CSE422: Artificial Intelligence

Project Report

Project Title : Car price prediction

Group: 2, Lab Section : 12, Fall 2024	
ID	**Name**
21301611	Tasnia Jannat Ayesha
22101137	Tanjila Fariha

Table of Contents

Introduction: Car price prediction

This project focuses on predicting the selling prices of cars using multiple machine learning models. The primary goal is to create a system that accurately estimates car prices based on key features like make, model, year of manufacture, mileage, fuel type, transmission, and ownership details. The problem it seeks to address is the difficulty in determining a fair and accurate price for used cars, which often varies significantly due to subjective judgment and market fluctuations.

The used car market is growing rapidly, and buyers and sellers frequently face challenges in arriving at a mutually agreeable price. Traditional valuation methods are either too simplistic or rely heavily on expert opinions, which can lead to inconsistencies and leave room for negotiation disputes. By using a data-driven approach, this project aims to provide more consistent and reliable price estimates, reducing guesswork and improving decision-making for all parties involved.

The motivation for undertaking this project stems from the increasing importance of transparency and efficiency in the automotive sector. Accurate price predictions can benefit individuals looking to buy or sell used cars and can also assist businesses such as car dealerships and online platforms in streamlining their operations. By applying advanced regression models and carefully preparing the data, this project aspires to deliver a scalable solution that addresses a real-world need effectively.

Dataset description

Source: Kaggle

Link: https://www.kaggle.com/datasets/jacksondivakarr/sample34

Reference: Jackson Divakar

Dataset description

1. **How many features:**

There are 16 features initially, such as: unnamed, name, year, selling_price, km_driven, fuel, seller_type, transmission, owner, seats, max_power (in bph), Mileage Unit, Mileage, Engine (CC), area, country.

2. **Is this a classification or regression problem ? Why do you think so?**

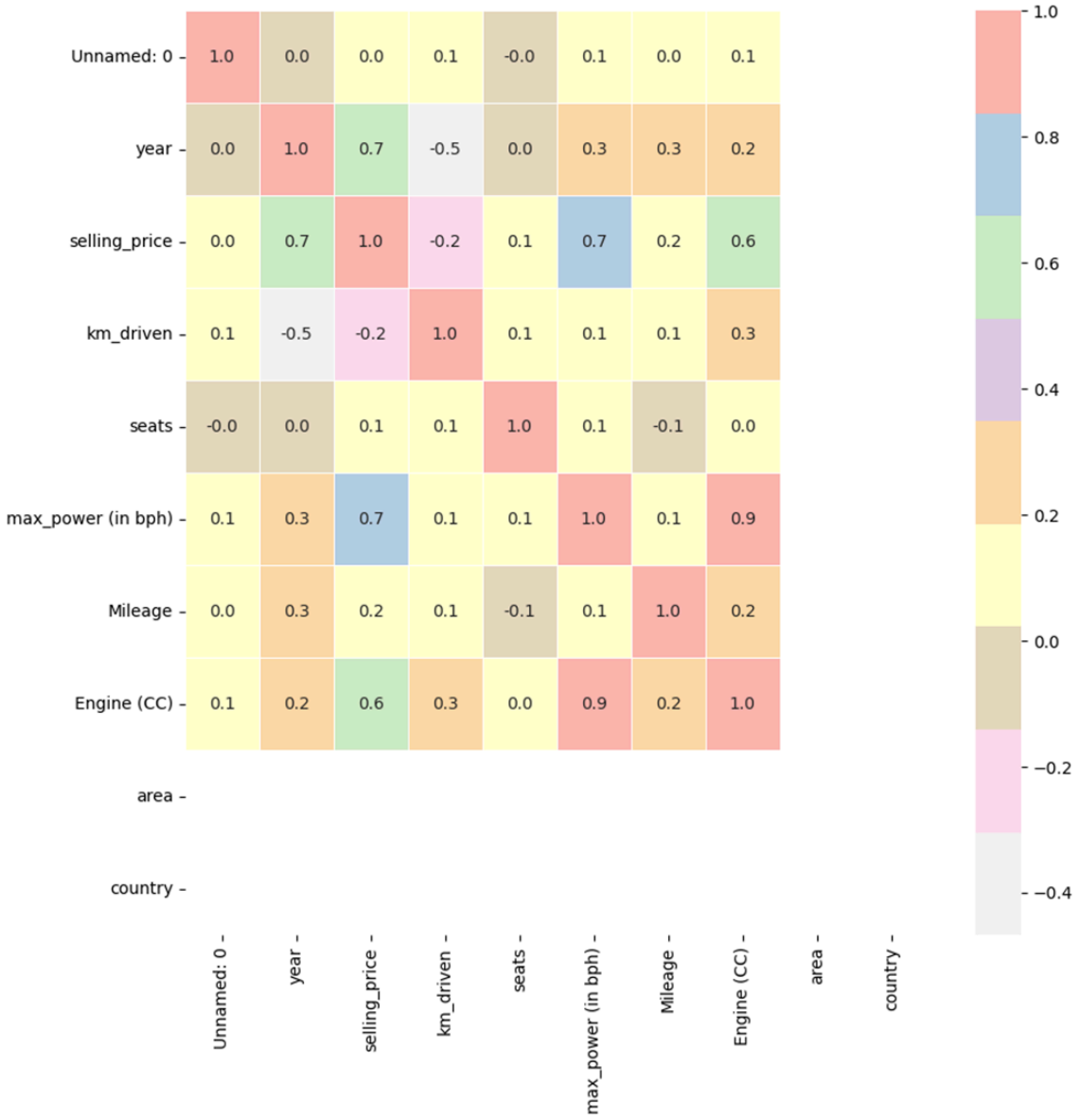
In our project, we are tackling a regression problem. Regression tasks involve predicting a continuous numeric value based on input features. Specifically, in this case, we aim to predict the price of a car based on various factors such as its brand, model, age, mileage, and other relevant features. Unlike classification, where the goal is to predict a category or class, regression deals with estimating a precise value. Therefore, the goal of our project is to predict the price of a car, which is a continuous numerical value, making it a classic regression problem.

3. **Data points:** 2095

4. **What kind of features are in your dataset? Quantitative or categorical?**

The dataset we are working with contains a combination of quantitative and categorical features, each providing unique insights into the factors that influence car prices. The quantitative features, such as selling price, mileage, and engine capacity, directly impact the car's value and are crucial for prediction. Meanwhile, the categorical features, such as brand, fuel type, transmission, and ownership status, categorize the cars and offer valuable context for understanding pricing trends based on specific characteristics. This blend of data types allows us to create a more accurate prediction model for car prices.

5. Correlation of all features:



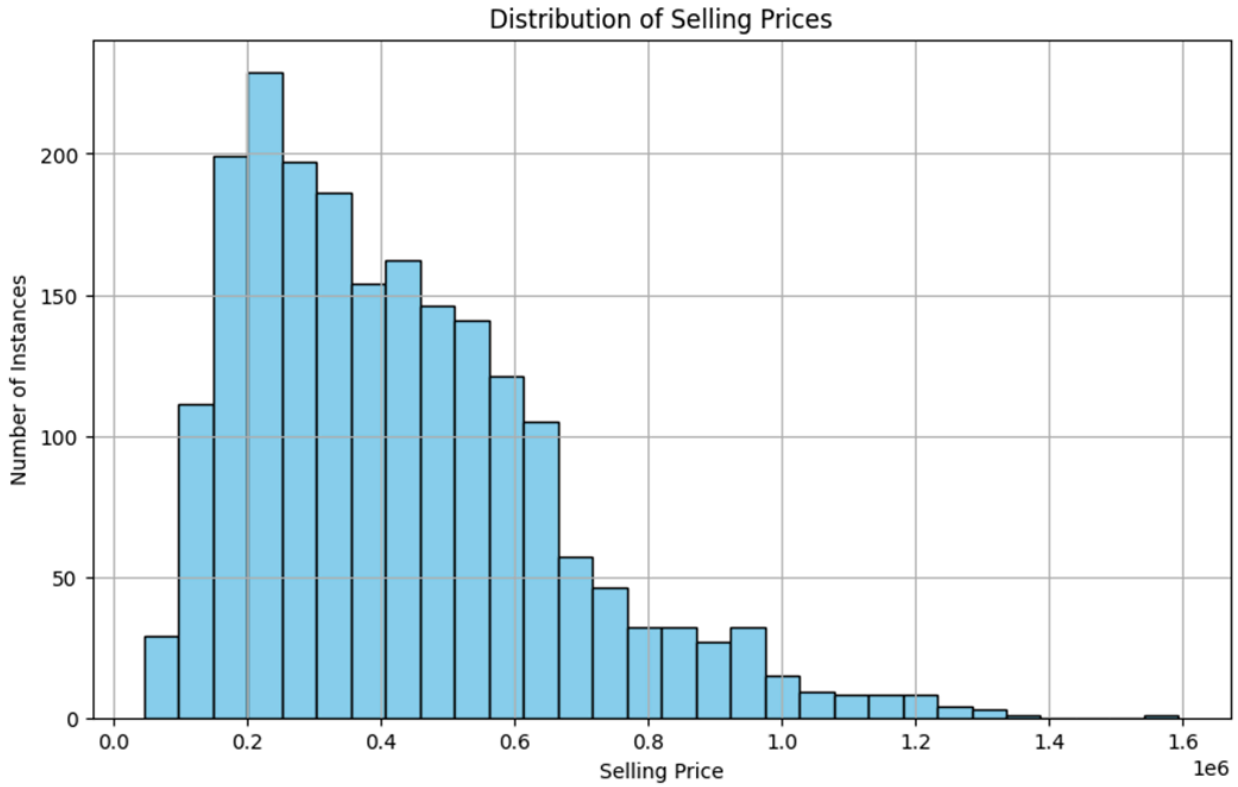
A correlation matrix is a table that shows the correlation coefficients between multiple features. It's a way to see how strongly different features in the dataset are related to each other. A correlation coefficient close to 1 means a strong positive relationship, close to -1 means a strong negative relationship, and close to 0 means little or no relationship.

Imbalanced Dataset

1. For the output feature, do all unique classes have an equal number of instances or not?

- No

2. Distribution of selling price:



Dataset Preprocessing

Faults: Null values

```
1 car.isnull().sum()
```

	0
Unnamed: 0	0
name	0
year	0
selling_price	0
km_driven	6
fuel	0
seller_type	12
transmission	0
owner	0
seats	0
max_power (in bph)	0
Mileage Unit	0
Mileage	0
Engine (CC)	0
area	2095
country	2095

Drop null values rows:

```
1
2 print("Number of rows with null values in seller_type column: ", car['seller_type'].isnull().sum())
3
4
5 car_subset = car[car['seller_type'].notnull()]
6
7 print("Shape after removing null values: ", car_subset.shape)
```

```
Number of rows with null values in seller_type column:  12
Shape after removing null values:  (2083, 14)
```

Drop null value column:

```
1 car = car.drop(['area', 'country'], axis = 1)
2 car.shape
```

```
(2095, 14)
```

Impute values:

```
1
2 nan_count_before = car['km_driven'].isna().sum()
3 print(f"Number of NaN values before filling: {nan_count_before}")
4
5
6 avg_km_driven = car['km_driven'].mean()
7
8 car['km_driven'] = car['km_driven'].fillna(avg_km_driven)
9
10 nan_count_after = car['km_driven'].isna().sum()
11 print(f"Number of NaN values after filling: {nan_count_after}")
12
13 car.head()
14
15
```

```
Number of NaN values before filling: 6
Number of NaN values after filling: 0
```

Fault: Categorical values:

```
1 car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2063 entries, 0 to 2094
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   name                2063 non-null   object
 1   year                2063 non-null   int64
 2   selling_price       2063 non-null   int64
 3   km_driven           2063 non-null   float64
 4   fuel                2063 non-null   object
 5   seller_type         2063 non-null   object
 6   transmission        2063 non-null   object
 7   owner               2063 non-null   object
 8   seats               2063 non-null   int64
 9   max_power (in bph)  2063 non-null   float64
 10  Mileage             2063 non-null   float64
 11  Engine (CC)         2063 non-null   int64
dtypes: float64(3), int64(4), object(5)
```

The object types are qualitative data while the float and int type data are quantitative.

Encoding

We need to replace the categorical data with numerical data so that the computer can understand and notice a pattern so that it can make a prediction. We do this via:

1. **One hot encoding:**

```
## feature encoding
# Transform the category_desc column
name_enc = pd.get_dummies(car['name'],dtype='int')
owner_enc = pd.get_dummies(car['owner'],dtype='int')
car = pd.concat([car, name_enc], axis=1)
car = pd.concat([car, owner_enc], axis=1)
```

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	seats	max_power (in bph)	...	Toyota	Volkswagen	First Owner	Fourth & Above Owner	Second Owner	Test Drive Car	Third Owner
0	Maruti	2014	450000	145500.0	Diesel	Individual	Manual	First Owner	5	74.00	...	0	0	1	0	0	0	0
1	Hyundai	2010	225000	127000.0	Diesel	Individual	Manual	First Owner	5	90.00	...	0	0	1	0	0	0	0
2	Hyundai	2017	440000	45000.0	Petrol	Individual	Manual	First Owner	5	81.86	...	0	0	1	0	0	0	0
3	Toyota	2011	350000	90000.0	Diesel	Individual	Manual	First Owner	5	67.10	...	1	0	1	0	0	0	0
4	Ford	2013	200000	169000.0	Diesel	Individual	Manual	First Owner	5	68.10	...	0	0	1	0	0	0	0

2. **Label encoding:**

```
from sklearn.preprocessing import LabelEncoder
enc_transmission = LabelEncoder()
car['transmission_enc'] = enc_transmission.fit_transform(car['transmission'])

enc_seller_type = LabelEncoder()
car['seller_type_enc'] = enc_seller_type.fit_transform(car['seller_type'])

enc_fuel = LabelEncoder()
car['fuel_enc'] = enc_fuel.fit_transform(car['fuel'])

```

name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	seats	max_power (in bph)	...	Toyota	Volkswagen	First Owner	Fourth & Above Owner	Second Owner	Test Drive Car	Third Owner	transmission_enc	seller_type_enc	fuel_enc
Maruti	2014	450000	145500.0	Diesel	Individual	Manual	First Owner	5	74.00	...	0	0	1	0	0	0	0	1	1	0
undai	2010	225000	127000.0	Diesel	Individual	Manual	First Owner	5	90.00	...	0	0	1	0	0	0	0	1	1	0
undai	2017	440000	45000.0	Petrol	Individual	Manual	First Owner	5	81.86	...	0	0	1	0	0	0	0	1	1	1
oyota	2011	350000	90000.0	Diesel	Individual	Manual	First Owner	5	67.10	...	1	0	1	0	0	0	0	1	1	0
Ford	2013	200000	169000.0	Diesel	Individual	Manual	First Owner	5	68.10	...	0	0	1	0	0	0	0	1	1	0

Fault: Outliers:

```
threshold = 1100000
car[car['selling_price'] > threshold]
```

	year	selling_price	km_driven	seats	max_power (in bph)	Mileage	Engine (CC)	Chevrolet	Ford	Honda	...	Toyota	Volkswagen	First Owner	Fourth & Above Owner	Second Owner	Test Drive Car	Third Owner	transmission_enc	seller_type_enc	fuel_enc
37	2016	1140000	50000.0	5	126.20	19.67	1582	0	0	0	...	0	0	1	0	0	0	0	1	1	0
87	2018	1210000	25000.0	5	88.70	21.38	1396	0	0	0	...	0	0	1	0	0	0	0	1	1	0
276	2017	1125000	105000.0	5	88.70	21.38	1396	0	0	0	...	0	0	1	0	0	0	0	1	1	0
285	2017	1150000	63309.0	5	126.20	19.67	1582	0	0	0	...	0	0	1	0	0	0	0	1	0	0

Only 28 rows have values upper than the threshold value.

Remove outliers:

```
car = car[car['selling_price'] <= threshold]
```

Fault: Irrelevant features:

Drop that feature:

```
car = car.drop(columns=['Unnamed: 0'])
car
```

```
car = car.drop(['Mileage Unit'], axis = 1)
car.head(5)
```

Feature scaling

One of the problems that the algorithm often faces is that it gets biased towards larger values. To avoid this problem, we scale the numerical data to bring it in a small range. Here we're using standard scaler:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
y_train=y_train/10000
y_test=y_test/10000
```

Data splitting

To train the model, data splitting was done at 70% for the training model and 30% for testing.

```
from sklearn.model_selection import train_test_split
X = car.drop('selling_price', axis=1)  # Features
y = car['selling_price']               # Labels

# Step 3: Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

# Step 4: Print the shapes of the splits
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (1424, 24)
X_test shape: (611, 24)
y_train shape: (1424,)
y_test shape: (611,)
```

Model training & testing

1. Linear Regression:

```
from sklearn.linear_model import LinearRegression
clf = LinearRegression()
clf.fit(X_train, y_train)
yprediction_lr = clf.predict(X_test)

```

2. Decision Tree:

```
from sklearn import tree
clf1 = tree.DecisionTreeRegressor()
clf1 = clf1.fit(X_train, y_train)
yprediction_dt=clf1.predict(X_test)
```

3. Random Forest:

```
[116]  from sklearn.ensemble import RandomForestRegressor
       rf_regressor = RandomForestRegressor(random_state=42)
       rf_regressor = rf_regressor.fit(X_train, y_train)
       yprediction_rf = rf_regressor.predict(X_test)

```

Model Selection Comparison Analysis

In this car price prediction project, we evaluated three machine learning models — Linear Regression, Decision Tree, and Random Forest — using four key regression evaluation metrics: Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R-Squared (R^2), and Mean Absolute Error (MAE). These metrics helped us assess how well each model performs in predicting car prices.

```python
from sklearn.metrics import mean_squared_error
import numpy as np


# Mean Squared Error (MSE)
mse_lr = mean_squared_error(y_test, yprediction_lr)
mse_dt = mean_squared_error(y_test, yprediction_dt)
mse_rf = mean_squared_error(y_test, yprediction_rf)


# Root Mean Squared Error (RMSE)
rmse_lr = np.sqrt(mse_lr)
rmse_dt = np.sqrt(mse_dt)
rmse_rf = np.sqrt(mse_rf)



# Print the results
print(f"Mean Squared Error LR(MSE): {mse_lr:.2f}")
print(f"Mean Squared Error DT(MSE): {mse_dt:.2f}")
print(f"Mean Squared Error RF(MSE): {mse_rf:.2f}")

print(f"Root Mean Squared Error LR(RMSE): {rmse_lr:.2f}")
print(f"Root Mean Squared Error DT(RMSE): {rmse_dt:.2f}")
print(f"Root Mean Squared Error RF(RMSE): {rmse_rf:.2f}")

```

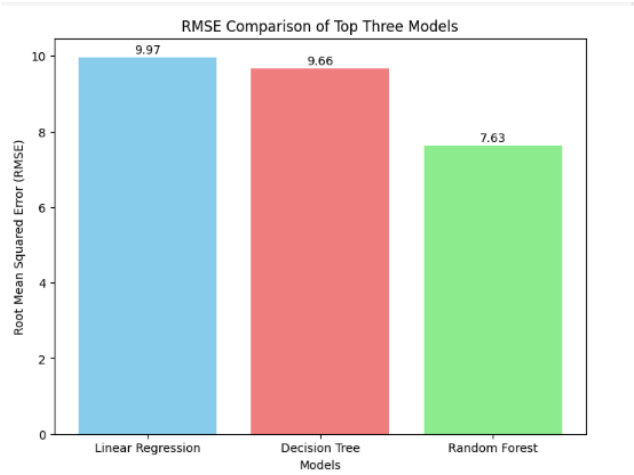
```
Mean Squared Error LR(MSE): 99.31
Mean Squared Error DT(MSE): 93.37
Mean Squared Error RF(MSE): 58.28
Root Mean Squared Error LR(RMSE): 9.97
Root Mean Squared Error DT(RMSE): 9.66
Root Mean Squared Error RF(RMSE): 7.63
```

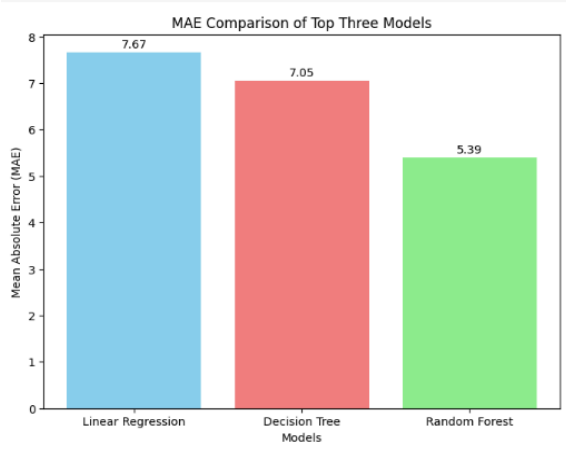
```python
from sklearn import metrics
# R² Score

r2_lr = metrics.r2_score(y_test, yprediction_lr)
r2_dt = metrics.r2_score(y_test, yprediction_dt)
r2_rf = metrics.r2_score(y_test, yprediction_rf)

print(f"R Squared Error LR(R²): {r2_lr:.2f}")
print(f"R Squared Error DT(R²): {r2_dt:.2f}")
print(f"R Squared Error RF(R²): {r2_rf:.2f}")
```

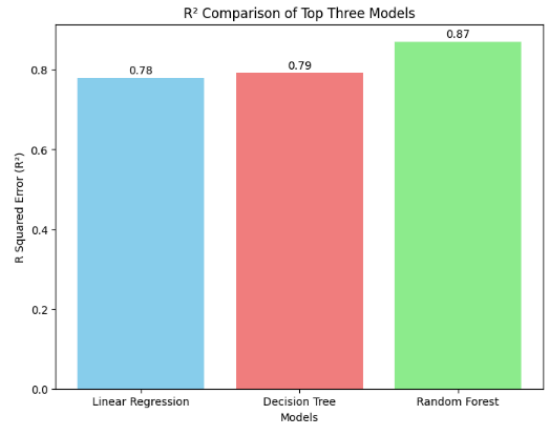
```
R Squared Error LR(R²): 0.78
R Squared Error DT(R²): 0.79
R Squared Error RF(R²): 0.87
```

```python
from sklearn.metrics import mean_absolute_error

mae_lr = mean_absolute_error(y_test, yprediction_lr)
mae_dt = mean_absolute_error(y_test, yprediction_dt)
mae_rf = mean_absolute_error(y_test, yprediction_rf)

print("Mean Absolute Error LR(MAE):", mae_lr)
print("Mean Absolute Error DT(MAE):", mae_dt)
print("Mean Absolute Error RF(MAE):", mae_rf)

```

```
Mean Absolute Error LR(MAE): 7.665969951927112
Mean Absolute Error DT(MAE): 7.047278696126568
Mean Absolute Error RF(MAE): 5.390200159812173
```


RMSE Comparison of Top Three Models
Root Mean Squared Error (RMSE)
9.97
9.66
7.63
Linear Regression
Decision Tree
Random Forest
Models


MAE Comparison of Top Three Models
Mean Absolute Error (MAE)
7.67
7.05
5.39
Linear Regression
Decision Tree
Random Forest
Models


R² Comparison of Top Three Models
R Squared Error (R²)
0.78
0.79
0.87
Linear Regression
Decision Tree
Random Forest
Models

After comparing the results, Random Forest emerged as the best-performing model across all metrics. It had the lowest error values (MSE, RMSE, and MAE), indicating more accurate predictions with minimal deviations from the actual prices. Additionally, it achieved the highest R^2 score, showing that it explains the most variance in car prices compared to the other models.

In contrast, Linear Regression performed the least effectively, with higher error values and a lower R^2 score, indicating that it struggles to capture the complex relationships in the dataset. The Decision Tree model performed better than Linear Regression but still fell behind Random Forest.

Overall, the analysis confirms that Random Forest is the most reliable model for predicting car prices in this dataset, providing a good balance between accuracy and robustness.

Conclusion

We performed data processing, feature scaling, dataset splitting and applied linear regression, random forest and decision tree regression models on the training set. Then we evaluated the accuracy of these models using the testing set. The evaluation metrics show that none of the models gives 100 percent accuracy. However, Random forest model has given the best accuracy among these three models. Thus, it still fulfills the goal of improving decision making by reducing guess work.