carPredictionNotebook

July 29, 2025

1 Car Price Prediction Project

[29]:
```
# Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report
```

[30]:
```
# Define data
df = pd.read_csv(r"Assets\car_data_v2.csv")
print(df.head())
```

```
         car_name    brand    model  vehicle_age  km_driven  mileage  \
0     Maruti Alto   Maruti     Alto            9     120000    19.70
1     Maruti Alto   Maruti     Alto            9      37000    20.92
2  Maruti Wagon R   Maruti  Wagon R            8      35000    18.90
3  Maruti Wagon R   Maruti  Wagon R            3      17512    20.51
4   Hyundai Venue  Hyundai    Venue            2      20000    18.15

   max_power  seats  selling_price
0      46.30      5         120000
1      67.10      5         226000
2      67.10      5         350000
3      67.04      5         410000
4     118.35      5        1050000
```

[31]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2119 entries, 0 to 2118
Data columns (total 9 columns):
```

```
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   car_name       2119 non-null   object
 1   brand          2119 non-null   object
 2   model          2119 non-null   object
 3   vehicle_age    2119 non-null   int64
 4   km_driven      2119 non-null   int64
 5   mileage        2119 non-null   float64
 6   max_power      2119 non-null   float64
 7   seats          2119 non-null   int64
 8   selling_price  2119 non-null   int64
dtypes: float64(2), int64(4), object(3)
memory usage: 149.1+ KB
None
```

[32]:
```python
print(df.describe())
```

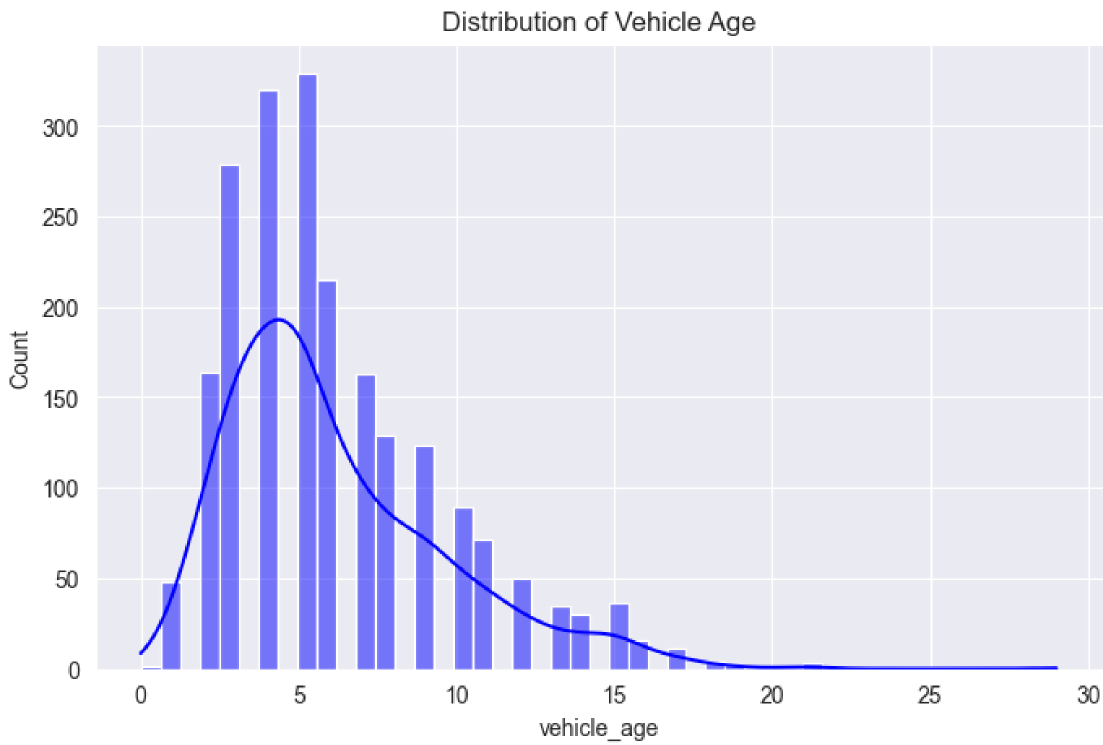
```
       vehicle_age      km_driven      mileage    max_power        seats  \
count  2119.000000    2119.000000  2119.000000  2119.000000  2119.000000
mean      6.153374   42207.621992    22.574856    61.802931     5.002832
std       3.524845   27950.561196     3.008683    13.112960     0.176503
min       0.000000     581.000000    14.400000    38.400000     4.000000
25%       4.000000   21000.000000    20.510000    53.260000     5.000000
50%       5.000000   38000.000000    22.740000    67.000000     5.000000
75%       8.000000   58494.000000    23.950000    67.050000     5.000000
max      29.000000  425785.000000    33.540000   123.370000     7.000000

       selling_price
count   2.119000e+03
mean    3.287744e+05
std     1.496699e+05
min     4.000000e+04
25%     2.490000e+05
50%     3.150000e+05
75%     3.900000e+05
max     1.240000e+06
```

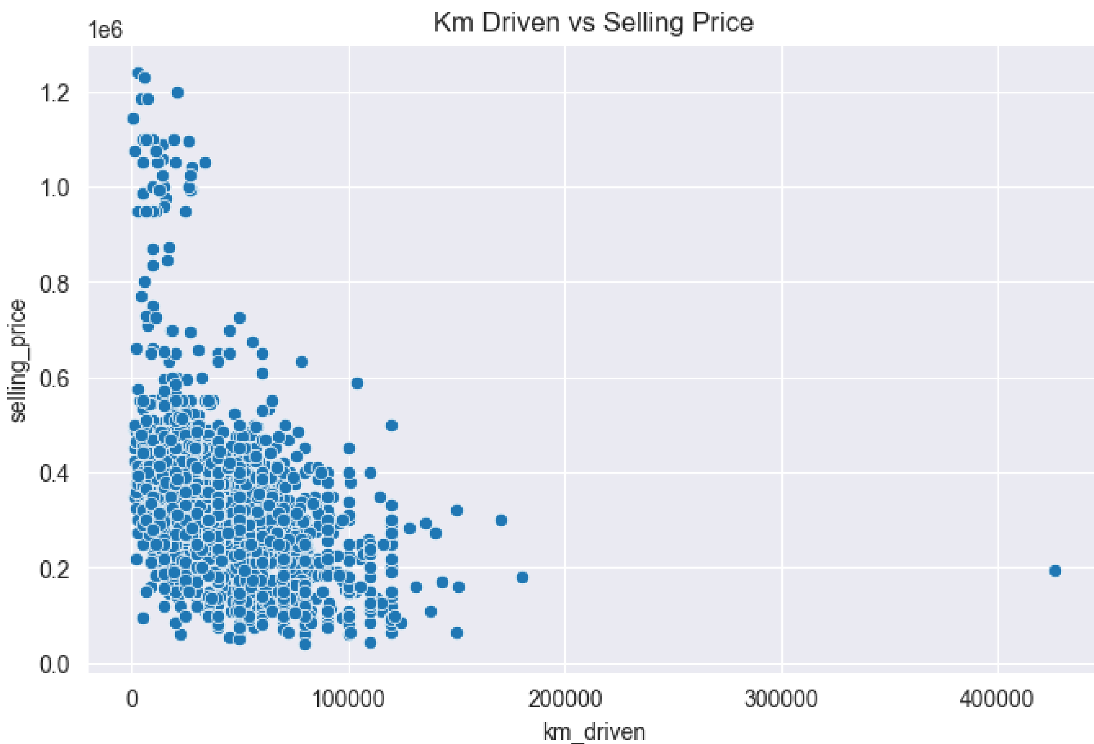
[33]:
```python
print(df.isnull().sum())
```

```
car_name         0
brand            0
model            0
vehicle_age      0
km_driven        0
mileage          0
max_power        0
seats            0
selling_price    0
dtype: int64
```

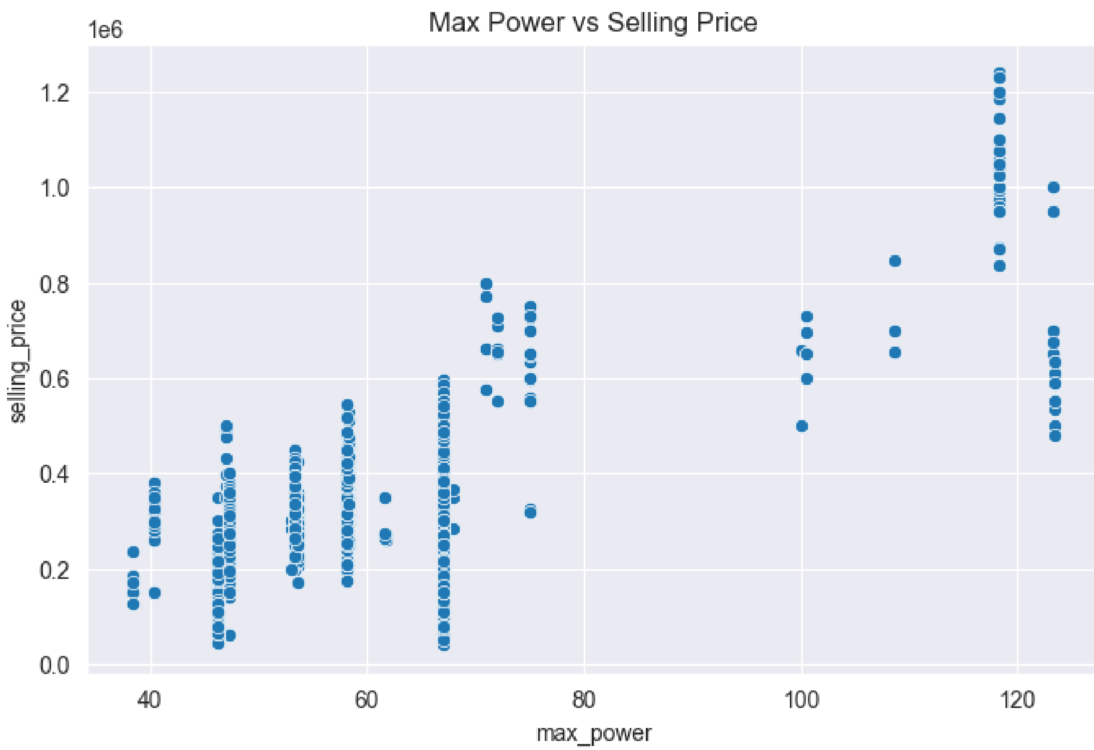
[34]:
```
# Check the distribution of vehicle ages in order to identify trends related to
  ↪vehicle age and pricing
plt.figure(figsize=(8,5))
sns.histplot(df['vehicle_age'], kde=True, color='blue')
plt.title('Distribution of Vehicle Age')
plt.show()
```



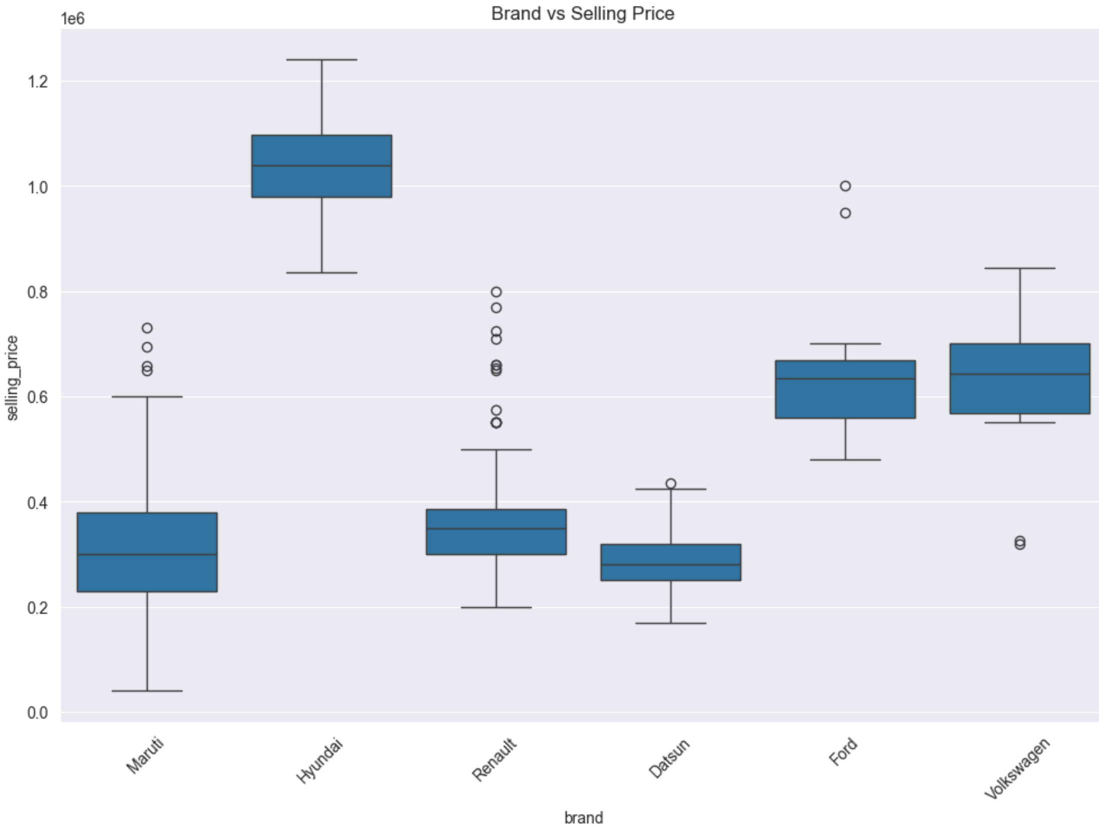
[35]:
```
 #Relationship between km_driven and selling_price
plt.figure(figsize=(8,5))
sns.scatterplot(x='km_driven', y='selling_price', data=df)
plt.title('Km Driven vs Selling Price')
plt.show()
```



[36]:
```
# Relationship between power vs Selling Price
plt.figure(figsize=(8,5))
sns.scatterplot(x='max_power', y='selling_price', data=df)
plt.title('Max Power vs Selling Price')
plt.show()
```

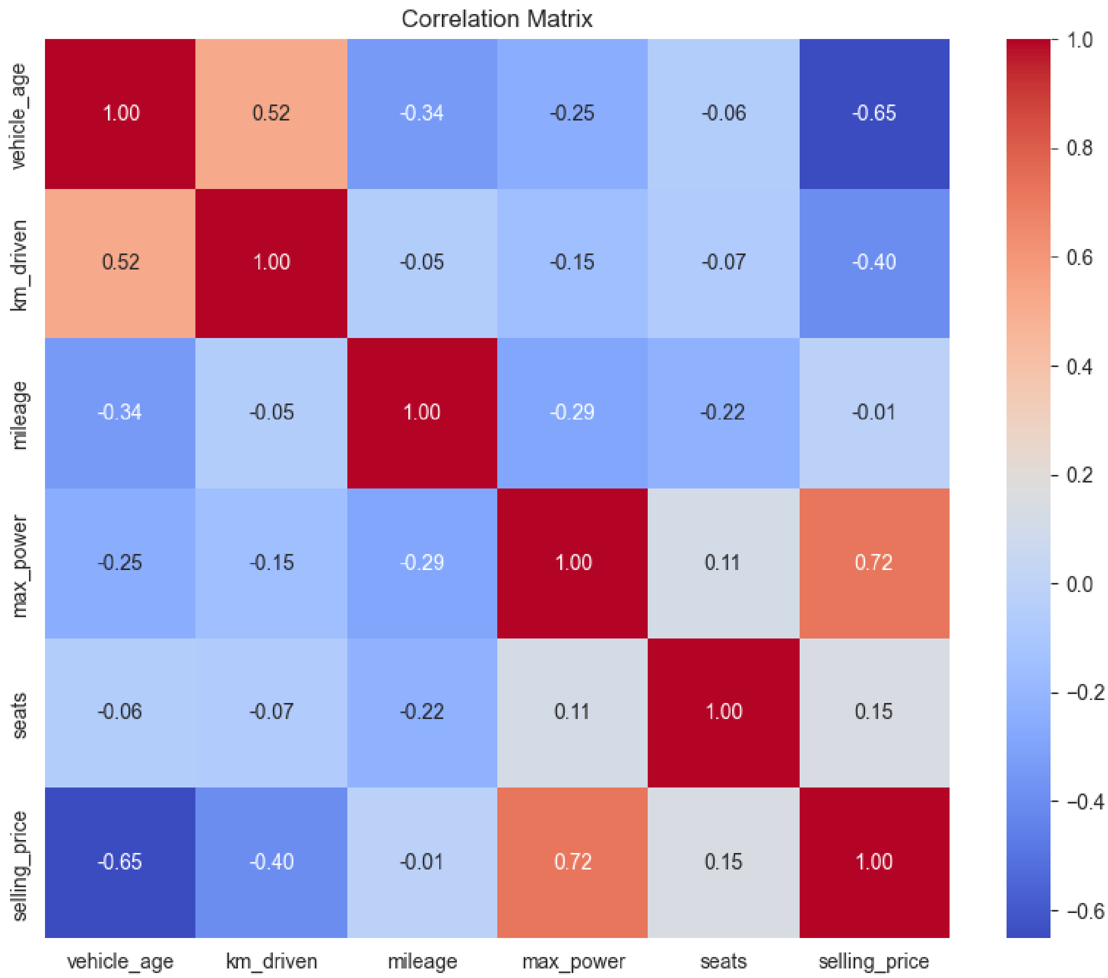


[37]:
```python
# Brand  Distribution of Selling Price
plt.figure(figsize=(12,8))
sns.boxplot(x='brand', y='selling_price', data=df)
plt.title('Brand vs Selling Price')
plt.xticks(rotation=45)
plt.show()
```

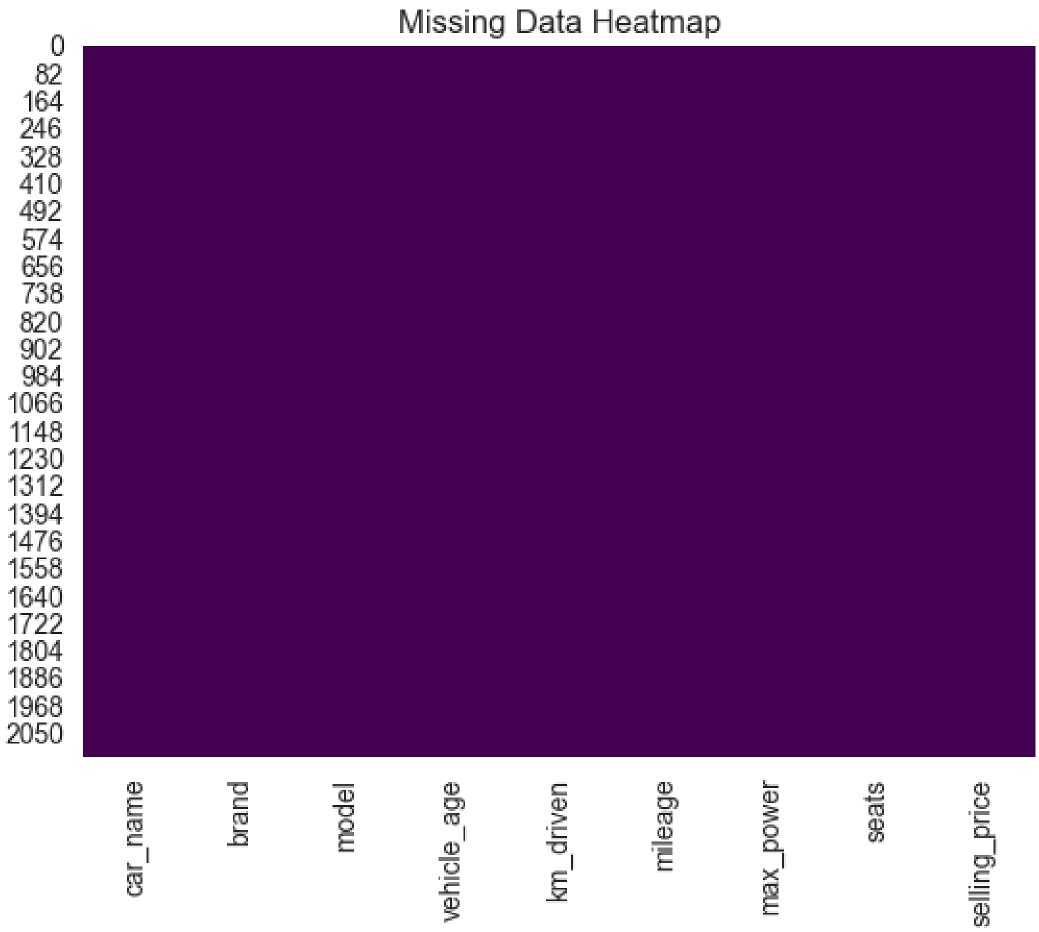


[38]:
```python
# Selection of the numeric columns for correlation analysis
numeric_columns = df.select_dtypes(include=['float64', 'int64'])

#Correlation matrix
plt.figure(figsize=(10,8))
sns.heatmap(numeric_columns.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```



[39]:
```
# Missing Value Check
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title('Missing Data Heatmap')
plt.show()
```



[40]:
```
# Check  for duplicate rows
df.duplicated().sum()

# # Fixing duplicate rows:
# df.drop_duplicates(inplace=True)
```

[40]:
```
np.int64(34)
```

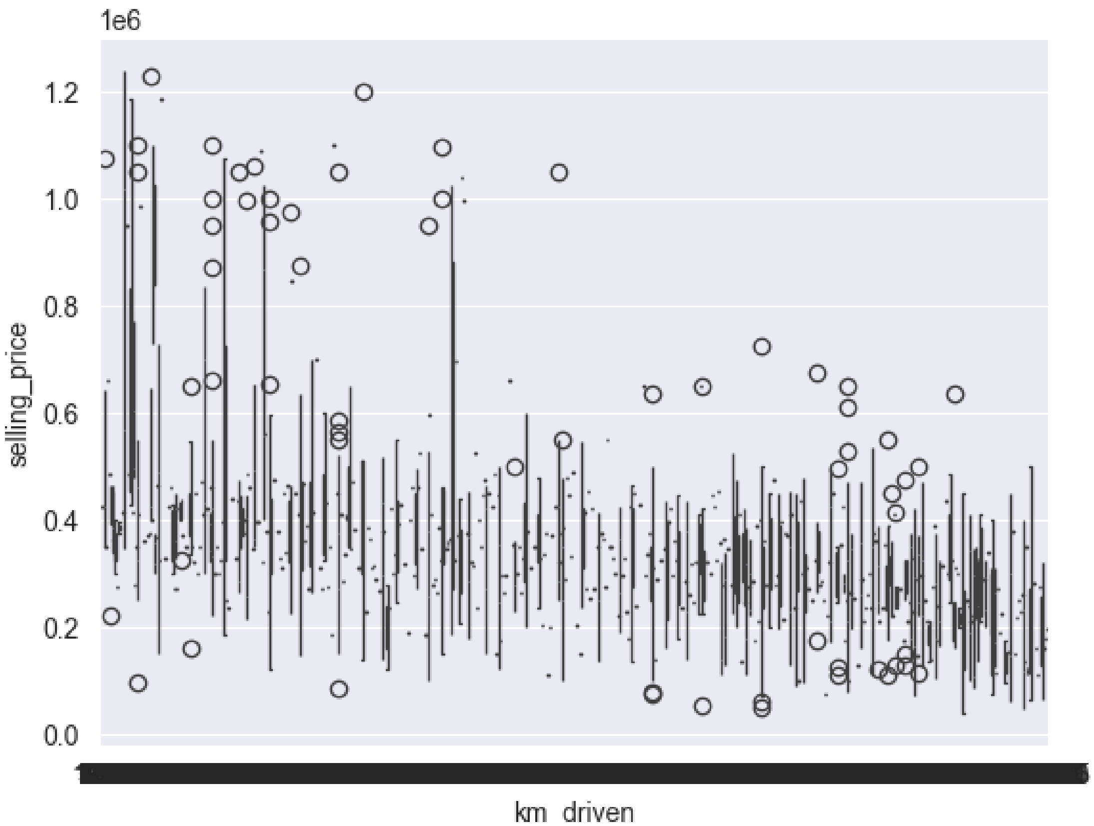
[41]:
```
#data types
df.dtypes
```

[41]:
```
car_name          object
brand             object
model             object
vehicle_age        int64
km_driven          int64
mileage          float64
max_power        float64
```

```
seats                int64
selling_price        int64
dtype: object
```

[42]:
```python
 #Outliers
sns.boxplot(x=df['km_driven'], y=df['selling_price'], data=df)
```

[42]:
```
<Axes: xlabel='km_driven', ylabel='selling_price'>
```



2 Modelling

LINEAR REGRESSION

[43]:
```python
# Feature selection
X = df.drop(['selling_price', 'car_name', 'model'], axis=1)
X = pd.get_dummies(X, drop_first=True)

y = df['selling_price']

# Train-test split
```

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

[44]:
```
# Modeling the training data
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
```

[44]: LinearRegression()

[45]:
```
# Evaluation
y_pred = lr_model.predict(X_test)
print("R² Score:", r2_score(y_test, y_pred))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
R² Score: 0.8129121015140068
RMSE: 58325.83941323817
```

[46]:
```
# Prediction of a new car's price
carInput = X_test.iloc[0:1]
predictedPrice = lr_model.predict(carInput)
print("Predicted Selling Price:", predictedPrice[0])
```

```
Predicted Selling Price: 399165.51263545104
```

LOGISTIC REGRESSION

[47]:
```
medianPrice = df['selling_price'].median()
df['price_category'] = np.where(df['selling_price'] > medianPrice, 1, 0)
```

[48]:
```
X = df.drop(['selling_price', 'price_category', 'car_name', 'model'], axis=1)
X = pd.get_dummies(X, drop_first=True)

y = df['price_category']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[49]:
```
# Modeling of the training data
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_trainScaled = scaler.fit_transform(X_train)
X_testScaled = scaler.transform(X_test)

log_model = LogisticRegression(class_weight='balanced', max_iter=2000)
log_model.fit(X_trainScaled, y_train)
y_pred = log_model.predict(X_testScaled)
```

[50]:
```
# Evaluation
y_pred = log_model.predict(X_test_scaled)

print("This is the Accuracy:", accuracy_score(y_test, y_pred))
print("This is the Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("The Classification Report:\n", classification_report(y_test, y_pred,␣
↪zero_division=0))
```

```
This is the Accuracy: 0.8608490566037735
This is the Confusion Matrix:
 [[164  34]
 [ 25 201]]
The Classification Report:
               precision    recall  f1-score   support

           0       0.87      0.83      0.85       198
           1       0.86      0.89      0.87       226

    accuracy                           0.86       424
   macro avg       0.86      0.86      0.86       424
weighted avg       0.86      0.86      0.86       424
```

[51]:
```
# Scale the sample input (use the same scaler used for training)
carInput = X_test.iloc[0:1]
carInputScaled = scaler.transform(carInput)

# Predict if it will be expensive/affordable
predictedCategory = log_model.predict(carInputScaled)
print("Predicted Category (0=Affordable, 1=Expensive):", predictedCategory[0])
```

```
Predicted Category (0=Affordable, 1=Expensive): 1
```

[ ]:
```

```