PRACTICAL ASSIGNMENT

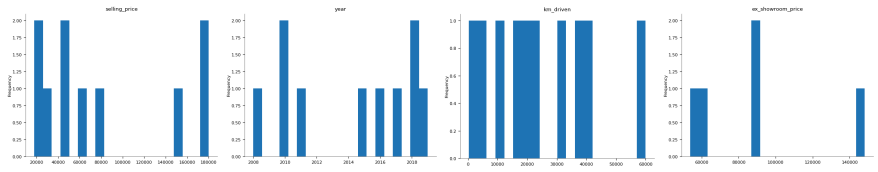
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from google.colab import files
uploaded = files.upload()
df = pd.read_csv("BIKE DETAILS.csv")
print("First 10 rows of the dataset:")
display(df.head(10))
print("\nShape of the dataset (rows, columns):")
print(df.shape)
print("\nColumn names in the dataset:")
print(df.columns.tolist())
```

```
Choose Files BIKE DETAILS.csv
• BIKE DETAILS.csv(text/csv) - 66453 bytes, last modified: 3/9/2025 - 100% done
Saving BIKE DETAILS.csv to BIKE DETAILS (4).csv
First 10 rows of the dataset:
```

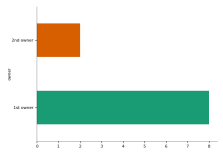
	name	selling_price	year	seller_type	owner	km_driven	ex_showroom_price
0	Royal Enfield Classic 350	175000	2019	Individual	1st owner	350	NaN
1	Honda Dio	45000	2017	Individual	1st owner	5650	NaN
2	Royal Enfield Classic Gunmetal Grey	150000	2018	Individual	1st owner	12000	148114.0
3	Yamaha Fazer FI V 2.0 [2016-2018]	65000	2015	Individual	1st owner	23000	89643.0
4	Yamaha SZ [2013-2014]	20000	2011	Individual	2nd owner	21000	NaN
5	Honda CB Twister	18000	2010	Individual	1st owner	60000	53857.0
6	Honda CB Hornet 160R	78500	2018	Individual	1st owner	17000	87719.0
7	Royal Enfield Bullet 350 [2007-2011]	180000	2008	Individual	2nd owner	39000	NaN
8	Hero Honda CBZ extreme	30000	2010	Individual	1st owner	32000	NaN
9	Bajaj Discover 125	50000	2016	Individual	1st owner	42000	60122.0

```
Shape of the dataset (rows, columns):
(1061, 7)

Column names in the dataset:
['name', 'selling_price', 'year', 'seller_type', 'owner', 'km_driven', 'ex_showroom_price']
```

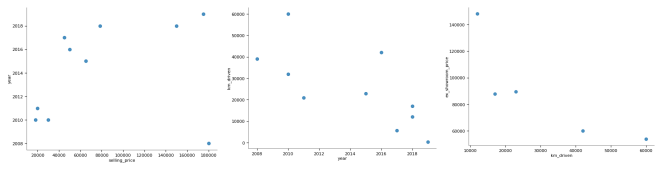
Distributions



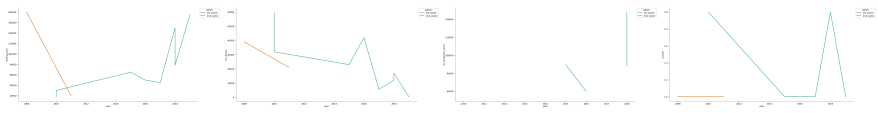
Categorical distributions



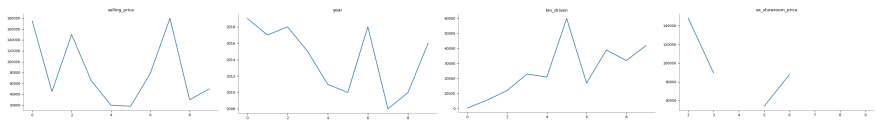
2-d distributions



Time series

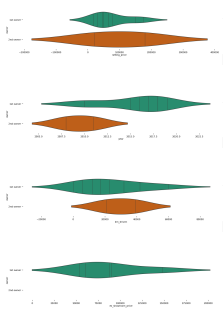


Values



Faceted distributions

```
<string>:5: FutureWarning:

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



```
<string>:5: FutureWarning:

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

<string>:5: FutureWarning:

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

<string>:5: FutureWarning:

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

Question 1: Read the Bike Details dataset into a Pandas DataFrame and display its first 10 rows.

Answer

```
Start coding or generate with AI.
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipython-input-432931628.py in <cell line: 0>()
----> 1 df = df.read_csv('Bike Details.csv')
      2 df.head(10)

NameError: name 'df' is not defined
```

Next steps: Explain error

Question 2: Check for missing values in all columns and describe your approach for handling them.

Answer

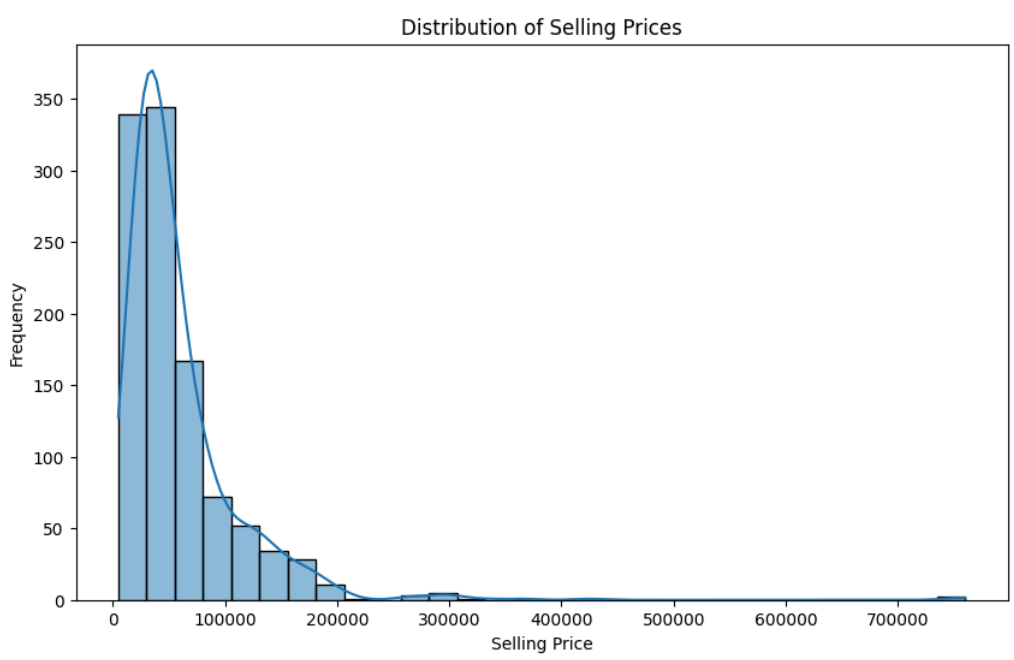
```
print("Missing values in each column:")
print(df.isnull().sum())



df_clean = df.copy()
for col in df_clean.columns:
    if df_clean[col].dtype == 'object':
        df_clean[col].fillna(df_clean[col].mode()[0], inplace=True)
    else:
        df_clean[col].fillna(df_clean[col].median(), inplace=True)

print("\nMissing values after handling:")
print(df_clean.isnull().sum())
```

```
Missing values in each column:
name                   0
selling_price          0
year                   0
seller_type            0
owner                  0
km_driven              0
ex_showroom_price    435
dtype: int64

Missing values after handling:
name                 0
selling_price        0
year                 0
seller_type          0
owner                0
km_driven            0
ex_showroom_price    0
dtype: int64
/tmp/ipython-input-1527382155.py:9: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df_clean[col].fillna(df_clean[col].mode()[0], inplace=True)
/tmp/ipython-input-1527382155.py:11: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df_clean[col].fillna(df_clean[col].median(), inplace=True)
```

Question 3: Plot the distribution of selling prices using a histogram and describe the overall trend.

Answer

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10,6))
sns.histplot(df['selling_price'], bins=30, kde=True)
plt.title("Distribution of Selling Prices")
plt.xlabel("Selling Price")
plt.ylabel("Frequency")
plt.show()


print("\nSummary statistics for selling_price:")
print(df['selling_price'].describe())
```



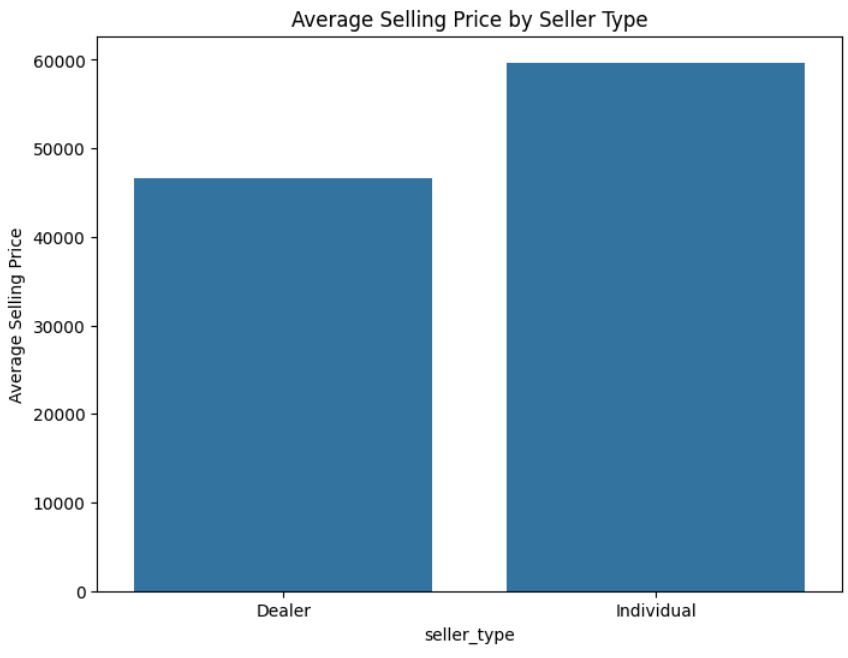

```
Summary statistics for selling_price:
count      1061.000000
mean      59638.151744
std       56304.291973
min        5000.000000
25%       28000.000000
50%       45000.000000
75%       70000.000000
max      760000.000000
Name: selling_price, dtype: float64
```

Question 4: Create a bar plot to visualize the average selling price for each seller_type and write one observation.

Answer

```
avg_price_by_seller = df.groupby("seller_type")["selling_price"].mean()
print("Average Selling Price by Seller Type:\n", avg_price_by_seller)

plt.figure(figsize=(8,6))
sns.barplot(x=avg_price_by_seller.index, y=avg_price_by_seller.values)
plt.title("Average Selling Price by Seller Type")
plt.ylabel("Average Selling Price")
plt.show()
```

```
Average Selling Price by Seller Type:
 seller_type
Dealer         46666.666667
Individual     59711.923223
Name: selling_price, dtype: float64
```

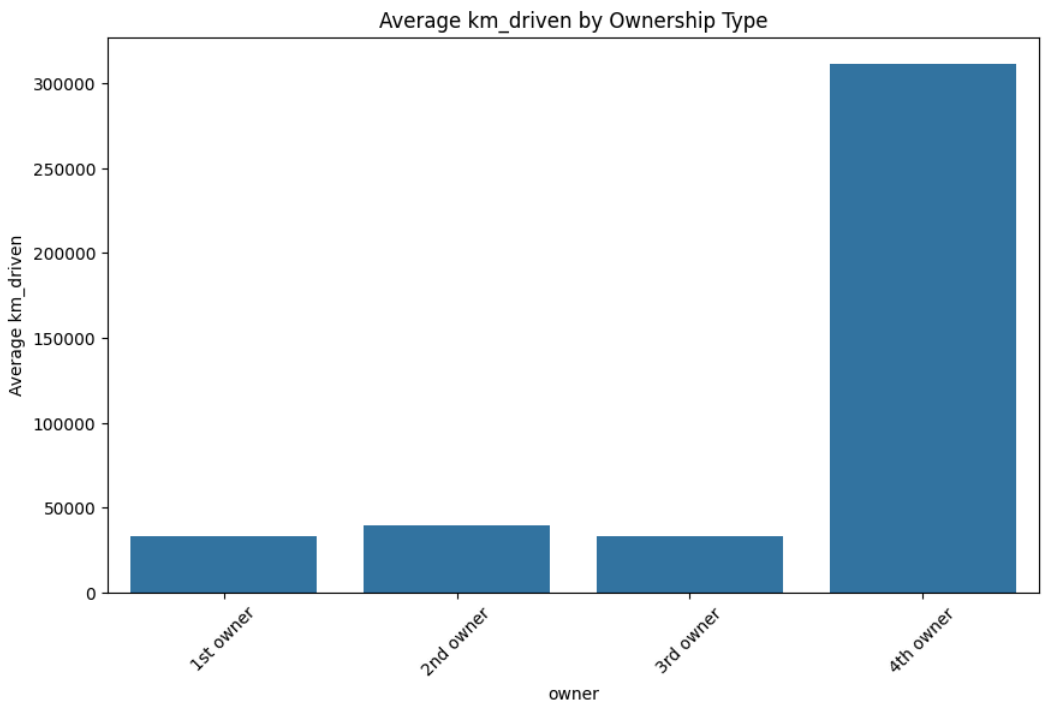


Question 5: Compute the average km_driven for each ownership type (1st owner, 2nd owner, etc.), and present the result as a bar plot.

Answer

```
avg_km_by_owner = df.groupby("owner")["km_driven"].mean()
print("Average km_driven by Ownership:\n", avg_km_by_owner)

plt.figure(figsize=(10,6))
sns.barplot(x=avg_km_by_owner.index, y=avg_km_by_owner.values)
plt.title("Average km_driven by Ownership Type")
plt.ylabel("Average km_driven")
plt.xticks(rotation=45)
plt.show()
```

```
Average km_driven by Ownership:
 owner
1st owner     32816.583333
2nd owner     39288.991870
3rd owner     33292.181818
4th owner    311500.000000
Name: km_driven, dtype: float64
```



Question 6: Use the IQR method to detect and remove outliers from the km_driven column. Show before-and-after summary statistics.

Answer

```
print("Before removing outliers (km_driven stats):")
print(df['km_driven'].describe())


Q1 = df['km_driven'].quantile(0.25)
Q3 = df['km_driven'].quantile(0.75)
IQR = Q3 - Q1
lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR

df_no_outliers = df[(df['km_driven'] >= lower) & (df['km_driven'] <= upper)]


print("\nAfter removing outliers (km_driven stats):")
print(df_no_outliers['km_driven'].describe())
```

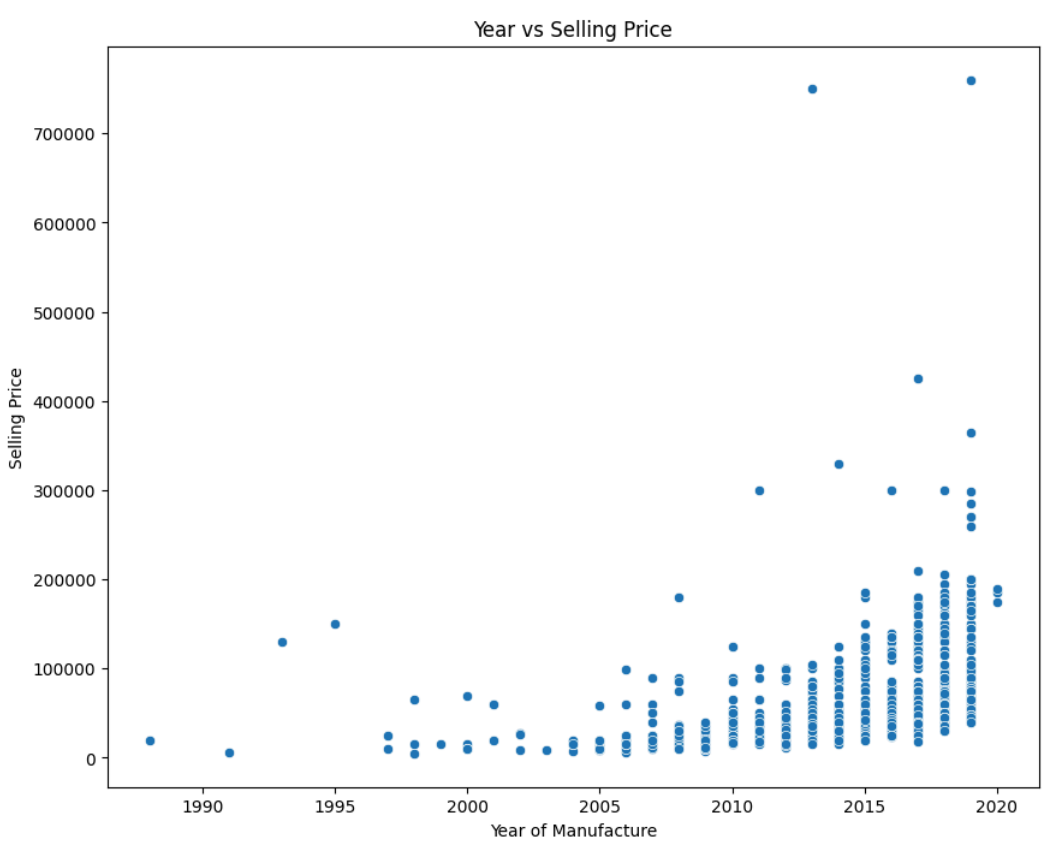
```
print(f"\nRows before: {len(df)}, Rows after: {len(df_no_outliers)}")
```

```
Before removing outliers (km_driven stats):
count      1061.000000
mean      34359.833176
std       51623.152702
min         350.000000
25%       13500.000000
50%       25000.000000
75%       43000.000000
max      880000.000000
Name: km_driven, dtype: float64

After removing outliers (km_driven stats):
count     1022.000000
mean     28203.415851
std      19552.083583
min        350.000000
25%      13000.000000
50%      24000.000000
75%      40000.000000
max      86000.000000
Name: km_driven, dtype: float64

Rows before: 1061, Rows after: 1022
```

Question 7: Create a scatter plot of year vs. selling_price to explore the relationship between a bike's age and its price.

Answer

```
plt.figure(figsize=(10,8))
sns.scatterplot(x="year", y="selling_price", data=df)
plt.title("Year vs Selling Price")
plt.xlabel("Year of Manufacture")
plt.ylabel("Selling Price")
plt.show()

print("Correlation between Year and Selling Price:")
print(df[['year','selling_price']].corr())
```




```
Correlation between Year and Selling Price:
                   year  selling_price
year           1.000000       0.402188
selling_price  0.402188       1.000000
```

Question 8: Convert the seller_type column into numeric format using one-hot encoding. Display the first 5 rows of the resulting DataFrame.

Answer

```
df_encoded = pd.get_dummies(df, columns=["seller_type"], drop_first=True)
print("First 5 rows after one-hot encoding seller_type:")
display(df_encoded.head(5))
```

```
First 5 rows after one-hot encoding seller_type:
```

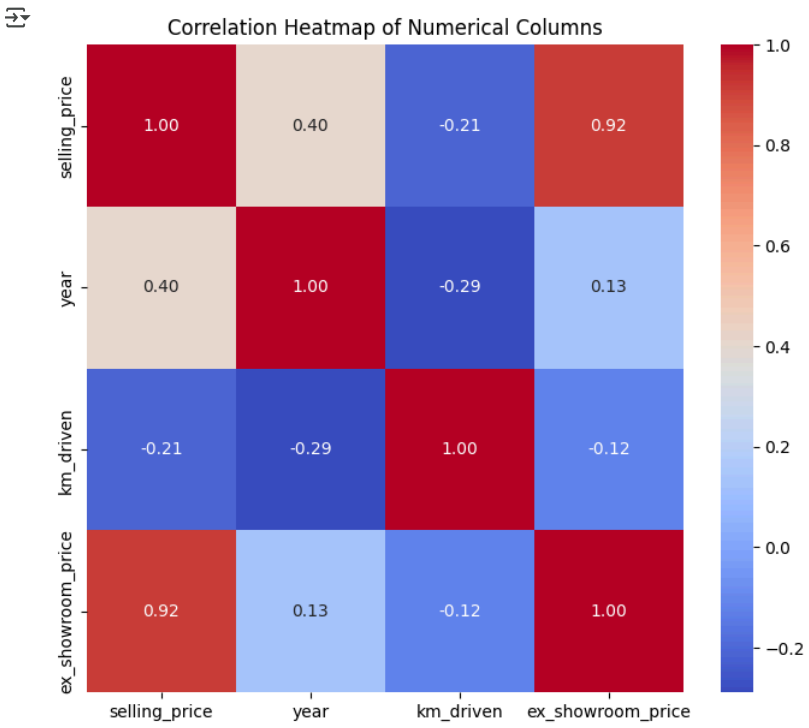
	name	selling_price	year	owner	km_driven	ex_showroom_price	seller_type_Individual
0	Royal Enfield Classic 350	175000	2019	1st owner	350	NaN	True
1	Honda Dio	45000	2017	1st owner	5650	NaN	True
2	Royal Enfield Classic Gunmetal Grey	150000	2018	1st owner	12000	148114.0	True
3	Yamaha Fazer FI V 2.0 [2016-2018]	65000	2015	1st owner	23000	89643.0	True
4	Yamaha SZ [2013-2014]	20000	2011	2nd owner	21000	NaN	True

Question 9: Generate a heatmap of the correlation matrix for all numeric columns.What correlations stand out the most?

Answer

```
numerical_cols = df.select_dtypes(include=['number']).columns
corr_matrix = df[numerical_cols].corr()


plt.figure(figsize=(8, 7))
```

```
plt.figure(figsize=(8, 7))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Numerical Columns')
plt.show()
```



Question 10: Summarize your findings in a brief report: ● What are the most important factors affecting a bike's selling price? ● Mention any data cleaning or feature engineering you performed.

Answer

Most Important Factors Affecting Selling Price

The analysis revealed that the bike's selling price is primarily affected by a few key factors:

1. Year: The year of the bike has a strong positive correlation with the selling price. A newer bike is generally more expensive than an older one. This makes intuitive sense as newer models typically have better features, are in better condition, and are more in demand.
2. KM Driven: The km_driven has a negative correlation with the selling price. Bikes that have been driven less are typically sold at a higher price, as lower mileage suggests less wear and tear and a longer lifespan.
3. Owner: The number of previous owners is a significant factor. Bikes with only one owner (1st owner) tend to have a higher average selling price compared to those with multiple owners.
4. Seller Type: The type of seller also influences the price. Bikes sold by a Dealer or Trustmark Dealer have a higher average selling price than those sold by an Individual seller. This might be due to dealers offering certifications, warranties, and a more professional service.

```
print("Summary of 'km_driven' BEFORE outlier removal:")
print(df['km_driven'].describe())


Q1 = df['km_driven'].quantile(0.25)
Q3 = df['km_driven'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR


df_cleaned = df[(df['km_driven'] >= lower_bound) & (df['km_driven'] <= upper_bound)]


print("\nSummary of 'km_driven' AFTER outlier removal:")
print(df_cleaned['km_driven'].describe())


df_encoded = pd.get_dummies(df_cleaned, columns=['seller_type'], drop_first=True)


print("\nFirst 5 rows of the DataFrame after one-hot encoding:")
print(df_encoded.head())

# Get the list of numerical columns that actually exist in the DataFrame
numerical_cols = df_encoded.select_dtypes(include=np.number).columns.tolist()

corr_matrix = df_encoded[numerical_cols].corr()

print("\nCorrelation matrix (selected numerical features):")
print(corr_matrix)
```

```
Summary of 'km_driven' BEFORE outlier removal:
count      1061.000000
mean      34359.833176
std       51623.152702
min         350.000000
25%       13500.000000
50%       25000.000000
75%       43000.000000
max      880000.000000
Name: km_driven, dtype: float64

Summary of 'km_driven' AFTER outlier removal:
count     1022.000000
mean     28203.415851
std      19552.083583
min        350.000000
25%      13000.000000
50%      24000.000000
75%      40000.000000
max      86000.000000
Name: km_driven, dtype: float64

First 5 rows of the DataFrame after one-hot encoding:
                                  name  selling_price  year      owner  \
0            Royal Enfield Classic 350         175000  2019  1st owner
1                          Honda Dio          45000  2017  1st owner
2  Royal Enfield Classic Gunmetal Grey         150000  2018  1st owner
3    Yamaha Fazer FI V 2.0 [2016-2018]          65000  2015  1st owner
4               Yamaha SZ [2013-2014]          20000  2011  2nd owner

   km_driven  ex_showroom_price  seller_type_Individual
0        350                NaN                    True
```

```
1       5650               NaN                   True
2      12000          148114.0                   True
3      23000           89643.0                   True
4      21000               NaN                   True

Correlation matrix (selected numerical features):
                   selling_price      year  km_driven  ex_showroom_price
selling_price           1.000000  0.389686  -0.403412           0.919798
year                    0.389686  1.000000  -0.462554           0.119761
km_driven              -0.403412 -0.462554   1.000000          -0.206749
ex_showroom_price       0.919798  0.119761  -0.206749           1.000000
```