# github

https://github.com/saugatshakya/predicting_car_price

# live demo

https://carprice-predict.ambitiousisland-1be3b1ed.southeastasia.azurecontainerapps.io/

In [1]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pickle
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Load the dataset
data = pd.read_csv("cars.csv")
print("Dataset loaded successfully!")
print(f"Original dataset shape: {data.shape}")
```

```
Dataset loaded successfully!
Original dataset shape: (8128, 13)
```

In [3]:
```
# Display basic information about the dataset
print("\nDataset Info:")
data.info()
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8128 entries, 0 to 8127
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   name           8128 non-null   object
 1   year           8128 non-null   int64
 2   selling_price  8128 non-null   int64
 3   km_driven      8128 non-null   int64
 4   fuel           8128 non-null   object
 5   seller_type    8128 non-null   object
 6   transmission   8128 non-null   object
 7   owner          8128 non-null   object
 8   mileage        7907 non-null   object
 9   engine         7907 non-null   object
 10  max_power      7913 non-null   object
 11  torque         7906 non-null   object
 12  seats          7907 non-null   float64
dtypes: float64(1), int64(3), object(9)
memory usage: 825.6+ KB
```

In [4]:
```
# Display first few rows
print("\nFirst 5 rows:")
data.head()
```

```
First 5 rows:
```

Out[4]:

| | name | year | selling_price | km_driven | fuel | seller_type | transmissio |
|---|---|---|---|---|---|---|---|
| **0** | Maruti Swift Dzire VDI | 2014 | 450000 | 145500 | Diesel | Individual | Manua |
| **1** | Skoda Rapid 1.5 TDI Ambition | 2014 | 370000 | 120000 | Diesel | Individual | Manua |
| **2** | Honda City 2017-2020 EXi | 2006 | 158000 | 140000 | Petrol | Individual | Manua |
| **3** | Hyundai i20 Sportz Diesel | 2010 | 225000 | 127000 | Diesel | Individual | Manua |
| **4** | Maruti Swift VXI BSIII | 2007 | 130000 | 120000 | Petrol | Individual | Manua |

In [5]:
```
# Check for missing values
print("\nMissing values in each column:")
print(data.isnull().sum())
```

```
Missing values in each column:
name               0
year               0
selling_price      0
km_driven          0
fuel               0
seller_type        0
transmission       0
owner              0
mileage          221
engine           221
max_power        215
torque           222
seats            221
dtype: int64
```

In [6]:
```
# Check for duplicates
print(f"\nNumber of duplicates: {data.duplicated().sum()}")
```

```
Number of duplicates: 1202
```

In [7]:
```
# Remove duplicates
```

```python
data = data.drop_duplicates()
print(f"Dataset shape after removing duplicates: {data.shape}")
```

Dataset shape after removing duplicates: (6926, 13)

In [8]:
```python
# Remove CNG and LPG fuel types as instructed
data = data[~data['fuel'].isin(['CNG', 'LPG'])]
print(f"Dataset shape after removing CNG/LPG: {data.shape}")
```

Dataset shape after removing CNG/LPG: (6832, 13)

In [9]:
```python
# Clean mileage column
data['mileage'] = (
    data['mileage']
    .str.replace('kmpl', '', regex=False)
    .str.replace('km/kg', '', regex=False)
    .str.strip()
)
data['mileage'] = pd.to_numeric(data['mileage'], errors='coerce')
```

In [10]:
```python
# Clean engine column
data['engine'] = (
    data['engine']
    .str.replace('CC', '', regex=False)
    .str.strip()
)
data['engine'] = pd.to_numeric(data['engine'], errors='coerce')
```

In [11]:
```python
# Clean max_power column
data['max_power'] = (
    data['max_power']
    .str.replace('bhp', '', regex=False)
    .str.strip()
)
data['max_power'] = pd.to_numeric(data['max_power'], errors='coerce
```

In [12]:
```python
# Fill missing values in seats with mode
data['seats'] = data['seats'].fillna(data['seats'].mode()[0])
```

In [13]:
```python
# Fill other missing values with median
for col in ['mileage', 'engine', 'max_power']:
    data[col] = data[col].fillna(data[col].median())
```

In [14]:
```python
# Drop torque column as instructed
data.drop(columns=['torque'], inplace=True)
```

In [15]:
```python
# Map owner values as instructed
owner_map = {
    'First Owner': 1,
    'Second Owner': 2,
    'Third Owner': 3,
    'Fourth & Above Owner': 4,
    'Test Drive Car': 5
}
data['owner'] = data['owner'].map(owner_map)
```

In [16]:
```
# Remove test drive cars as instructed
data = data[data['owner'] != 5]
print(f"Dataset shape after removing test drive cars: {data.shape}")
```

```
Dataset shape after removing test drive cars: (6827, 12)
```

In [17]:
```
# Extract brand from name
data['brand'] = data['name'].str.split().str[0]
data['brand'] = data['brand'].fillna('Unknown')
data.drop(columns=['name'], inplace=True)
```

In [18]:
```
#see all brand column unique values
print("\nUnique brands in the dataset:")
print(data['brand'].unique())
```

```
Unique brands in the dataset:
['Maruti' 'Skoda' 'Honda' 'Hyundai' 'Toyota' 'Ford' 'Renault' 'Mahin
dra'
 'Tata' 'Chevrolet' 'Fiat' 'Datsun' 'Jeep' 'Mercedes-Benz' 'Mitsubis
hi'
 'Audi' 'Volkswagen' 'BMW' 'Nissan' 'Lexus' 'Jaguar' 'Land' 'MG' 'Vo
lvo'
 'Daewoo' 'Kia' 'Force' 'Ambassador' 'Ashok' 'Isuzu' 'Opel' 'Peugeo
t']
```

In [19]:
```
# Apply log transformation to selling price as instructed
data['selling_price'] = np.log(data['selling_price'])
```

In [20]:
```
# Display final dataset info
print("\nFinal dataset info:")
data.info()
```

```
Final dataset info:
<class 'pandas.core.frame.DataFrame'>
Index: 6827 entries, 0 to 8125
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   year           6827 non-null   int64  
 1   selling_price  6827 non-null   float64
 2   km_driven      6827 non-null   int64  
 3   fuel           6827 non-null   object 
 4   seller_type    6827 non-null   object 
 5   transmission   6827 non-null   object 
 6   owner          6827 non-null   int64  
 7   mileage        6827 non-null   float64
 8   engine         6827 non-null   float64
 9   max_power      6827 non-null   float64
 10  seats          6827 non-null   float64
 11  brand          6827 non-null   object 
dtypes: float64(5), int64(3), object(4)
memory usage: 693.4+ KB
```

In [21]:
```
print("\nFirst 5 rows of cleaned data:")
data.head()
```

```
First 5 rows of cleaned data:
```

Out[21]:

| | year | selling_price | km_driven | fuel | seller_type | transmission | owner |
|---|---|---|---|---|---|---|---|
| **0** | 2014 | 13.017003 | 145500 | Diesel | Individual | Manual | 1 |
| **1** | 2014 | 12.821258 | 120000 | Diesel | Individual | Manual | 2 |
| **2** | 2006 | 11.970350 | 140000 | Petrol | Individual | Manual | 3 |
| **3** | 2010 | 12.323856 | 127000 | Diesel | Individual | Manual | 1 |
| **4** | 2007 | 11.775290 | 120000 | Petrol | Individual | Manual | 1 |

In [22]:

```
# Check for remaining missing values
print("\nRemaining missing values:")
print(data.isnull().sum())
```

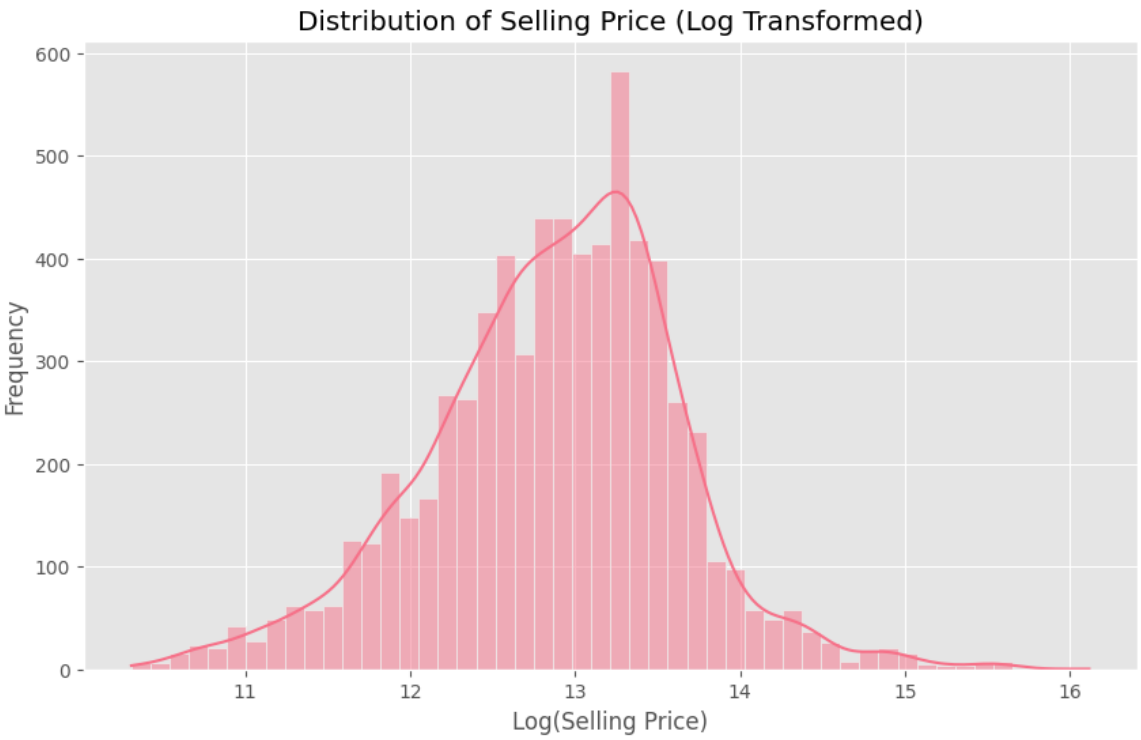
```
Remaining missing values:
year             0
selling_price    0
km_driven        0
fuel             0
seller_type      0
transmission     0
owner            0
mileage          0
engine           0
max_power        0
seats            0
brand            0
dtype: int64
```

In [23]:

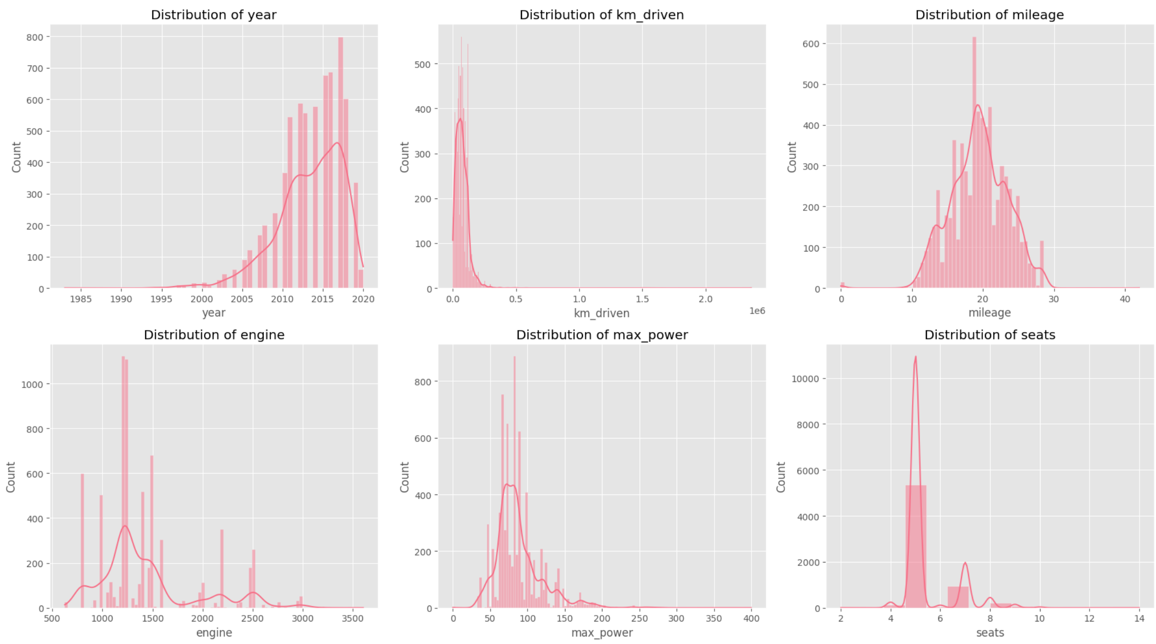
```
# Set style for plots
plt.style.use('ggplot')
sns.set_palette("husl")
```

In [24]:

```
# Visualize the distribution of selling price (log transformed)
plt.figure(figsize=(10, 6))
sns.histplot(data['selling_price'], bins=50, kde=True)
plt.title("Distribution of Selling Price (Log Transformed)")
plt.xlabel("Log(Selling Price)")
plt.ylabel("Frequency")
plt.show()
```

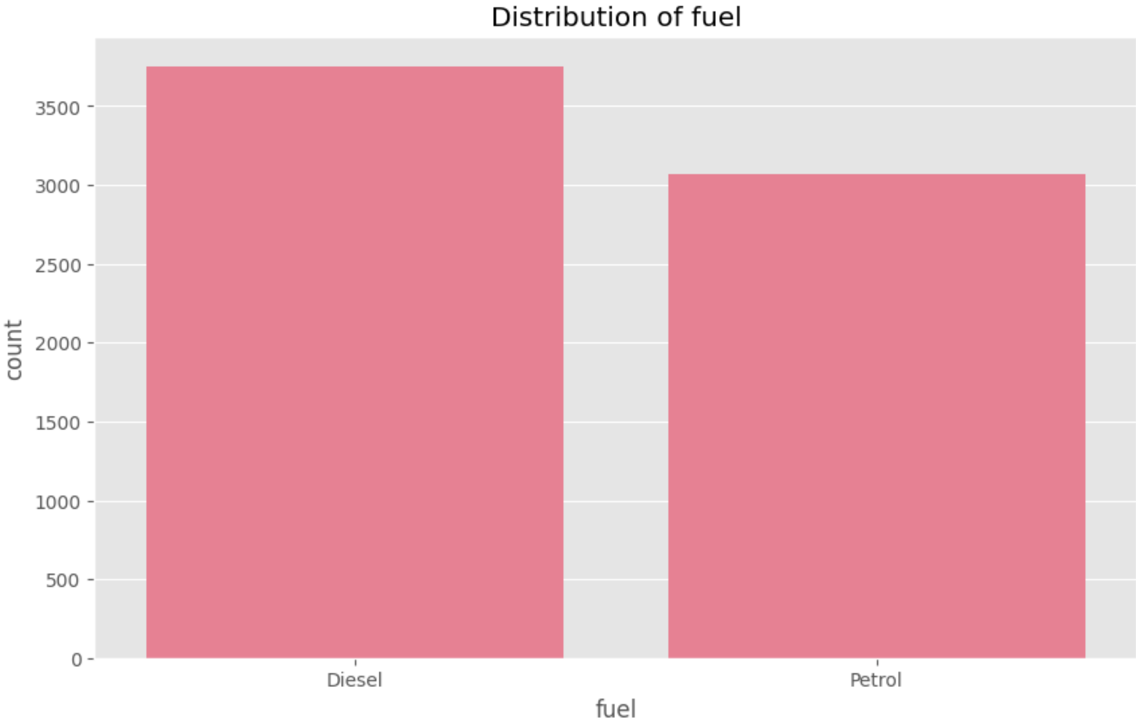


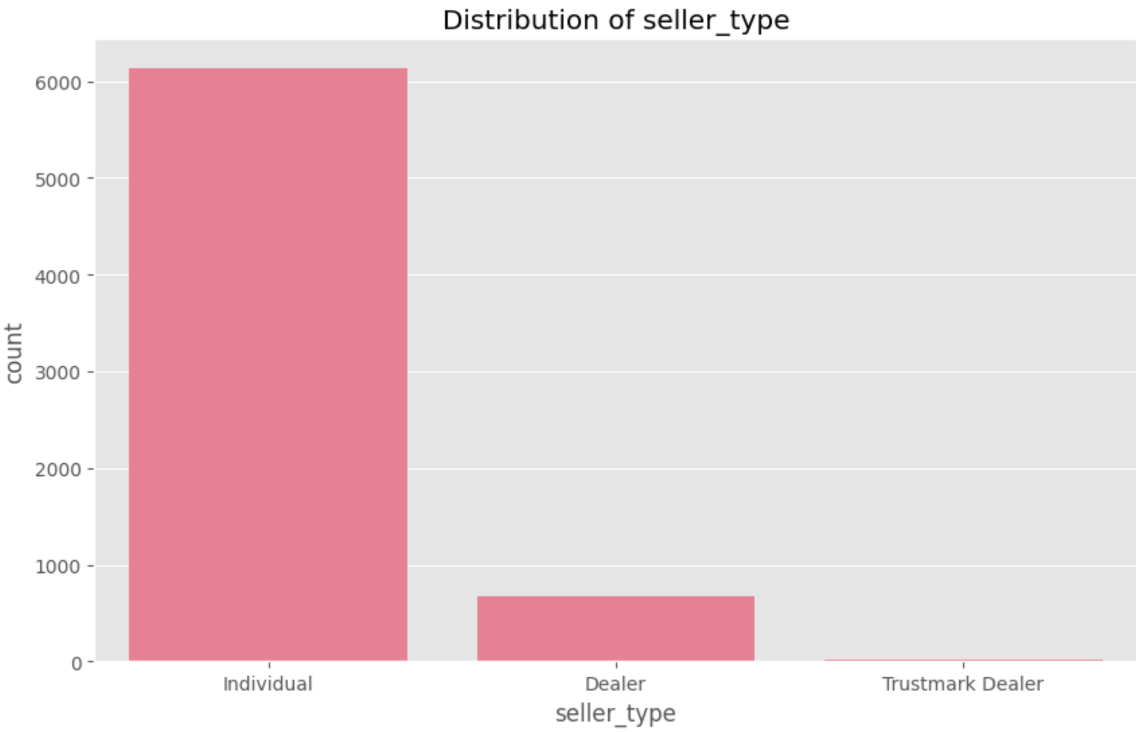

In [25]:
```
# Visualize numerical features
num_cols = ['year', 'km_driven', 'mileage', 'engine', 'max_power',

fig, axes = plt.subplots(2, 3, figsize=(18, 10))
axes = axes.ravel()

for i, col in enumerate(num_cols):
    sns.histplot(data[col], kde=True, ax=axes[i])
    axes[i].set_title(f"Distribution of {col}")

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

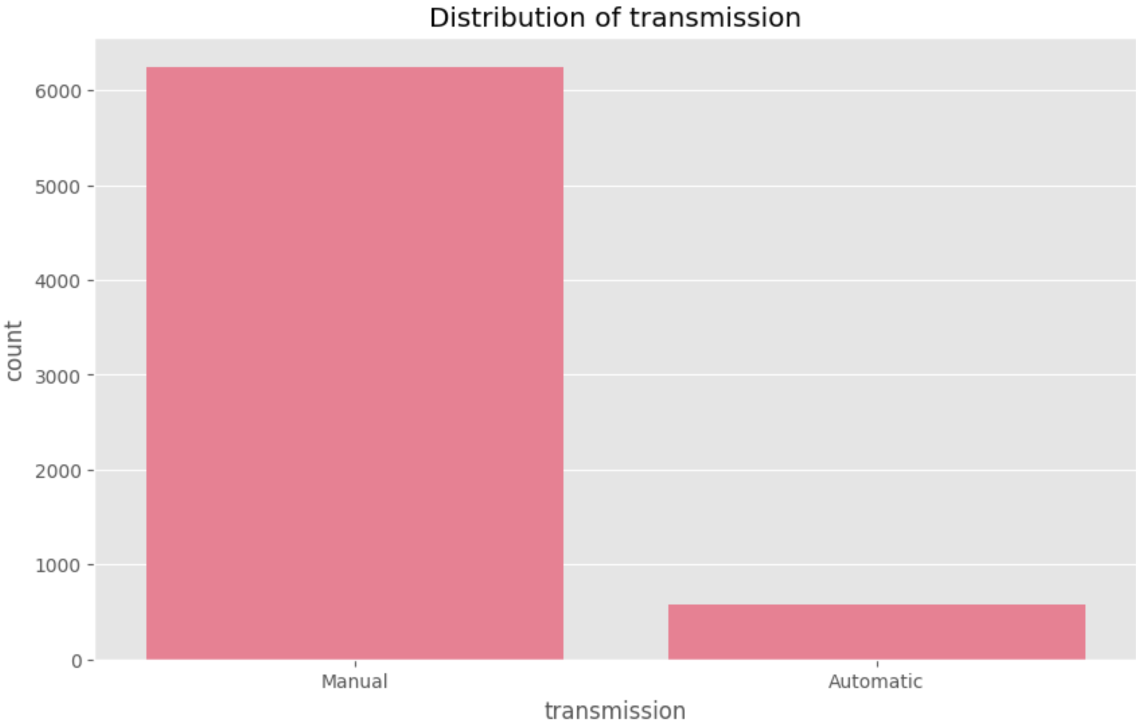


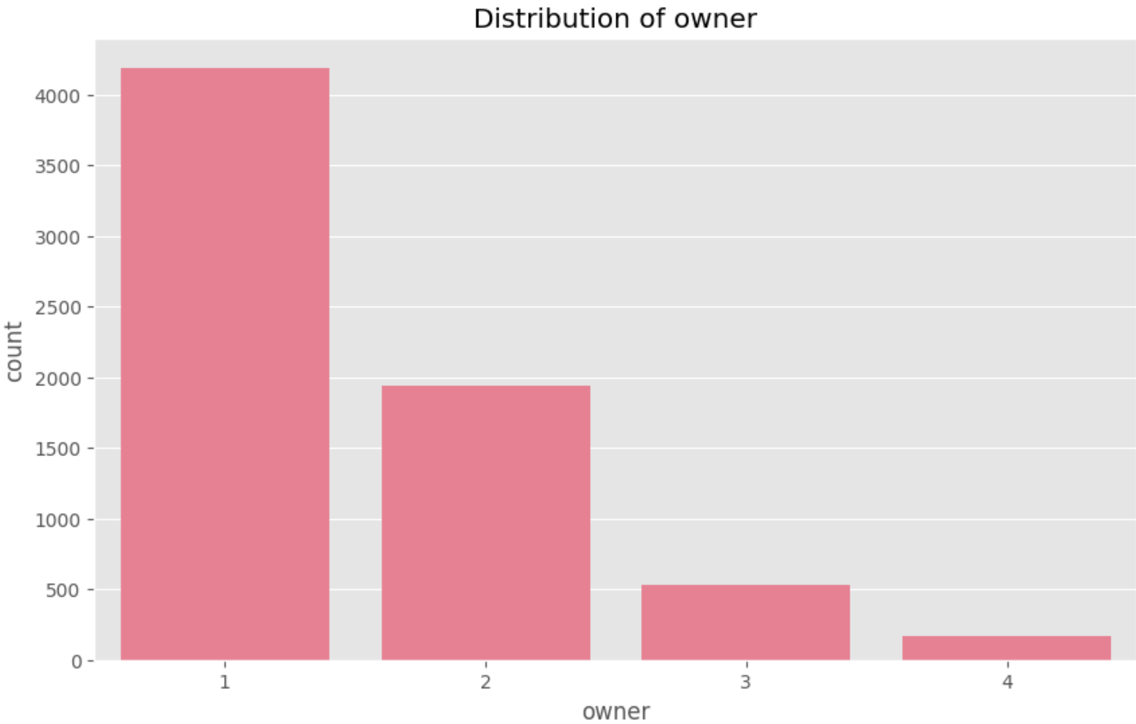

In [26]:
```
# Visualize categorical features
cat_cols = ['fuel', 'seller_type', 'transmission', 'owner', 'brand'
```

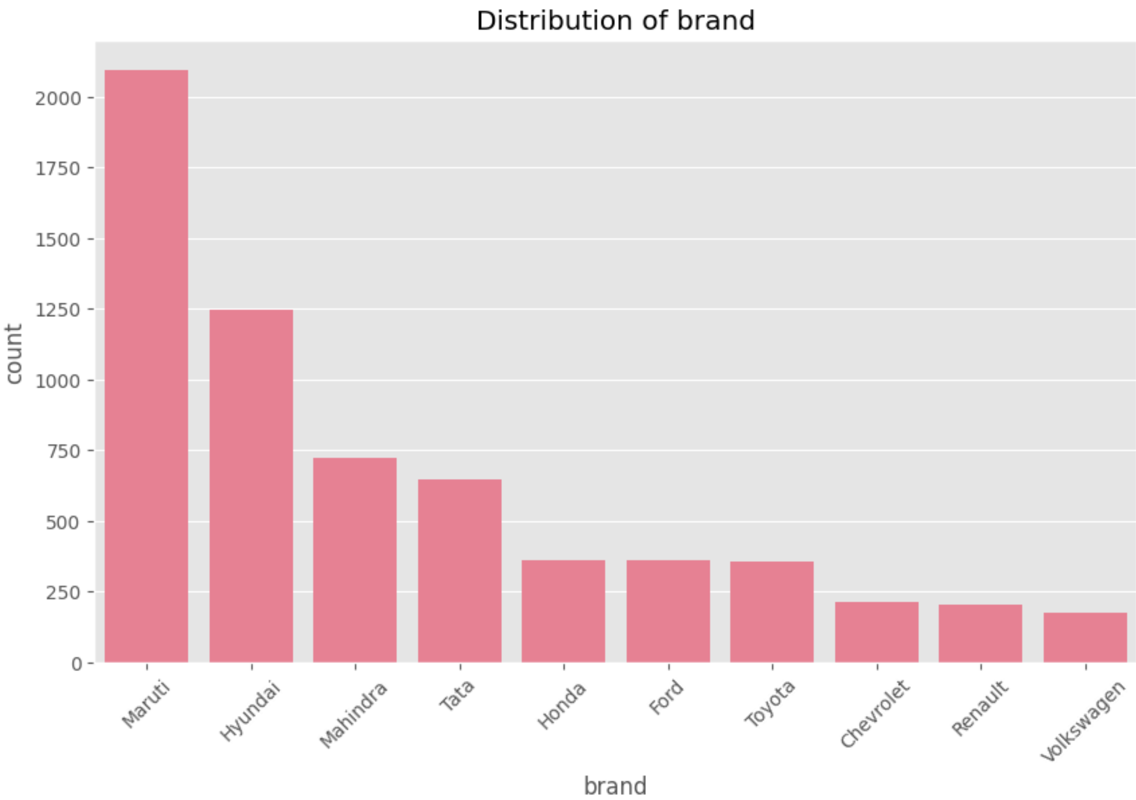
```
for col in cat_cols:
    plt.figure(figsize=(10, 6))
    if col == 'brand':  # For brand, show only top 10
        top_brands = data['brand'].value_counts().nlargest(10).inde
        sns.countplot(data=data[data['brand'].isin(top_brands)], x=
        plt.xticks(rotation=45)
    else:
        sns.countplot(data=data, x=col)
    plt.title(f"Distribution of {col}")
    plt.show()
```



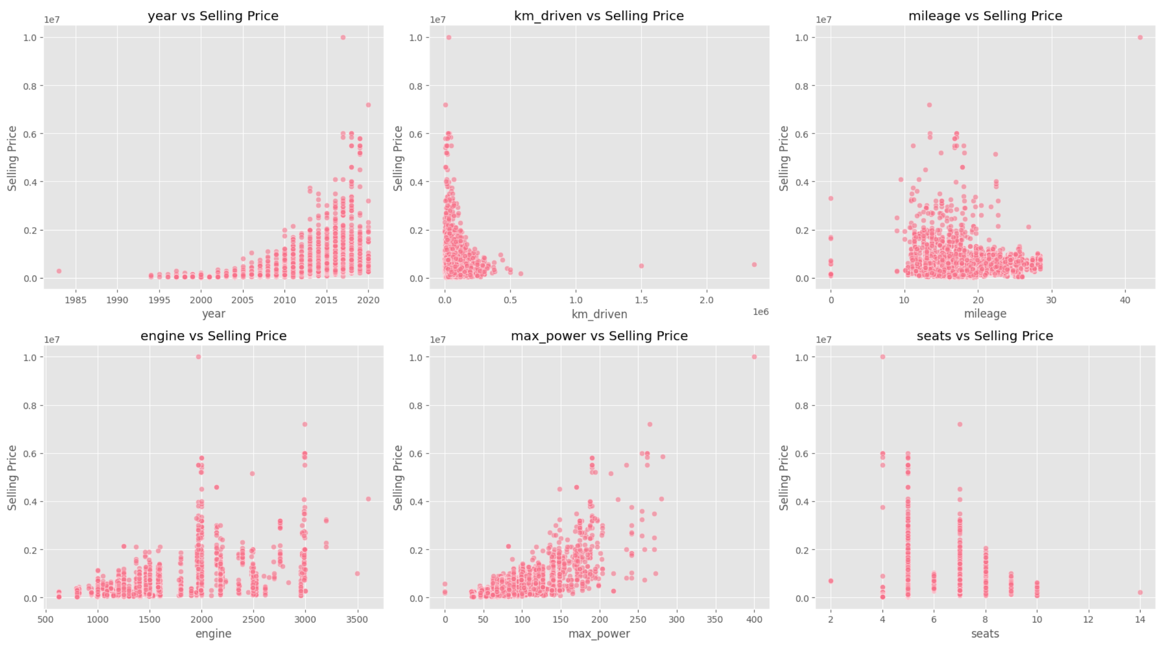




Distribution of transmission
count
0
1000
2000
3000
4000
5000
6000
Manual
Automatic
transmission


Distribution of owner
count
0
500
1000
1500
2000
2500
3000
3500
4000
1
2
3
4
owner

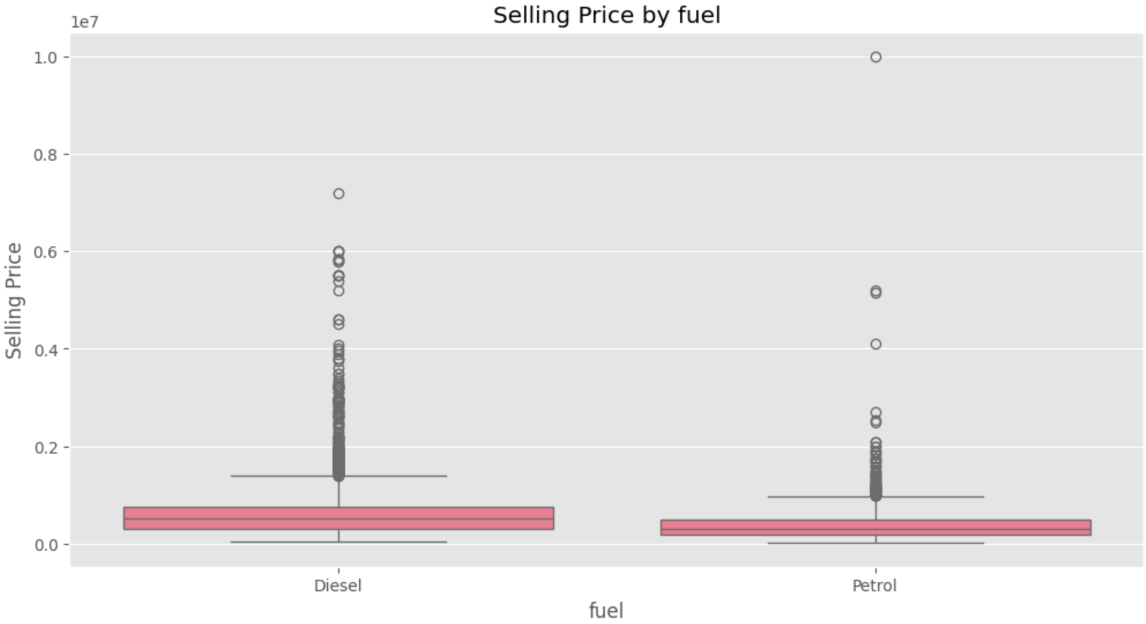


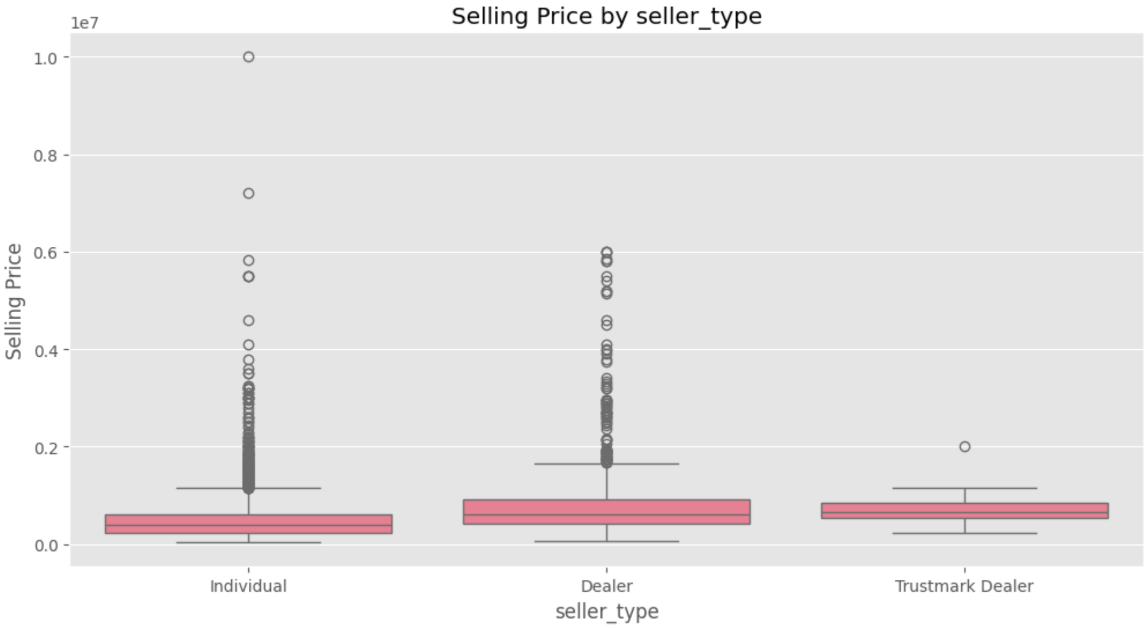

In [27]:
```python
# Analyze relationship between features and selling price
# Numerical features vs selling price
fig, axes = plt.subplots(2, 3, figsize=(18, 10))
axes = axes.ravel()

for i, col in enumerate(num_cols):
    sns.scatterplot(data=data, x=col, y=np.exp(data['selling_price'
    axes[i].set_title(f"{col} vs Selling Price")
    axes[i].set_ylabel("Selling Price")

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

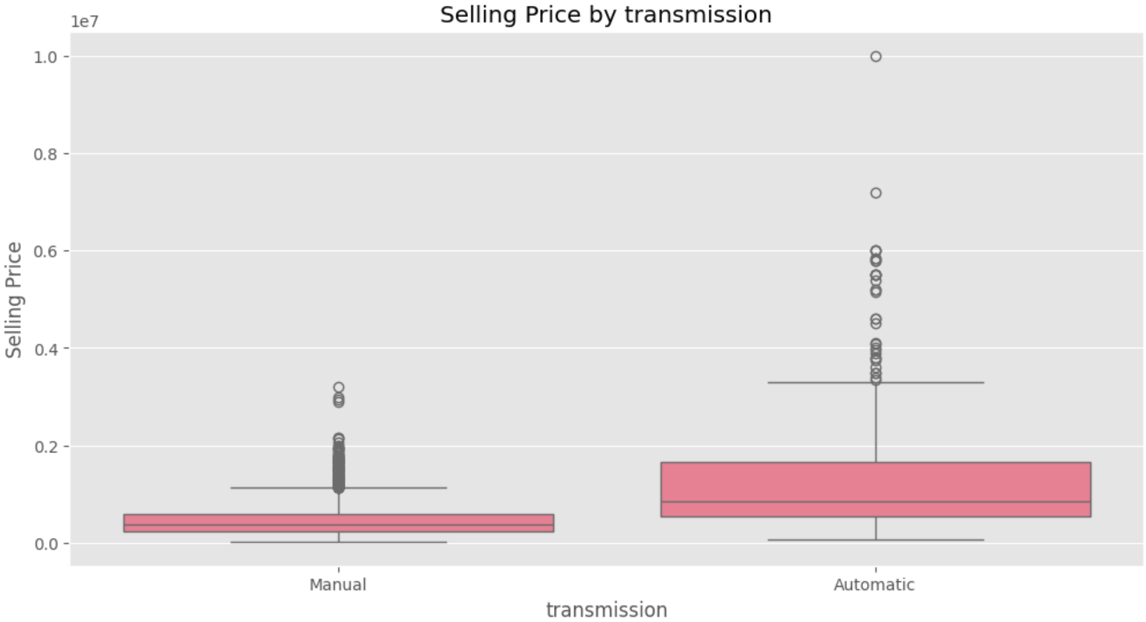


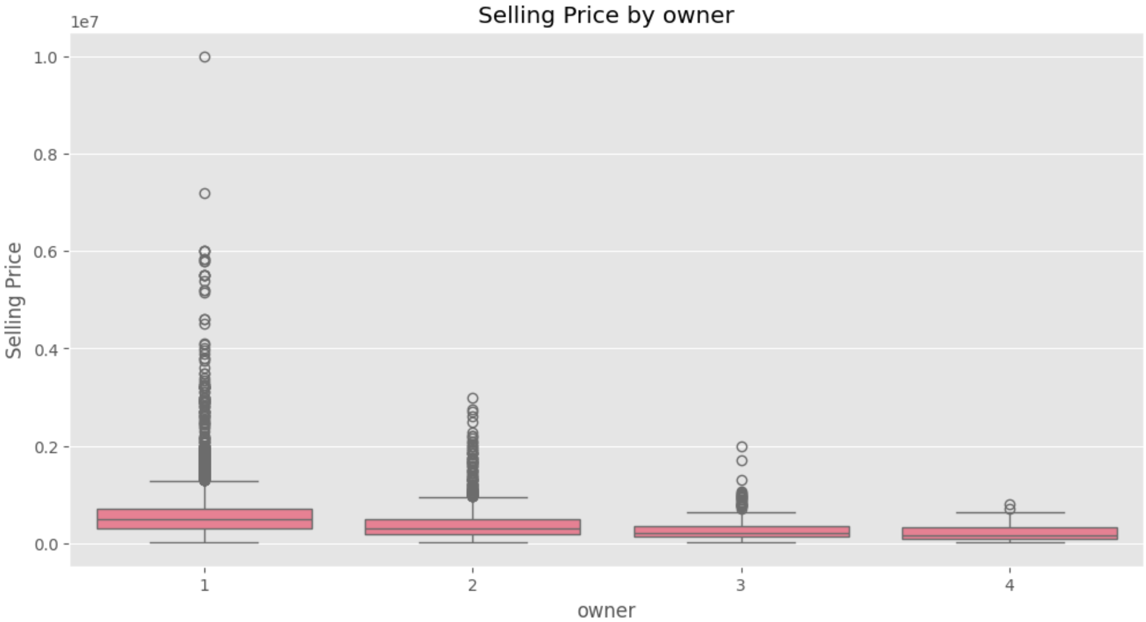

In [28]:
```python
# Categorical features vs selling price
```

```
for col in cat_cols:
    plt.figure(figsize=(12, 6))
    if col == 'brand':  # For brand, show only top 10
        top_brands = data['brand'].value_counts().nlargest(10).inde
        sns.boxplot(data=data[data['brand'].isin(top_brands)], x=co
        plt.xticks(rotation=45)
    else:
        sns.boxplot(data=data, x=col, y=np.exp(data['selling_price'
    plt.title(f"Selling Price by {col}")
    plt.ylabel("Selling Price")
    plt.show()
```



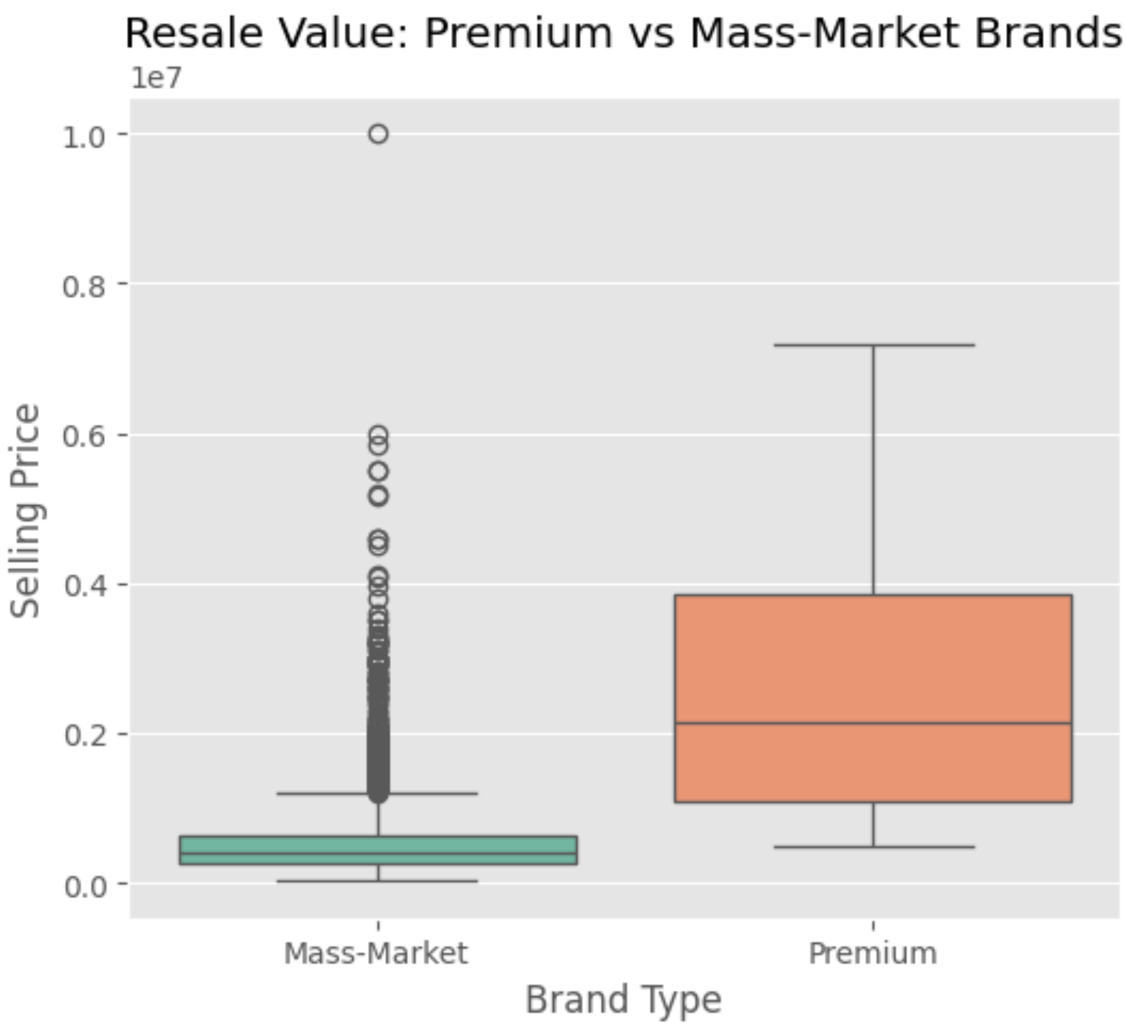




Selling Price by transmission
1e7
1.0
0.8
0.6
0.4
0.2
0.0
Selling Price
Manual
Automatic
transmission


Selling Price by owner
1e7
1.0
0.8
0.6
0.4
0.2
0.0
Selling Price
1
2
3
4
owner

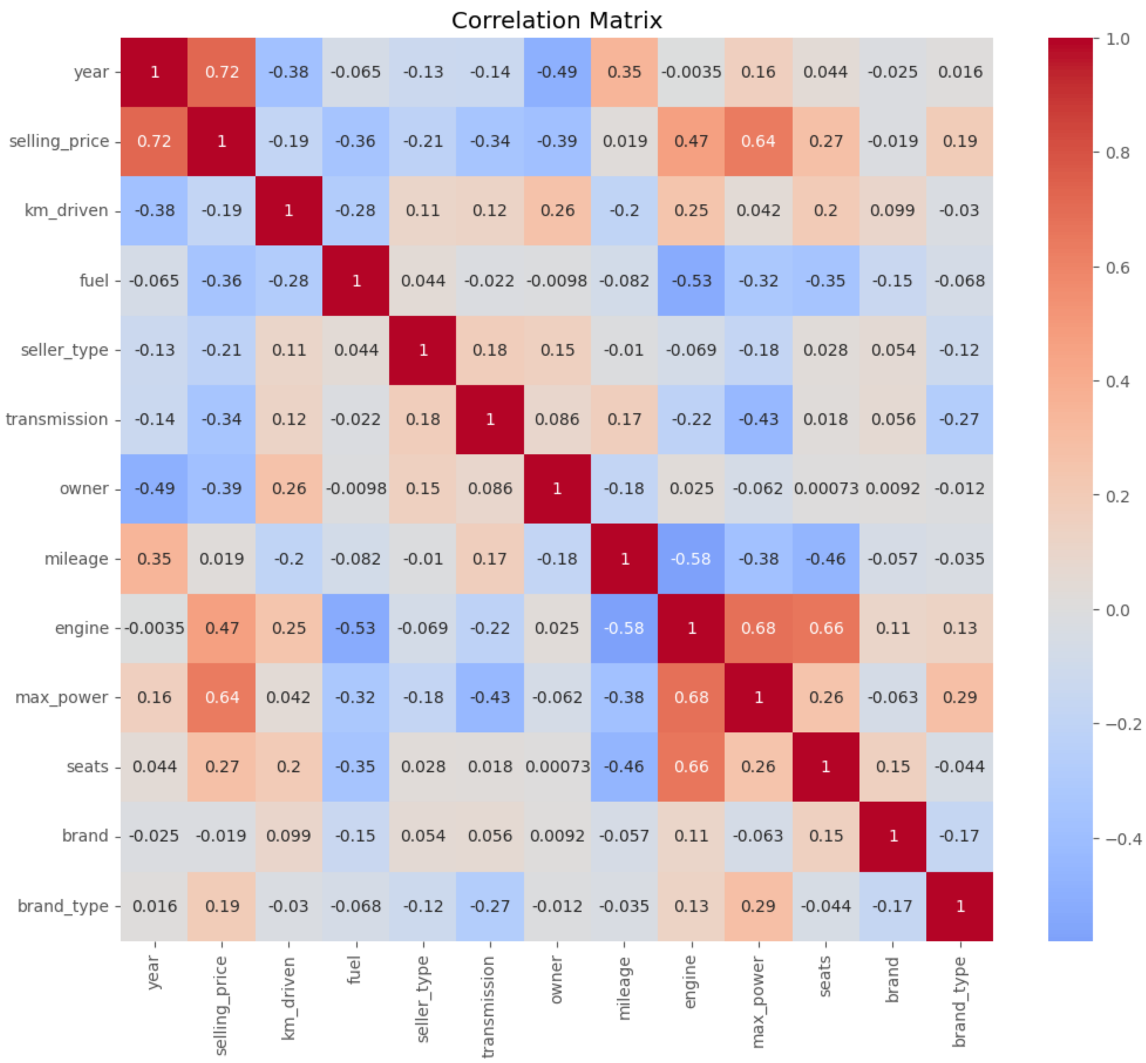

Selling Price by brand
1e6
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Selling Price
Maruti
Hyundai
Mahindra
Tata
Honda
Ford
Toyota
Chevrolet
Renault
Volkswagen
brand

In [29]:
```
# Focus only on premium vs non-premium
data["brand_type"] = data["brand"].apply(lambda x: "Premium" if x in

plt.figure(figsize=(6, 5))
sns.boxplot(
    data=data,
    x="brand_type",
    y=np.exp(data["selling_price"]),
    palette="Set2"
)

plt.title("Resale Value: Premium vs Mass-Market Brands")
plt.ylabel("Selling Price")
plt.xlabel("Brand Type")
plt.show()
```




In [30]:
```
# Correlation analysis
from sklearn.preprocessing import LabelEncoder
# Create a copy for correlation analysis
corr_data = data.copy()

# Encode categorical variables
cat_cols = corr_data.select_dtypes(exclude='number').columns
le_dict = {}
for col in cat_cols:
    le = LabelEncoder()
```

```
    corr_data[col] = le.fit_transform(corr_data[col].astype(str))
    le_dict[col] = le

# Calculate correlation matrix
corr_matrix = corr_data.corr()

# Plot correlation heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0)
plt.title("Correlation Matrix")
plt.show()
```




In [31]:

```
# Correlation with target
corr_with_target = corr_matrix['selling_price'].drop('selling_price
corr_sorted = corr_with_target.reindex(corr_with_target.abs().sort_

print("Feature correlations with selling price:")
print(corr_sorted)
```

```
Feature correlations with selling price:
year             0.718678
max_power        0.637513
engine           0.468379
owner           -0.389101
fuel            -0.356654
transmission    -0.343871
seats            0.273511
seller_type     -0.212444
brand_type       0.188333
km_driven       -0.185280
mileage          0.018881
brand           -0.018835
Name: selling_price, dtype: float64
```

In [32]:
```
# Plot feature importance based on correlation
plt.figure(figsize=(10, 8))
corr_sorted.abs().plot(kind='barh', color='skyblue')
plt.title("Feature Correlation with Selling Price (Absolute Values)'
plt.xlabel("Absolute Correlation Coefficient")
plt.show()
```

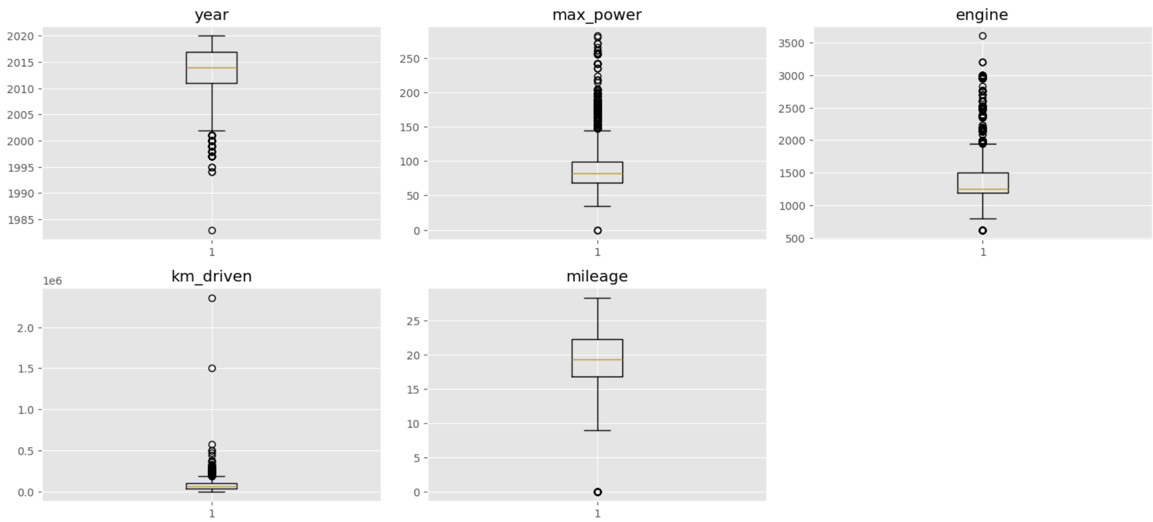

In [33]:
```
# Prepare features and target for modeling
feature_cols = ['year', 'max_power', 'engine', 'brand', 'km_driven'
X = data[feature_cols]
y = data["selling_price"]
```

In [34]:
```
# Encode brand column as others are already encoded
le = LabelEncoder()
X.loc[:, 'brand'] = le.fit_transform(X['brand'].astype(str))
```

In [35]:
```
# Split data into train and test sets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=

print(f"Training set shape: {X_train.shape}")
print(f"Testing set shape: {X_test.shape}")
```

```
Training set shape: (5461, 6)
Testing set shape: (1366, 6)
```

In [36]:
```
# Check for outliers in numerical features
num_cols = X_train.select_dtypes(include=['int64', 'float64']).colu

plt.figure(figsize=(15, 10))
for i, col in enumerate(num_cols):
    plt.subplot(3, 3, i+1)
    plt.boxplot(X_train[col])
    plt.title(col)
plt.tight_layout()
plt.show()
```




In [37]:
```
# Define function to count outliers
def outlier_count(col, data=X_train):
    q75, q25 = np.percentile(data[col], [75, 25])
    iqr = q75 - q25
    min_val = q25 - (iqr * 1.5)
    max_val = q75 + (iqr * 1.5)
    outlier_count = len(np.where((data[col] > max_val) | (data[col]
    outlier_percent = round(outlier_count / len(data[col]) * 100, 2

    if outlier_count > 0:
        print(f"{col}: {outlier_count} outliers ({outlier_percent}%

print("Outlier analysis:")
for col in num_cols:
    outlier_count(col)
```

```
Outlier analysis:
year: 61 outliers (1.12%)
max_power: 307 outliers (5.62%)
engine: 978 outliers (17.91%)
km_driven: 138 outliers (2.53%)
mileage: 15 outliers (0.27%)
```

In [38]:
```
# Scale features
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [39]:
```
# Train and evaluate multiple models
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
```

In [40]:
```
# Define models to evaluate
models = {
    "Linear Regression": LinearRegression(),
    "SVR": SVR(),
    "K-Neighbors Regressor": KNeighborsRegressor(),
    "Decision Tree": DecisionTreeRegressor(random_state=42),
    "Random Forest": RandomForestRegressor(random_state=42)
}
```

In [41]:
```
# Evaluate models using cross-validation and print mse and r2
print("Model evaluation using 5-fold cross-validation:")
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

results = {}
for name, model in models.items():
    cv_scores = cross_val_score(model, X_train_scaled, y_train, cv=
    results[name] = {
        'Mean MSE': -cv_scores.mean(),
        'Std MSE': cv_scores.std()
    }
    print(f"{name}: Mean MSE = {-cv_scores.mean():.4f} (±{cv_scores
```

```
Model evaluation using 5-fold cross-validation:
Linear Regression: Mean MSE = 0.1038 (±0.0052)  R² = 0.8239
SVR: Mean MSE = 0.0692 (±0.0038)  R² = 0.8713
K-Neighbors Regressor: Mean MSE = 0.0679 (±0.0038)  R² = 0.8831
Decision Tree: Mean MSE = 0.0938 (±0.0010)  R² = 0.8411
Random Forest: Mean MSE = 0.0573 (±0.0038)  R² = 0.9016
```

In [42]:
```
# Hyperparameter tuning for Random Forest
print("\nHyperparameter tuning for Random Forest...")
param_grid = {'bootstrap': [True], 'max_depth': [5, 10, 20, None],
```

```
               'n_estimators': [10, 11, 12, 13, 15, 20, 25,]}

rf = RandomForestRegressor(random_state=98)
grid = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    cv=kfold,
    n_jobs=-1,
    return_train_score=True,
    refit=True,
    scoring='neg_mean_squared_error'
)

grid.fit(X_train_scaled, y_train)

print(f"Best parameters: {grid.best_params_}")
print(f"Best CV score: {-grid.best_score_:.4f}")
```

```
Hyperparameter tuning for Random Forest...
Best parameters: {'bootstrap': True, 'max_depth': 10, 'n_estimator
s': 25}
Best CV score: 0.0572
```

In [43]:

```
# Evaluate on test set
best_model = grid.best_estimator_
y_pred = best_model.predict(X_test_scaled)
test_mse = mean_squared_error(y_test, y_pred)
test_r2 = r2_score(y_test, y_pred)

print(f"\nTest set performance:")
print(f"MSE: {test_mse:.4f}")
print(f"R² Score: {test_r2:.4f}")
```

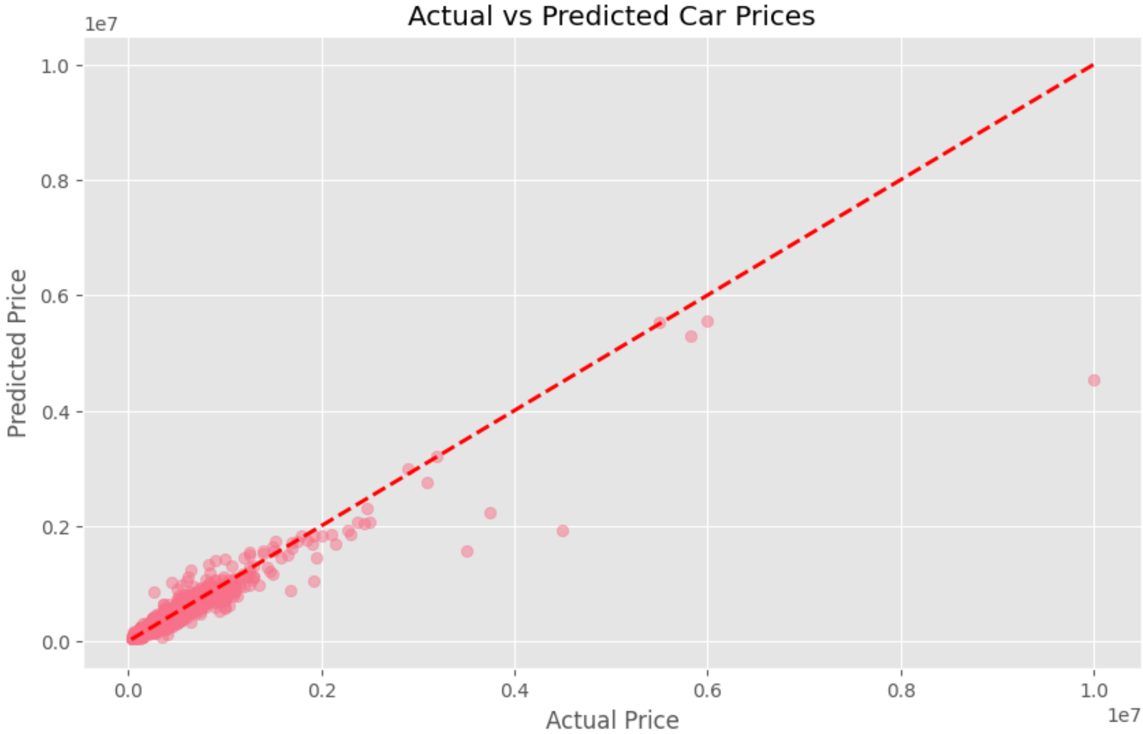
```
Test set performance:
MSE: 0.0534
R² Score: 0.9040
```

In [44]:

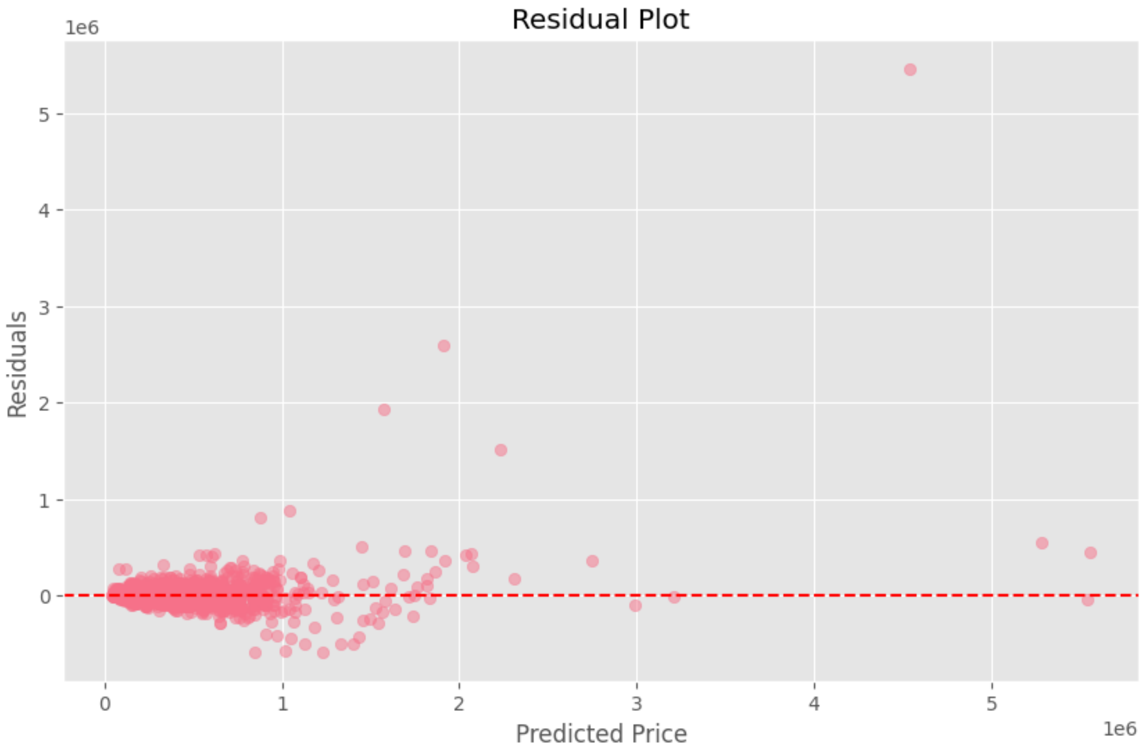
```
# Convert back to original scale for interpretation
y_test_orig = np.exp(y_test)
y_pred_orig = np.exp(y_pred)
```

In [45]:

```
# Plot actual vs predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test_orig, y_pred_orig, alpha=0.5)
plt.plot([y_test_orig.min(), y_test_orig.max()], [y_test_orig.min()
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual vs Predicted Car Prices")
plt.show()
```

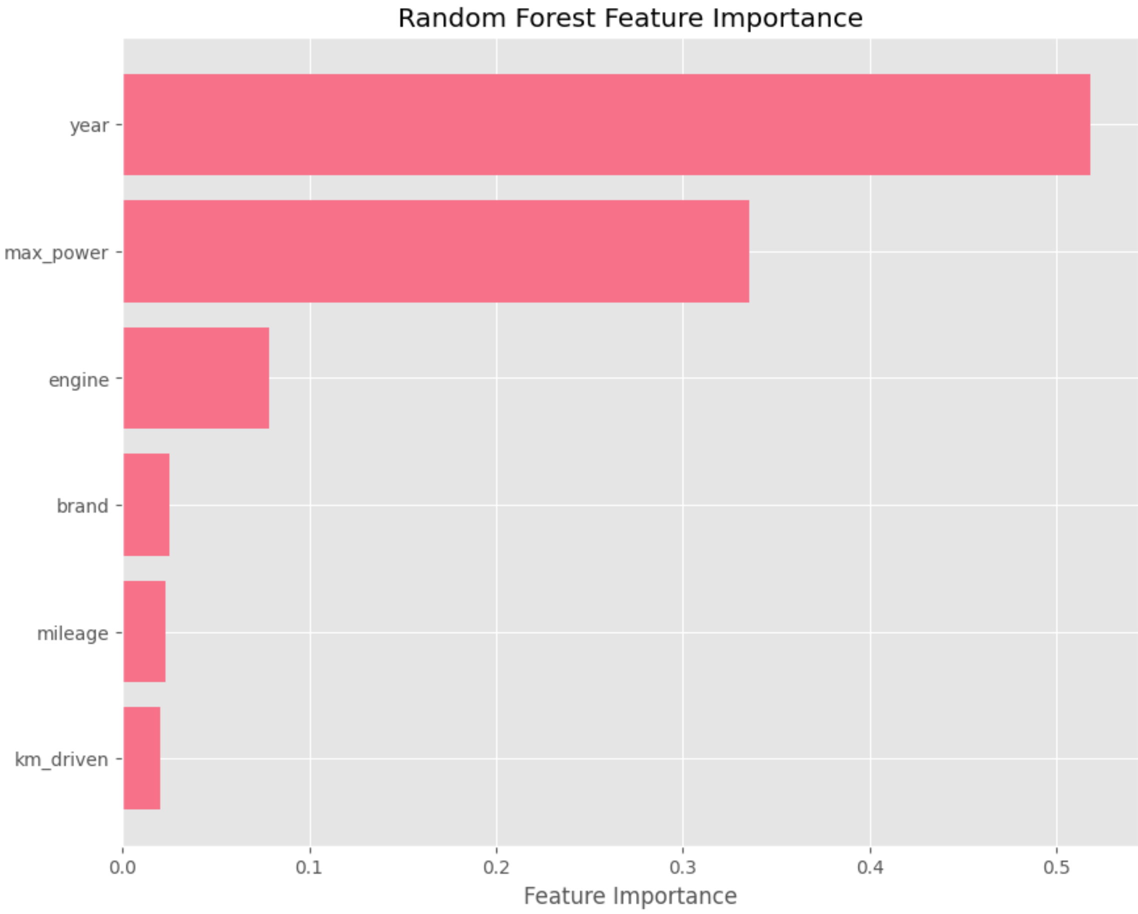



In [46]:
```
# Plot residuals
residuals = y_test_orig - y_pred_orig
plt.figure(figsize=(10, 6))
plt.scatter(y_pred_orig, residuals, alpha=0.5)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel("Predicted Price")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()
```



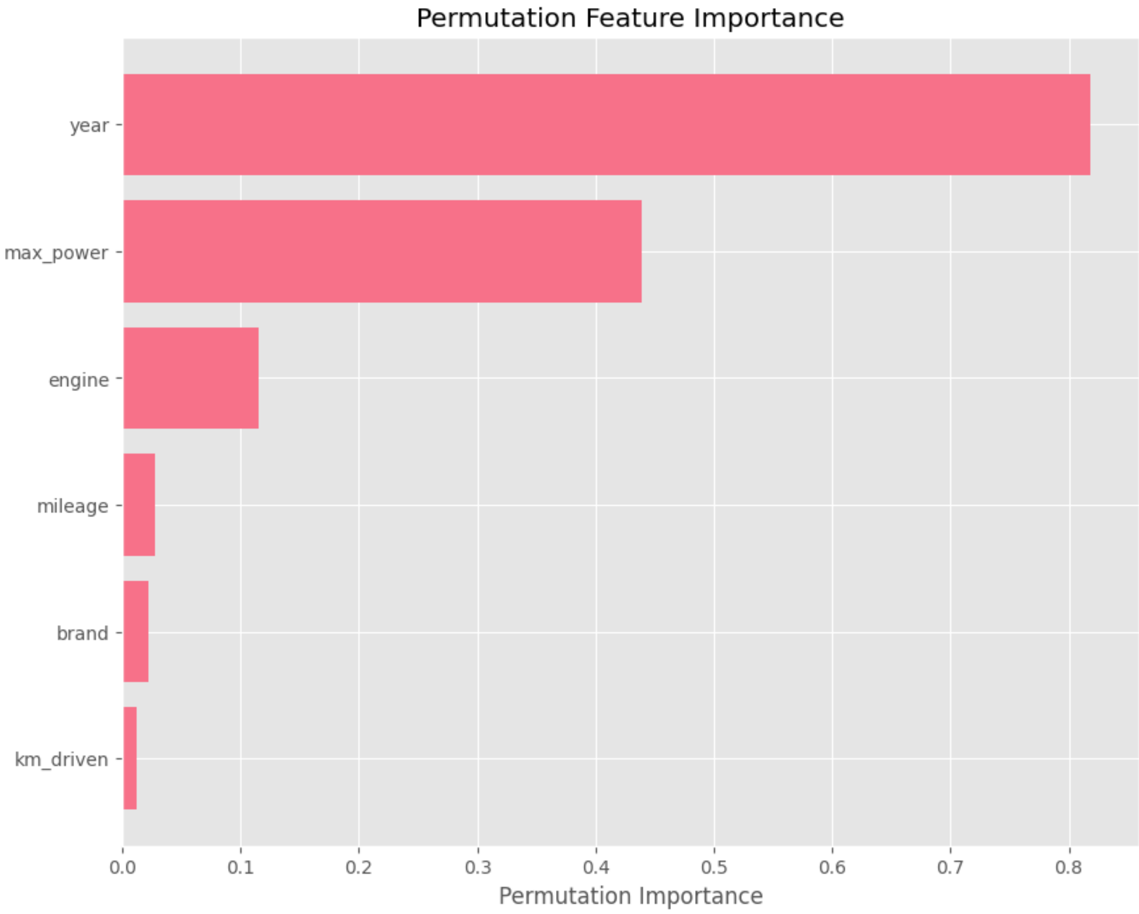

In [47]:
```
# Feature importance analysis
feature_importance = best_model.feature_importances_
```

```
sorted_idx = np.argsort(feature_importance)

plt.figure(figsize=(10, 8))
plt.barh(range(len(sorted_idx)), feature_importance[sorted_idx])
plt.yticks(range(len(sorted_idx)), [X.columns[i] for i in sorted_id
plt.xlabel("Feature Importance")
plt.title("Random Forest Feature Importance")
plt.show()
# numerical values of feature importance
for i in sorted_idx:
    print(f"{X.columns[i]}: {feature_importance[i]:.4f}")
```




```
km_driven: 0.0202
mileage: 0.0229
brand: 0.0250
engine: 0.0784
max_power: 0.3353
year: 0.5182
```

In [48]:
```
# Permutation importance
from sklearn.inspection import permutation_importance

perm_importance = permutation_importance(best_model, X_test_scaled,
sorted_idx = perm_importance.importances_mean.argsort()

plt.figure(figsize=(10, 8))
plt.barh(range(len(sorted_idx)), perm_importance.importances_mean[s
plt.yticks(range(len(sorted_idx)), [X.columns[i] for i in sorted_id
plt.xlabel("Permutation Importance")
plt.title("Permutation Feature Importance")
plt.show()
```




In [49]:
```
# Save the model and preprocessing objects for Flask app
model_data = {
    'model': best_model,
    'scaler': scaler,
    'le_dict': le_dict,
    'feature_cols': feature_cols,
    'feature_importance': feature_importance
}

with open('app/model/car_price_model.pkl', 'wb') as f:
    pickle.dump(model_data, f)

print("Model and preprocessing objects saved to 'car_price_model.pk
```

```
Model and preprocessing objects saved to 'car_price_model.pkl'
```

In [50]:
```
# Create a prediction function for demonstration
def predict_car_price(input_data, model_data):
    """
    Predict car price based on input features.

    Parameters:
    input_data (dict): Dictionary containing feature values
    model_data (dict): Contains model, scaler, label encoders, and 

    Returns:
    float: Predicted car price
    """
    model = model_data['model']
    scaler = model_data['scaler']
```

```python
    le_dict = model_data['le_dict']
    feature_cols = model_data['feature_cols']

    # Prepare input data in the correct order
    processed_data = []
    for col in feature_cols:
        if col in le_dict:  # Categorical feature
            # Handle unseen labels
            try:
                encoded_val = le_dict[col].transform([input_data[co
            except ValueError:
                # If label not seen during training, use the first 
                encoded_val = 0
            processed_data.append(encoded_val)
        else:  # Numerical feature
            processed_data.append(input_data[col])

    # Convert to array and scale
    sample = np.array(processed_data).reshape(1, -1)
    sample_scaled = scaler.transform(sample)

    # Predict and transform back from log scale
    pred_log_price = model.predict(sample_scaled)[0]
    pred_price = np.exp(pred_log_price)

    return pred_price
```

In [51]:

```python
# Example prediction
example_car = {
    'year': 2018,
    'km_driven': 35000,
    'engine': 1498,
    'max_power': 110,
    'brand': 'Honda',
    'mileage': 17.0
}

predicted_price = predict_car_price(example_car, model_data)
print(f"\nPredicted price for example car: ${predicted_price:,.2f}"
```

```
Predicted price for example car: $773,165.07
```

# 🚗 Car Price Prediction Analysis Report

## Project Overview

This project aimed to build a **car price prediction system** for **Chaky's company** using a dataset of **8,128 cars**.

Key features in the dataset included:

- `year`

- `km_driven`
- `fuel`
- `owner`
- `mileage`
- `engine`
- `max_power`
- `brand`

---

## Data Preparation

We performed several preprocessing steps:

- **Owner mapping** → (e.g., *First Owner → 1, Second Owner → 2*).
- **Removed CNG and LPG cars** (different mileage units: *km/kg vs kmpl*).
- **Cleaned numeric columns** → stripped `"kmpl"` from mileage and `"CC"` from engine.
- **Extracted brand** → first word from car name.
- **Dropped torque** (inconsistent units).
- **Removed Test Drive Cars** (unusually inflated prices).
- **Log-transformed `selling_price`** to stabilize variation (range: 29,999 → 10,000,000).

---

## 🔍 Exploratory Data Analysis (EDA)

To guide feature selection, we performed both **correlation analysis** and evaluated **Random Forest feature importance**. These complementary approaches allowed us to capture both linear relationships and more complex, non-linear contributions of features to car price prediction. It is important to note that correlation does not imply causation — some features may interact or overlap, so their effects should be interpreted carefully.

From this analysis, we selected **six key features** that provide strong predictive power while remaining interpretable. **Year** is highly correlated with price (0.718) and has the highest feature importance (~0.518), reflecting the premium placed on newer cars. **Max_power** (0.637 correlation, ~0.335 importance) captures the value associated with performance-oriented or premium models, while **Engine** size (0.468 correlation, ~0.078 importance) distinguishes larger, more luxurious or powerful vehicles. **Km_driven** shows a weak negative correlation (-0.185, ~0.020 importance), indicating that higher mileage slightly reduces price. **Mileage** (0.152 correlation, ~0.043 importance) now contributes positively, showing that fuel-efficient cars are slightly more valued. **Brand**, although minimally correlated (-0.018) with a small importance (~0.025), contributes subtly through perceived prestige. Several features were excluded: **Owner**, **Fuel Type**, **Transmission**, and **Seats**, either overlapping with the selected features or showing low variability. Removing these did not reduce model performance ($R^2$ remained ~0.90), confirming that the six selected features capture the most relevant information for predicting car prices.

---

## ✅ Selected Features

We chose **6 features** for predictive power and interpretability:

| Feature | Correlation | Importance | Rationale |
|---|---|---|---|
| **Year** | **0.718** | ~0.518 | Newer cars command higher prices due to less wear and modern tech. |
| **Max_power** | **0.637** | ~0.335 | Stronger engines → premium/sport models. |
| **Engine** | **0.468** | ~0.078 | Bigger engines → luxury/performance cars. |
| **Km_driven** | **-0.185** | ~0.020 | More mileage reduces price, but weaker than expected. |
| **Mileage** | **0.018** | ~0.022 | Higher efficiency slightly boosts value. |
| **Brand** | **-0.018** | ~0.025 | Premium perception boosts price (e.g., BMW vs Maruti). |

### Examples:

- **Year**: 2018 Honda City → 925,000 vs 2006 Honda City → 158,000.
- **Max_power**: Jeep Compass (160.77 bhp, 2,100,000) vs Maruti Alto (47.3 bhp, 275,000).
- **Engine**: Toyota Fortuner (2982 CC, 1,500,000) vs Maruti 800 (796 CC, 45,000).
- **Km_driven**: Swift (145,500 km → 450,000) vs Swift (35,000 km → 675,000).
- **Mileage**: Maruti Swift (23 kmpl → 675,000) vs older hatchback (18 kmpl → 420,000).
- **Brand**: Mercedes-Benz B Class → 1,450,000 vs Maruti Alto → 275,000.

---

## ❌ Skipped Features

| Feature | Correlation | Reason for Exclusion |
|---|---|---|

| | | |
|---|---|---|
| **Owner** | -0.389 | Dropping didn't reduce $R^2$ (0.90). Effect captured by `year` + `km_driven` . |
| **Fuel Type** | -0.356 | Correlates with `engine` size. Adds redundancy. |
| **Transmission** | -0.343 | Overlaps with `max_power` and `engine` . |
| **Seats** | 0.273 | Low variation (mostly 5 seats). Limited influence. |

---

## 📊 Model Comparison

We tested multiple ML models with the selected 6 features:

| Model | $R^2$ Score |
|---|---|
| **Random Forest** | **0.9016** |
| Decision Tree | 0.8411 |
| Linear Regression | 0.8239 |
| K-Neighbors (KNN) | 0.8831 |
| Support Vector Regr. | 0.8713 |

---

## 🚀 Conclusion

This first AI project successfully demonstrates how **machine learning can estimate car prices** using a small, impactful feature set.

For Chaky's company, this system now accounts for **fuel efficiency** through mileage, offering:

- **Ease of data collection**
- **Strong predictive power**
- **Simple web-app integration**