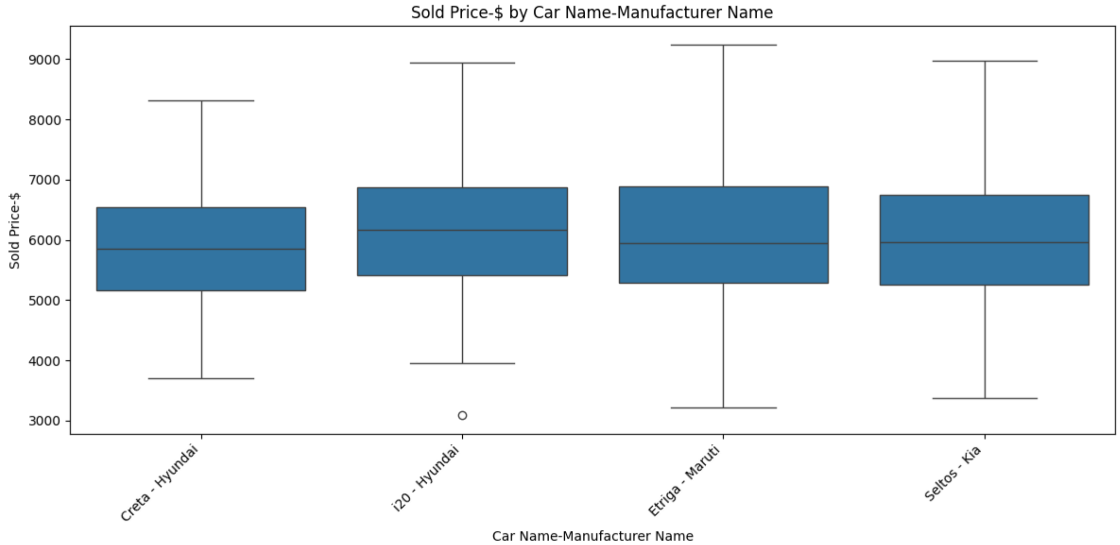


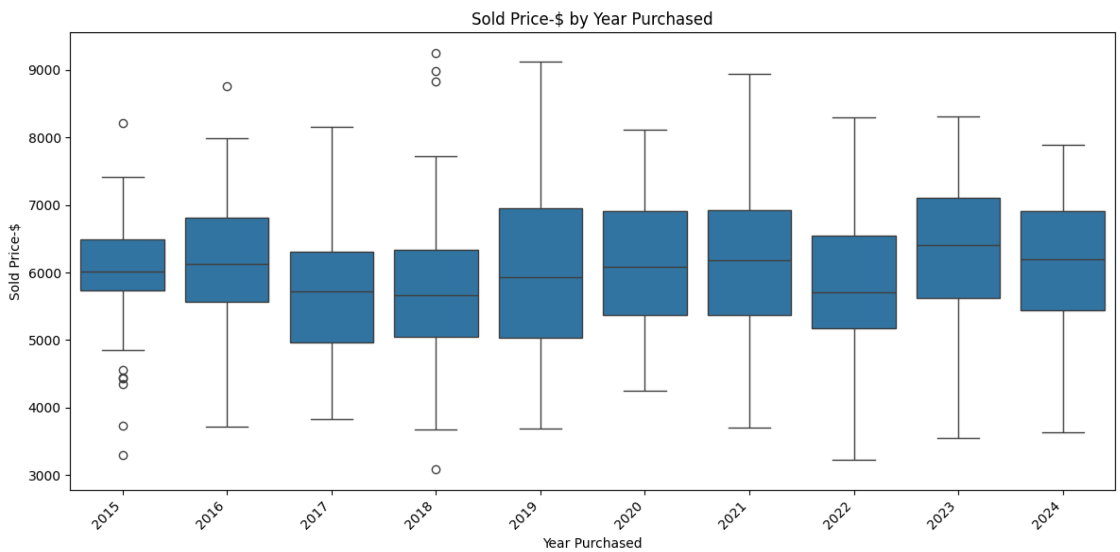

Sold Price-$ by Color
Sold Price-$
9000
8000
7000
6000
5000
4000
3000
Blue
Red
White
Gray
Black
Color


Sold Price-$ by Gearbox
Sold Price-$
9000
8000
7000
6000
5000
4000
3000
Automatic
Manual
Gearbox


Sold Price-$ by Energy
9000
8000
7000
6000
5000
4000
3000
Sold Price-$
Hybrid
Diesel
Electric
Petrol
Energy


Sold Price-$ by Manufactured Year
9000
8000
7000
6000
5000
4000
3000
Sold Price-$
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
Manufactured Year

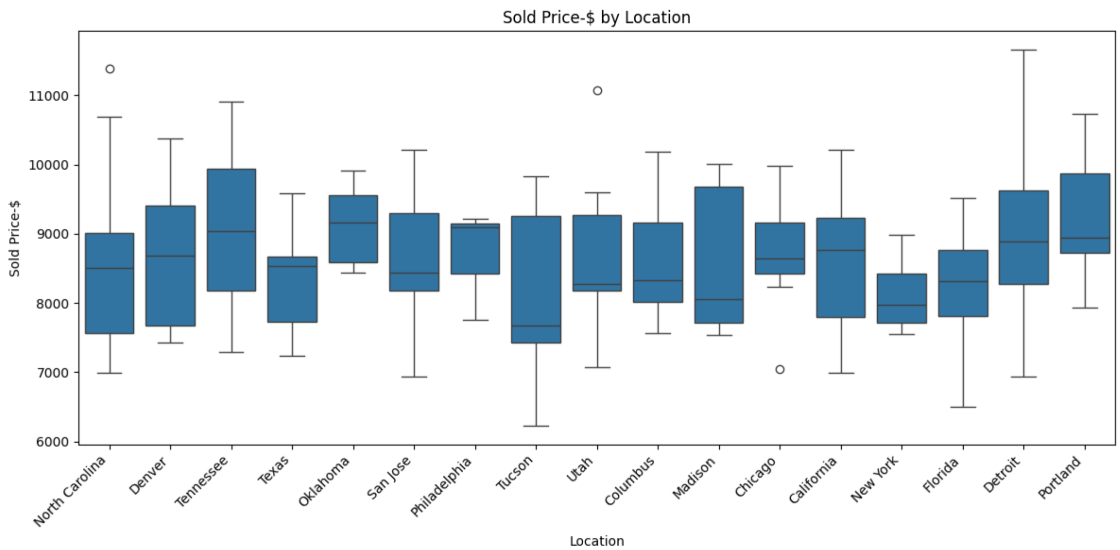


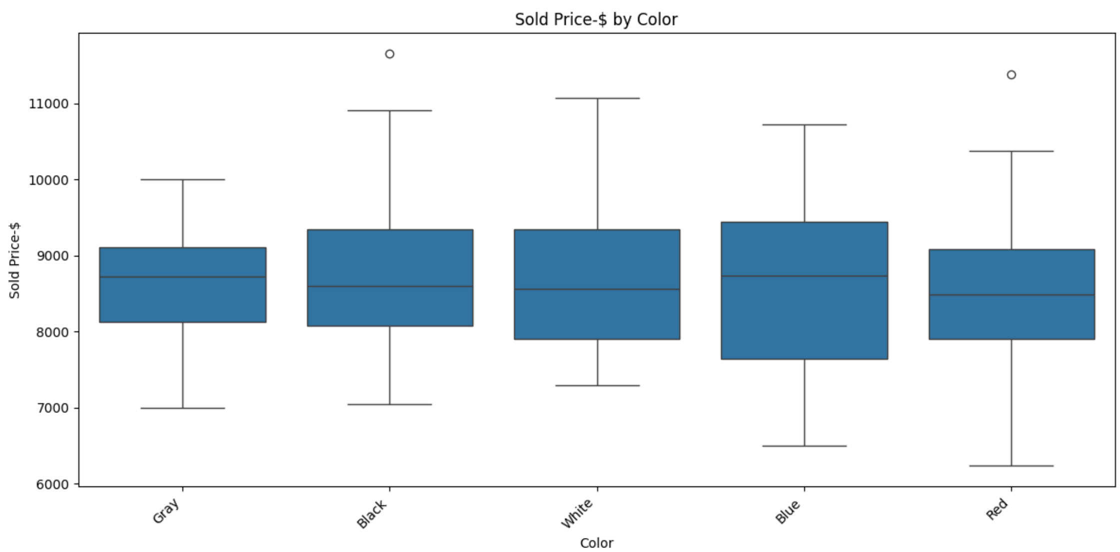


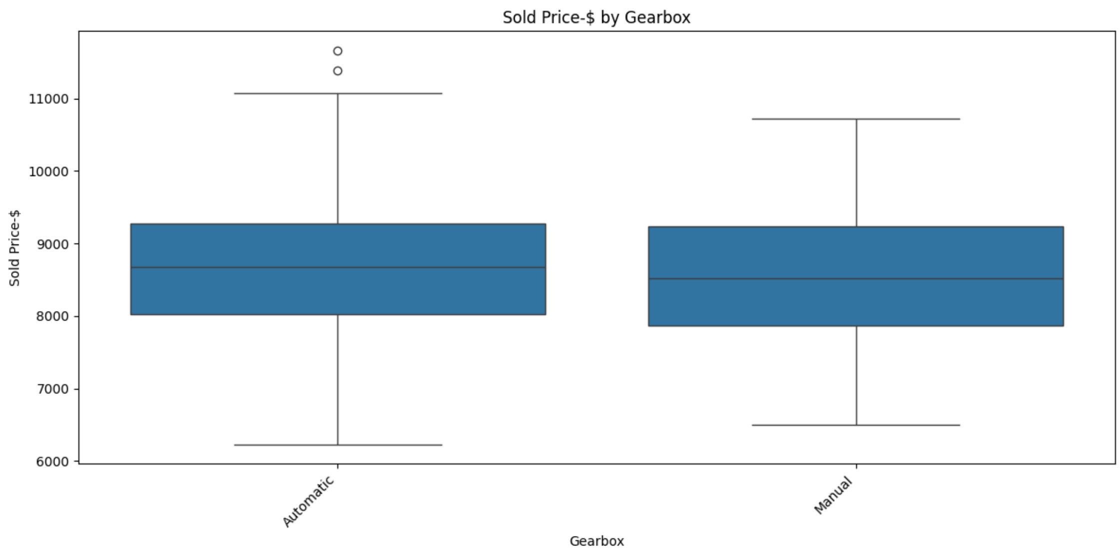
11 Convertible Analysis

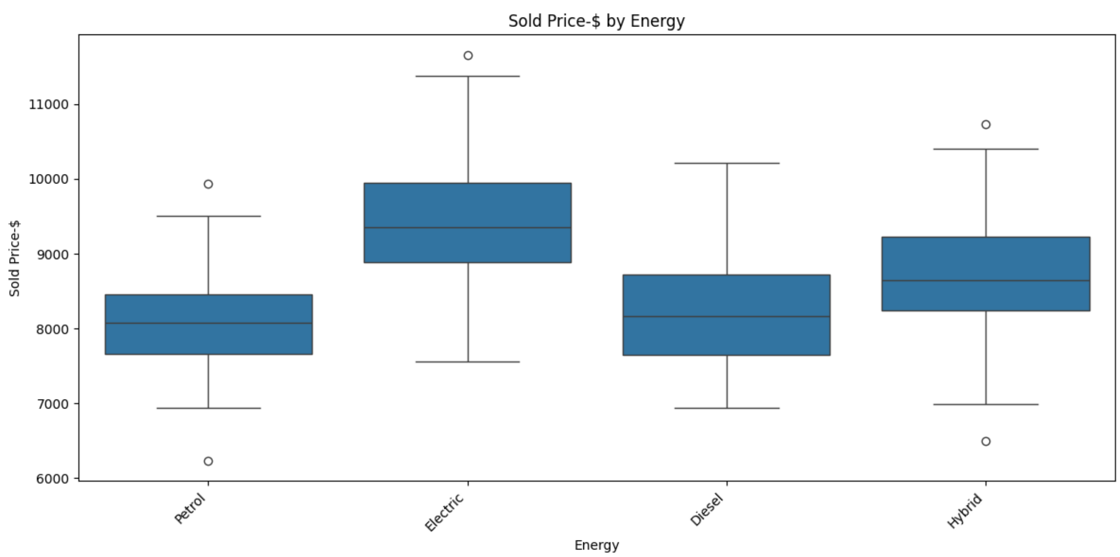
[38]:
```
# Define the target variable
target = 'Sold Price-$'
columns = convertible_categorical_data.columns
skipped_columns = ['Sold Price-$','Sold Date','Purchased Date','Car Type','Year␣
↪Sold']
# Loop through each categorical variable and plot boxplot
for col in columns:
```

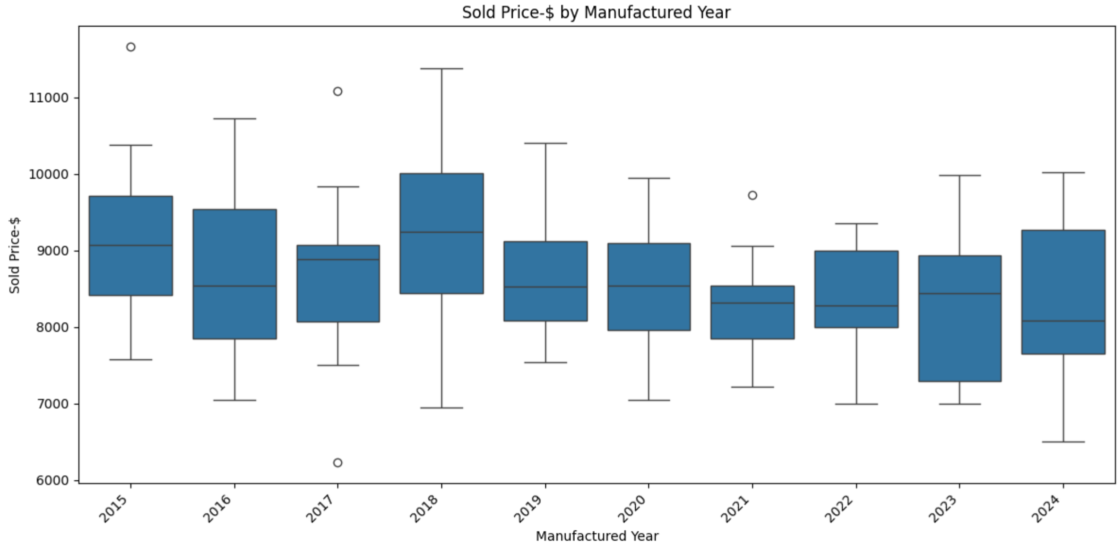
```
    if col not in skipped_columns:
        plt.figure(figsize=(12, 6))
        sns.boxplot(x=convertible_categorical_data[col],␣
↪y=convertible_categorical_data[target])
        plt.title(f'{target} by {col}')
        plt.xticks(rotation=45, ha='right')
        plt.tight_layout()
        plt.show()
```

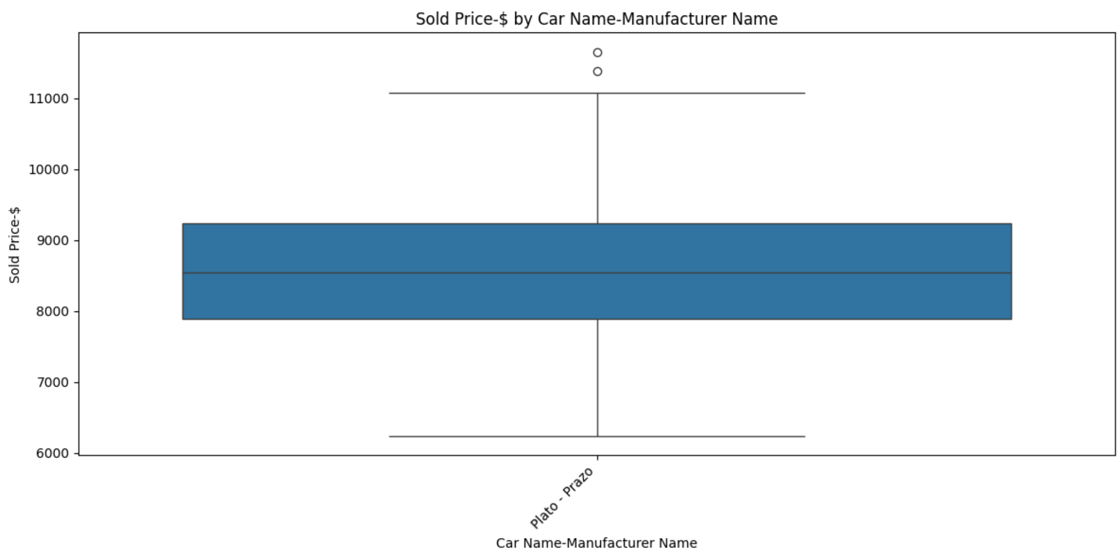


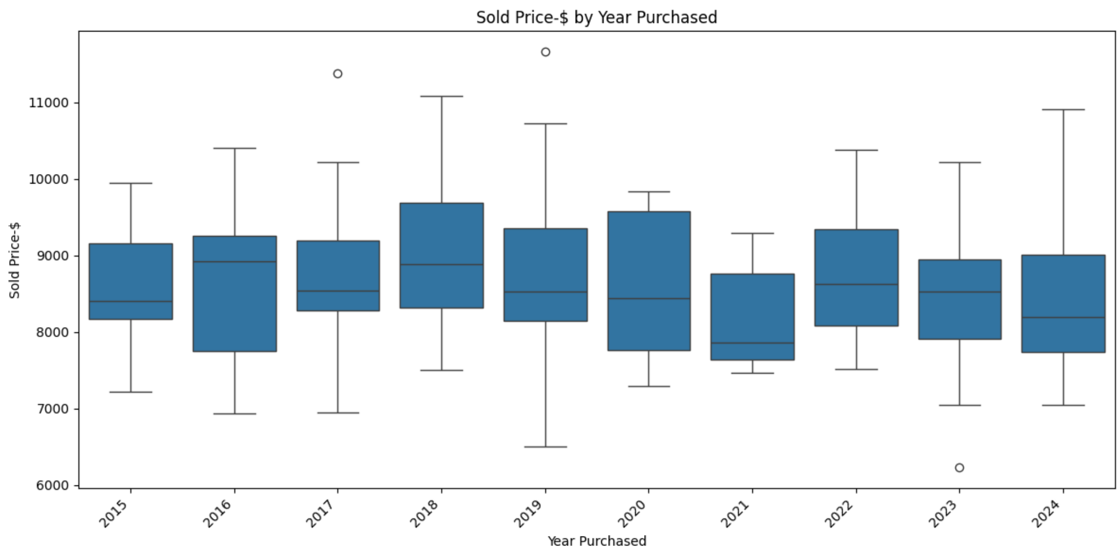



Sold Price-$ by Gearbox
11000
10000
9000
8000
7000
6000
Sold Price-$
Automatic
Manual
Gearbox


Sold Price-$ by Energy
11000
10000
9000
8000
7000
6000
Sold Price-$
Petrol
Electric
Diesel
Hybrid
Energy

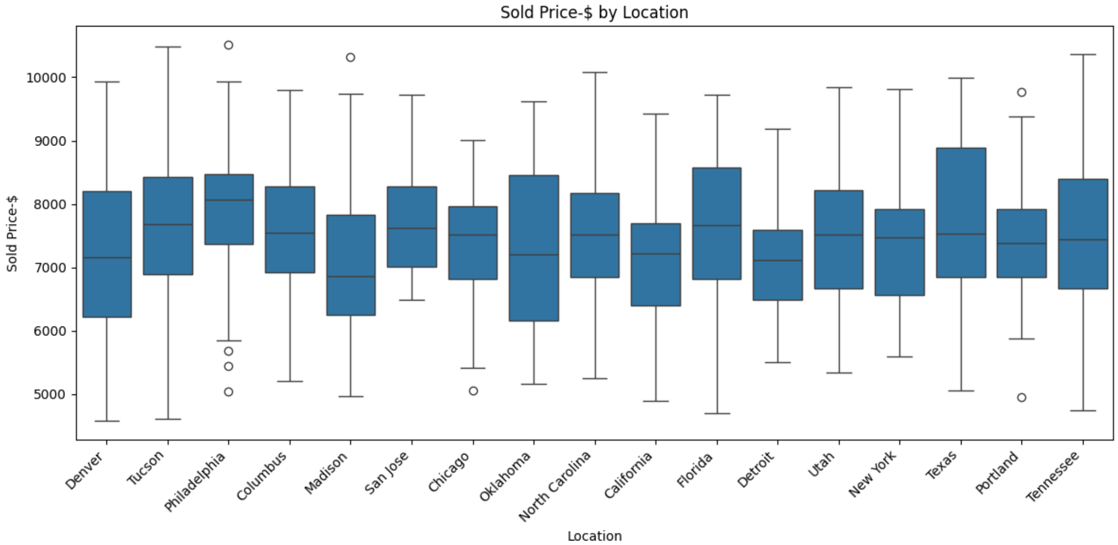

Sold Price-$ by Manufactured Year
Sold Price-$
Manufactured Year
6000
7000
8000
9000
10000
11000
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024

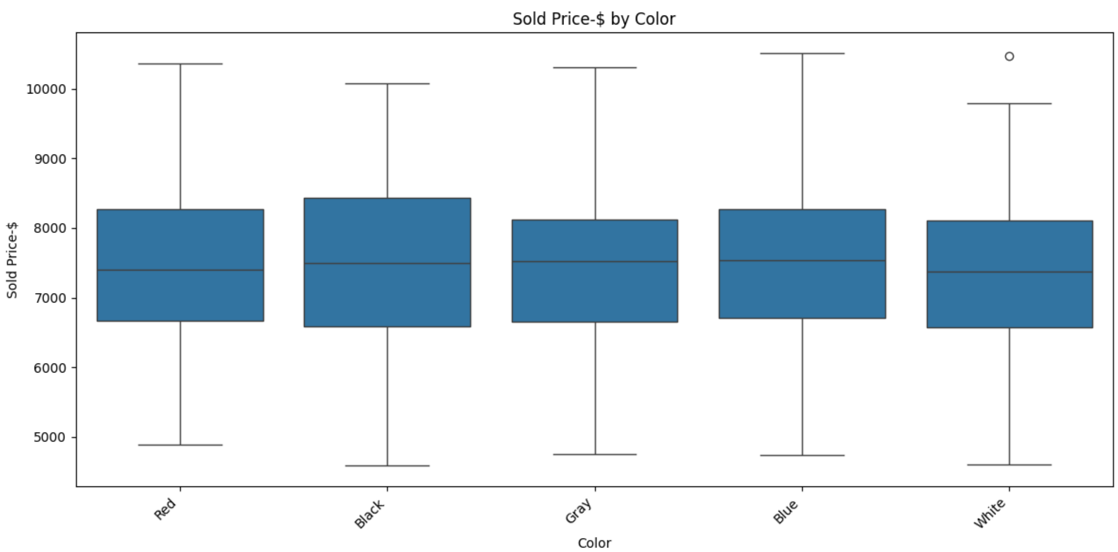

Sold Price-$ by Car Name-Manufacturer Name
Sold Price-$
Car Name-Manufacturer Name
6000
7000
8000
9000
10000
11000
Plato - Prazo

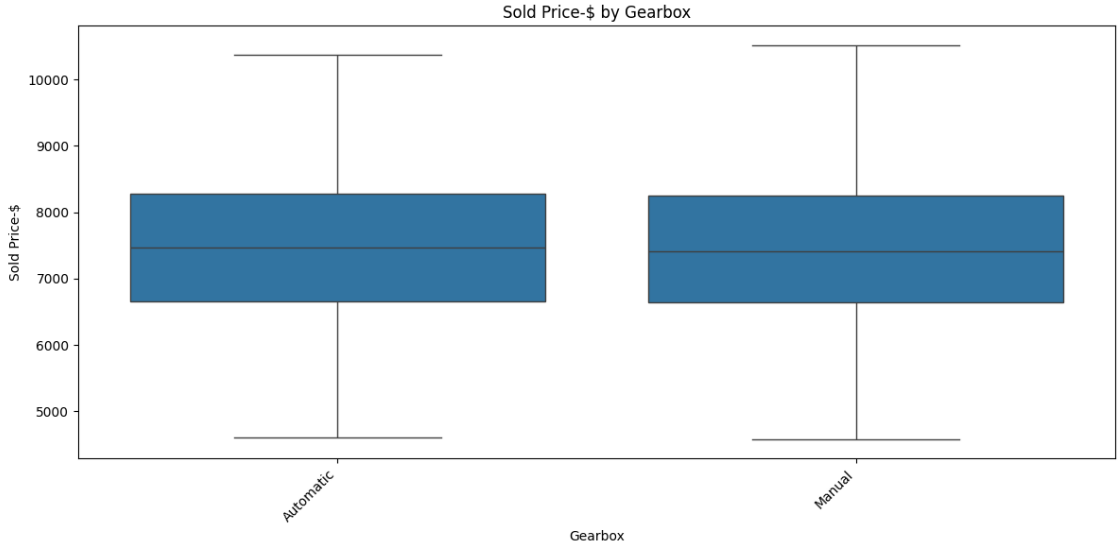


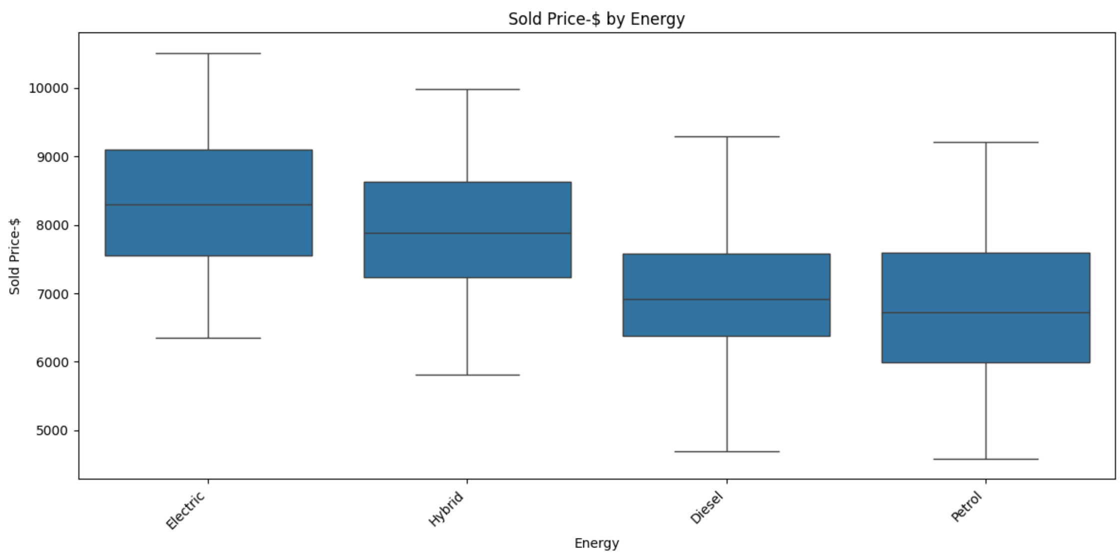
12 SUV Analysis

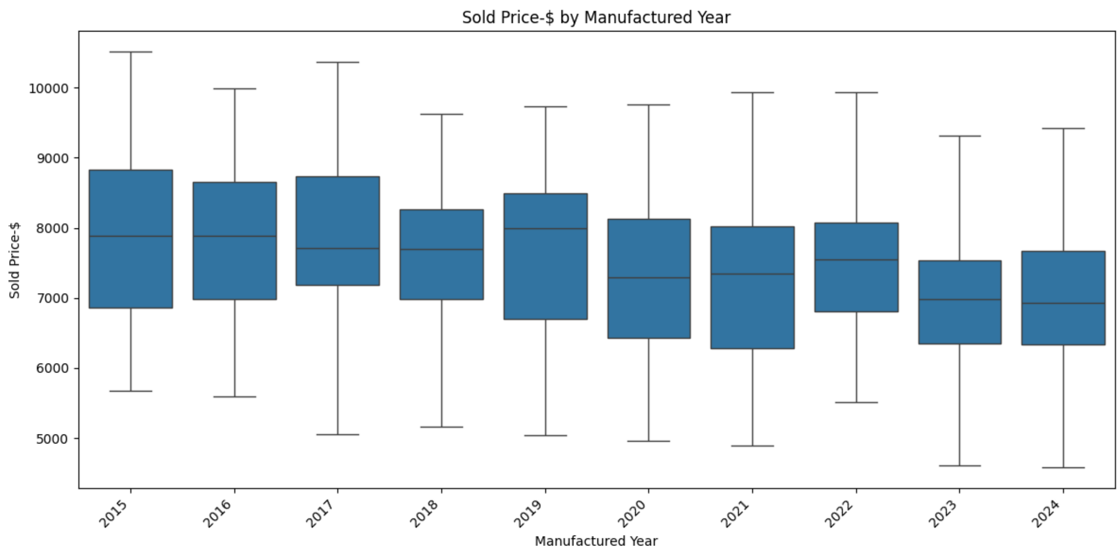
[39]:
```
# Define the target variable
target = 'Sold Price-$'
columns = SUV_categorical_data.columns
skipped_columns = ['Sold Price-$','Sold Date','Purchased Date','Car Type','Year␣
↪Sold']
# Loop through each categorical variable and plot boxplot
for col in columns:
    if col not in skipped_columns:
        plt.figure(figsize=(12, 6))
        sns.boxplot(x=SUV_categorical_data[col], y=SUV_categorical_data[target])
        plt.title(f'{target} by {col}')
        plt.xticks(rotation=45, ha='right')
        plt.tight_layout()
        plt.show()
```

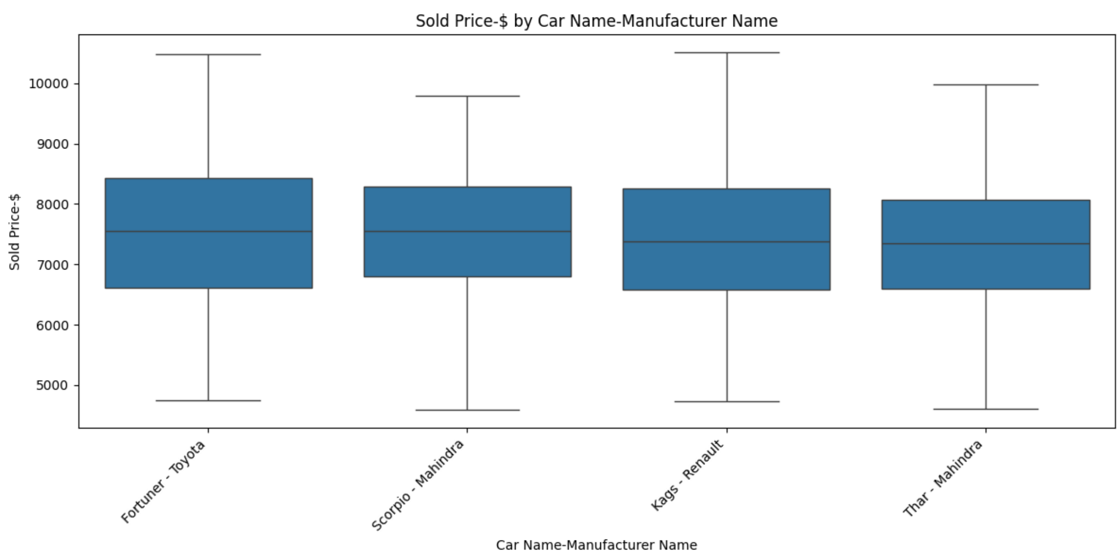

Sold Price-$ by Location
Sold Price-$
10000
9000
8000
7000
6000
5000
Denver
Tucson
Philadelphia
Columbus
Madison
San Jose
Chicago
Oklahoma
North Carolina
California
Florida
Detroit
Utah
New York
Texas
Portland
Tennessee
Location

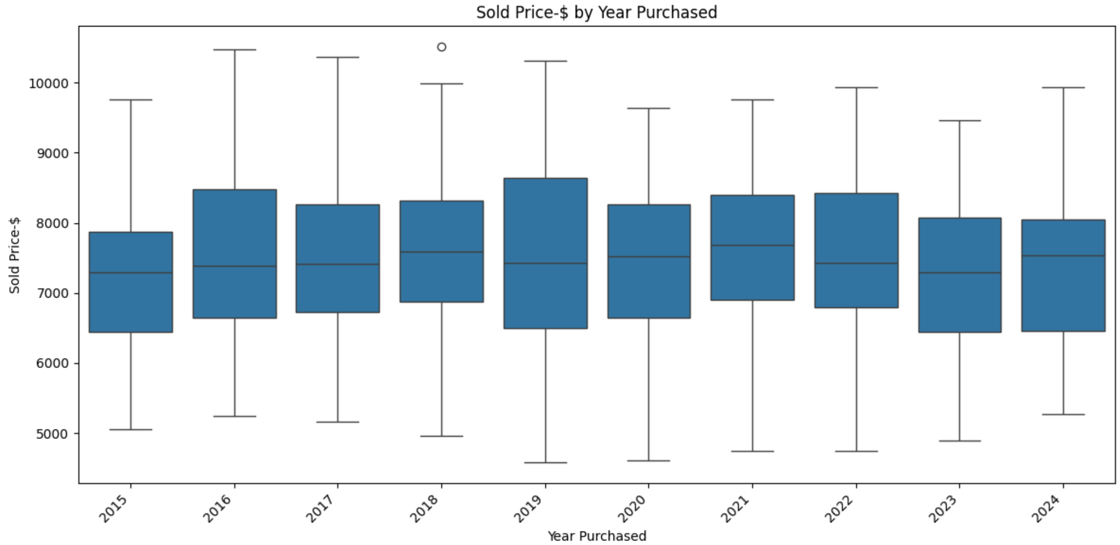

Sold Price-$ by Color
Sold Price-$
10000
9000
8000
7000
6000
5000
Red
Black
Gray
Blue
White
Color


Sold Price-$ by Gearbox
10000
9000
8000
7000
6000
5000
Sold Price-$
Automatic
Manual
Gearbox


Sold Price-$ by Energy
10000
9000
8000
7000
6000
5000
Sold Price-$
Electric
Hybrid
Diesel
Petrol
Energy

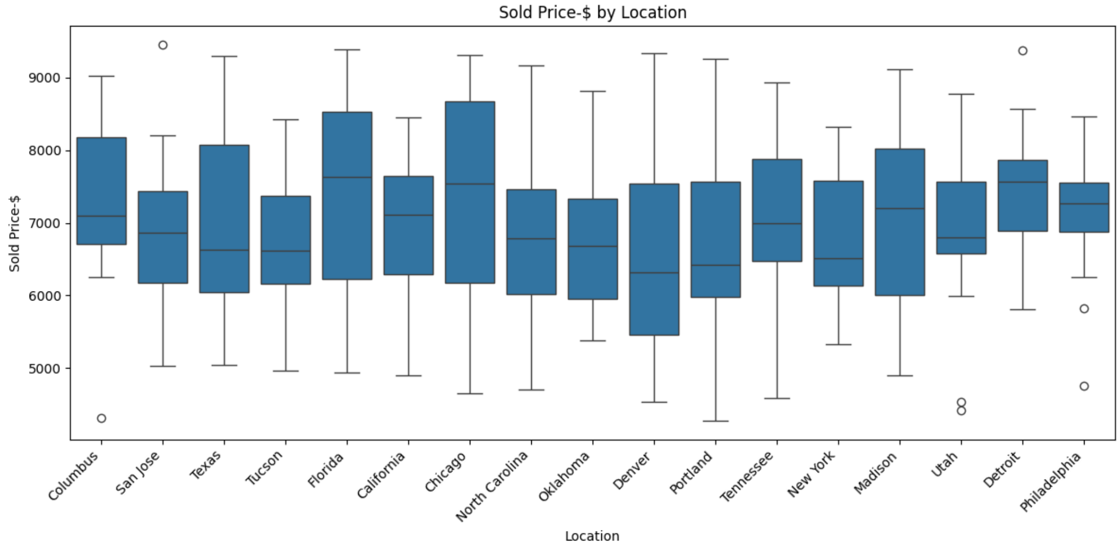

Sold Price-$ by Manufactured Year
Sold Price-$
10000
9000
8000
7000
6000
5000
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
Manufactured Year

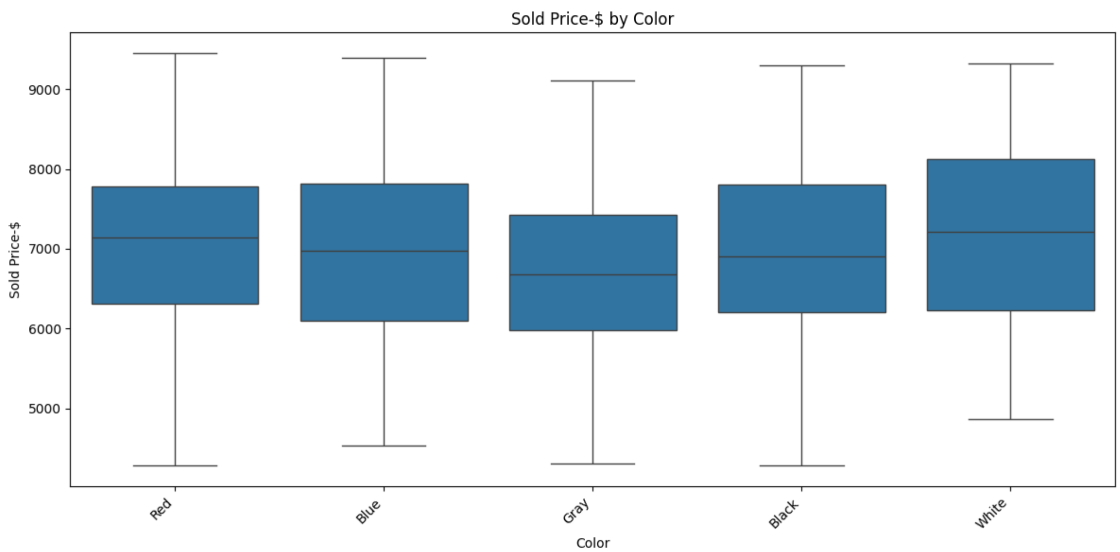

Sold Price-$ by Car Name-Manufacturer Name
Sold Price-$
10000
9000
8000
7000
6000
5000
Fortuner - Toyota
Scorpio - Mahindra
Kags - Renault
Thar - Mahindra
Car Name-Manufacturer Name

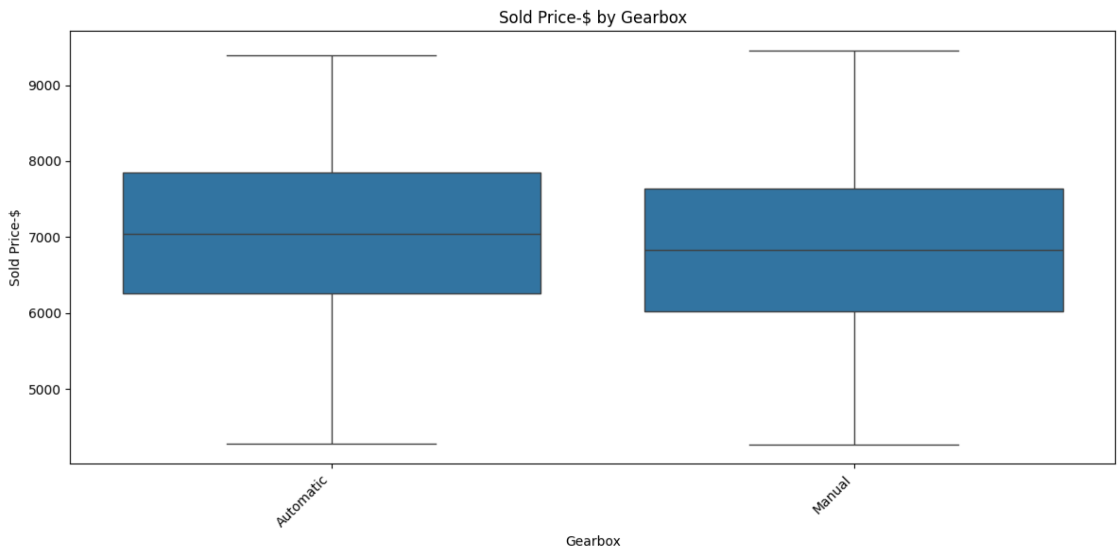


13 Truck Analysis

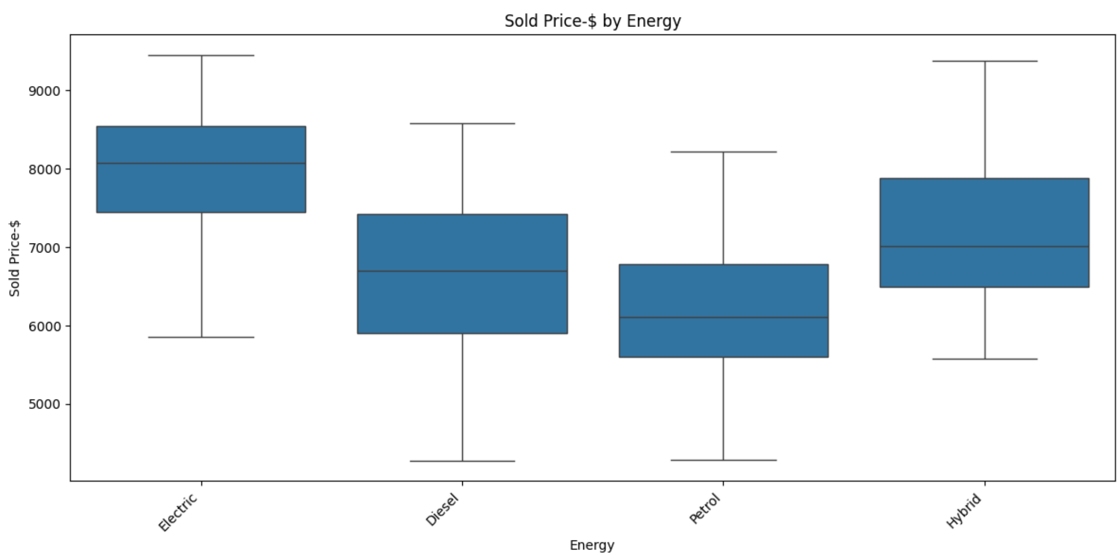
[40]:
```
# Define the target variable
target = 'Sold Price-$'
columns = truck_categorical_data.columns
skipped_columns = ['Sold Price-$','Sold Date','Purchased Date','Car Type','Year␣
↪Sold']
# Loop through each categorical variable and plot boxplot
for col in columns:
    if col not in skipped_columns:
        plt.figure(figsize=(12, 6))
        sns.boxplot(x=truck_categorical_data[col],␣
↪y=truck_categorical_data[target])
        plt.title(f'{target} by {col}')
        plt.xticks(rotation=45, ha='right')
        plt.tight_layout()
        plt.show()
```


Sold Price-$ by Location
Sold Price-$
9000
8000
7000
6000
5000
Columbus
San Jose
Texas
Tucson
Florida
California
Chicago
North Carolina
Oklahoma
Denver
Portland
Tennessee
New York
Madison
Utah
Detroit
Philadelphia
Location

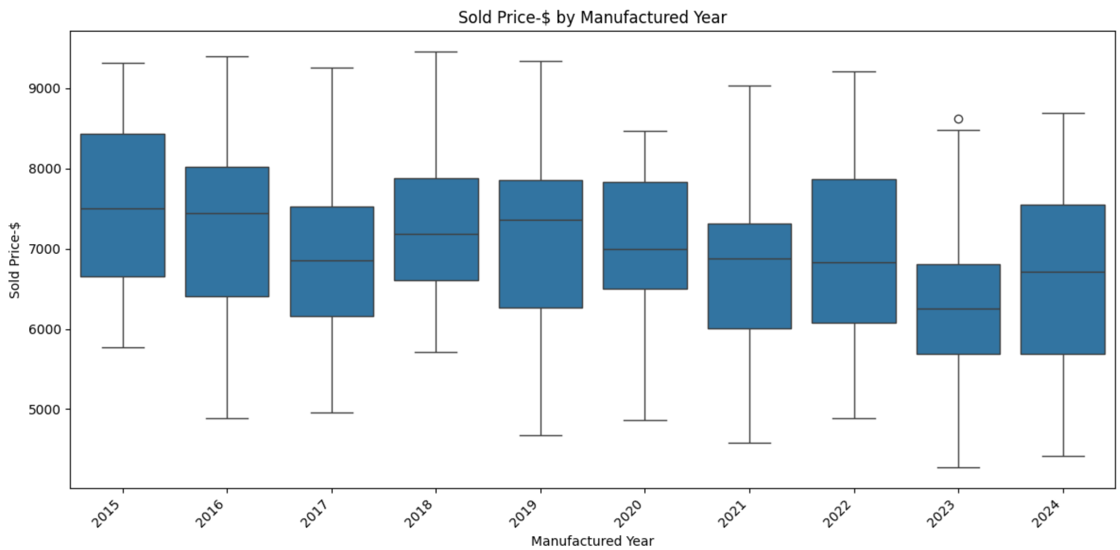

Sold Price-$ by Color
Sold Price-$
9000
8000
7000
6000
5000
Red
Blue
Gray
Black
White
Color

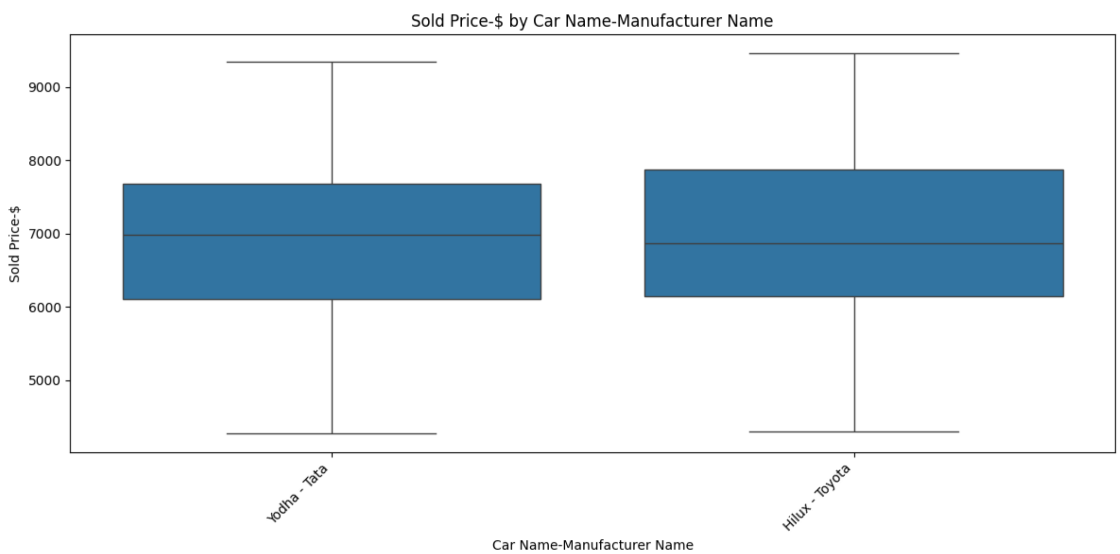


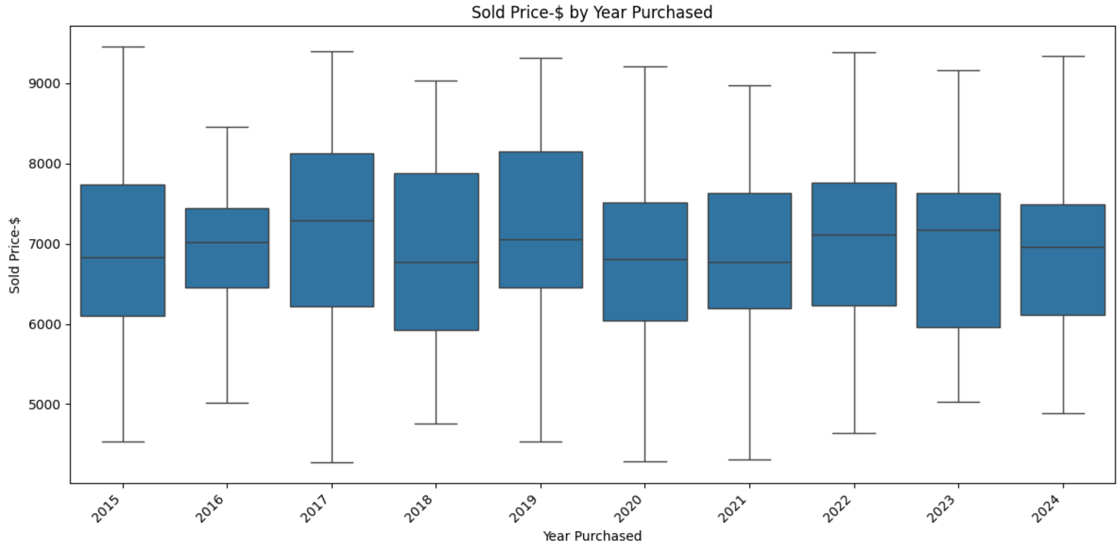
Sold Price-$ by Gearbox
Sold Price-$
9000
8000
7000
6000
5000
Automatic
Manual
Gearbox



Sold Price-$ by Energy
Sold Price-$
9000
8000
7000
6000
5000
Electric
Diesel
Petrol
Hybrid
Energy

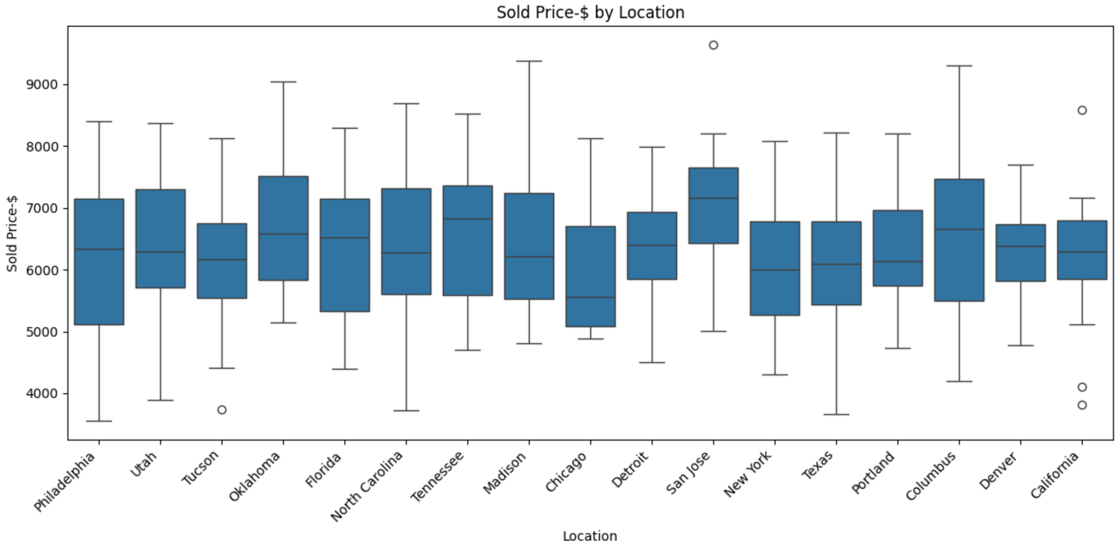

Sold Price-$ by Manufactured Year
Sold Price-$
9000
8000
7000
6000
5000
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
Manufactured Year

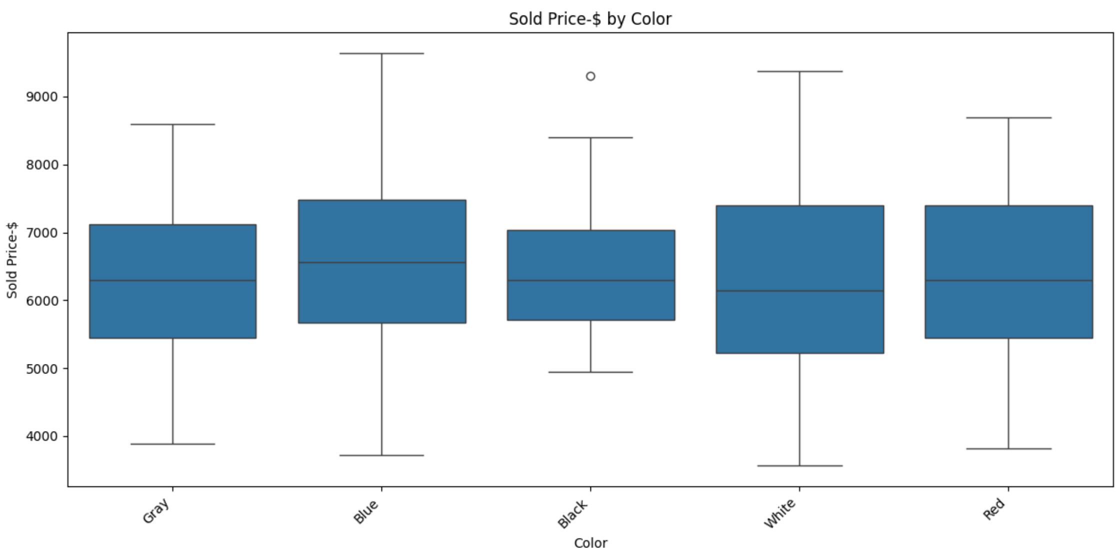

Sold Price-$ by Car Name-Manufacturer Name
Sold Price-$
9000
8000
7000
6000
5000
Yodha - Tata
Hilux - Toyota
Car Name-Manufacturer Name

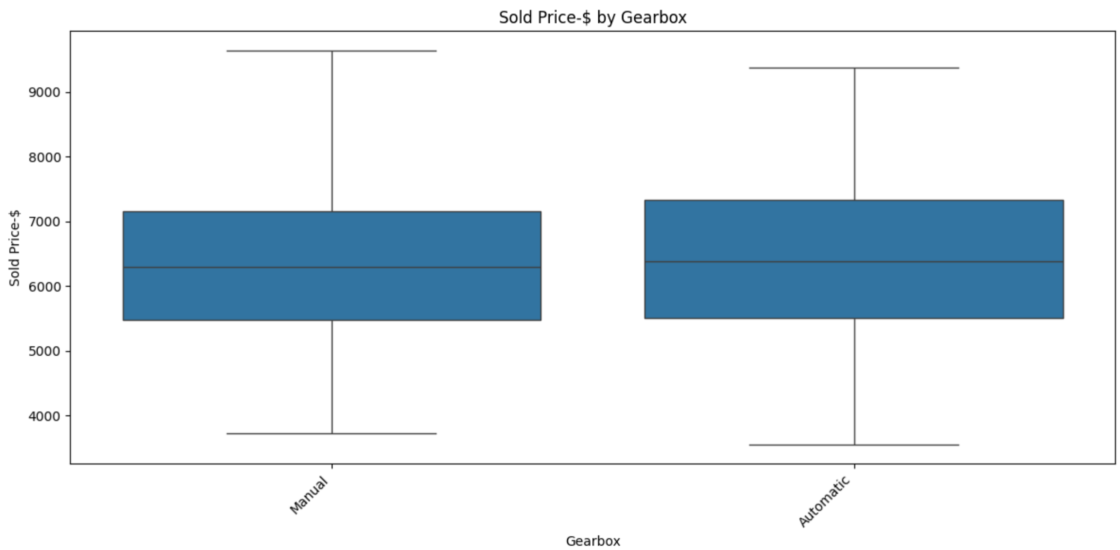


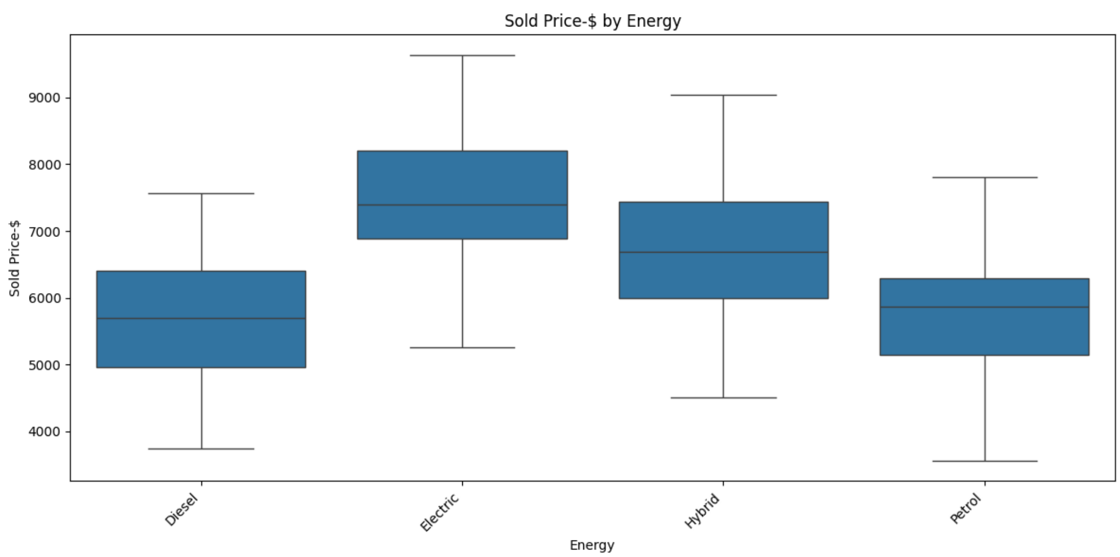
14 Sedan Analysis

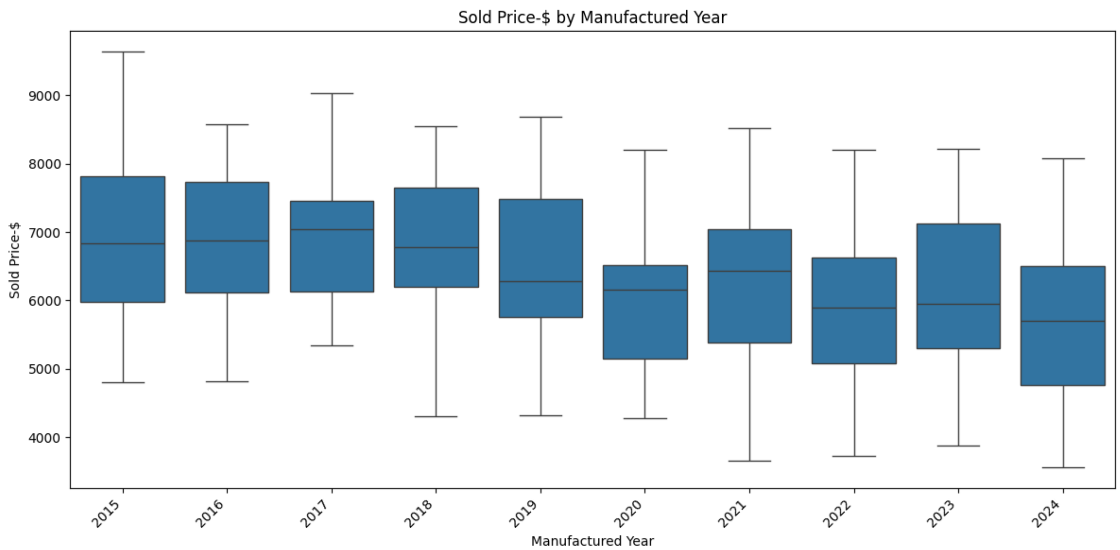
[41]:
```
# Define the target variable
target = 'Sold Price-$'
columns = sedan_categorical_data.columns
skipped_columns = ['Sold Price-$','Sold Date','Purchased Date','Car Type','Year␣
↪Sold']
# Loop through each categorical variable and plot boxplot
for col in columns:
    if col not in skipped_columns:
        plt.figure(figsize=(12, 6))
        sns.boxplot(x=sedan_categorical_data[col],␣
↪y=sedan_categorical_data[target])
        plt.title(f'{target} by {col}')
        plt.xticks(rotation=45, ha='right')
        plt.tight_layout()
        plt.show()
```

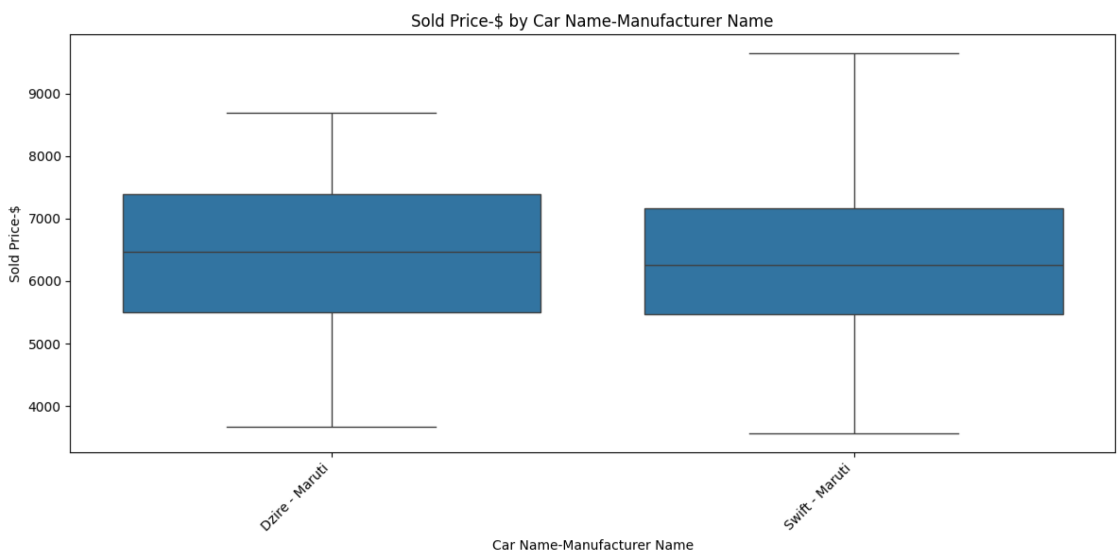

Sold Price-$ by Location
Sold Price-$
9000
8000
7000
6000
5000
4000
Philadelphia
Utah
Tucson
Oklahoma
Florida
North Carolina
Tennessee
Madison
Chicago
Detroit
San Jose
New York
Texas
Portland
Columbus
Denver
California
Location

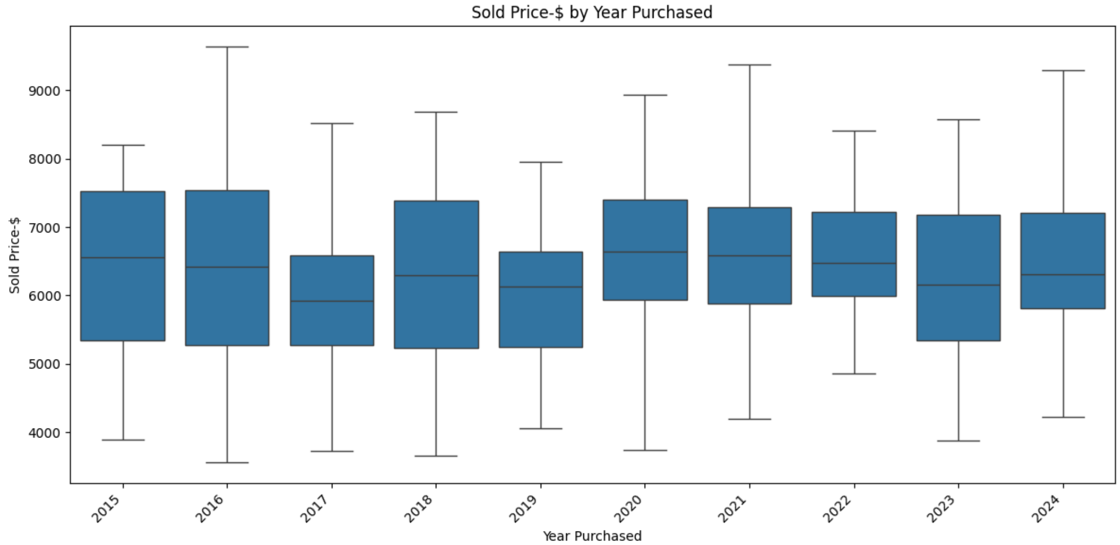

Sold Price-$ by Color
Sold Price-$
9000
8000
7000
6000
5000
4000
Gray
Blue
Black
White
Red
Color


Sold Price-$ by Gearbox
9000
8000
7000
6000
5000
4000
Sold Price-$
Manual
Automatic
Gearbox


Sold Price-$ by Energy
9000
8000
7000
6000
5000
4000
Sold Price-$
Diesel
Electric
Hybrid
Petrol
Energy


Sold Price-$ by Manufactured Year
Sold Price-$
9000
8000
7000
6000
5000
4000
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
Manufactured Year


Sold Price-$ by Car Name-Manufacturer Name
Sold Price-$
9000
8000
7000
6000
5000
4000
Dzire - Maruti
Swift - Maruti
Car Name-Manufacturer Name



15 (4.1) Identify which values related with Sold Price (multiple box plots of diff type of data in a certain category)

Categorical Data Analysis Summary

Through examining the distribution of Sold Price for each variable within each Car Type specifically, it can be observed the primary categorical variable that affects the Sold Price is **Energy** (the type of fuel the Car Runs on).

For other categorical variables, the box-and-whisker plots for each type of the categorical variable largely overlap. Hence, they would not be ideal variables for a machine learning

16 (4.2) Utilise Random Forest to get a model which can predict an ideal Sold Price based on the most relevant categorical variables

Machine Learning Technique: Random Forest

Utilising a Random Forest Model, we are able to train it on the Car Type and Energy of the car in order to predict the Sold Price of the car. This would give a seller better insight on a reasonable price to sell the car at.

The scikitlearn library, which we opted to use, comes with a built-in OneHotEncoder to encode the Categorical Variables for processing.

[42]:
```
X = sold_cars_categorical_data[['Car Type', 'Energy']]
y = sold_cars_categorical_data['Sold Price-$']

categorical_features = ['Car Type', 'Energy']
```

```python
categorical_transformer = OneHotEncoder(handle_unknown='ignore')

preprocessor = ColumnTransformer(
    transformers=[('cat', categorical_transformer, categorical_features)]
)

model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor(n_estimators=100,␣
 ↪random_state=42,max_depth=8))
])

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

rmse = math.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"Root Mean Squared Error: {rmse:.2f}")
print(f"R² Score: {r2:.2f}")
```

```
Root Mean Squared Error: 924.01
R² Score: 0.54
```

[43]:
```python
new_car = pd.DataFrame([{
    'Car Type': 'SUV',
    'Energy': 'Electric'
}])

predicted_price = model.predict(new_car)
print(f"Predicted sale price: ${predicted_price[0]:,.2f}")
```

```
Predicted sale price: $8,349.31
```

[44]:
```python
new_car = pd.DataFrame([{
    'Car Type': 'Hatchback',
    'Energy': 'Petrol'
}])

predicted_price = model.predict(new_car)
print(f"Predicted sale price: ${predicted_price[0]:,.2f}")
```

```
Predicted sale price: $5,399.71
```

17 (5) Beyond the Content of the Course

Two Layer Perceptron

A two-layer perceptron is a type of artificial neural network that consists of two layers of neurons: Input layer: This layer takes in the input data. Hidden layer: A layer of neurons that applies weights, biases, and an activation function to transform the input. Each neuron in the hidden layer processes the weighted sum of the inputs, adds a bias, and passes the result through an activation function (like ReLU or Sigmoid) to introduce non-linearity.

If we're considering a two-layer perceptron for just one variable determining another variable: The input layer has a single neuron (representing one variable). The hidden layer can still consist of one or more neurons (depending on the complexity). The output layer will have one neuron to predict the dependent variable.

[45]:
```
engine_power = sold_cars_numerical_data['Engine Power-HP'].values.reshape(-1, 1)
sold_price = sold_cars_numerical_data['Sold Price-$'].values.reshape(-1, 1)

engine_scaler = StandardScaler()
price_scaler = StandardScaler()

engine_power_scaled = engine_scaler.fit_transform(engine_power)
sold_price_scaled = price_scaler.fit_transform(sold_price)

X_train, X_test, y_train, y_test = train_test_split(engine_power_scaled,␣
 ↪sold_price_scaled, test_size=0.2, random_state=42)

model = Sequential()

model.add(Input(shape=(1,)))  # Use Input layer
model.add(Dense(64, activation='relu'))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=1000, verbose=0)

loss = model.evaluate(X_test, y_test)
```

```
14/14            0s 3ms/step - loss:
0.6921
```

[46]:
```
sample_hp = np.array([[150]])
sample_hp_scaled = engine_scaler.transform(sample_hp)

predicted_price_scaled = model.predict(sample_hp_scaled)

predicted_price = price_scaler.inverse_transform(predicted_price_scaled)

print(f"Predicted price for 150 HP: ${predicted_price[0][0]:,.2f}")
```

```
1/1              0s 54ms/step
Predicted price for 150 HP: $6,099.84
```

[47]:

```
# Retrieve HP Bands
df = sold_cars_numerical_data.copy()
bins = np.arange(int(df['Engine Power-HP'].min()) - 10,
                 int(df['Engine Power-HP'].max()) + 10, 10)
df['HP Band'] = pd.cut(df['Engine Power-HP'], bins)

# Aggregate by HP Band
band_stats = df.groupby('HP Band', observed=True).agg({
    'Sold Price-$': 'sum',
    'Engine Power-HP': ['sum', 'count', 'mean']
})

band_stats.columns = ['_'.join(col).strip() for col in band_stats.columns.
 ↪values]

# Calculate average price per HP and per car
band_stats['Avg Price per HP'] = band_stats['Sold Price-$_sum'] /␣
 ↪band_stats['Engine Power-HP_sum']
band_stats['Avg Price per Car'] = band_stats['Sold Price-$_sum'] /␣
 ↪band_stats['Engine Power-HP_count']

# Display
display_stats = band_stats[['Avg Price per HP', 'Avg Price per Car', 'Engine␣
 ↪Power-HP_count']]
display_stats = display_stats.sort_index()
print(display_stats)
```

```
            Avg Price per HP  Avg Price per Car  Engine Power-HP_count
HP Band
(90, 100]          67.034856        6703.485632                    348
(100, 110]         58.521017        6027.664706                    170
(110, 120]         54.936289        6419.734158                    647
(120, 130]         56.817487        7386.273292                    161
(140, 150]         40.165430        6024.814570                    151
(170, 180]         42.529818        7442.718232                    181
(190, 200]         36.406850        7281.369942                    346
(240, 250]         34.524149        8631.037267                    161
```