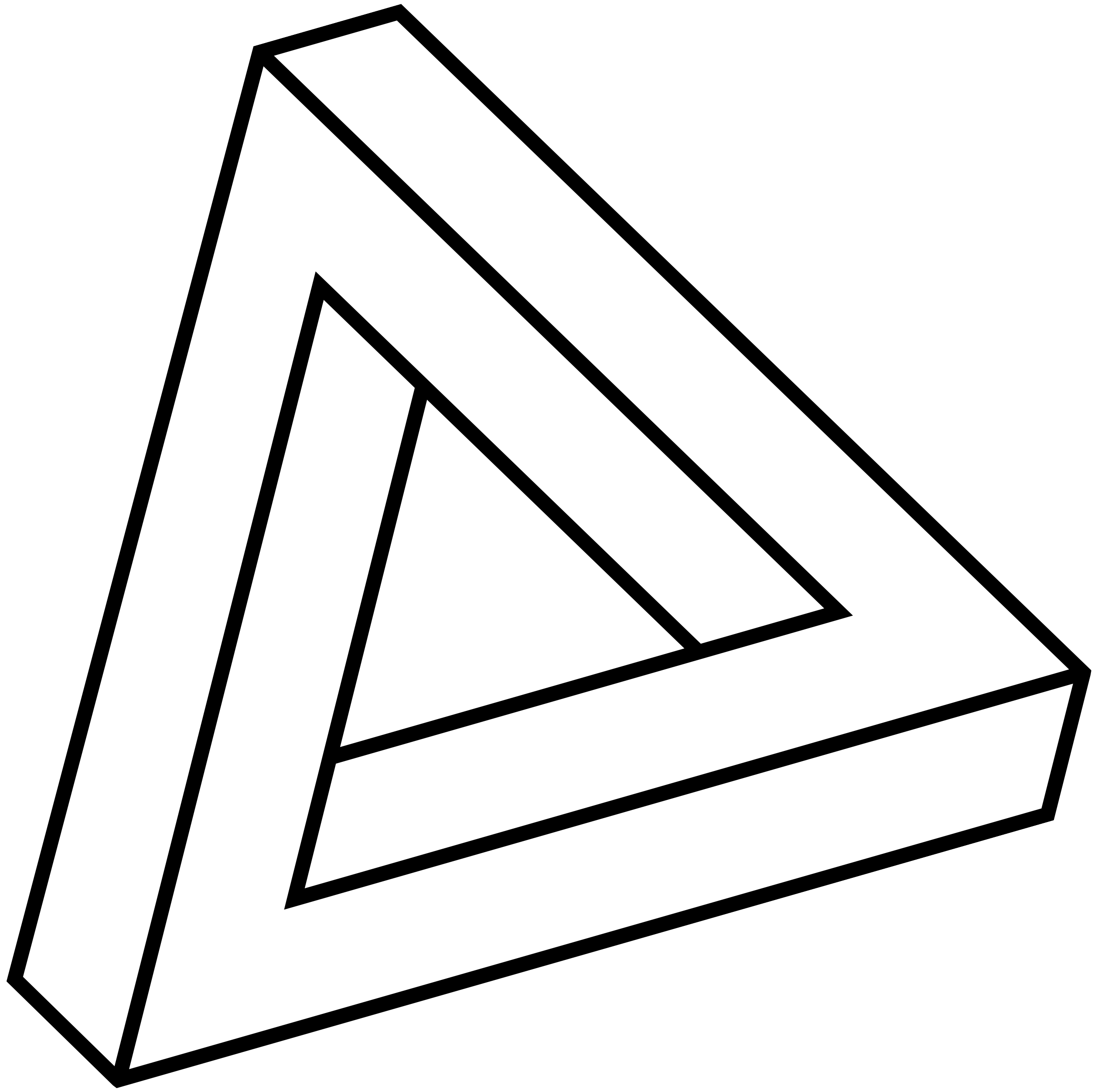
CAR PRICE PREDICTION

PRESENTATION



Content

1

Data preparation and EDA

Data transformation , detect outliers, check duplicated values and imputate or drop the null values

2

Data modeling and Evaluate

Using the distinct regression models and evaluate to find the best result for selling price prediction

3

Model Prediction

Perform model predict the selling price for brands and information about car by using Streamlit

Executive Summary

- Data preparation is to handle all values.
- Exploratory Data Analysis (EDA) is to perform the Data Analytic.

Result Summary

- The best model for car price prediction is **XGBoost** model.
- The primary factors which have high influent for model training is

year and for the mid-high are **max_power**, **is_popular**, **is_luxury** and **engine**

Features selection

Features selection is to choosing subset of features in a large set of variable. The goal is to improve and effective the model performance, reduce overfitting and enhance the interpretability.

The 13 features from data set aren’t enough to effective the model performance, so we will have to create new features from original 13 features in data set.

which are.....

age

brands

high_mileage

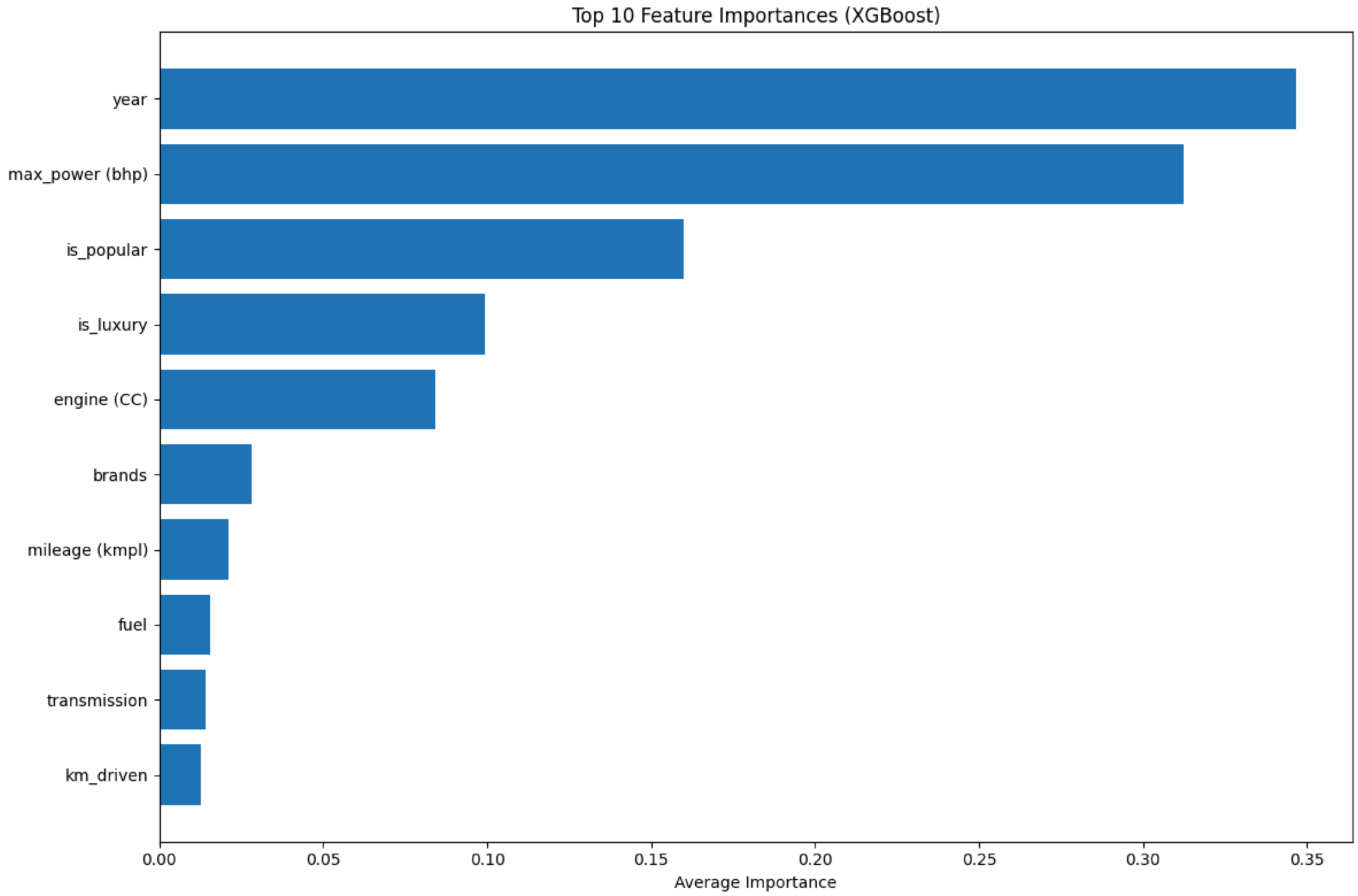
fuel_efficiency

is_popular

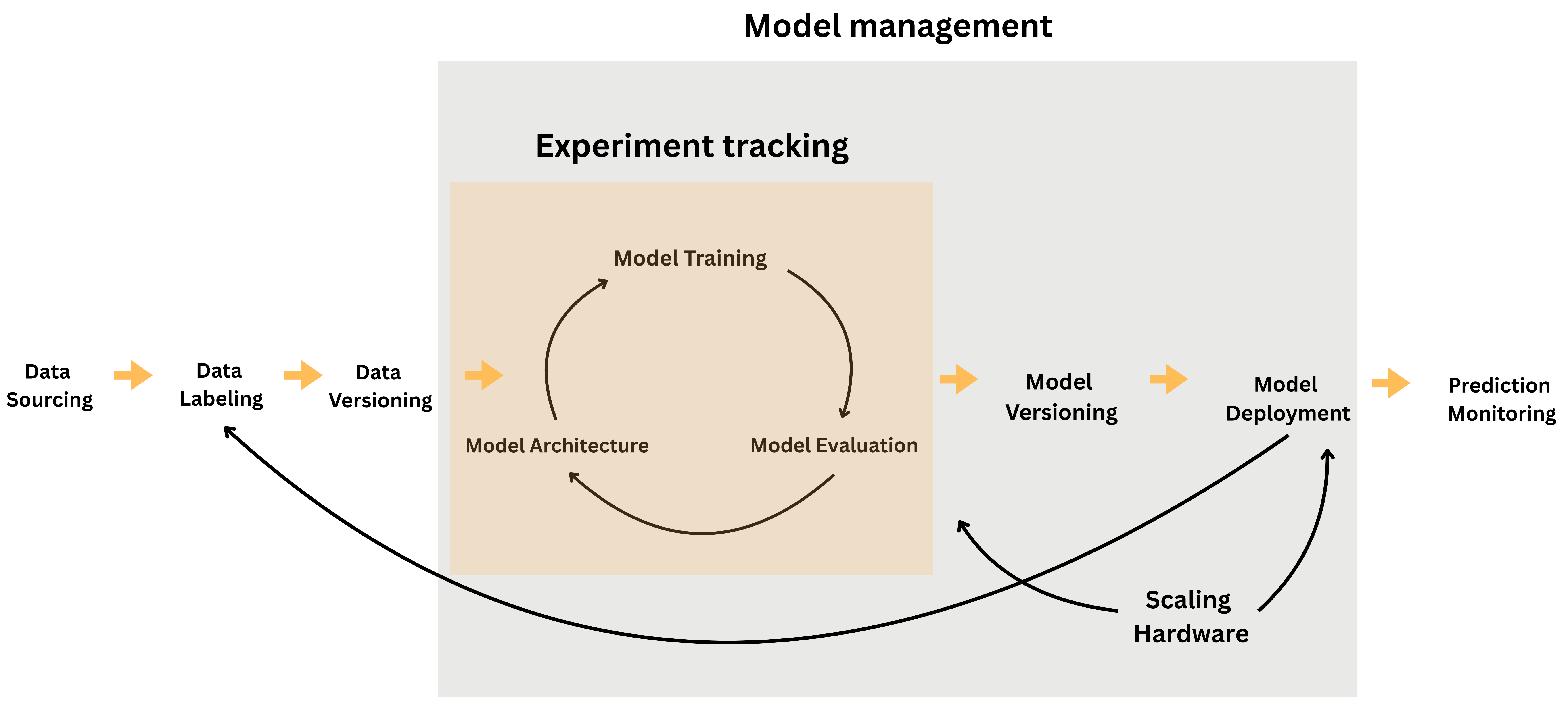
is_luxury

age_category

Features importance



MLOPs in model deployment



Model for car price prediction

Model Performance Comparison:

	Model	Train R²	Test R²	MAE	RMSE
0	Linear Regression	0.747	0.736	113404.94	154842.57
1	Ridge Regression	0.747	0.736	113385.53	154848.90
2	Lasso Regression	0.747	0.736	113404.94	154842.57
3	Random Forest	0.973	0.886	66799.70	101696.78
4	Gradient Boosting	0.966	0.895	64583.00	97784.77
5	XGBoost	0.962	0.896	64576.10	97150.58

- XGBoost
  Best model
- Linear Regression
- Ridge Regression
- Lasso Regression
- Random Forest
- Gradient Boosting

Data Description

The data was collected by Kaggle :

https://www.kaggle.com/code/mohaiminul101/car-price-prediction

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	1248 CC	74 bhp	190Nm@ 2000rpm	5.0
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14 kmpl	1498 CC	103.52 bhp	250Nm@ 1500-2500rpm	5.0
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.7 kmpl	1497 CC	78 bhp	12.7@ 2,700(kgm@ rpm)	5.0
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.0 kmpl	1396 CC	90 bhp	22.4 kgm at 1750-2750rpm	5.0
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.1 kmpl	1298 CC	88.2 bhp	11.5@ 4,500(kgm@ rpm)	5.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
8123	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	First Owner	18.5 kmpl	1197 CC	82.85 bhp	113.7Nm@ 4000rpm	5.0
8124	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	Fourth & Above Owner	16.8 kmpl	1493 CC	110 bhp	24@ 1,900-2,750(kgm@ rpm)	5.0
8125	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	First Owner	19.3 kmpl	1248 CC	73.9 bhp	190Nm@ 2000rpm	5.0
8126	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	23.57 kmpl	1396 CC	70 bhp	140Nm@ 1800-3000rpm	5.0
8127	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	23.57 kmpl	1396 CC	70 bhp	140Nm@ 1800-3000rpm	5.0

8128 rows × 13 columns

The data set was consisted by the column of car information following details :

- name
- year
- km_driven
- selling_price
- fuel
- transmission
- seller_type
- owner
- mileage
- engine
- max_power
- torque
- seats

Data preparation

DATA VALIDATION FOR MODELING

- Total Data : 8128 Records
- Total Columns : 13 Columns
- Categorical Columns : 6 Columns
- Numerical Columns : 5 Columns
- Null values : 1100 values
- Duplicated values : 1202 values

DATA CLEANSING

- Check any missing values and duplicated values
- Drop the Duplicated values
- Detect outlier values and remove

Data preparation

DATA TRANSFORMATION

```
car_df.drop("torque", axis=1, inplace=True)
```

- drop the **torque** columns

```
car_df = car_df.rename(columns={'mileage':'mileage (kmpl)'})
car_df['mileage (kmpl)'] = car_df['mileage (kmpl)'].str.split().str[0].astype(float)
```

```
car_df = car_df.rename(columns={'engine': 'engine (CC)'})
car_df['engine (CC)'] = car_df['engine (CC)'].str.split().str[0]
car_df['engine (CC)'] = pd.to_numeric(car_df['engine (CC)'], errors='coerce')
```

```
car_df = car_df.rename(columns={'max_power': 'max_power (bhp)'})
car_df['max_power (bhp)'] = car_df['max_power (bhp)'].str.split().str[0]
car_df['max_power (bhp)'] = pd.to_numeric(car_df['max_power (bhp)'], errors='coerce')
```

- Split **mileage**, **engine**, **max_power**, columns into single values

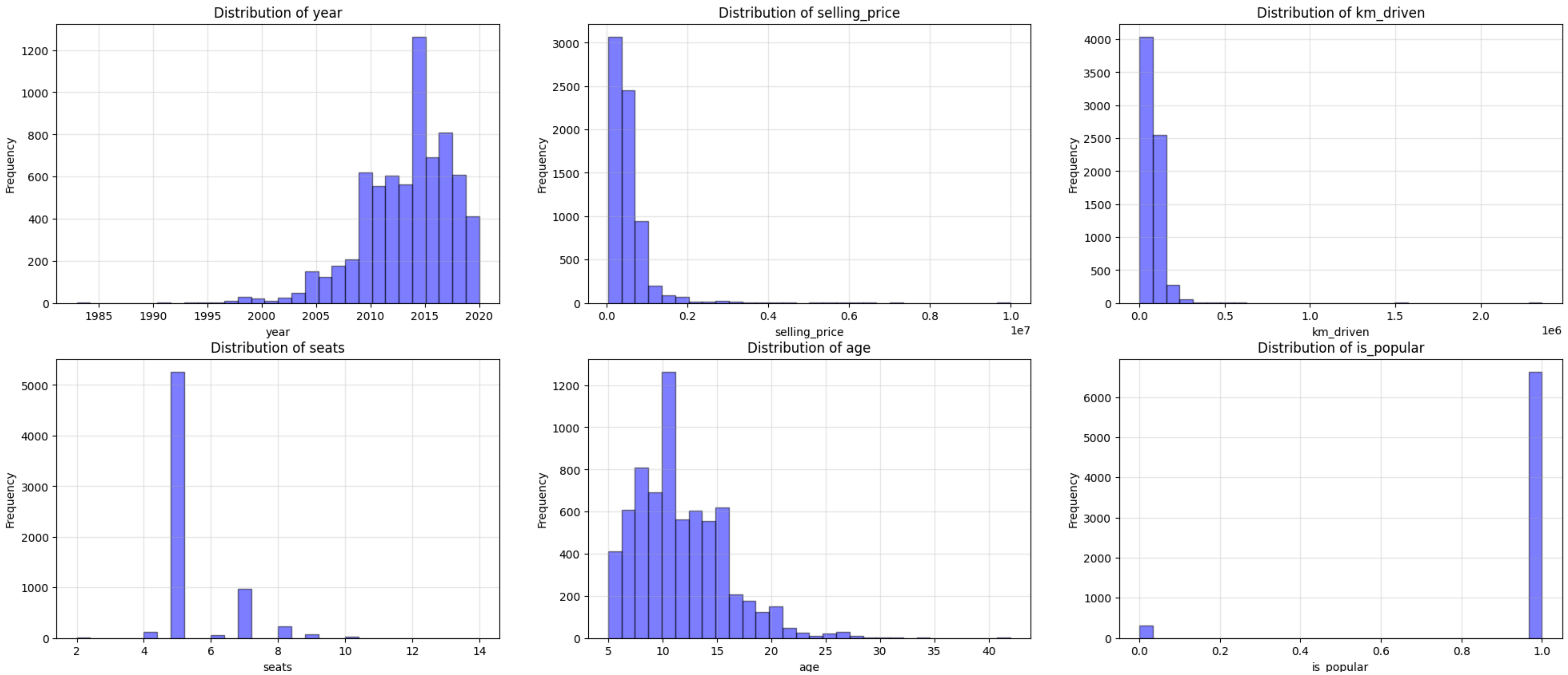
Create new feature like is_luxury, is_popular, age ETC

FOR EXAMPLE

```
luxury_brands = ['BMW', 'Mercedes-Benz', 'Audi', 'Jaguar', 'Land', 'Volvo','Lexus']
car_df["is_luxury"] = car_df['brands'].isin(luxury_brands).astype(int)
car_df
```

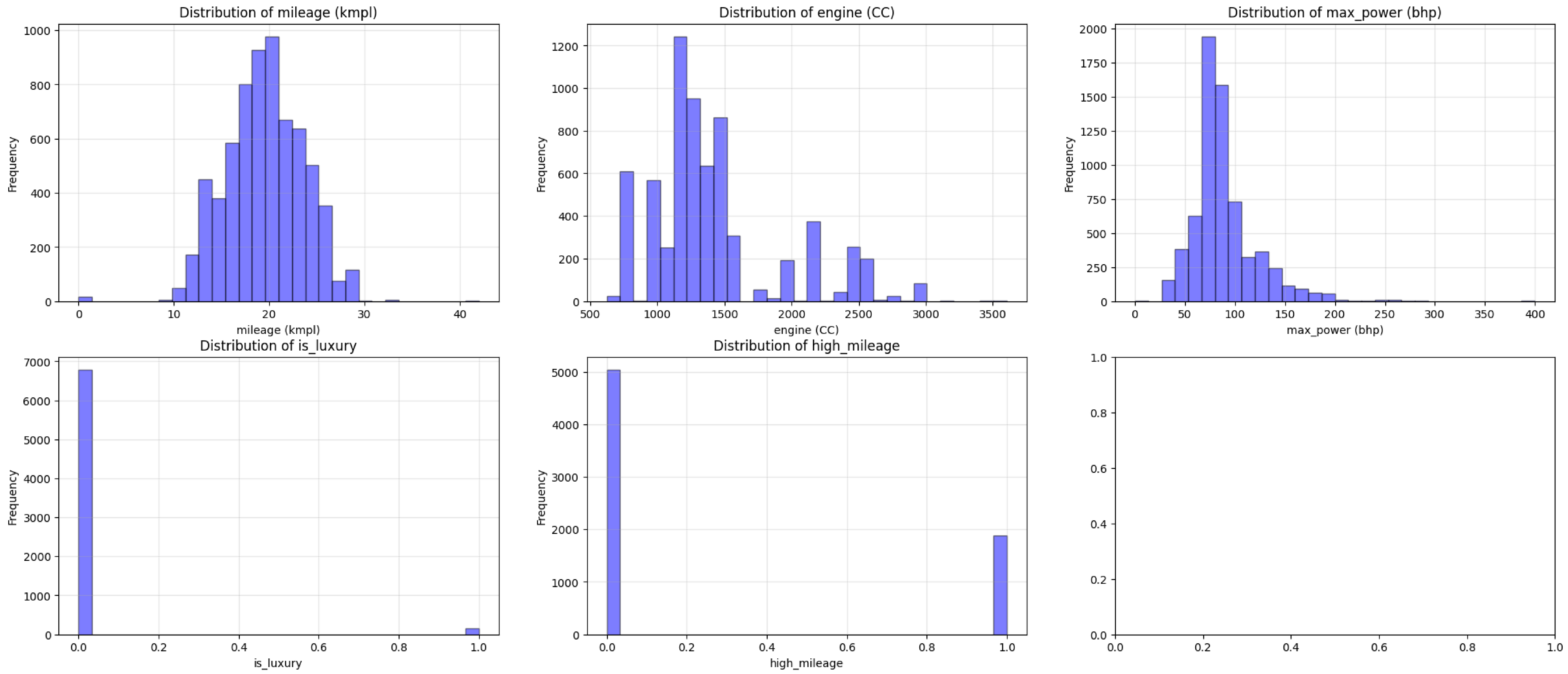
```
popular_brands = [
    'Maruti', 'Hyundai', 'Honda', 'Toyota', 'Tata', 'Mahindra',
    'Ford', 'Renault', 'Volkswagen', 'Skoda', 'Nissan', 'Chevrolet'
]

car_df["is_popular"] = car_df['brands'].isin(popular_brands).astype(int)
car_df
```

Exploratory Data Analysis

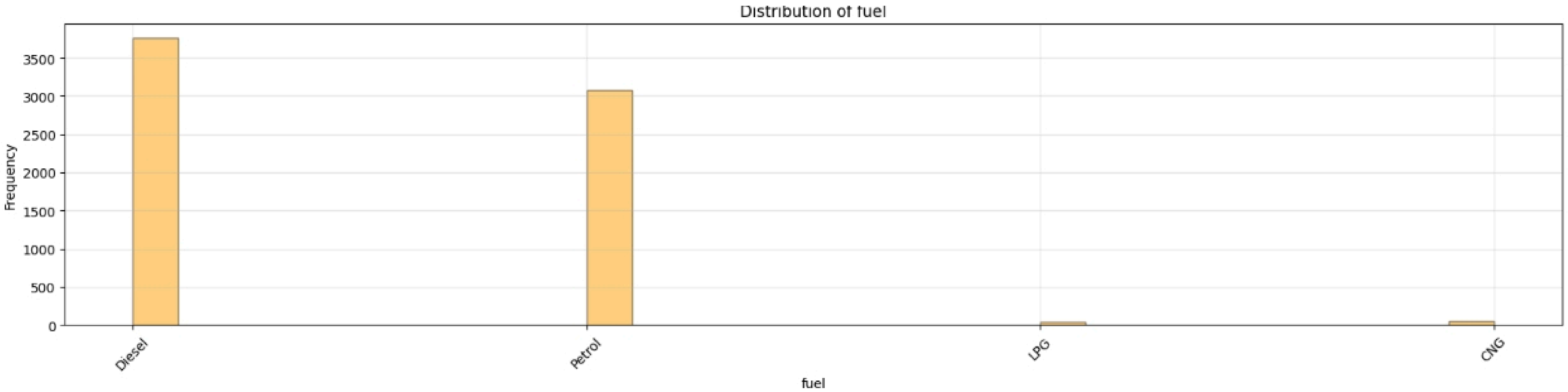


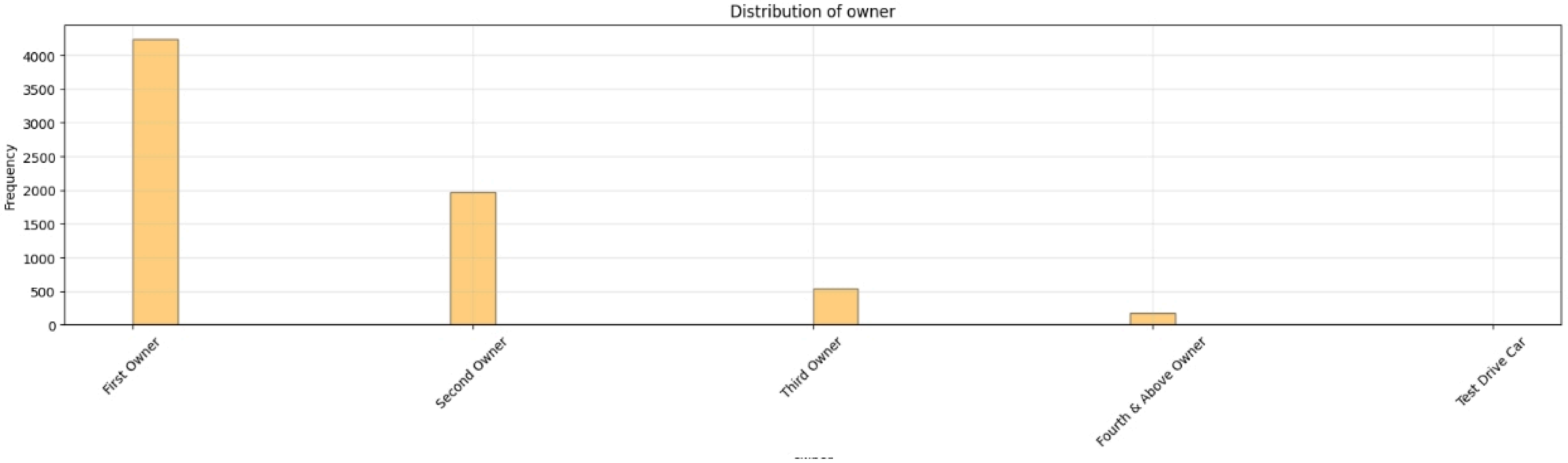

```
<class 'pandas.core.frame.DataFrame'>
Index: 6926 entries, 0 to 8125
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   name             6926 non-null   object
 1   year             6926 non-null   int64
 2   selling_price    6926 non-null   int64
 3   km_driven        6926 non-null   int64
 4   fuel             6926 non-null   object
 5   seller_type      6926 non-null   object
 6   transmission     6926 non-null   object
 7   owner            6926 non-null   object
 8   mileage (kmpl)   6718 non-null   float64
 9   engine (CC)      6718 non-null   float64
 10  max_power (bhp)  6720 non-null   float64
 11  seats            6718 non-null   float64
 12  age              6926 non-null   int64
 13  brands           6926 non-null   object
 14  is_popular       6926 non-null   int64
 15  is_luxury        6926 non-null   int64
 16  high_mileage     6926 non-null   int64
 17  age_category     6926 non-null   category
dtypes: category(1), float64(4), int64(7), object(6)
memory usage: 980.9+ KB
```

Exploratory Data Analysis

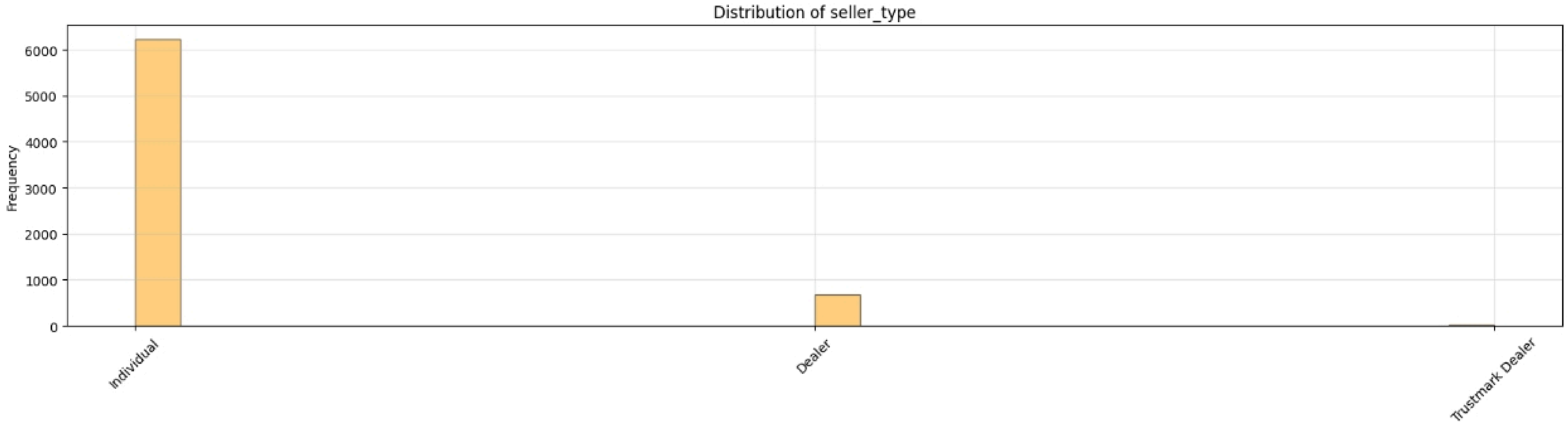


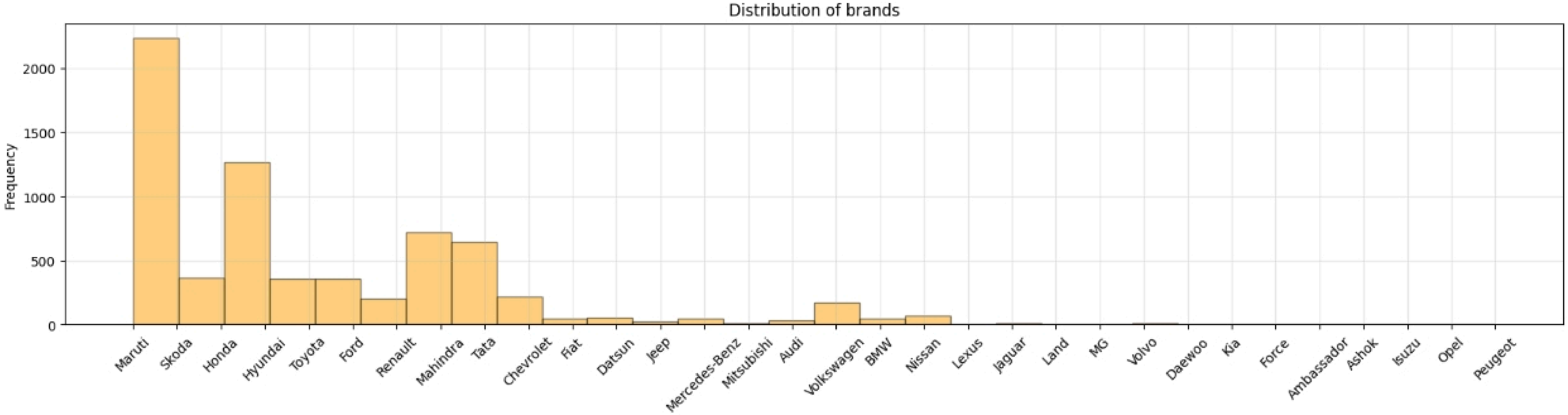
Exploratory Data Analysis



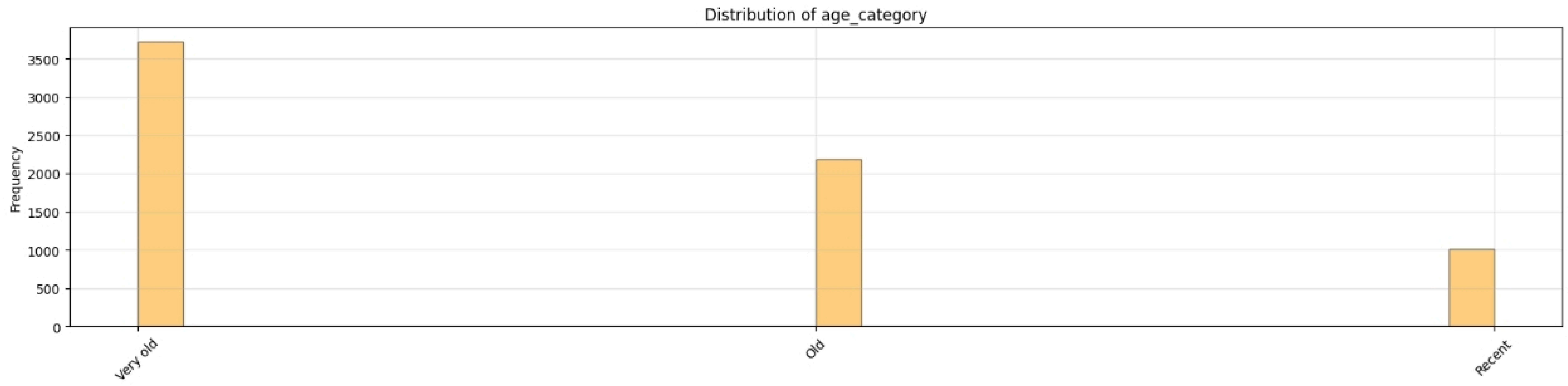


Exploratory Data Analysis



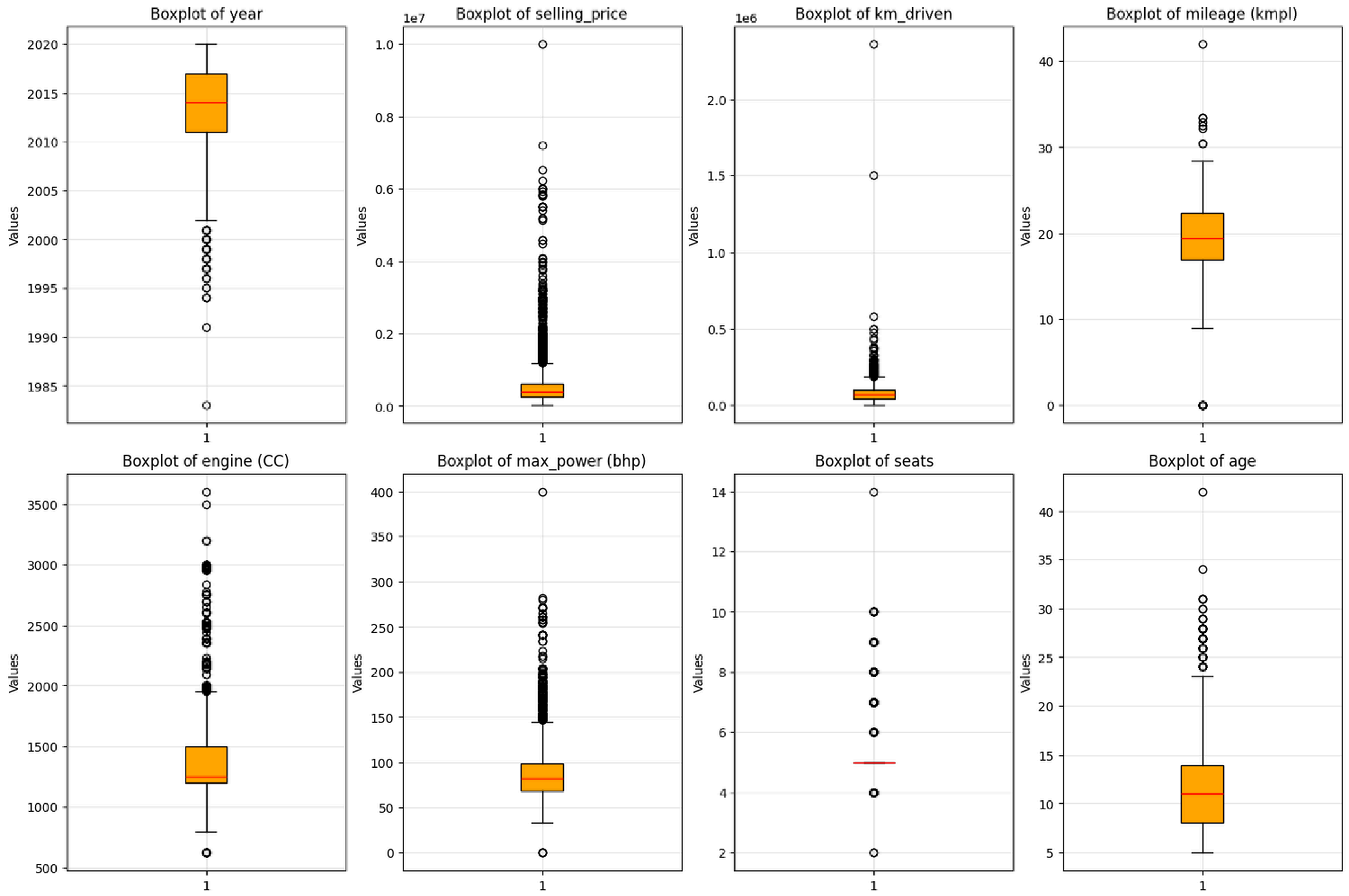


Exploratory Data Analysis



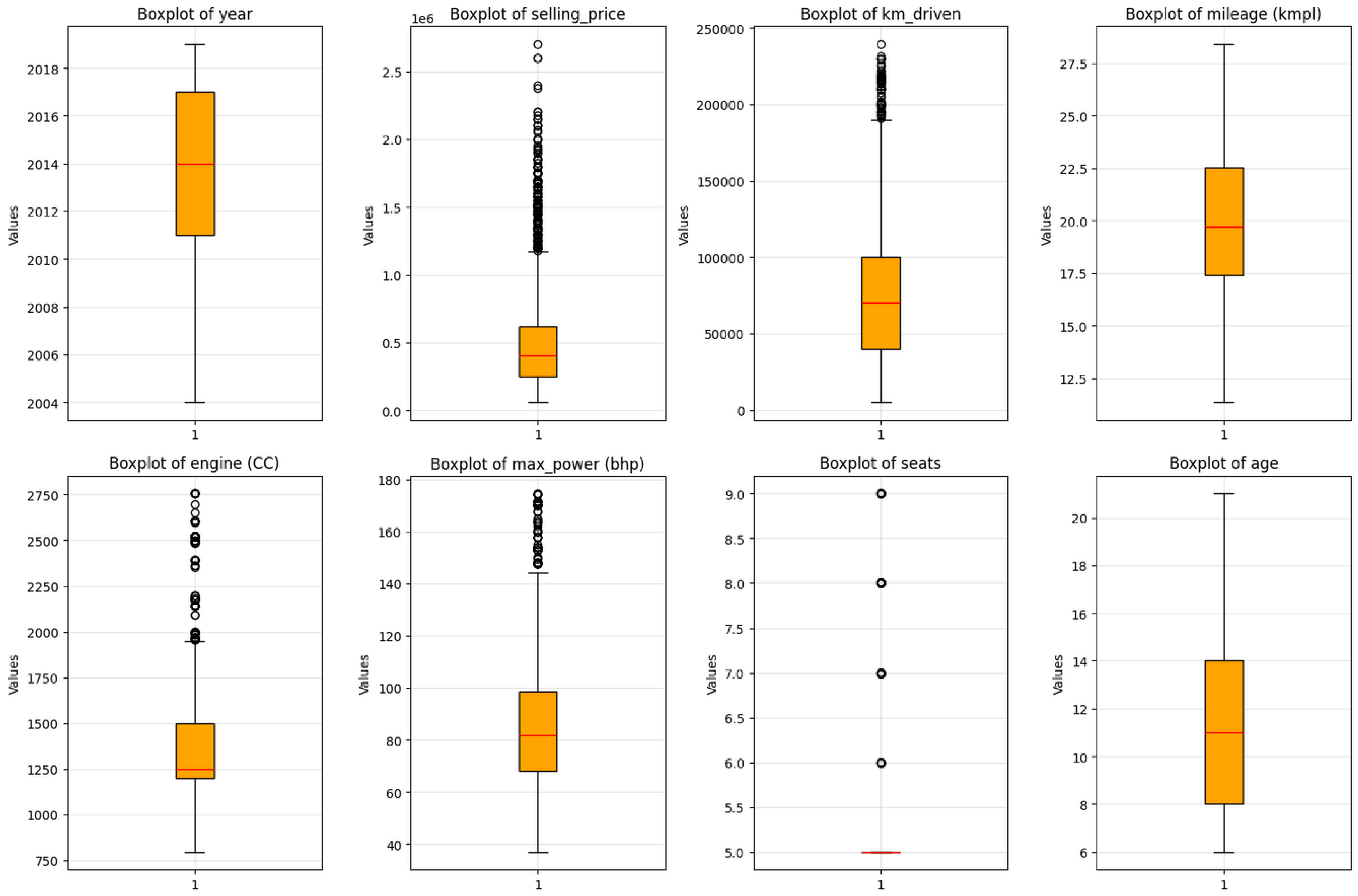
Exploratory Data Analysis

Outliers before cleansing

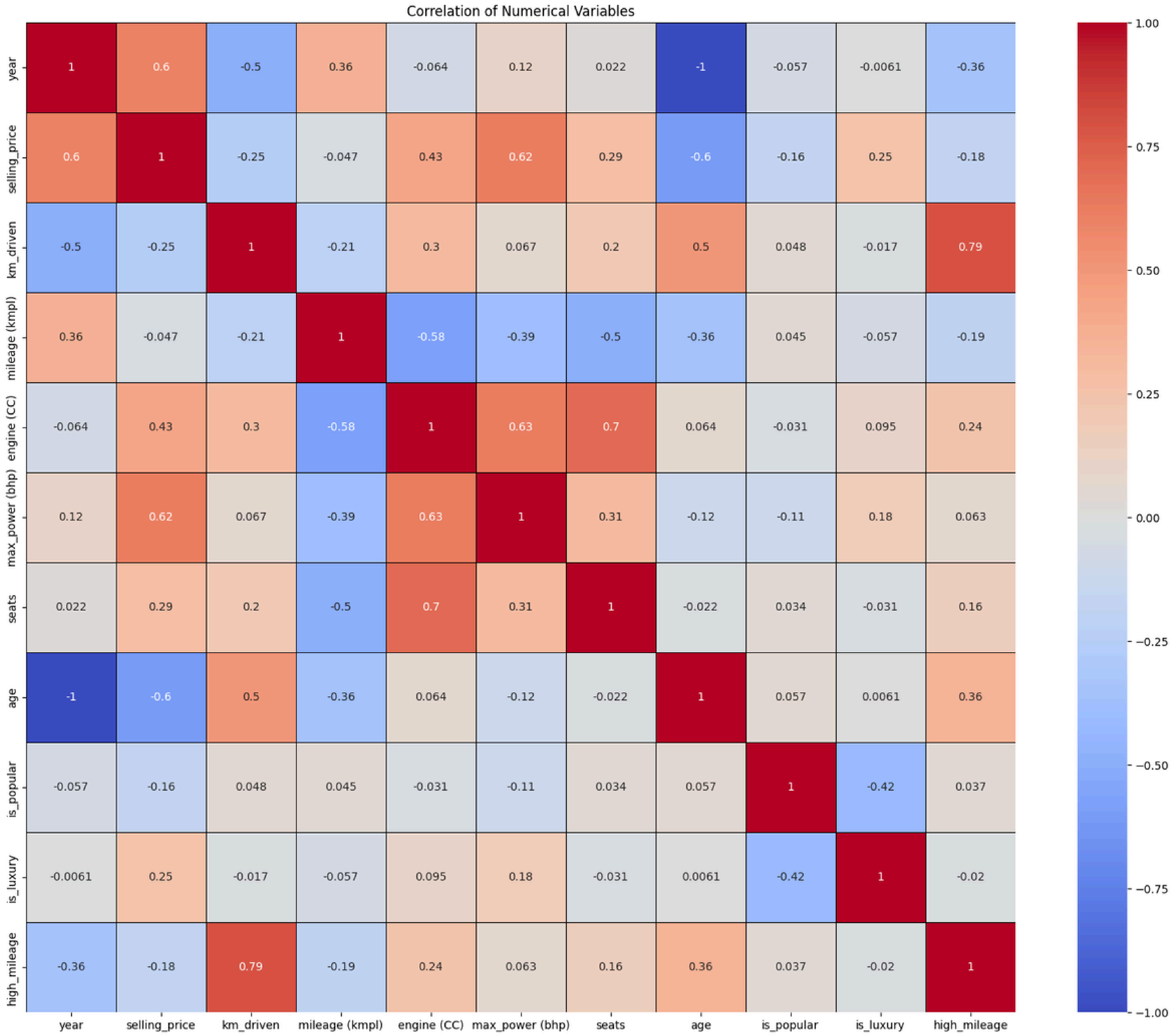


Exploratory Data Analysis

Outliers after cleansing



Exploratory Data Analysis



Data Modeling

Use LabelEncoder to encode the categorical values to prepare for modeling

For example

```
car_df["fuel"] = LabelEncoder.fit_transform(car_df["fuel"])
car_df["fuel"].unique()
```

```
array([1, 3, 2, 0])
```

```
car_df["seller_type"] = LabelEncoder.fit_transform(car_df["seller_type"])
car_df["seller_type"].unique()
```

```
array([1, 0, 2])
```

```
car_df["brands"] = LabelEncoder.fit_transform(car_df["brands"])
car_df["brands"].unique()
```

```
array([16, 21,  9, 10, 23,  8, 20, 15, 22,  4,  5, 12, 17,  2, 24, 19,  3,
       14, 13,  6,  7, 25, 18,  1,  0, 11])
```

Drop the ‘selling_price’ feature for and split for training

```
y = car_df['selling_price']
X = car_df.drop('selling_price', axis=1)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Split the data for 70% training and 30% for testing

Prepare models for prediction, which are regression models and tree-based models

- Linear Regression
- Ridge Regression
- Lasso Regression
- Random Forest
- Gradient Boosting
- XGBoost

Train Model

Example for coding

```python
models = {
    "Linear Regression": Pipeline([
        ('scaler', StandardScaler()),
        ('model', LinearRegression())
    ]),

    "Ridge Regression": Pipeline([
        ('scaler', StandardScaler()),
        ('model', GridSearchCV(
            Ridge(),
            param_grid={"alpha": [0.1, 1.0, 10.0]},
            cv=10
        ))
    ]),
    "Lasso Regression": GridSearchCV(
        Pipeline([
            ('scaler', StandardScaler()),
            ('model', Lasso(random_state=42))
        ]),
        param_grid={"model__alpha": [0.001, 0.01, 0.1, 1.0, 10.0]},
        cv=10,
    ),
    "Random Forest": GridSearchCV(
        RandomForestRegressor(random_state=42),
        param_grid={
            "n_estimators": [100, 200],
            "max_depth": [None, 10, 20],
            "min_samples_split": [2, 5]
        },
        cv=10,
        n_jobs=-1
    ),
```

```python
results = []
trained_models = {}

for name, model in models.items():
    print(f"Training {name}...")
    model.fit(X_train, y_train)

    if hasattr(model, 'best_estimator_'):
        best_model = model.best_estimator_
    elif hasattr(model, 'named_steps') and hasattr(model.named_steps['model'], 'best_estimator_'):
        best_model = model
    else:
        best_model = model

    trained_models[name] = best_model

    y_pred = best_model.predict(X_test)
    y_pred_train = best_model.predict(X_train)

    r2_train = r2_score(y_train, y_pred_train)
    r2_test = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))

    results.append({
        "Model": name,
        "Train R²": round(r2_train, 3),
        "Test R²": round(r2_test, 3),
        "MAE": round(mae, 2),
        "RMSE": round(rmse, 2)
    })

results_car_df = pd.DataFrame(results)
print("="*60)
print("Model Performance Comparison:")
print(results_car_df)
```

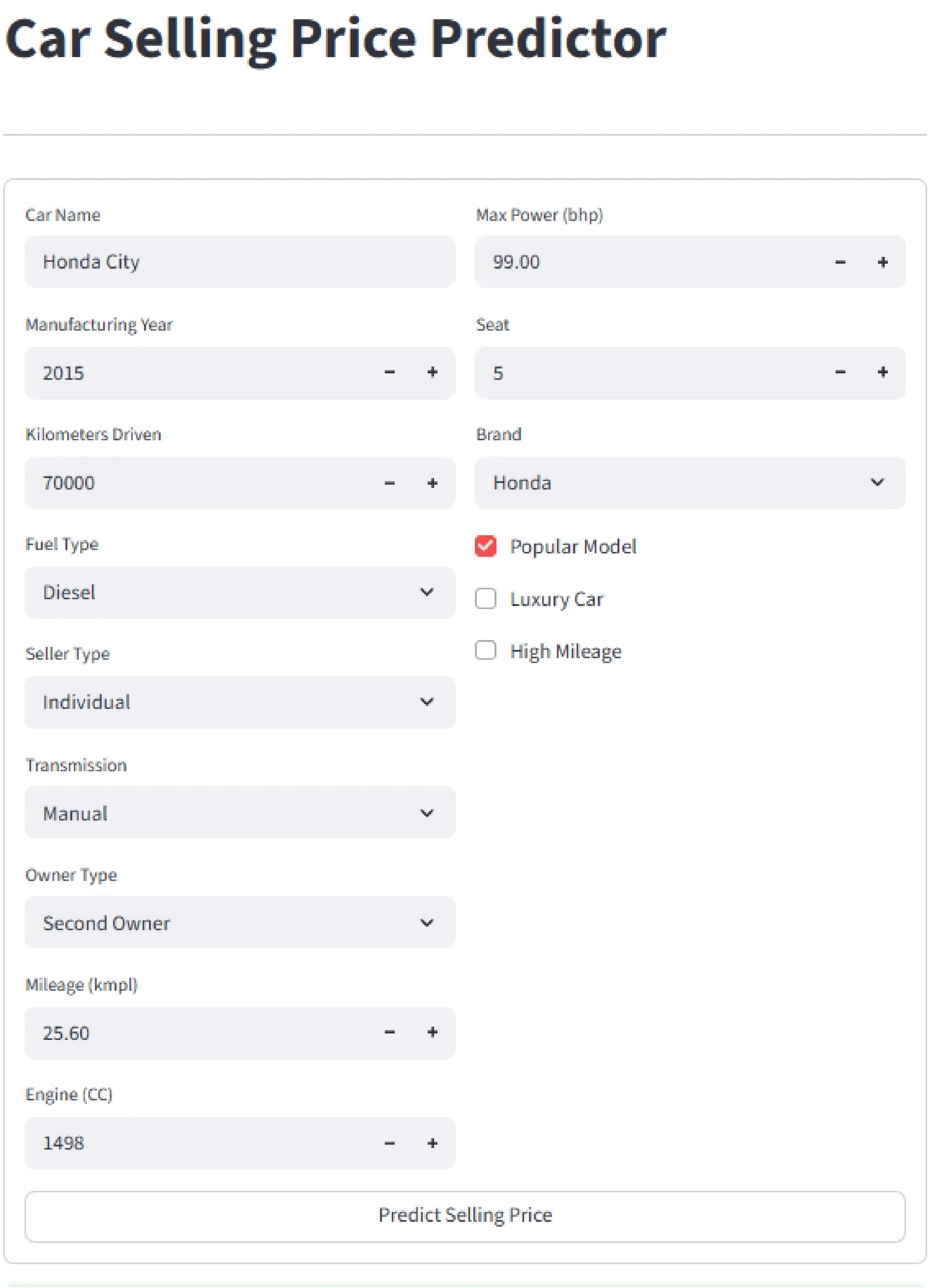
Evaluation

Model Performance Comparison:

	Model	Train R²	Test R²	MAE	RMSE
0	Linear Regression	0.747	0.736	113404.94	154842.57
1	Ridge Regression	0.747	0.736	113385.53	154848.90
2	Lasso Regression	0.747	0.736	113404.94	154842.57
3	Random Forest	0.973	0.886	66799.70	101696.78
4	Gradient Boosting	0.966	0.895	64583.00	97784.77
5	XGBoost	0.962	0.896	64576.10	97150.58

The 3 Linear models perform similar result , but show the large MAE and RMSE. The Random Forest with a an excellent train score but low on test score , which is show some overfitting. The Gradient and XGBoost models are perform with the best result among these models, but the best choice is XGBoost with lowest MAE and RMSE score.

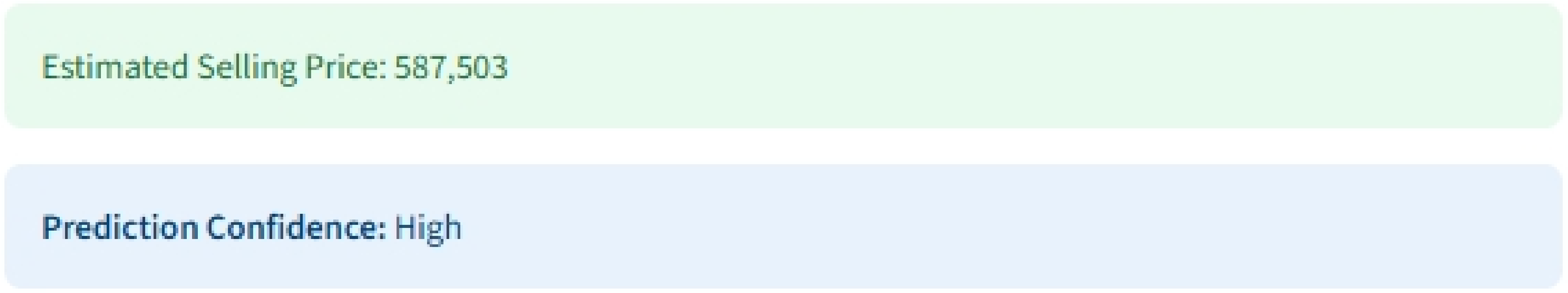
Evaluation



Save the best model as pkl file and then use Streamlit to perform a car price model prediction

Input the information of the car, for example Honda City 2015 which the average price in second hand or above in range 400000 - 600000

The model prediction was 587,503 which is a excellent and fascinate result.



End