

Car Price Prediction Project Report

1. Project Objective

The objective of this project is to predict the selling price of used cars based on various factors such as manufacturing year, brand, mileage, engine power, and other key features. The project applies machine learning algorithms to analyze car data and build models that estimate accurate resale prices.

2. Dataset Information

Column Name	Description
name	Full name of the car model (includes brand and variant). Example: Maruti Swift Dzire VDI. Used to extract the brand.
year	Year of manufacture. Newer cars tend to have higher resale value.
selling_price	The target variable — actual selling price of the car (in INR).
km_driven	Total distance driven in kilometers; higher values reduce price.
fuel	Type of fuel — Petrol, Diesel, CNG, LPG, or Electric.
seller_type	Type of seller — Individual, Dealer, or Trustmark Dealer.
transmission	Transmission type — Manual or Automatic.
owner	Ownership count — First Owner, Second Owner, etc.
mileage(km/ltr/kg)	Fuel efficiency of the car in km per liter or km per kg.
engine	Engine capacity in cubic centimeters (CC).
max_power	Maximum power produced by the engine (in

bhp).

seats	Number of seats in the car.
brand	Derived feature from 'name', representing the car's brand.

The dataset used in this project is obtained from CarDekho, a well-known car marketplace website. It contains details of cars sold in India, along with their technical specifications and ownership information.

Dataset Name: cardekho.csv

Total Rows: ~6000

Total Columns: 12

Column Descriptions

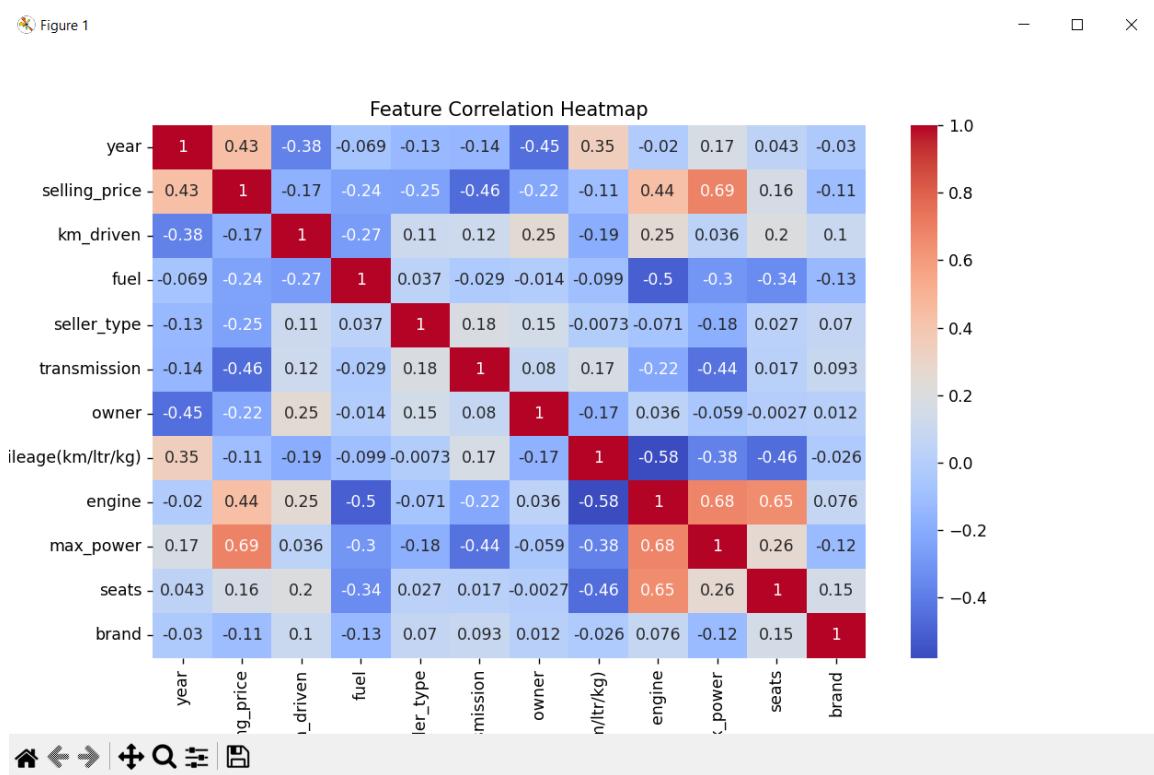
ats	name	year	selling_price	km_driven	fuel	...	owner	mileage(km/ltr/kg)	engine	max_power	se
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	...	First Owner	23.40	1248.0	74	5.0
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	...	Second Owner	21.14	1498.0	103.52	5.0
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	...	Third Owner	17.70	1497.0	78	5.0
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	...	First Owner	23.00	1396.0	90	5.0

3. Feature Impact on Selling Price

Feature	Effect on Selling Price
year	Newer cars generally have higher resale value.
km_driven	More kilometers driven → lower price.
fuel	Petrol cars retain better resale value than diesel or CNG in many cases.
seller_type	Dealer cars may have slightly higher price due to warranty/service.
transmission	Automatic cars often priced higher than manual ones.
owner	More previous owners → lower price.

mileage(km/ltr/kg)	Better mileage increases resale value.
engine	Higher engine capacity increases performance and price.
max_power	Higher power usually means higher price.
seats	Slightly affects price depending on car type (SUV, Sedan, etc.).
brand	Premium brands have higher resale prices.

Figure 1



4. Data Cleaning

Extracted brand name from name column.

Converted max_power values to numeric (float).

Changed year column to integer type.

Dropped unnecessary name column.

```
""
# ✎ STEP 2: Data Cleaning
# =====

# Extract brand name from 'name' column
df['brand'] = df['name'].str.split().str.get(0)

# Convert 'max_power' to float
df['max_power'] = df['max_power'].str.split().str.get(0).astype('float')

# Ensure 'year' is integer
df['year'] = df['year'].astype('int')

# Drop unnecessary column
df.drop(columns=['name'], inplace=True)

print("\nAfter Cleaning:")
print(df.info())
```

```
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
 --- 
 0   year              8128 non-null    int64  
 1   selling_price     8128 non-null    int64  
 2   km_driven         8128 non-null    int64  
 3   fuel               8128 non-null    object  
 4   seller_type        8128 non-null    object  
 5   transmission       8128 non-null    object  
 6   owner              8128 non-null    object  
 7   mileage(km/ltr/kg) 7907 non-null    float64 
 8   engine             7907 non-null    float64 
 9   max_power          7912 non-null    float64 
 10  seats              7907 non-null    float64 
 11  brand              8128 non-null    object  
 dtypes: float64(4), int64(3), object(5)
 memory usage: 762.1+ KB
None
```

5. Handling Missing Values

Missing values were filled using mean or mode strategies:

mileage(km/ltr/kg) → Mean

engine → Mean

seats → Mode

max_power → Mode

```
# =====
# ✅ STEP 3: Handling Missing Values
# =====

print("\nMissing Values Before:")
print(df.isnull().sum())

df['mileage(km/ltr/kg)'] = df['mileage(km/ltr/kg)'].fillna(df['mileage(km/ltr/kg)'].mean())
df['engine'] = df['engine'].fillna(df['engine'].mean())
df['seats'] = df['seats'].fillna(df['seats'].mode()[0])
df['max_power'] = df['max_power'].fillna(df['max_power'].mode()[0])
```

Missing Values Before:	Missing Values After:
year 0	year 0
selling_price 0	selling_price 0
km_driven 0	km_driven 0
fuel 0	fuel 0
seller_type 0	seller_type 0
transmission 0	transmission 0
owner 0	owner 0
mileage(km/ltr/kg) 221	mileage(km/ltr/kg) 0
engine 221	engine 0
max_power 216	max_power 0
seats 221	seats 0
brand 0	brand 0
dtype: int64	dtype: int64

6. Removing Duplicates

Duplicate rows were identified and dropped to maintain data integrity.

```
# =====
# ⚡ STEP 4: Remove Duplicates
# =====
print("\nDuplicate Rows Before:", df.duplicated().sum())
df.drop_duplicates(inplace=True)
print("Duplicate Rows After:", df.duplicated().sum())
```

```
Duplicate Rows Before: 1221
Duplicate Rows After: 0
```

7. Encoding Categorical Data

Categorical features (fuel, seller_type, transmission, owner, and brand) were converted into numeric form using Label Encoding.

```
# ⚡ STEP 5: Encoding Categorical Variables
# =====
le = LabelEncoder()
cat_columns = ['fuel', 'seller_type', 'transmission', 'owner', 'brand']

for col in cat_columns:
    df[col] = le.fit_transform(df[col])

print("\nAfter Label Encoding:")
print(df.info())
```

8. Reducing Noise (Grouping Rare Brands)

Car brands with fewer than 30 occurrences were grouped under label 0, representing rare brands, to prevent overfitting and improve model stability.

```
# =====
# 🚗 STEP 6: Reducing Noise (Grouping Rare Brands)
# =====
brands_to_remove = df['brand'].value_counts()[df['brand'].value_counts() < 30].index.tolist()
df['brand'] = df['brand'].apply(lambda x: 0 if x in brands_to_remove else x) # 0 → represents rare brands
```

```
print("\nTop Brand Frequencies:")
print(df['brand'].value_counts().head())

# =====
```

Top Brand Frequencies:

brand	count
20	2158
11	1263
19	719
28	646
10	362

Name: count, dtype: int64

9. Feature Scaling

All numerical features were scaled between 0 and 1 using MinMaxScaler to bring them to the same range and help models converge faster.

```
Data After Normalization:
   year selling_price km_driven      fuel seller_type ... mileage(km/ltr/kg)   engine max_power seats   brand
0  0.837838        450000  0.061640  0.333333       0.5 ...      0.557143  0.209396  0.1850  0.25  0.666667
1  0.837838        370000  0.050837  0.333333       0.5 ...      0.503333  0.293289  0.2588  0.25  0.900000
2  0.621622        158000  0.059310  1.000000       0.5 ...      0.421429  0.292953  0.1950  0.25  0.333333
3  0.729730        225000  0.053803  0.333333       0.5 ...      0.547619  0.259060  0.2250  0.25  0.366667
4  0.648649        130000  0.050837  1.000000       0.5 ...      0.383333  0.226174  0.2205  0.25  0.666667
```

[5 rows x 12 columns]

10. Train-Test Split

The dataset was split into:

Training Set: 80%

Testing Set: 20%

This helps evaluate the model's generalization ability.

```
# =====
# 🧠 STEP 8: Splitting the Dataset
#
x = df.drop(columns=['selling_price'])
y = df['selling_price']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42, shuffle=True)
```

11. Model Training and Evaluation

Five regression models were trained and tested:

Linear Regression

Decision Tree Regressor

Random Forest Regressor

Gradient Boosting Regressor

XGBoost Regressor

Evaluation Metrics

R² Score

Mean Absolute Error (MAE)

Mean Squared Error (MSE)

Root Mean Squared Error (RMSE)

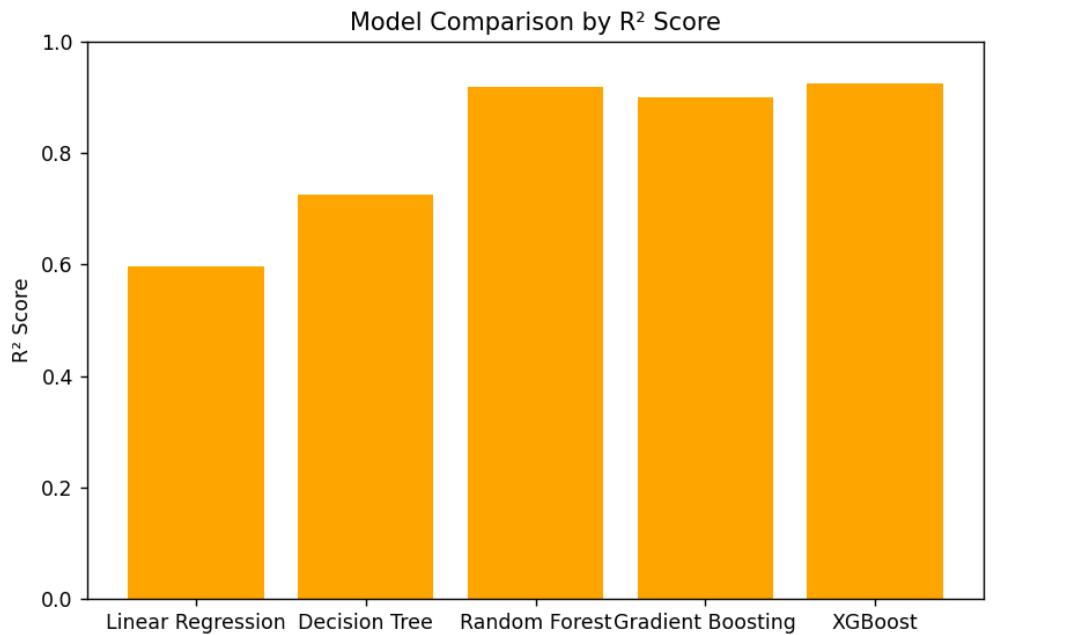
```
↳ Model Performance Results:
Linear Regression: {'R² Score': 0.5959264105839471, 'RMSE': 298999.7691912058, 'MAE': 173334.6937713843, 'MSE': 89400861976.39433}
Decision Tree: {'R² Score': 0.7263296323916257, 'RMSE': 246667.64230960264, 'MAE': 166191.70264422282, 'MSE': 60549284591.80654}
Random Forest: {'R² Score': 0.9194225939384688, 'RMSE': 133520.28835053823, 'MAE': 73808.12969488118, 'MSE': 17827667401.210873}
Gradient Boosting: {'R² Score': 0.8992979962659375, 'RMSE': 149265.58121114044, 'MAE': 84263.29141077785, 'MSE': 22280213734.299564}

XGBoost: {'R² Score': 0.9254964590072632, 'RMSE': 128389.3671875, 'MAE': 72072.1953125, 'MSE': 16483829760.0}
```

12. Model Comparison

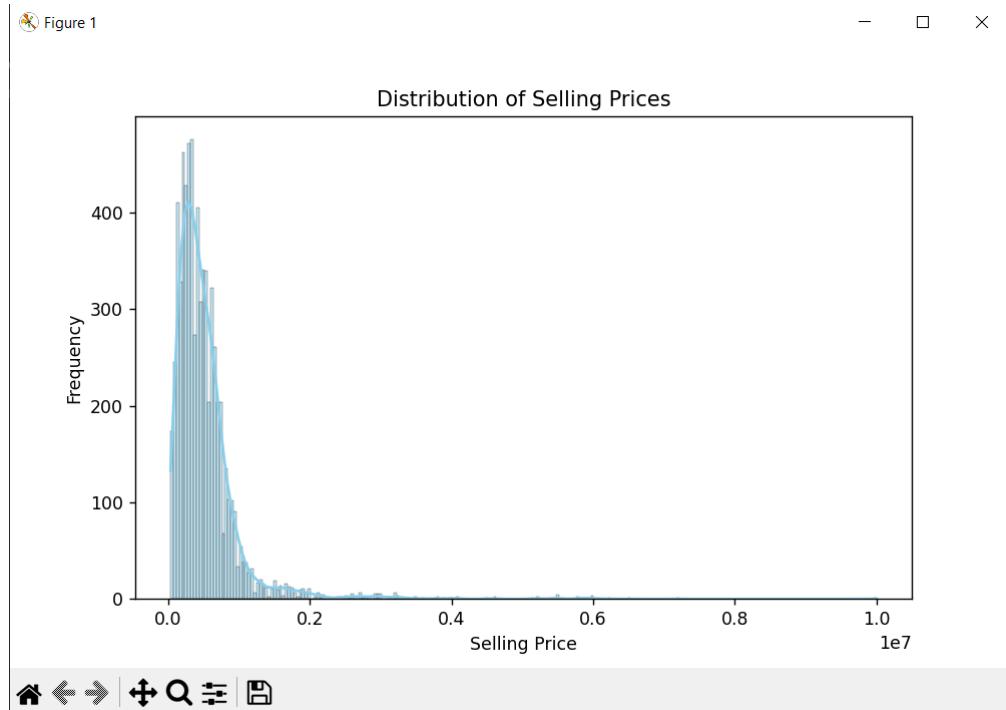
A bar chart was created to compare R² Scores of all models.

Figure 1



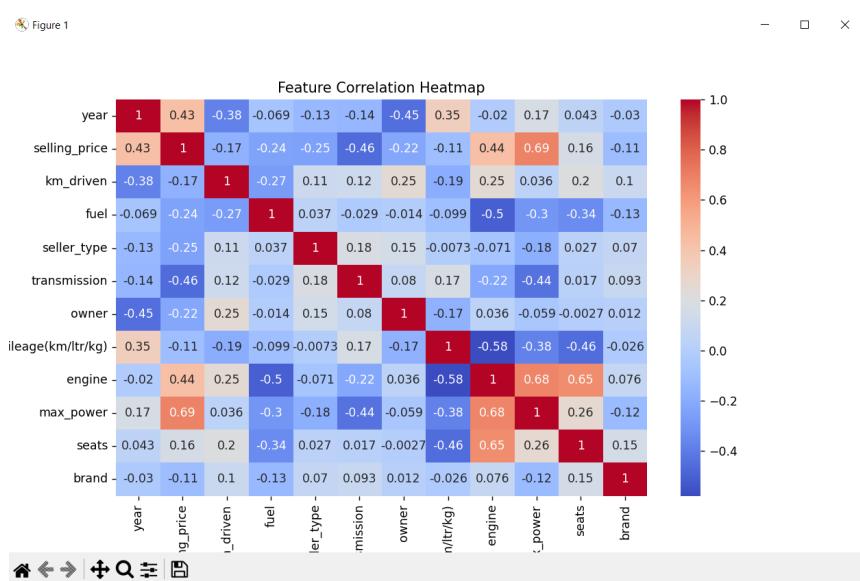
13. Selling Price Distribution

A histogram with KDE curve shows how selling prices are distributed in the dataset.



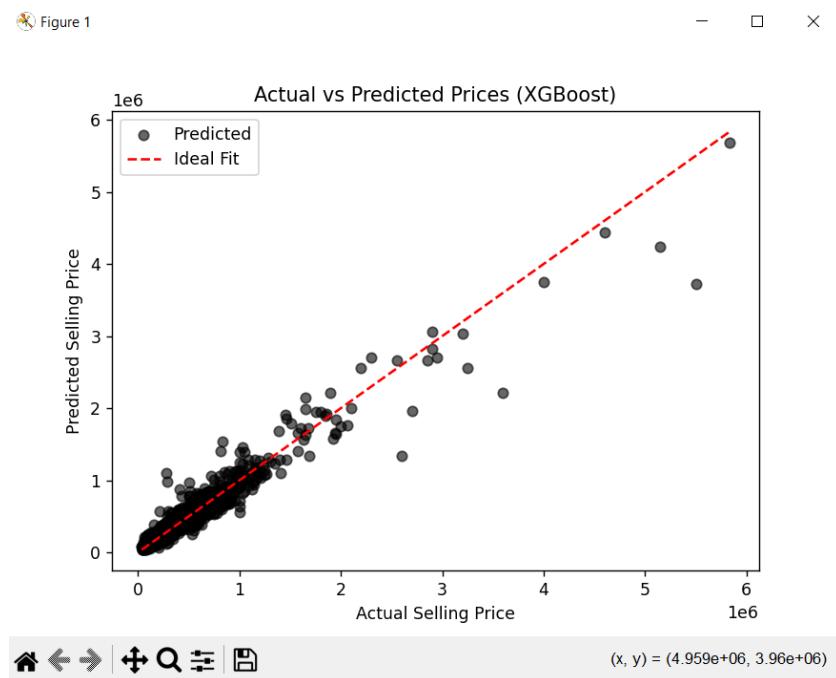
14. Correlation Heatmap

A heatmap displays the correlation among all numerical features, helping visualize how strongly each variable relates to selling price.



15. Actual vs Predicted (XGBoost)

A scatter plot compares actual vs predicted prices using the XGBoost model.
A red diagonal line indicates the ideal prediction line.



16. Conclusion

The XGBoost and Random Forest models achieved the highest accuracy and R^2 score, making them the best-performing models for this dataset.
Data cleaning, encoding, and feature scaling significantly improved model performance.
The project successfully demonstrates how machine learning can predict used car prices with high accuracy using real-world data.

17. Future Enhancements

Add more real-world features like car condition, location, and service history.
Perform hyperparameter tuning for optimized results.
Deploy the model using Flask or Streamlit for user interaction.
Integrate the system with a live dataset API for real-time predictions.