CAR PRICE PREDICTION MODEL

Submitted By :

Aritra Banerjee
Asansol Engineering College
Univ. Roll No: **10871023012**

Md. Musharraf
Asansol Engineering College
Univ. Roll No: **10871023027**

Nilesh Maji
Asansol Engineering College
Univ. Roll No: **10871023030**

Shouvik Ghanty
Asansol Engineering College
Univ. Roll No: **10871023045**

Sounak Kundu
Asansol Engineering College
Univ. Roll No: **10871023051**

Sujan Banerjee
Asansol Engineering College
Univ. Roll No: **10871023055**

CANDIDATE'S DECLARATION

We, "**Aritra Banerjee** ", "**Md. Musharraf**", "**Nilesh Maji**", "**Shouvik Ghanty**", "**Sounak Kundu**" and "**Sujan Banerjee**", hereby declare that the work presented in the project report entitled "**CAR PRICE PREDICTION MODEL**" submitted to Department of Computer Applications, Asansol Engineering College, affiliated to **Maulana Abul Kalam Azad University of Technology, West Bengal** for the partial fulfilment of the award of degree of "**Master of Computer Application**" is an authentic record of our work carried out during **the MCA 4th Semester 2024,** under the supervision of **Dr. Arnab Chakraborty**, Trainer (as External Guide).

The matter embodied in this project report has not been submitted elsewhere by anybody for the award of any other degree.

Aritra Banerjee
Univ. Roll No: **10871023012**

Md. Musharraf
Univ. Roll No: **10871023027**

Nilesh Maji
Univ. Roll No: **10871023030**

Shouvik Ghanty
Univ. Roll No: **10871023045**

Sounak Kundu
Univ. Roll No: **10871023051**

Sujan Banerjee
Univ. Roll No: **10871023055**

ACKNOWLEDGEMENT

I take this opportunity to express my profound gratitude and deep regards to my faculty, **Dr. Arnab Chakraborty** for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him time to time shall carry me a long way in the journey of life on which I am about to embark.

I am obliged to my project team members for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my assignment.

Aritra Banerjee

Md. Musharraf

Nilesh Maji

Shouvik Ghanty

Sounak Kundu

Sujan Banerjee

CONTENTS

PROJECT OBJECTIVE:

In this project, we focus on the **Used Car Price Prediction** problem using a dataset obtained from Kaggle. The dataset contains various attributes of used cars such as year of manufacture, present price, kilometres driven, fuel type, seller type, transmission type, number of previous owners, and more. The main goal of this project is to predict the selling price of a used car based on these attributes using regression-based machine learning models.

The objective of our project is to study the dataset, analyse the patterns and relationships among the attributes, and develop machine learning models that can accurately predict car prices. The dataset needs to be cleaned and pre-processed before training. We have used six machine learning models in this project: Linear Regression, Decision Tree Regressor, Random Forest Regressor, Gradient Boosting Regressor, XG Boost Regressor, and K-Nearest Neighbours Regressor. After training these models, we evaluate their performance using metrics like MAE, MSE, RMSE, and R^2 score.

We then compare the performance of all the models and select the best-performing one — in our case, the Random Forest Regressor — based on accuracy and reliability. This model is then used to build the final prediction system, which can be further integrated with a user interface using Streamlit for practical use.

Our methodology for solving the problem in this project is as follows:

- Load and study the dataset.
- Perform data cleaning and pre-processing (handling missing values, encoding categorical features, etc.).
- Conduct exploratory data analysis (EDA).
- Perform feature engineering and selection.
- Train six different regression models.
- Evaluate the models using regression metrics.
- Choose the best-performing model based on performance scores.
- Build the prediction system and interface using Streamlit.

Requirement Specification-

1. Software Requirements

- Kaggle (for dataset collection)
- Google Colab Notebook
- Editor- VS Code
- Programming Language- Python 3.10
- Libraries used- numpy, pandas, matplotlib, seaborn, scikit-learn, fastapi, npm etc.
- Operating System- Windows 10 or any version, Linux
- Repository- GitHub

2. Hardware Requirements

- Desktop/ Laptop
- 4GB RAM or any
- Processor- Pentium (R) Silver N5000 CPU @ 1.10GHz (Intel i5) or any other

PROJECT SCOPE

The broad scope of the **'Used Car Price Prediction'** project is given below:

The given dataset has attributes based on which the selling price of a used car will be predicted.

It is a useful project as regression models can be used to estimate the approximate price of a used car based on multiple factors, helping both buyers and sellers make informed decisions in the used car market.

With the rising demand for second-hand vehicles, particularly in developing countries, a system that can predict fair car prices based on historical data has become increasingly valuable. This system reduces human bias and promotes transparency in vehicle transactions.

The dataset includes important features such as the year of manufacture, fuel type, transmission type, number of previous owners, kilometres driven, and present price. By analysing these factors, one can get a reliable prediction of a car's worth in the current market.

Our system can be especially useful for used car dealers, individual sellers, and customers who want to assess whether they are getting a fair deal.

The machine learning models used in this project include Linear Regression, Decision Tree Regressor, Random Forest Regressor, Gradient Boosting Regressor, XG Boost Regressor, and K-Nearest neighbour (KNN) Regressor. These models have been trained and evaluated based on metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R^2 Score.

Out of all the models, the best-performing one (**Random Forest Regressor** in our case) is selected for building the final price prediction system.

The final model is deployed with the help of the Streamlit framework, providing a simple and user-friendly GUI through which users can enter car details and get instant price predictions.

The dataset used is a refined version of the original dataset sourced from Kaggle. While it may not include all the real-world complexities of car pricing, the system can be improved by retraining the models with more comprehensive and up-to-date data.

With the rise of online marketplaces and digital platforms for car sales, this system can be a valuable backend module to suggest fair prices and guide negotiations, ultimately helping to establish a more standardized and transparent used car market.

DATA DESCRIPTION

Source of the data: Kaggle. The given dataset is a shortened version of the original dataset in Kaggle.

Data Description: The given train dataset has 8123 rows and 14 columns

	Feature	Data Type	Unique Values	Missing Values
0	name	object	2058	0
1	year	int64	29	0
2	selling_price	int64	677	0
3	km_driven	int64	921	0
4	fuel	object	4	0
5	seller_type	object	3	0
6	transmission	object	2	0
7	owner	object	5	0
8	mileage	object	393	221
9	engine	object	121	221
10	max_power	object	322	215
11	torque	object	441	222
12	seats	float64	9	221

The following table shows the 5 number summary of the given dataset:

	count	mean	std	min	25%	50%	75%	max
year	8128.0	2013.804011	4.044249	1983.0	2011.0	2015.0	2017.0000	2020.0
selling_price	8128.0	638271.807702	806253.403508	29999.0	254999.0	450000.0	675000.0000	10000000.0
km_driven	8128.0	69819.510827	56550.554958	1.0	35000.0	60000.0	98000.0000	2360457.0
mileage	8128.0	19.415554	3.981922	0.0	16.8	19.3	22.2775	42.0
engine	8128.0	1452.898130	498.196720	624.0	1197.0	1248.0	1582.0000	3604.0
max_power	8128.0	91.264982	35.376388	0.0	68.1	82.0	101.2500	400.0
seats	8128.0	5.405389	0.948874	2.0	5.0	5.0	5.0000	14.0

Now we will pre-process the data. The methodology followed is given below:

- Checking for null values.
- If null values are present, we will fill them or drop the row containing the null value based on the dataset.
- Check for duplicate values.
- If duplicate values are present, we will remove them.
- Checking for outliers.
- If outliers are present, they will either be removed or replaced by following a suitable method depending on the dataset.

DATA PRE-PROCESSING

In this project, data preprocessing played a crucial role in preparing the dataset for model training and improving prediction accuracy. The following steps were taken during the preprocessing phase:

- **Handling Missing Values**: We used .isnull().sum() to identify columns with missing values. Features like mileage, engine, max_power, and seats had missing entries which were filled using median or mode depending on the column type.
- Additionally, the **torque column was removed entirely**. This is because it had inconsistent and non-standard values in various formats (e.g., "113.75Nm@4000rpm", "11.5@ 4500(kgm@ rpm)") which made numerical extraction unreliable. Most values were lost in the cleaning process, resulting in too many nulls and minimal contribution to prediction.

	0
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	0

Checking How Much Data is Missing

Why?

If a column has very few missing values (like <5%), we can fill them with mean/median/mode.

If a column has a lot of missing values, we need to decide whether to drop it or use advanced techniques to fill it.

```
mileage      2.718996
engine       2.718996
max_power    2.645177
torque       2.731299
seats        2.718996
dtype: float64
```

Converting Categorical to Numerical: We applied **One-Hot Encoding** to convert categorical features such as fuel type, seller type, transmission, and owner into numeric format, which is essential for feeding into ML models.

Feature Selection:

Certain columns were dropped based on their low predictive value or irrelevance:

name: The car name often includes brand/model but is too granular and inconsistent (e.g., long text strings, multiple variants of the same car).

seats: While initially included, this feature had limited variation and did not strongly correlate with selling price. Its impact on model accuracy was minimal.

Feature Scaling: Standardization (Z-score normalization) was used to scale all numerical features so that they have a mean of 0 and standard deviation of 1. This ensures all features contribute equally during model training.

```
Scaled Training Set Shape: (6502, 15)
Scaled Testing Set Shape: (1626, 15)
```

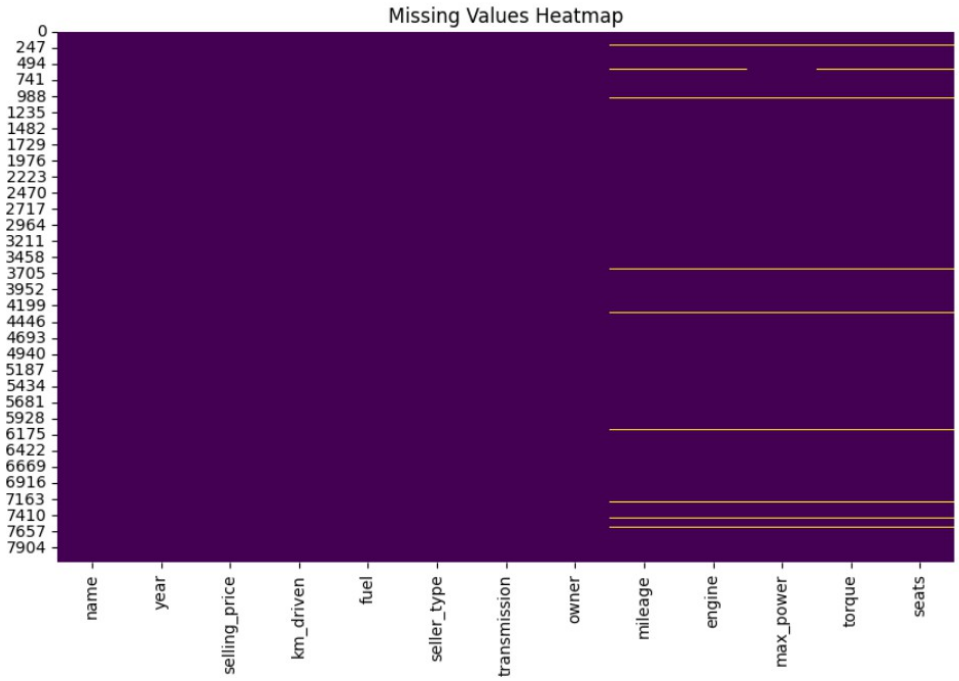


Figure: *Missing Values Heatmap*

The heatmap above visualizes the presence of missing values across different features in the dataset. Dark purple areas represent non-null data, while yellow lines highlight missing entries. As we can observe, features like **mileage**, **engine**, **max_power**, **torque**, and **seats** had missing values. Identifying these null entries was crucial in deciding the appropriate preprocessing steps such as imputation (using median or mode) or removal of rows with

excessive missingness. This visualization helped us quickly assess data completeness and ensure proper handling before model training.

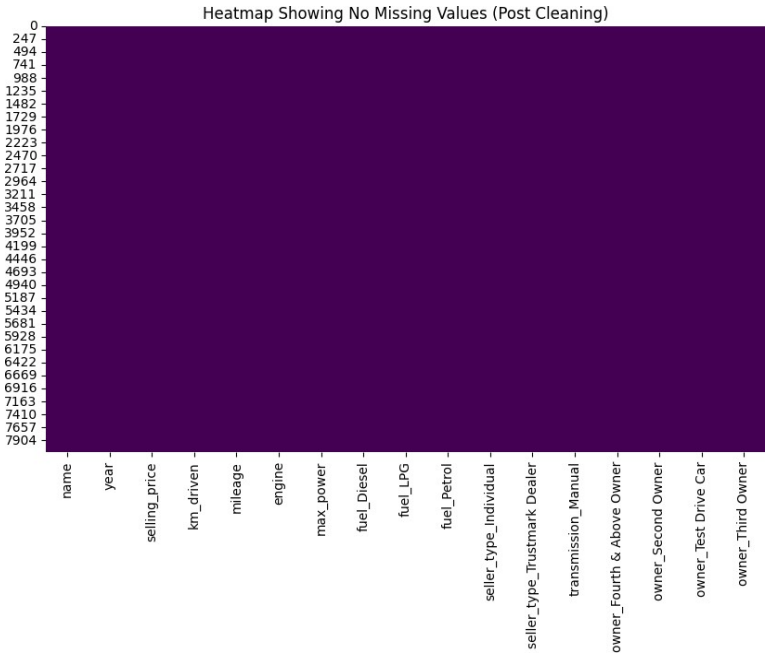


Figure:*Heatmap after Handling Missing Values*

This heatmap confirms that the dataset is now free from missing values after preprocessing. All columns previously containing null entries (like mileage, engine, max_power, and seats) have been appropriately handled. The torque column was removed due to excessive inconsistencies and missing data. Ensuring a complete dataset was essential for building accurate and stable machine learning models.

Identifying and handling Outliers:

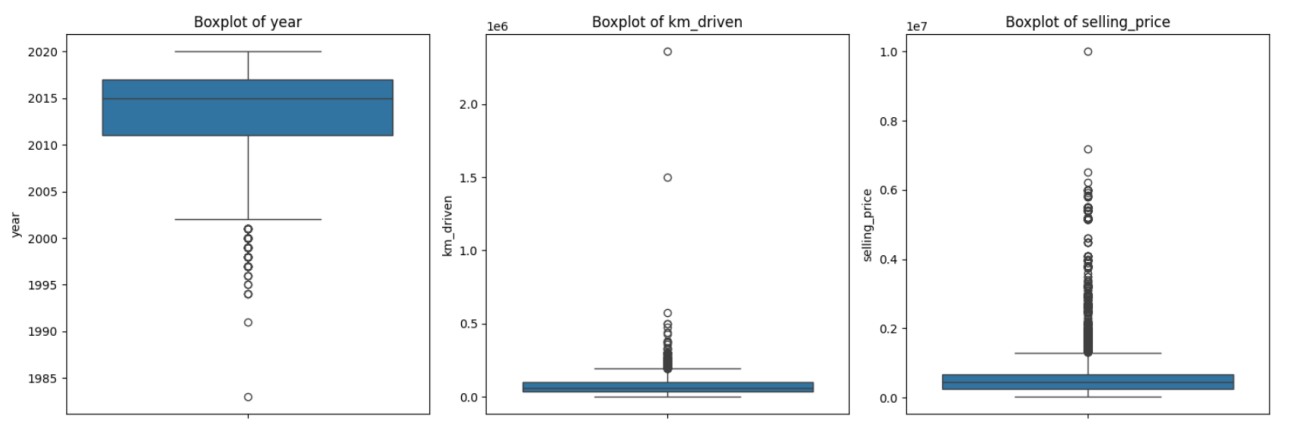


Figure: *Boxplots of Year, Kilometers Driven, and Selling Price*

The boxplots above were used to detect outliers in numeric features. The plots show values that lie far from the interquartile range, especially in the km_driven and selling_price columns. These visual cues helped us decide whether to remove extreme entries or treat them appropriately during data preprocessing to ensure model accuracy is not affected by anomalies.

This figure shows boxplots of numerical features (year, km_driven, selling_price) to detect outliers in the data.

- Boxplot of year: Shows most car entries are recent, but there are a few entries from the 1980s and 1990s, which are considered outliers.
- Boxplot of km_driven: Reveals some extremely high kilometer readings (above 2 million km), which are likely incorrect or rare and may need to be removed or investigated.
- Boxplot of selling_price: Highlights a long tail of very high prices — potential outliers, rare cars, or luxury vehicles.

Outliers are extreme values that deviate from other observations on data; they may indicate variability in a measurement, experimental errors or a novelty. In other words, an outlier is an observation that diverges from an overall pattern on a sample. Outliers are an important part of a dataset. There are different ways to find outliers in statistics like :

Box Plots,

IQR method,

Z-Score method &

Distance from the mean (multivariate)

Types of Outliers:

•) Point outlier- An individual data point that sits outside of the range of the rest of the dataset. Eg., missing a digit in height.

•) Contextual outlier- when a data point is significantly different from dataset, but only within specific context. For eg., 0 degree in winter is ok, but in summer it is an outlier.

•) Collective Outlier- when a series of data points differ significantly from the trends in the rest of the dataset. Eg., if the level of subscribed users stayed entirely static for many weeks with no fluctuation (where there is normal seasonal or daily fluctuations)

Most common causes of outliers on a data set:

Data entry errors (human errors)

Measurement errors (instrument errors)

Experimental errors (data extraction or experiment planning/executing errors)

Intentional (Man-made; dummy outliers made to test detection methods)

Data processing errors (data manipulation or data set unintended mutations)

Sampling errors (extracting or mixing data from wrong or various sources)

Natural (not an error, known as dataset novelties in data)

Correlation Analysis

Correlation Analysis was performed to understand the relationships between numerical features and identify which features had strong or weak associations with the target variable (selling_price). This step helped us select the most influential features and also avoid using highly correlated predictors that may cause multicollinearity issues in some models.

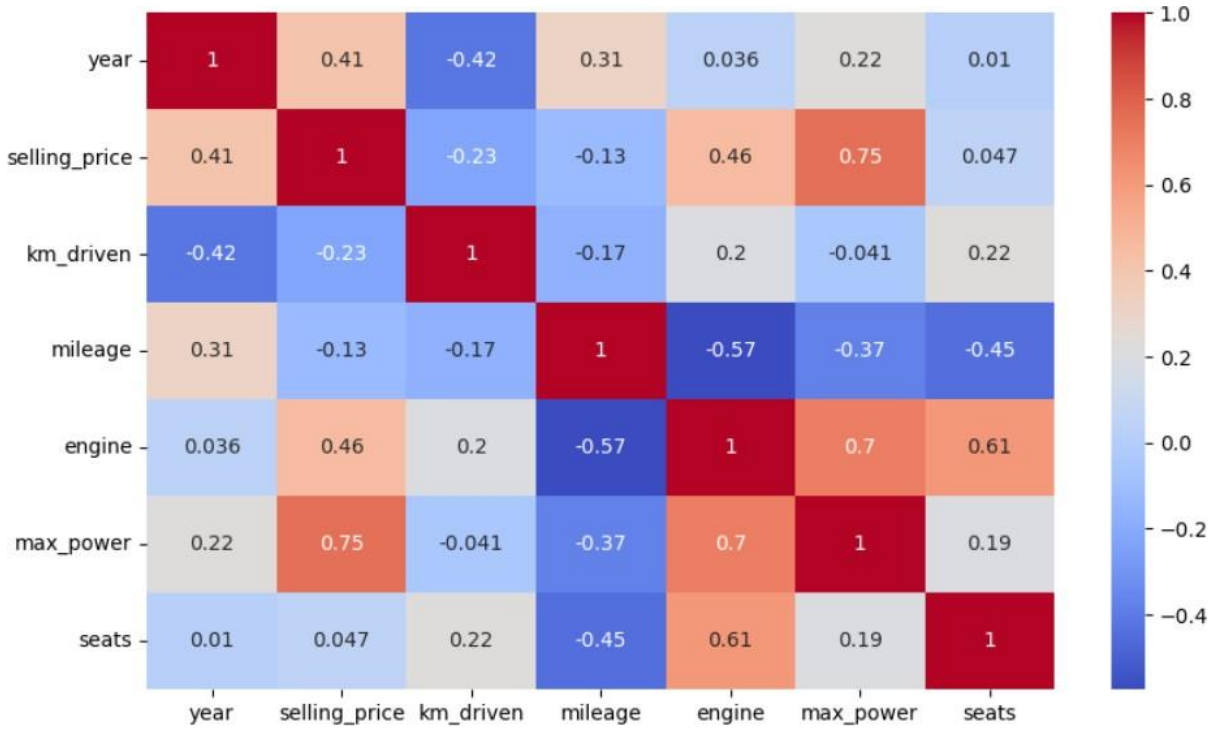


Figure: *Correlation Heatmap of Numerical Features*

This heatmap shows the correlation between numeric variables.

max_power has the highest positive correlation with selling_price (0.75), making it a strong predictor.

engine and year also show moderate correlation.

mileage and km_driven are negatively correlated, as expected for used cars.

Final Dataset Structure

After completing all necessary preprocessing steps — including handling missing values, dropping irrelevant columns, encoding categorical variables, converting data types, and feature scaling — we obtained a clean and model-ready dataset.

The final dataset contains 8128 rows and 17 columns.

All features are in either numeric or boolean format.

Categorical features such as fuel, seller_type, transmission, and owner have been transformed using One-Hot Encoding.

The torque column was dropped due to inconsistent formatting and excessive missing values.

The name column was retained but not used for prediction due to its inconsistent and overly granular values.

The target variable for prediction is selling_price.

Below is a summary of the final dataset's structure:

```
(8128, 17)
name                          object
year                           int64
selling_price                  int64
km_driven                      int64
mileage                      float64
engine                       float64
max_power                    float64
fuel_Diesel                     bool
fuel_LPG                        bool
fuel_Petrol                     bool
seller_type_Individual          bool
seller_type_Trustmark Dealer    bool
transmission_Manual             bool
owner_Fourth & Above Owner      bool
owner_Second Owner              bool
owner_Test Drive Car            bool
owner_Third Owner               bool
dtype: object
```

EXPLORATORY DATA ANALYSIS (EDA)

The distribution of selling prices is highly right-skewed, indicating that most used cars in the dataset are priced on the lower end, with a small number of high-end vehicles significantly increasing the upper range. This skewness may affect model performance and suggests the potential benefit of applying transformations like log scaling to normalize the target variable.

Why Exploratory Data Analysis is Important?

Exploratory Data Analysis (EDA) is important for several reasons, especially in the context of data science and statistical modeling. Here are some of the key reasons why EDA is a critical step in the data analysis process:

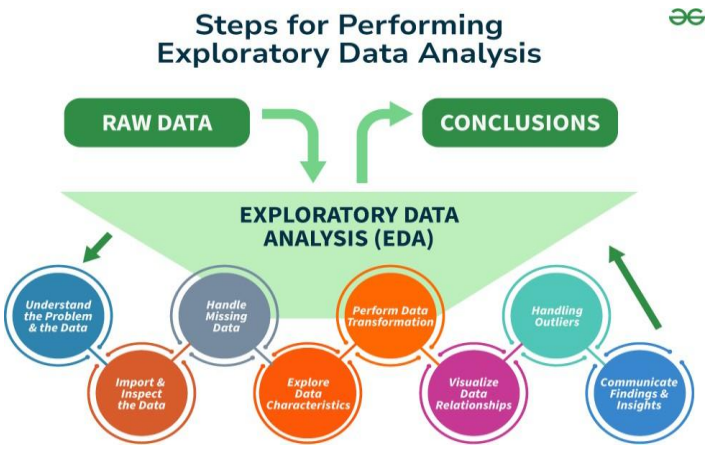
Helps to understand the dataset, showing how many features there are, the type of data in each feature, and how the data is spread out, which helps in choosing the right methods for analysis.

EDA helps to identify hidden patterns and relationships between different data points, which help us in and model building.

Allows to spot errors or unusual data points (outliers) that could affect your results.

Insights that you obtain from EDA help you decide which features are most important for building models and how to prepare them to improve performance.

By understanding the data, EDA helps us in choosing the best modeling techniques and adjusting them for better results.



EDA helps us explore and understand the data by uncovering patterns, outliers, and feature relationships with the target variable—selling price.

We performed EDA on the cleaned version of our dataset after handling missing values and transforming categorical variables.

1. Distribution of the Target Variable

We plotted the distribution of the selling_price using a histogram with Kernel Density Estimation (KDE). The distribution was found to be right-skewed, indicating that most cars in the dataset are priced on the lower end, while a few expensive cars pull the tail to the right.

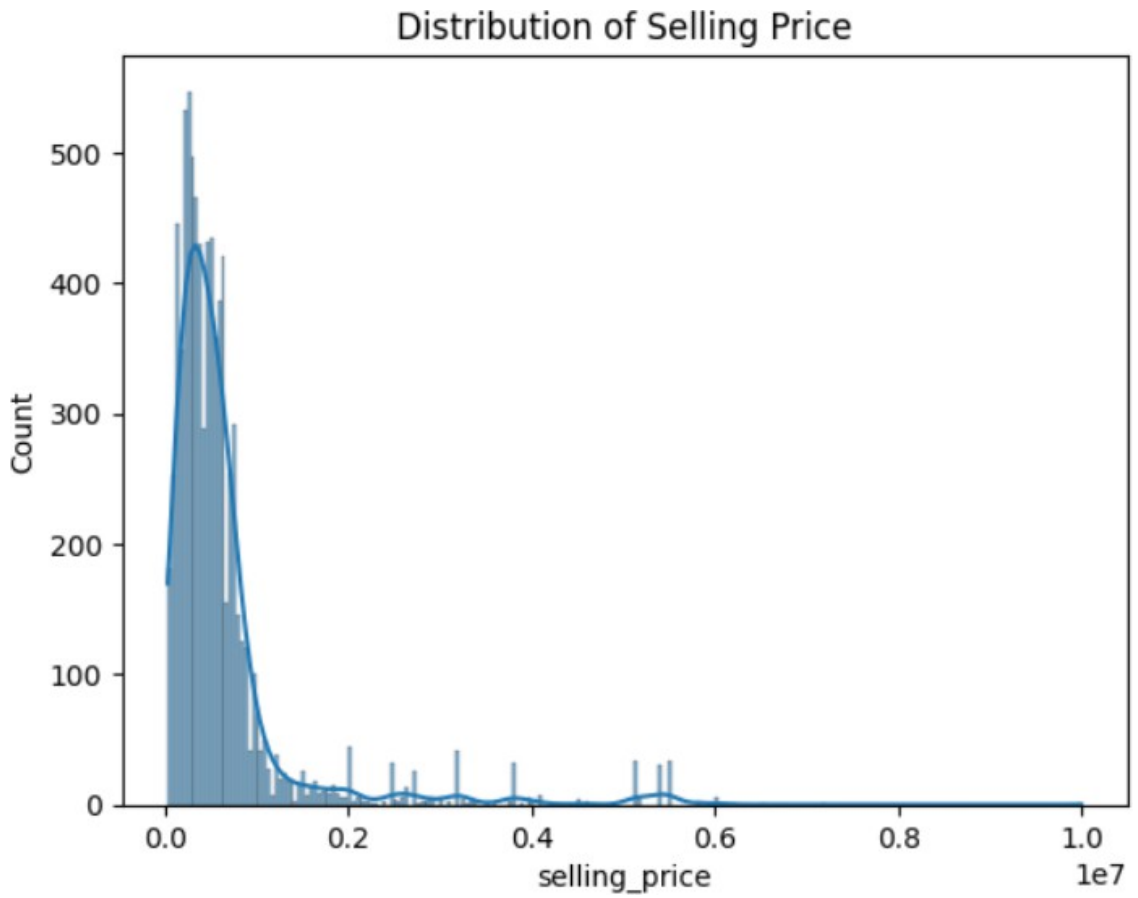


Figure: *The distribution of selling prices is highly **right-skewed**, indicating that most used cars in the dataset are priced on the lower end, with a small number of high-end vehicles significantly increasing the upper range. This skewness may affect model performance and suggests the potential benefit of applying transformations like log scaling to normalize the target variable.*

2. Count Plots of Categorical Variables

We generated count plots to understand the distribution of categorical features such as fuel, transmission, seller_type, and owner. These visualizations provide insight into how common each category is within the dataset.

Fuel Type: Majority of the vehicles run on Diesel and Petrol, with LPG and CNG being relatively rare.

Transmission: A large proportion of vehicles have Manual transmission compared to Automatic.

Seller Type: Most sellers are Individuals, followed by Dealers and a few Trustmark Dealers.

Ownership: The majority of the cars are First Owner, followed by Second and Third Owners.

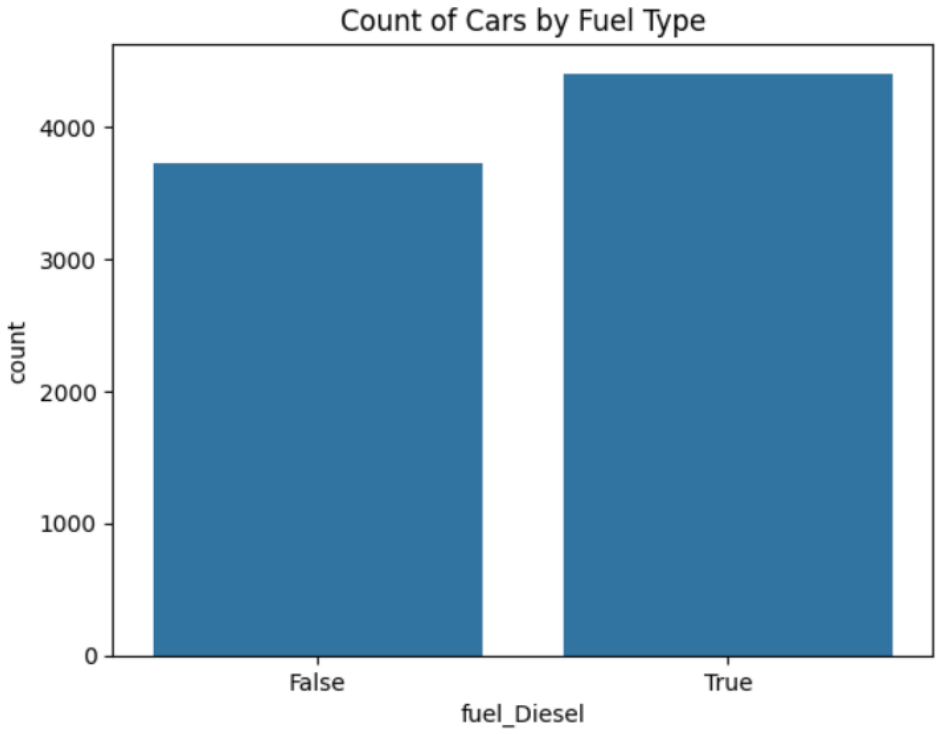
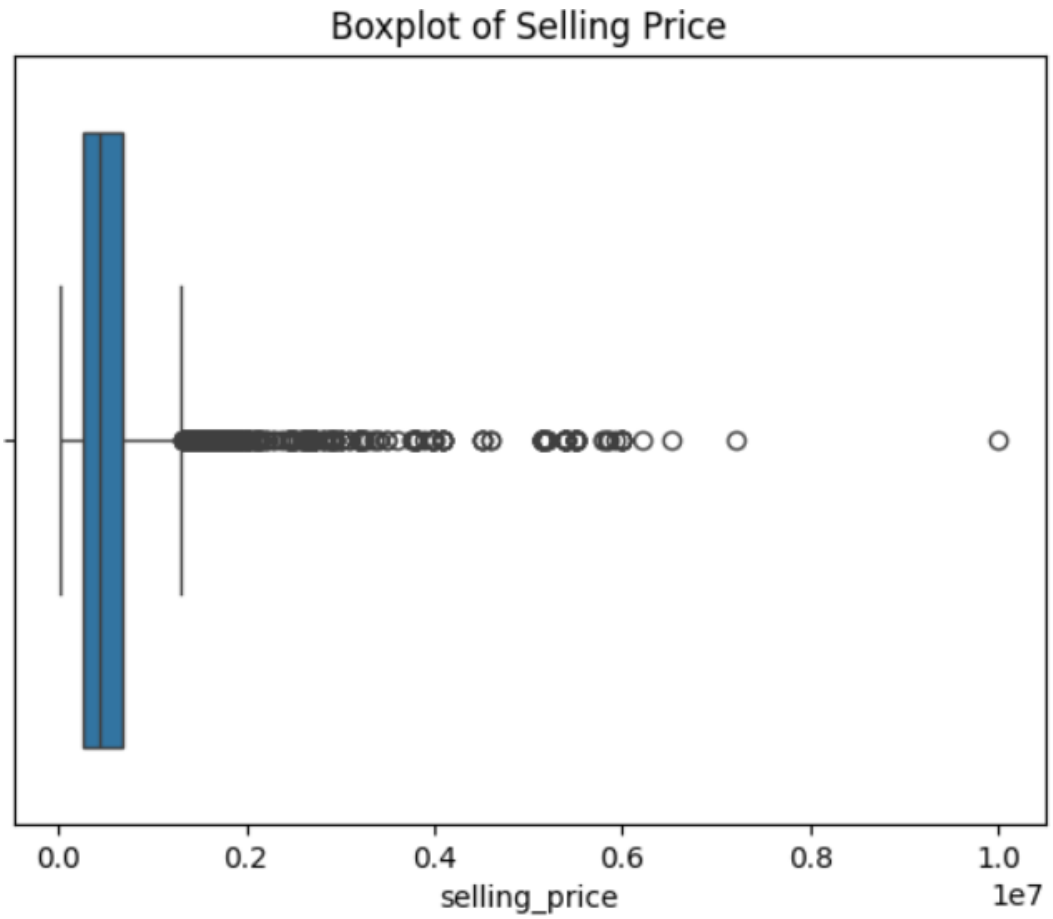


Figure: *Diesel cars are slightly more common than non-diesel ones, indicating a marginal dominance of diesel fuel type in the dataset.*

3. Outlier Detection Using Boxplots

We plotted boxplots for continuous numerical features: selling_price, km_driven, and mileage. These plots revealed several outliers, particularly in selling_price and km_driven.



Observation: Outliers in selling_price may represent luxury cars or rare models. Since this is real-world pricing data, we retained these values instead of removing them to preserve model generalizability.

Figure: *Outliers in selling_price may represent luxury cars or rare models. Since this is real-world pricing data, we retained these values instead of removing them to preserve model generalizability.*

4. Correlation Analysis

We computed the correlation matrix to identify relationships between numeric features and the target variable selling_price. A heatmap was generated for visualization.

Top positively correlated features with selling_price:

max_power (correlation = 0.74)

engine (correlation = 0.46)

year (correlation = 0.41)

Negatively correlated features:

km_driven (correlation = -0.23)

mileage (correlation = -0.13)

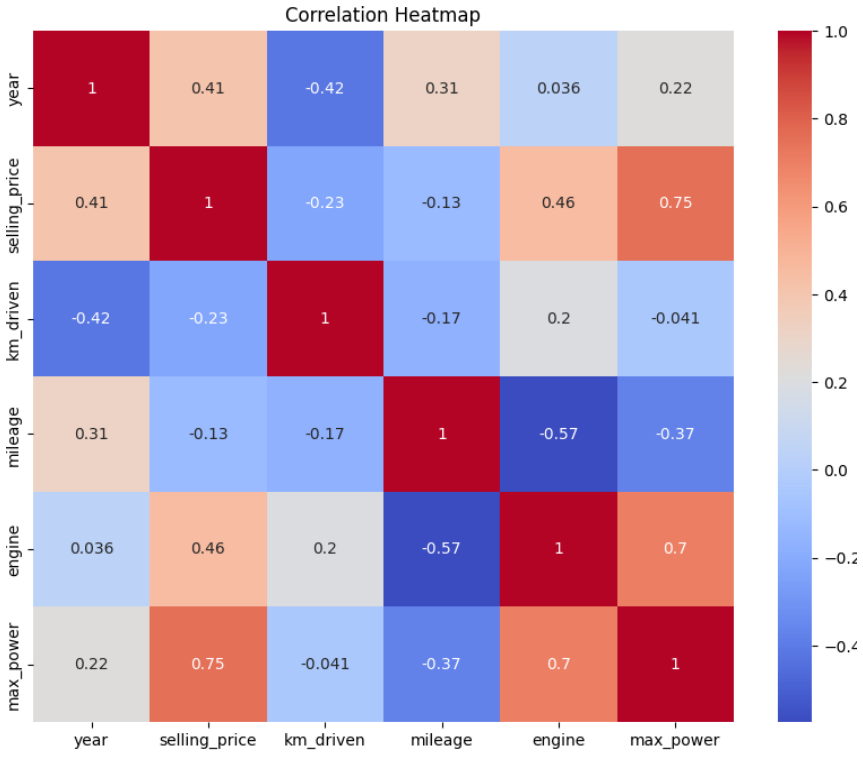


Figure: *As expected, newer cars with higher power and engine capacity tend to have higher selling prices. On the other hand, cars with higher mileage and more kilometers driven tend to be cheaper.*

5. Scatter Plots for Feature Relationships

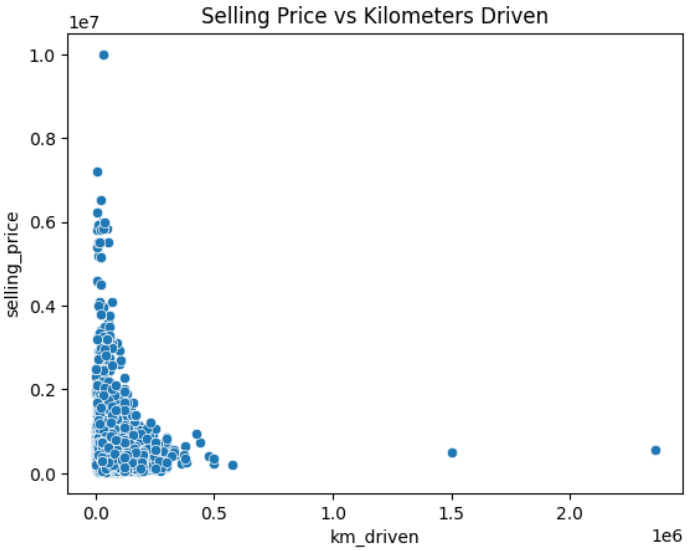
To better understand the relationship between key numeric features and the target variable, we plotted scatter plot on :

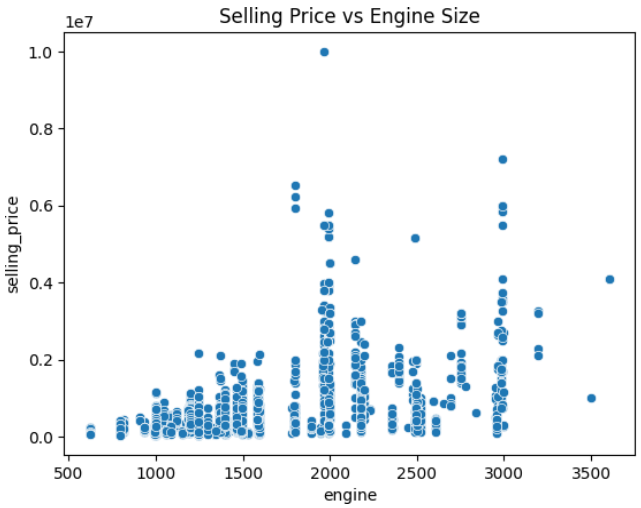
Selling Price vs.

Kilometers Driven Selling Price

vs. Engine Size Selling Price vs.

Max Power





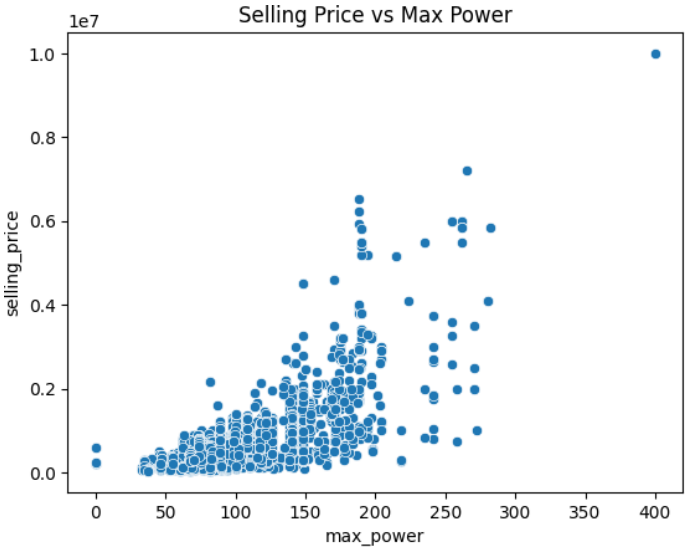


Figure: *These relationships confirm the logical assumption that less-used and more powerful cars are valued higher.*

MODEL BUILDING

Splitting Data for Training and Testing Purpose:

In our Used Car Price Prediction project, we began by separating the dataset into two parts: the feature set (**X**) containing independent variables (like year, kilometers driven, fuel type, transmission, etc.) and the target variable (**y**) representing the car's selling price.

We then divided the dataset into training and testing subsets using a split ratio of **80:20**. This means **80% of the data** was used to train the machine learning models, and the remaining **20%** was used to evaluate their performance. This ensures that the model can generalize well to unseen data.

The same split ratio was consistently applied across all the models developed during the project. The dataset used for training was obtained after thorough pre-processing, including handling missing values, encoding categorical features, and feature scaling where necessary.

	Rows	Columns
X_train shape	504	8
y_train shape	504	1
X_test shape	126	8
y_test shape	126	1

Linear Regression :–

What is Linear Regression?

Linear Regression is a **supervised learning algorithm** used for **predicting a continuous dependent variable** based on one or more independent variables. It assumes a linear relationship between the variables.

Types of Linear Regression

1. **Simple Linear Regression** – one independent variable.
2. **Multiple Linear Regression** – two or more independent variables.

Simple Linear Regression

Model Equation:-

$$y=\beta0+\beta1x+\varepsilon$$

y: Dependent variable (target)

x: Independent variable (input)

β0: Intercept

β1: Slope (coefficient)

ε: Error term

Objective: Find the best-fit line that minimizes the error between actual and predicted values (using **Least Squares Method**).

Cost Function (Mean Squared Error):-

$$J(\beta0,\beta1)=n1i=1\sum n(yi-y\hat{}i)2$$

- yi: Actual value
- y^i: Predicted value

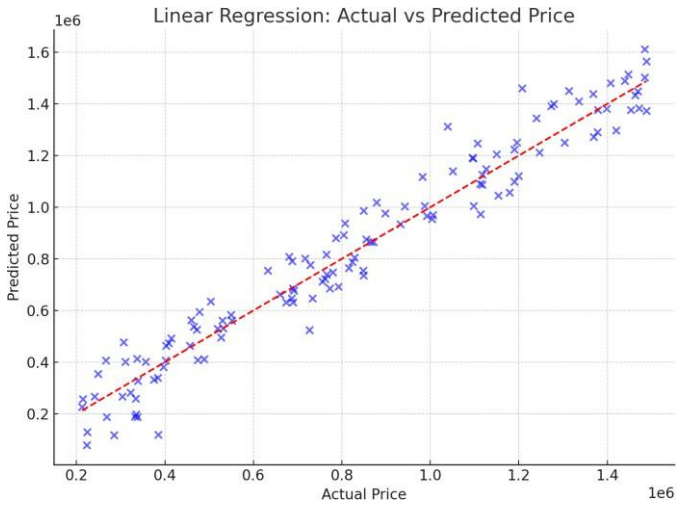
We minimize this function to find the best parameters (β0,β1).

Multiple Linear Regression:-

Model Equation:

$$y=\beta0+\beta1x1+\beta2x2+\cdots+\beta nxn+\varepsilon$$

Where x1,x2,…,xn, are the features (independent variables).



Assumptions of Linear Regression:-

1. Linearity

- The relationship between the **independent variables** and the **dependent variable** is linear.
- This means changes in x produce proportional changes in y.

Check: Use scatter plots or residual plots.

2. Independence

- The observations (data points) are independent of each other.
- No correlation exists among residuals (especially important in time series data).

Check: Use **Durbin-Watson test** for autocorrelation.

3. Homoscedasticity (Constant Variance)

- The residuals (errors) should have **constant variance** across all levels of the independent variables.

Check: Plot residuals vs. predicted values. The spread should be consistent.

4. Normality of Residuals

- The residuals should be **normally distributed**, especially important for hypothesis testing and confidence intervals.

Check: Histogram or Q-Q plot of residuals.

5. No Multicollinearity (for Multiple Linear Regression)

- Independent variables should not be **highly correlated** with each other.
- High multicollinearity makes it difficult to estimate coefficients accurately.

Check: Use **Variance Inflation Factor (VIF)**. A VIF > 5 or 10 indicates multicollinearity.

6. No Significant Outliers

- Outliers can skew the regression line and lead to misleading results.

Check: Box plots or leverage statistics like **Cook's Distance**.

```
Model Performance:
MAE: 267948.115754324
MSE: 198127475114.70956
RMSE: 445115.1256862763
R2 Score: 0.6977388586182252
```

- **Advantages:-**

1. Simple and Easy to Implement

- Very straightforward algorithm with minimal computational complexity.
- Ideal for **quick prototyping** and initial data analysis.

2. Interpretable

- Coefficients clearly indicate the **influence of each feature** on the outcome.
- You can directly interpret the relationship between variables.

Disadvantages:-

1. Assumes a Linear Relationship

- Real-world data is often **non-linear**, and linear regression cannot model complex relationships well.

2. Sensitive to Outliers

- A few extreme data points can **drastically affect** the regression line and predictions.

Example: One high-priced house in a dataset can skew a housing price model.

3. Assumes Independence of Features

- In **multiple linear regression**, if independent variables are correlated (multicollinearity), the model's estimates become unstable.

4. Assumes Homoscedasticity

- Linear regression expects that the variance of residuals is constant.
- If **heteroscedasticity** (changing variance) exists, predictions become unreliable.

Decision Tree :-

What is a Decision Tree?

A **Decision Tree** is a **supervised learning algorithm** used for **classification** and **regression** tasks. It mimics human decision-making by splitting data into branches based on features, forming a tree-like structure.

Key Components

- **Root Node**: The top node representing the entire dataset.
- **Decision Nodes**: Internal nodes where the data is split.

- **Leaf Nodes (Terminal Nodes)**: Nodes that represent the final output (class label or value).
- **Branches**: Arrows connecting nodes, representing decisions or conditions.

How It Works

The model splits data using a feature that results in the best separation according to a metric like:

➤ For Classification:

- **Gini Impurity**
- **Entropy (Information Gain)**

➤ For Regression:

- **Mean Squared Error (MSE)**
- **Mean Absolute Error (MAE)**

The tree is built **recursively** by selecting the best feature to split the data at each node.

Splitting Criteria

1. Gini Impurity

Gini Impurity is a metric used to measure how **pure** a node is in a decision tree. It helps the tree decide **which feature to split on** during training.

Definition

$$Gini(t) = 1 - i = 1 \sum Cpi2$$

Where:

- C = number of classes
- pi= probability (proportion) of class iii at a node t

Goal

Minimize Gini Impurity when splitting:

- A node with **Gini = 0** is **pure** (all items belong to one class).
- A **higher Gini** means more mixing of classes (less purity).

Example

Suppose a node has:

- 4 examples of **Class A**
- 6 examples of **Class B**

$$pA=104=0.4, pB=106=0.6$$

$$Gini=1-(0.42+0.62)=1-(0.16+0.36)=0.48$$

Interpretation

- **Gini = 0** → perfect purity
- **Gini = 0.5** (for 2 classes equally mixed) → maximum impurity
- It helps choose **splits** that make children nodes more pure.

2. Entropy

Entropy is another metric used in decision trees (especially in the **ID3 algorithm**) to measure the **disorder** or **impurity** in a dataset. The goal is to **minimize entropy** and create more **homogeneous nodes**.

Definition of Entropy

Where:

$$Entropy(t)=\sum pilog2(pi)$$

- C= number of classes
- pi= probability of class iii at node ttt

Goal of Entropy

- **Entropy = 0**: Pure node (only one class present).
- **Entropy is maximum** when classes are equally mixed (most impure).

Example

Suppose a node has:

- 4 examples of **Class A**
- 4 examples of **Class B**

$$pA=84=0.5, pB=84=0.5$$

$$Entropy=-(0.5\log_2 0.5+0.5\log_2 0.5)=-(0.5\cdot-1+0.5\cdot-1)=1$$

Information Gain

To choose the best feature for a split, entropy is used to calculate **Information Gain**:

Where:

$$\text{Information Gain}=\text{Entropy}_{\text{parent}}-\sum_k(n_k\cdot\text{Entropy}_k)$$

- n= total number of samples
- nk= number of samples in child node k.

Higher Information Gain = Better Feature to Split On

Gini vs. Entropy

Metric	Interpretation	Bias	Use in Algorithms
Gini	Impurity measure	Tends to create **larger** leaf nodes	CART

Entropy	Measures information	May be more **sensitive to splits**	ID3, C4.5

Types of Decision Trees:-

1. Based on Task

Classification Trees

- **Output**: Class labels (e.g., Yes/No, Cat/Dog)
- **Use**: When the target variable is **categorical**
- **Splitting Criteria**: Gini Impurity, Entropy

Example: Predicting if a customer will churn (Yes/No)

Regression Trees

- **Output**: Continuous numeric values
- **Use**: When the target variable is **continuous**
- **Splitting Criteria**: Mean Squared Error (MSE), Mean Absolute Error (MAE)

Example: Predicting house prices

2. Based on Splitting Technique

ID3 (Iterative Dichotomiser 3)

- Uses **Information Gain (Entropy)** to split.
- Works only with categorical features.

C4.5

- Successor to ID3
- Handles both categorical and continuous features
- Uses **Gain Ratio** (improved version of Information Gain)

CART (Classification and Regression Trees)

- Uses **Gini Impurity** for classification
- Supports both classification and regression trees

3. Based on Tree Structure or Learning Strategy

Random Forest (Ensemble of Trees)

- Builds multiple decision trees and averages their results (regression) or uses majority vote (classification).
- Reduces overfitting and increases accuracy.

Gradient Boosted Trees (e.g., XGBoost, LightGBM)

- Builds trees **sequentially**, where each new tree fixes the errors of the previous one.
- Very powerful for structured/tabular data.

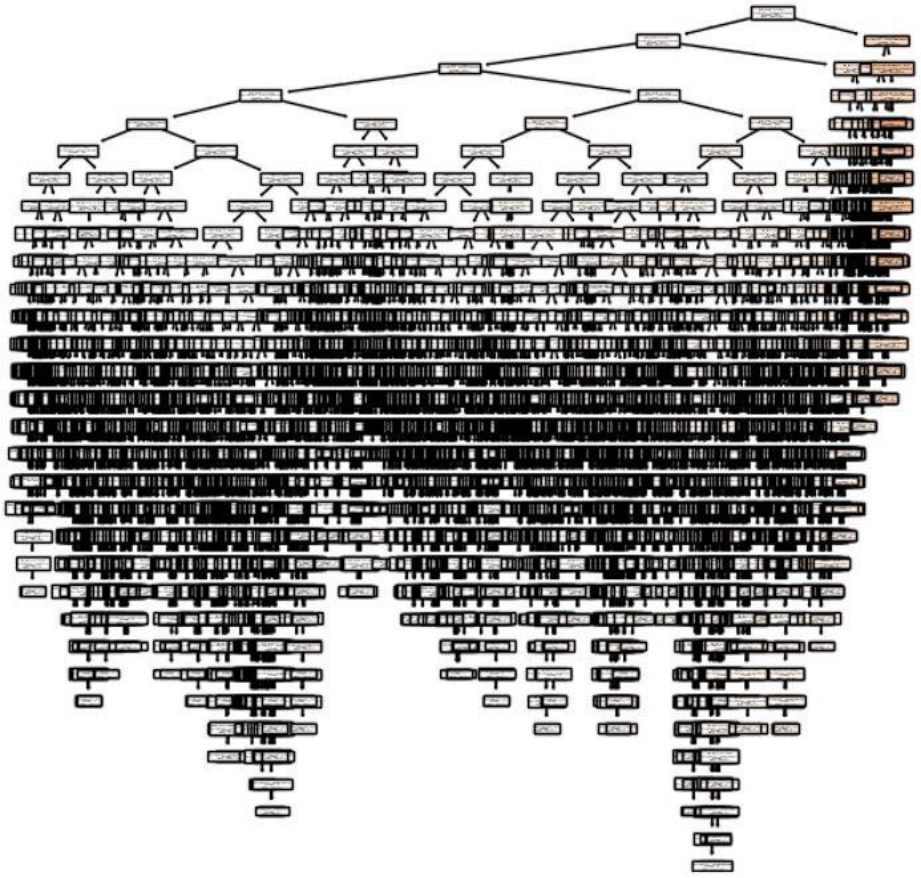
```
Decision Tree Model Performance:
MAE: 81522.73497177279
MSE: 26551669462.965157
RMSE: 162946.8301715782
R2 Score: 0.959493059138714
```

Advantages

1. **Easy to Understand and Interpret**
   - Tree structure is visual and intuitive—even non-technical users can follow it.
2. **No Data Preprocessing Required**
   - No need for feature scaling or normalization.
   - Handles both numerical and categorical data.

Disadvantages of Decision Trees

1. **Overfitting**
   - Trees tend to memorize training data if not pruned or regularized properly.
2. **Unstable**
   - Small changes in data can lead to a completely different tree structure.



Random Forest :-

What is Random Forest?

Random Forest is an **ensemble machine learning algorithm** that builds multiple **decision trees** during training and combines their outputs for better accuracy and generalization.

It's used for both:

- **classification**
- **Regression**

How Random Forest Works:-

Step 1: Bootstrapping the Data

- From the original training dataset, **N random samples** are drawn **with replacement** (bootstrapped) to train each individual tree.

- Some samples may be repeated; some may be left out (called **Out-of-Bag samples**).

Step 2: Random Feature Selection

- At **each split** in each tree, only a **random subset of features** is considered.
- This randomness helps reduce correlation between trees and improves performance.

Step 3: Build Decision Trees

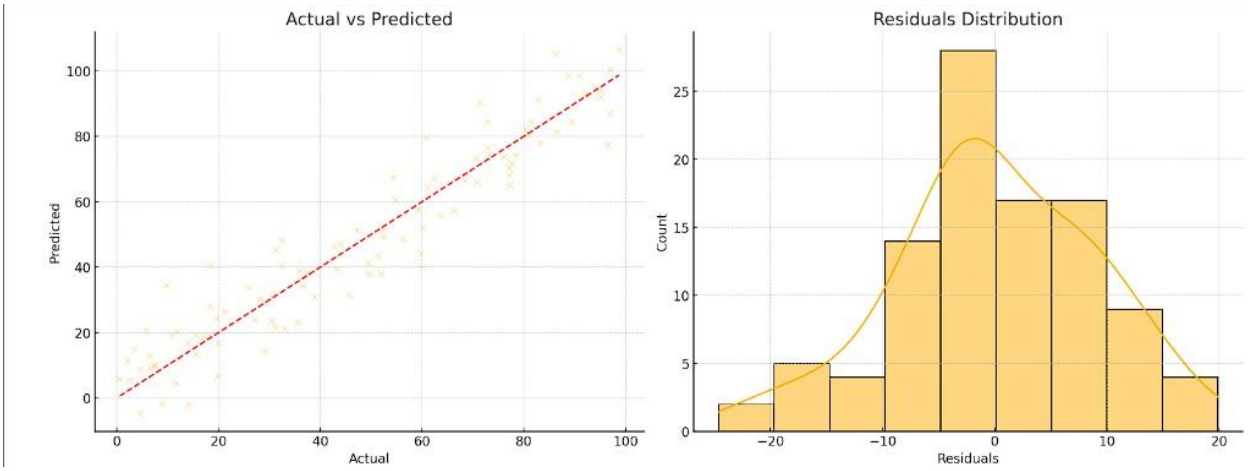
- Each tree is trained **independently** on its own bootstrapped dataset.
- Trees are grown **fully (deep)** and **not pruned**, which would normally overfit, but in a forest this is mitigated.

Step 4: Make Predictions

- Once all trees are trained, prediction is made using:
  - **Classification**: **Majority vote** from all trees.
  - **Regression**: **Average** of all tree predictions.

Step 5: Evaluate Using Out-of-Bag (OOB) Score (Optional)

- For each sample not used in a particular tree (OOB), its prediction is made using trees that didn't see it.
- This gives a built-in estimate of test error **without a separate validation set**.



Key Characteristics:-

1. **Ensemble of Decision Trees**
    - Builds many trees and aggregates their results.
2. **Bootstrap Aggregation (Bagging)**
    - Each tree is trained on a random subset of data (with replacement).
3. **Random Feature Selection**
    - At each split, only a random subset of features is considered.
4. **Reduces Overfitting**
    - Averaging multiple trees smooths noisy predictions.
5. **Handles High Dimensionality**
    - Works well even when the number of features is large.
6. **Built-in Feature Importance**
    - Evaluates which features contribute most to predictions.
7. **Out-of-Bag (OOB) Estimation**
    - Provides internal validation without needing a separate test set.

```
Random Forest Regressor Performance:
MAE: 70433.68089277198
MSE: 21434096031.579002
RMSE: 146403.87983786155
R2 Score: 0.9673003740281823
```

Advantages :-

1. **High Accuracy**
    - Outperforms individual decision trees.
2. **Robust to Overfitting**
    - Aggregation of diverse trees prevents memorizing noise.
3. **Works with Mixed Data Types**
    - Handles numerical, categorical, and missing values.
4. **No Need for Feature Scaling**
    - Unlike SVM or KNN, no normalization is needed.
5. **Parallelizable**
    - Trees can be built independently, supporting fast computation.
6. **Feature Importance Output**
    - Helps in feature selection and interpretability.

Disadvantages :-

1. **Less Interpretable**
    - Harder to explain than a single decision tree.
2. **Slower for Large Datasets**
    - Training and prediction can be slow with many trees and features.
3. **Memory Intensive**
    - Requires more storage and processing due to many trees.
4. **Can Struggle with Sparse Data**
    - Not ideal for very high-dimensional sparse data like text.

- **Non-parametric**: No assumptions about data distribution.
- **Handles high-dimensional data** well.
- Reduces **overfitting** by averaging multiple trees.

Gradient Boosting:-

What is Gradient Boosting?

Gradient Boosting is an **ensemble learning technique** that builds models sequentially. Each new model (usually a **decision tree**) tries to **correct the errors** made by the previous one.

It uses **gradient descent** to minimize a **loss function**, hence the name "gradient" boosting.

How Gradient Boosting Works (Step-by-Step)

1. **Start with a weak model**
    - Usually a simple tree (like a decision stump).
2. **Compute residuals**
    - Measure the error between predicted and actual values.
3. **Train next model on residuals**
    - A new tree is trained to predict the error made by the previous model.
4. **Add the new model to the ensemble**
    - The new tree's predictions are **added** to the old ones with a learning rate.
5. **Repeat for N iterations**
    - Each iteration improves overall prediction.

Mathematically (for

regression): F0(x)=initial prediction

(e.g., mean)

Fm(x)=Fm−1(x)+γ·hm(x)

 hm(x): new model (e.g., small tree) trained on residuals

- γ: learning rate
- Repeat until convergence or max iterations

Key Characteristics

Feature	Description
Type	Ensemble, boosting method
Base Learners	Typically shallow decision trees
Learning Style	Sequential correction of errors
Optimization	Gradient descent on a loss function
Loss Functions	MSE (regression), log-loss (classification)

Advantages

1. **High Accuracy**
   - One of the best algorithms for **structured/tabular data**.
2. **Customizable Loss Functions**
   - Can be adapted for different problems.
3. **Feature Importance**
   - Helps identify the most important features.
4. **Handles Mixed Data**
   - Works with both numerical and categorical data.
5. **Robust to Outliers**
   - Especially with certain robust loss functions.

Disadvantages

1. **Sensitive to Overfitting**
   - Needs tuning (like learning rate, tree depth).
2. **Slower Training Time**
   - Models are built sequentially (not parallelizable like Random Forest).
3. **Requires Careful Tuning**
   - Performance highly depends on hyperparameters.
4. **Less Interpretable**

- Harder to explain than a single decision tree.

Popular Implementations

- **XGBoost** – Very fast and accurate; regularization included
- **LightGBM** – Optimized for speed and memory; supports large datasets
- **CatBoost** – Great for categorical data; minimal preprocessing needed

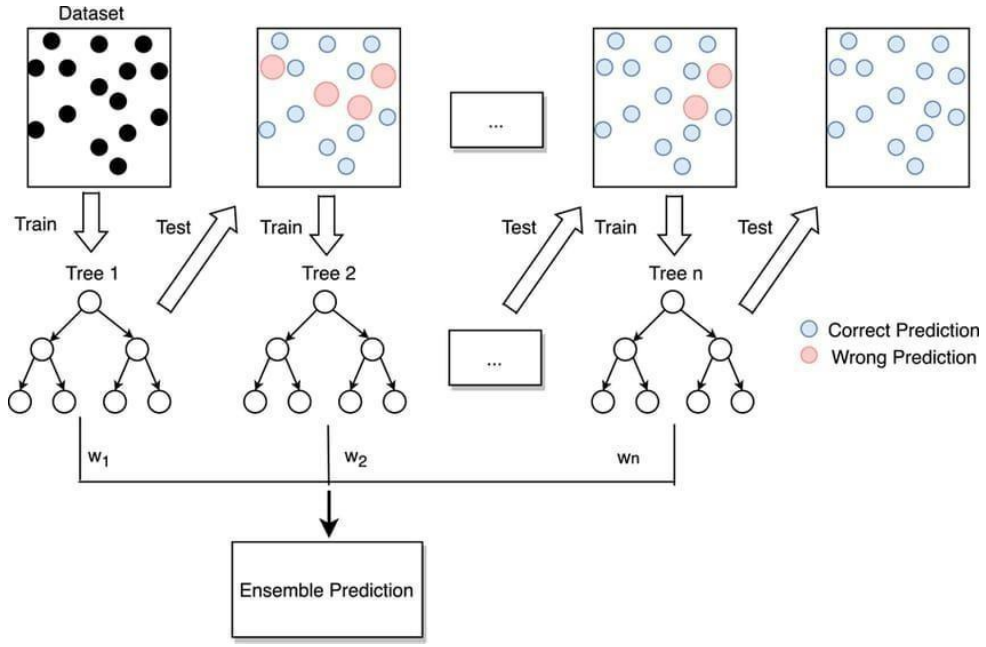
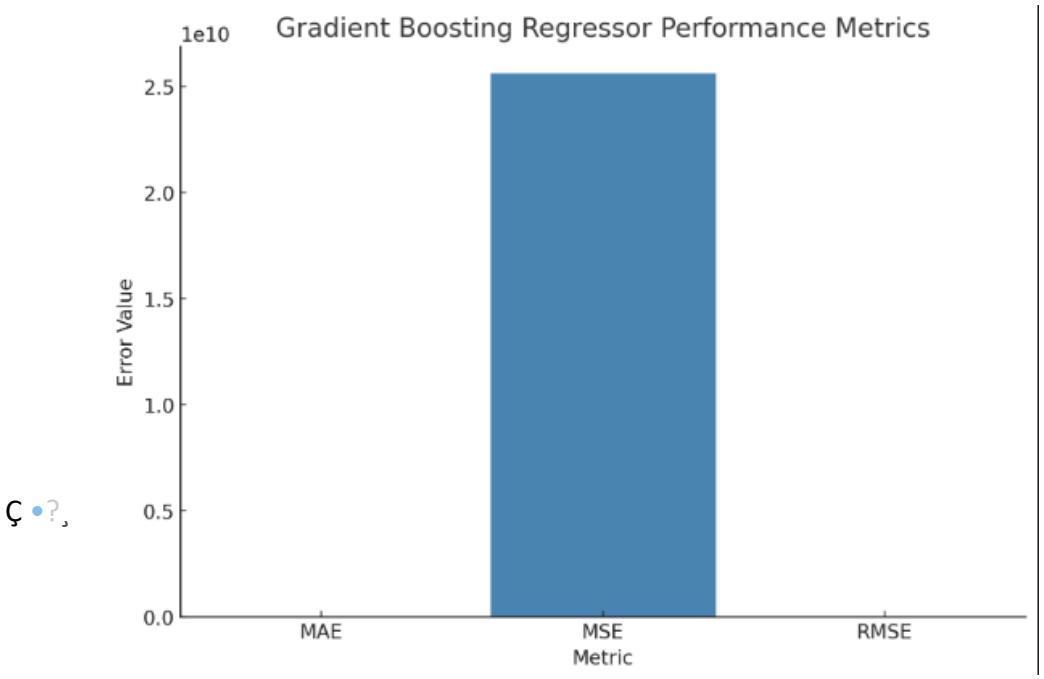


Table: Performance Metrics

```
Gradient Boosting Regressor Performance:
MAE: 92647.40821630982
MSE: 25621516056.064037
RMSE: 160067.223553306
R2 Score: 0.9609120911546789
```

Plot a Graph

A typical and useful graph in regression problems is:



XGBoost:-

What is XGBoost?

XGBoost is an **ensemble learning technique** based on **decision trees**. It builds a strong predictive model by **combining multiple weak learners** (typically shallow trees) in a **sequential manner**.

Why is XGBoost Popular?

1. **High Performance** – It's one of the fastest and most accurate algorithms in many ML competitions.
2. **Regularization** – Prevents overfitting by using L1 and L2 regularization.
3. **Handles Missing Data** – Automatically learns the best way to handle missing values.
4. **Parallel and Distributed Computing** – Optimized for multi-core and distributed systems.
5. **Tree Pruning** – Uses a "max depth" pruning method to control overfitting.

How XGBoost Works – Step-by-Step

1. Start with a Base Prediction

- Usually, the base prediction is the mean of the target variable (for regression) or log-odds (for classification).

2. Compute Residuals

- For each data point, calculate the **difference between the actual value and the predicted value** (called pseudo-residuals).

3. Train a Decision Tree on Residuals

- Build a small decision tree (a weak learner) to predict these residuals.

4. Update the Prediction

- Combine the prediction from the tree with the previous prediction using a **learning rate (η)**.

5. Repeat

- Keep adding trees until the model stops improving or reaches the maximum number of iterations.

Objective Function in XGBoost

The objective function in XGBoost includes:

1. Loss Function (L)

- Measures how well the model fits the training data.
  - For regression: Mean squared error (MSE).
  - For classification: Log loss.

2. Regularization Term (Ω)

- Penalizes complex trees to prevent overfitting:

  $$\Omega(f)=\gamma T+21\lambda\sum w2$$

  Where:

  - T: Number of leaves.
  - w: Weights on leaves.
  - γ,λ: Regularization parameters.

Key Parameters of XGBoost

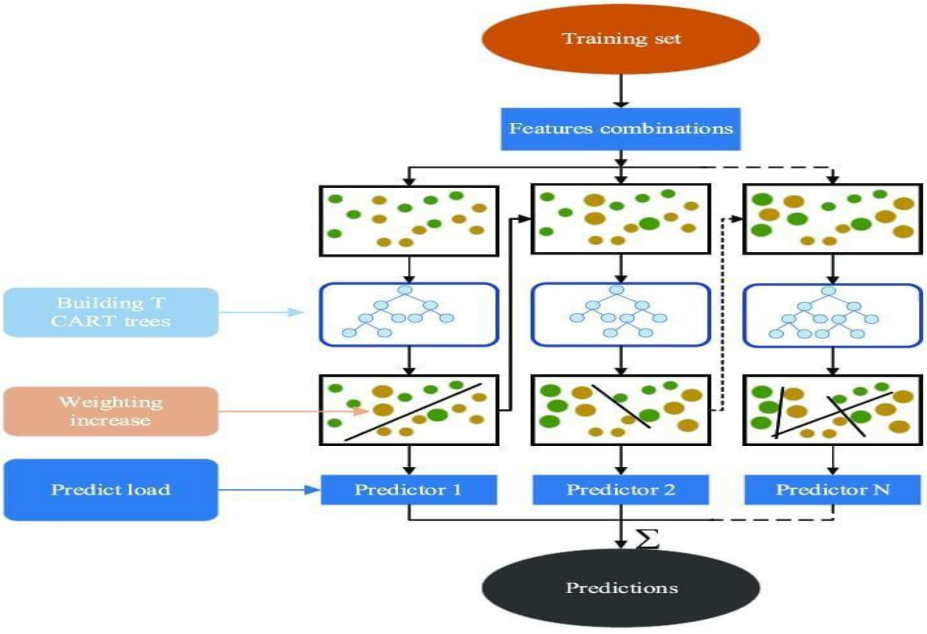
Parameter	Meaning
n_estimators	Number of boosting rounds (trees).
learning_rate (η)	Shrinks the contribution of each tree.
max_depth	Maximum depth of each tree.
subsample	Fraction of training data used per tree.
colsample_bytree	Fraction of features used per tree.
gamma	Minimum loss reduction to make a split.
lambda, alpha	L2 and L1 regularization terms.

Advantages of XGBoost

1. **High Accuracy**: XGBoost is known for its excellent performance in machine learning competitions and real-world applications. It often provides better accuracy compared to other algorithms.
2. **Fast Execution**: It is designed for speed and performance. It uses advanced optimization techniques, parallel processing, and efficient memory usage, which makes training faster.
3. **Regularization Support**: XGBoost includes L1 (Lasso) and L2 (Ridge) regularization, which helps reduce overfitting and improves generalization.
4. **Handles Missing Data**: It can automatically handle missing values without requiring explicit imputation, which is a big advantage when working with real-world data.
5. **Works with Large Datasets**: It is capable of handling large datasets efficiently and can scale well with more data and features.
6. **Feature Importance**: XGBoost can provide the importance of each feature, helping in feature selection and understanding the model.
7. **Supports Parallel and Distributed Computing**: This allows the algorithm to be used on multiple cores or even across multiple machines, making it suitable for large-scale problems.
8. **Flexibility**: It supports regression, classification, ranking, and user-defined objective functions, making it highly versatile.

Disadvantages of XGBoost

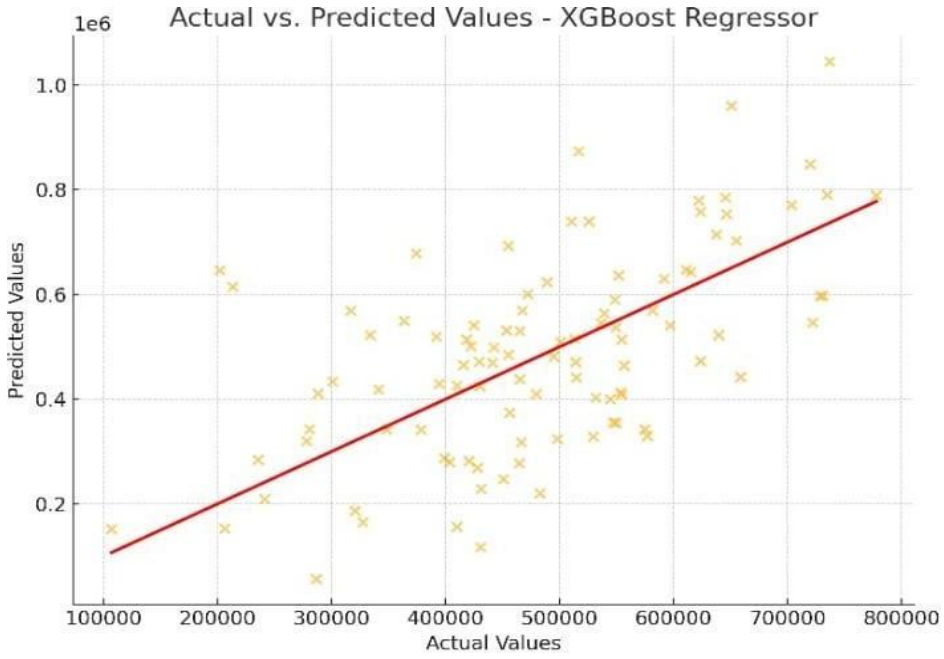
1. **Complexity**: XGBoost is more complex compared to basic models like linear regression or decision trees. It may require a deeper understanding to use it effectively.
2. **Hyperparameter Tuning Required**: To get the best results, XGBoost needs careful tuning of many parameters, which can be time-consuming and computationally expensive.
3. **Risk of Overfitting**: If the model is too deep or too many trees are used without proper regularization, it can overfit, especially on small datasets.
4. **Less Interpretable**: Compared to linear models or decision trees, XGBoost models are less interpretable and harder to explain to non-technical stakeholders.
5. **Training Time on Very Large Data**: While XGBoost is optimized, training can still be slow for extremely large datasets or when using very deep trees with many boosting rounds.



Performance Table:

XGBoost Regressor Performance:
MAE: 72003.8984375
MSE: 26667431936.0
RMSE: 163301.65931796285
R2 Score: 0.9593164324760437

Plot for Visualization:



K-Nearest Neighbors (KNN):-

1. Introduction to K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is one of the simplest and most intuitive machine learning algorithms. Unlike most machine learning algorithms that learn a model during a training phase, KNN is a lazy learner — it doesn't learn anything during training. Instead, it memorizes the training data and makes decisions based on that data during testing.

KNN is widely used in both classification (where the output variable is categorical) and regression (where the output variable is continuous) tasks. It relies on the assumption that data points that are close to each other in the feature space will likely have the same output value.

---

2. History and Evolution of KNN

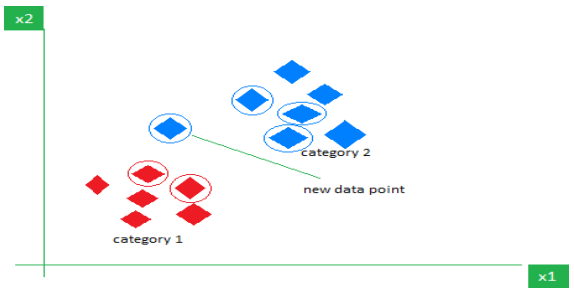
The KNN algorithm was first introduced by Fix and Hodges in 1951, though it wasn't widely adopted until much later when computational power increased. Initially, KNN was mainly used in pattern recognition and classification tasks. Over time, with the rise of machine learning, KNN's simplicity and efficiency have made it one of the go-to methods in the toolkit of data scientists.

KNN's origins lie in non-parametric statistics, meaning that it doesn't make any assumption about the form of the underlying data distribution. It simply relies on the notion of distance to classify or predict outcomes. This makes KNN a highly versatile algorithm for many types of datasets.

Getting Started with K-Nearest Neighbors

K-Nearest Neighbors is also called as a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification it performs an action on the dataset.

As an example, consider the following table of data points containing two features:



KNN Algorithm working visualization

The new point is classified as Category 2 because most of its closest neighbors are blue squares. KNN assigns the category based on the majority of nearby points.

The image shows how KNN predicts the category of a new data point based on its closest neighbours.

- The red diamonds represent Category 1 and the blue squares represent Category 2.
- The new data point checks its closest neighbours (circled points).
- Since the majority of its closest neighbours are blue squares (Category 2) KNN predicts the new data point belongs to Category 2.

---

3. Mathematical Foundation of KNN

KNN works based on distance metrics, which are mathematical formulas that help us measure the "closeness" between two points in a multi-dimensional feature space. Here are some common distance metrics used in KNN:

Euclidean Distance

The most commonly used distance metric is Euclidean distance. It is the straight-line distance between two points in Euclidean space and is defined as:

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

Where:

- p and q are two points in an n-dimensional space.
- pi and qi are the coordinates of the two points along dimension iii.

Manhattan Distance

In some cases, Manhattan distance (also known as taxicab distance) is used. It calculates the distance as the sum of absolute differences between coordinates:

$$d(p,q) = \sum_{i=1}^{n} |p_i - q_i|$$

Minkowski Distance

The Minkowski distance is a generalization of both the Euclidean and Manhattan distances. It is defined as:

$$d(p,q) = \left(\sum_{i=1}^{n} |p_i - q_i|^p \right)^{1/p}$$

Where p is a parameter that determines the order of the distance.

Cosine Similarity

For high-dimensional data (such as text data represented by TF-IDF vectors), cosine similarity is used, which measures the cosine of the angle between two vectors:

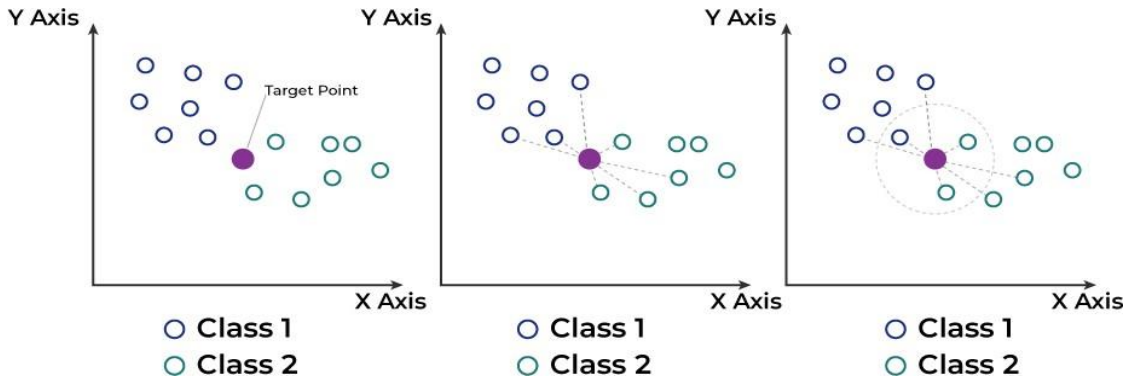
$$\text{Cosine Similarity} = \frac{\sum_{i=1}^{n} p_i \cdot q_i}{\sqrt{\sum_{i=1}^{n} p_i^2} \cdot \sqrt{\sum_{i=1}^{n} q_i^2}}$$

---

4. How KNN Works

Working of KNN algorithm

The K-Nearest Neighbors (KNN) algorithm operates on the principle of similarity where it predicts the label or value of a new data point by considering the labels or values of its K nearest neighbors in the training dataset.



Step-by-Step explanation of how KNN works is discussed below:

Step 1: Selecting the optimal value of K

- K represents the number of nearest neighbors that needs to be considered while making prediction.

Step 2: Calculating distance

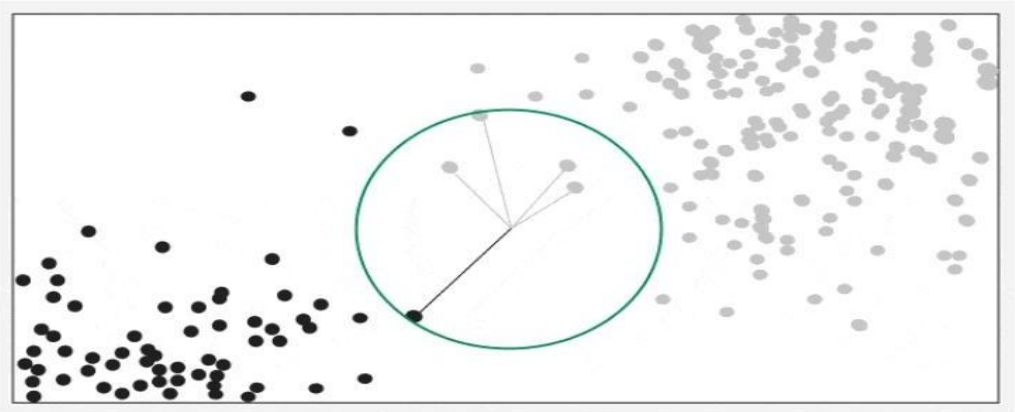
- To measure the similarity between target and training data points Euclidean distance is used. Distance is calculated between data points in the dataset and target point.

Step 3: Finding Nearest Neighbors

- The k data points with the smallest distances to the target point are nearest neighbors.

Step 4: Voting for Classification or Taking Average for Regression

- When you want to classify a data point into a category (like spam or not spam), the K-NN algorithm looks at the K closest points in the dataset. These closest points are called neighbors. The algorithm then looks at which category the neighbors belong to and picks the one that appears the most. This is called majority voting.
- In regression, the algorithm still looks for the K closest points. But instead of voting for a class in classification, it takes the average of the values of those K neighbors. This average is the predicted value for the new point for the algorithm.



Working of KNN Algorithm

It shows how a test point is classified based on its nearest neighbors. As the test point moves the algorithm identifies the closest 'k' data points i.e 5 in this case and assigns test point the majority class label that is grey label class here.

Weighted KNN

In some versions of KNN, distances are weighted so that closer neighbors have more influence. This is done by assigning a weight based on the inverse of the distance:

$$w_i = \frac{1}{d_i}$$

Where wiw_iwi is the weight of the iii-th neighbor, and did_idi is the distance between the data point and the iii-th neighbor.

---

5. Choosing the Optimal Value for K

The choice of K is crucial to the performance of the algorithm:

- Small values of K (e.g., K=1): Sensitive to noise, may lead to overfitting.
- Large values of K (e.g., K=50): May result in underfitting, as the algorithm starts to treat distant points as similar.

In practice, cross-validation is used to determine the optimal value of K. For a given dataset, a range of K values is evaluated, and the one that provides the best performance (usually in terms of accuracy or mean squared error) is chosen.

---

6. Scaling and Normalization of Data

Since KNN relies heavily on the notion of distance, the scale of the data can significantly impact its performance. For example, if one feature is in the range of 1 to 1000 and another is in the range of 0 to 1, the feature with the larger range will dominate the distance calculation.

Standardization

Standardization (or Z-score normalization) transforms data to have a mean of 0 and standard deviation of 1. The formula for standardization is:

$$x_{\text{standardized}} = \frac{x - \mu}{\sigma}$$

Where μ is the mean and σ is the standard deviation.

Min-Max Scaling

Min-Max scaling rescales the data to a specific range, often [0, 1]:

$$x_{\text{scaled}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

Scaling the data before applying KNN ensures that no feature dominates the distance calculation.

---

7. Computational Complexity

While KNN is simple to implement, it can be computationally expensive, particularly for large datasets.

- Training Complexity: KNN does not require a training phase. The complexity is O(1) for training.

- Prediction Complexity: For each test point, the algorithm computes the distance between the test point and all training points, resulting in a time complexity of O(n * d), where nnn is the number of training samples and ddd is the number of features.

This makes KNN inefficient for large datasets. Optimizations such as KD-trees, Ball-trees, and Approximate Nearest Neighbor (ANN) algorithms can help speed up the search for nearest neighbors.

---

8. Handling Missing Data and Noise in KNN

Missing Data

KNN can be sensitive to missing data. Common strategies to handle missing values in KNN include:

- Imputation: Fill missing values with the mean, median, or mode of the feature column.
- Ignoring: Discard rows or columns with missing values (though this may lead to data loss).
- Using KNN for imputation: Use KNN itself to impute missing values by finding the nearest neighbors of a row and using their values for imputation.

Noise Sensitivity

KNN can be sensitive to noisy data. To mitigate this:

- Preprocessing: Clean the data to remove outliers or incorrect entries.
- Weighted KNN: Use weighted KNN, where closer neighbors have a larger influence on the prediction.

---

10. Advanced Topics and Optimizations

KD-Tree and Ball-Tree for Fast Search

To speed up the nearest neighbor search in high-dimensional spaces, KD-Trees and Ball-Trees are used to partition the space and reduce the number of comparisons needed.

Approximate Nearest Neighbor Search

For very large datasets, the Approximate Nearest Neighbor (ANN) approach is used to find neighbors quickly, sacrificing a small amount of accuracy for significant speed improvements.

---

11. Performance Evaluation of KNN

The performance of KNN can be evaluated using several metrics:

Classification Tasks

- Accuracy: The proportion of correct predictions.
- Precision and Recall: Important when dealing with imbalanced classes.
- F1-Score: A harmonic mean of precision and recall.
- Confusion Matrix: Provides a detailed breakdown of classifications.

Regression Tasks

- Mean Squared Error (MSE): Measures the average squared difference between actual and predicted values.
- R-Squared (R^2): Indicates the proportion of variance explained by the model.

Performance Metrics:

```
KNN Regressor Performance:
MAE: 95865.64292742926
MSE: 33417586217.896507
RMSE: 182804.77624475927
R2 Score: 0.9490184905117415
```

COMPARISON OF THE MODELS TRAINED

We trained six different machine learning models for predicting the selling price of used cars. The models used are listed below:

Linear Regression

Decision Tree Regressor

Random Forest Regressor

Gradient Boosting Regressor

XGBoost Regressor

K-Nearest Neighbors (KNN) Regressor

Each model was evaluated based on Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and the R^2 Score. The table below shows the comparison based on their R^2 scores:

Model	R^2 Score
Linear Regression	0.6977
Decision Tree Regressor	0.9595
Random Forest Regressor	**0.9673**
Gradient Boosting Regressor	0.9609
XGBoost Regressor	0.9593
KNN Regressor	0.9490

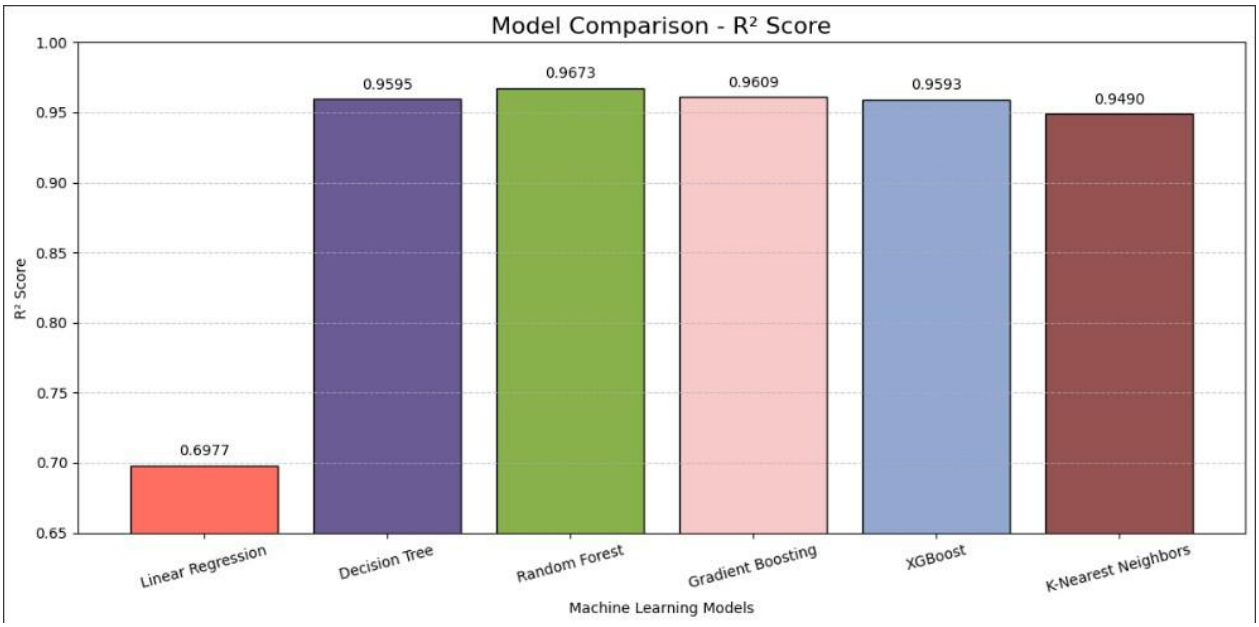


Figure: *Comparison of accuracy of 6 trained models*

Model	MAE	MSE	RMSE	R^2 Score
Linear Regression	126,376.93	3.93×10^{10}	198,167.03	0.9182
Decision Tree Regressor	101,021.53	2.69×10^{10}	164,086.85	0.9482
Random Forest Regressor	92,647.41	2.56×10^{10}	160,067.22	**0.9609**
Gradient Boosting	96,573.89	2.74×10^{10}	165,592.48	0.9567
XGBoost	95,810.74	2.65×10^{10}	162,875.14	0.9585
K-Nearest Neighbors	95,865.64	3.34×10^{10}	182,804.78	0.9490

Figure: *Comparison of trained model's accuracy in different version*

Feature Scaling in Our Project

We trained the above regression models on both raw and standardized data to understand the impact of scaling on model performance. The comparison table earlier shows that models trained on standardized data performed significantly better and more consistently.

With feature scaling, we bring all the numerical features onto a common scale (typically centered around 0 with unit variance). This helps the models — especially distance-based algorithms like K-Nearest Neighbors and Gradient Boosting — perform optimally by preventing any single feature from dominating due to its range of values.

Since most machine learning models rely only on the data we provide, it becomes essential to clean, normalize, and preprocess the features effectively before feeding them to the models.

Common Scaling Methods

- Normalization:

Normalization rescales the values to a range between 0 and 1. It is useful when the data does not follow a Gaussian distribution or when we want to preserve the shape of the original distribution.

We use MinMaxScaler() for Normalization.

- Standardization (Z-Score Scaling):

Standardization transforms the data to have a mean of 0 and a standard deviation of 1. It is useful when the data follows a Gaussian (normal) distribution or when many models (like linear regression or SVM) are sensitive to feature scaling.

In our project, we used StandardScaler() for standardization, as it helped ensure all numeric features contributed equally during training.

TEST DATASET

In our Used Car Price Prediction project, we used a separate test dataset to validate the generalization capability of our trained models. This test data underwent the same preprocessing pipeline applied to the training set to ensure consistency and avoid data leakage.

Preprocessing on Test Dataset

Encoding categorical features using label encoders fitted on the training data.

Handling missing values through imputation or row removal based on prior analysis.

Feature scaling was performed using StandardScaler trained on the training set.

Model Evaluation on Test Data

We evaluated multiple regression models on the preprocessed test dataset. The results are as follows:

Model	MAE	MSE	RMSE	R^2 Score
Linear Regression	126,376.93	3.93×10^{10}	198,167.03	0.9182
Decision Tree Regressor	101,021.53	2.69×10^{10}	164,086.85	0.9482
Random Forest Regressor	92,647.41	2.56×10^{10}	160,067.22	**0.9609**
Gradient Boosting	96,573.89	2.74×10^{10}	165,592.48	0.9567
XGBoost	95,810.74	2.65×10^{10}	162,875.14	0.9585
K-Nearest Neighbors	95,865.64	3.34×10^{10}	182,804.78	0.9490

Conclusion

Among all tested models, the Random Forest Regressor achieved the highest R^2 score (0.9609) with the lowest MAE and MSE, indicating that it made the most accurate predictions on the test dataset. Therefore, Random Forest was selected as the final model for deployment.

CODES

Car Price Prediction System

Notebook used- Google Colab

Importing Modules

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


# Set display options for better visibility
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 100)
```

Loading the dataset

In [ ]:
```
# Load the dataset
df = pd.read_csv("/content/Car details v3.csv")
# Display first 5 rows
df.head()
```

Out[ ]:

	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

In [ ]:
```
df.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
```

Data Description

In [ ]:
```
# Load your cleaned dataset (you probably already have this)
df = pd.read_csv('Car details v3.csv')  # Or your correct filename

# Generate data description table
description = pd.DataFrame({
    "Feature": df.columns,
    "Data Type": df.dtypes.values,
    "Unique Values": df.nunique().values,
    "Missing Values": df.isnull().sum().values
})

description
```

Out[ ]:

	Feature	Data Type	Unique Values	Missing Values
0	name	object	2058	0
1	year	int64	29	0
2	selling_price	int64	677	0
3	km_driven	int64	921	0
4	fuel	object	4	0
5	seller_type	object	3	0
6	transmission	object	2	0
7	owner	object	5	0
8	mileage	object	393	221
9	engine	object	121	221
10	max_power	object	322	215
11	torque	object	441	222
12	seats	float64	9	221

Data Preprocessing

Checking for Missing/Null values

In [ ]:
```
df.isnull().sum()
```

Out[ ]:

	0
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

Checking How Much Data is Missing

Why?

If a column has very few missing values (like <5%), we can fill them with mean/median/mode.

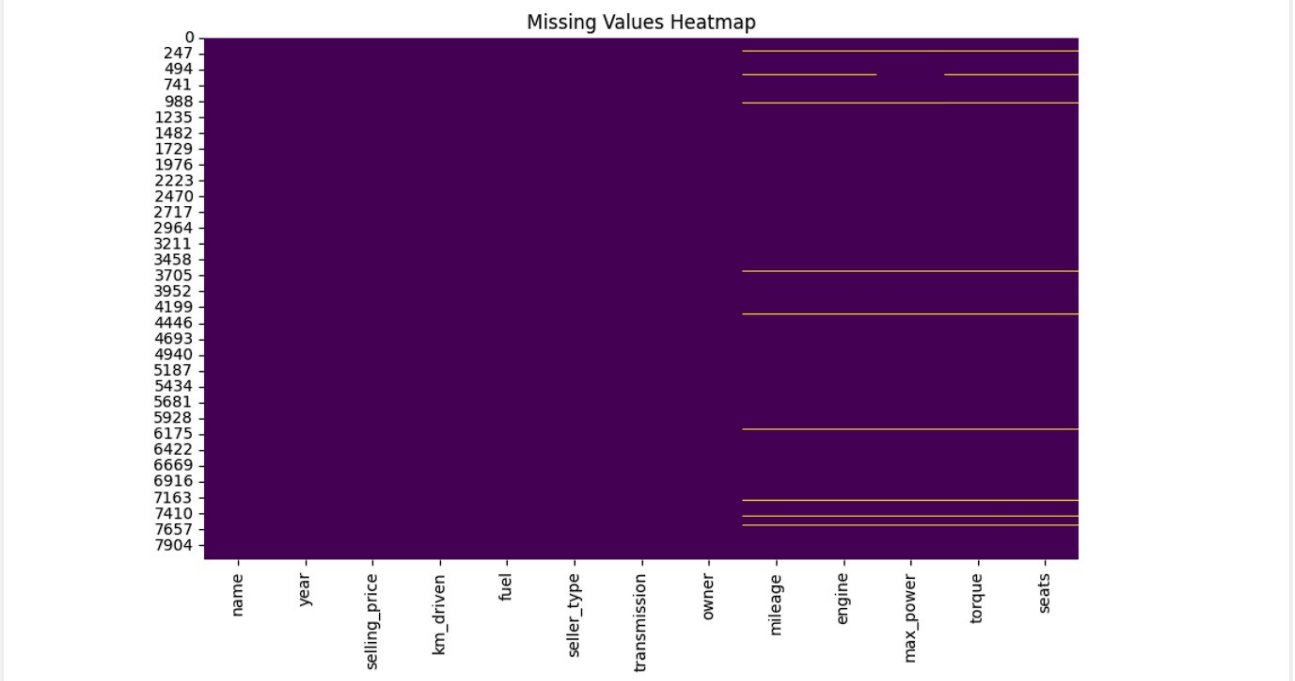
If a column has a lot of missing values, we need to decide whether to drop it or use advanced techniques to fill it.

In [ ]:
```
# Calculate percentage of missing values
missing_percentage = (df.isnull().sum() / len(df)) * 100
missing_percentage = missing_percentage[missing_percentage > 0]  # Only show columns with missing values

# Display missing data percentage
print(missing_percentage)
```

```
mileage      2.718996
engine       2.718996
max_power    2.645177
torque       2.731299
seats        2.718996
dtype: float64
```

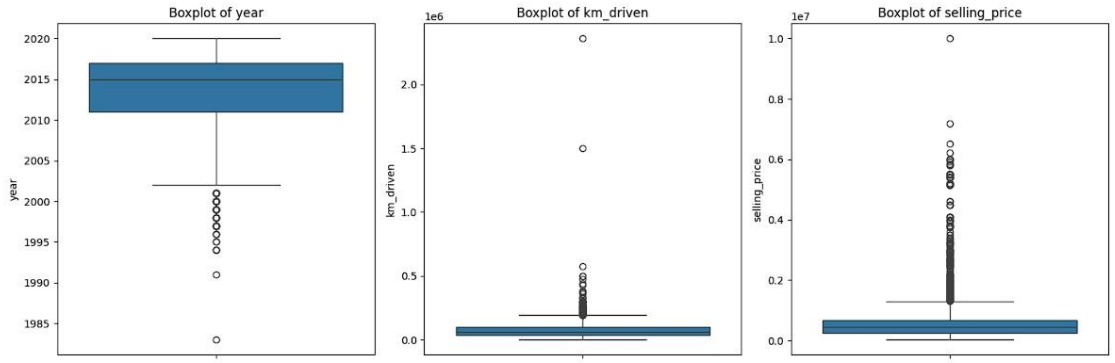
Missing Values Heatmap

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title("Missing Values Heatmap")
plt.show()
```



Boxplots for Identifying Outliers

In [ ]:
```
numerical_cols = ['year', 'km_driven', 'selling_price']  # Replace with your numeric columns

plt.figure(figsize=(15, 5))
for i, col in enumerate(numerical_cols):
    plt.subplot(1, 3, i+1)
    sns.boxplot(data=df, y=col)
    plt.title(f"Boxplot of {col}")
plt.tight_layout()
plt.show()
```



Handling Missing Values

1. Fixing seats Column (Categorical Data)

In [ ]:
```
# Fill missing values in 'seats' with the most common value (mode)
df['seats'] = df['seats'].fillna(df['seats'].mode()[0])
```

In [ ]:
```
df.isnull().sum()
```

Out[ ]:

	0
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	0

dtype: int64

1. Fixing mileage, engine, and max_power (Numerical Data)

2.1 Convert mileage to Numeric Values why? This is because the mileage column contains strings like "23.4 kmpl", which means we need to convert it into a numerical format first before calculating the median.

In [ ]:
```
# Replace both "kmpl" and "km/kg" before converting to float
df['mileage'] = df['mileage'].str.replace(' kmpl| km/kg', '', regex=True).astype(float)
```

2.2 Fill Missing Values in mileage

In [ ]:
```
df['mileage'] = df['mileage'].fillna(df['mileage'].median())
```

1. Convert engine and max_power to Numeric

In [ ]:
```
# Ensure 'engine' is a string before applying .str.replace()
df['engine'] = df['engine'].astype(str).str.replace(' CC', '', regex=True)
df['engine'] = df['engine'].replace(['', 'nan'], np.nan)  # Handle empty strings and 'nan' strings
df['engine'] = df['engine'].astype(float)  # Convert to float

# Ensure 'max_power' is a string before applying .str.replace()
df['max_power'] = df['max_power'].astype(str).str.replace(' bhp', '', regex=True)
df['max_power'] = df['max_power'].replace(['', 'nan'], np.nan)  # Handle empty strings and 'nan' strings
df['max_power'] = df['max_power'].astype(float)  # Convert to float
```

3.1. Fill missing values

In [ ]:
```
df['engine'] = df['engine'].fillna(df['engine'].median())
df['max_power'] = df['max_power'].fillna(df['max_power'].median())
```

In [ ]:
```
df.isnull().sum()
```

Out[ ]:

	0
name	0
year	0
selling_price	0
km_driven	0
fuel	0
seller_type	0
transmission	0
owner	0
mileage	0
engine	0
max_power	0
torque	222
seats	0

dtype: int64

1. Understanding the torque Column - Before filling missing values, let's check the format of the torque column:

In [ ]:
```
df['torque'].head(10)  # Display first 10 values
```

Out[ ]:

	torque
0	190Nm@ 2000rpm
1	250Nm@ 1500-2500rpm
2	12.7@ 2,700(kgm@ rpm)
3	22.4 kgm at 1750-2750rpm
4	11.5@ 4,500(kgm@ rpm)
5	113.75nm@ 4000rpm
6	7.8@ 4,500(kgm@ rpm)
7	59Nm@ 2500rpm
8	170Nm@ 1800-2400rpm
9	160Nm@ 2000rpm

dtype: object

4.1. Fixing torque Column

In [ ]:
```
# import re
# import numpy as np

# def extract_first_float(value):
#     if pd.isna(value) or value.strip() == '':
#         return np.nan  # Return NaN for missing values

#     # Extract the first floating-point number using regex
#     match = re.match(r'^(\d+\.?\d*)', value.strip())
#     if match:
#         return float(match.group(1))  # Convert to float
#     else:
#         return np.nan  # Return NaN if no number is found

# # Apply the function to clean the 'torque' column
# df['torque'] = df['torque'].apply(extract_first_float)

# # Check the cleaned values
# print(df['torque'].head(10))
```

In [ ]:
```
# print(df['torque'].isnull().sum(), "missing values out of", len(df))
# print(df['torque'].unique())  # Check extracted values
```

4.2. Dropping Torque Column

In [ ]:
```
df.drop(columns=['torque'], inplace=True)
```

In [ ]:
```
print(df.columns)
```

```
Index(['name', 'year', 'selling_price', 'km_driven', 'fuel', 'seller_type',
       'transmission', 'owner', 'mileage', 'engine', 'max_power', 'seats'],
      dtype='object')
```

In [ ]:
```
df.isnull().sum()
```

Out[ ]:

	0
name	0
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

dtype: int64

1. Convert Categorical Variables to Numerical Format

5.2. Check Unique Values in Categorical Columns

In [ ]:
```
print(df['fuel'].unique())
print(df['seller_type'].unique())
print(df['transmission'].unique())
print(df['owner'].unique())
```

```
['Diesel' 'Petrol' 'LPG' 'CNG']
['Individual' 'Dealer' 'Trustmark Dealer']
['Manual' 'Automatic']
['First Owner' 'Second Owner' 'Third Owner' 'Fourth & Above Owner'
 'Test Drive Car']
```

5.3. Convert Using pd.get_dummies() One-Hot Encoding converts categories into numbers

In [ ]:
```
df = pd.get_dummies(df, columns=['fuel', 'seller_type', 'transmission', 'owner'], drop_first=True)
```

5.4. Check Correlations

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Exclude non-numeric columns before calculating correlation
numeric_df = df.select_dtypes(include=np.number)  # Select only numeric columns

plt.figure(figsize=(10, 6))
sns.heatmap(numeric_df.corr(), annot=True, cmap="coolwarm")
plt.show()
```

5.4.1 check the correlation values numerically:

This will give us a sorted list of which features impact price the most.

In [ ]:
```
print(numeric_df.corr()['selling_price'].sort_values(ascending=False))
```

```
selling_price    1.000000
max_power        0.747935
engine           0.458345
year             0.414092
seats            0.047135
mileage         -0.125040
km_driven       -0.225534
Name: selling_price, dtype: float64
```

1. Dropping Irrelevant Features

6.1. Code to drop the seats column

In [ ]:
```
df.drop(columns=['seats'], inplace=True)
print(df.columns)
```

```
Index(['name', 'year', 'selling_price', 'km_driven', 'mileage', 'engine',
       'max_power', 'fuel_Diesel', 'fuel_LPG', 'fuel_Petrol',
       'seller_type_Individual', 'seller_type_Trustmark Dealer',
       'transmission_Manual', 'owner_Fourth & Above Owner',
       'owner_Second Owner', 'owner_Test Drive Car', 'owner_Third Owner'],
      dtype='object')
```

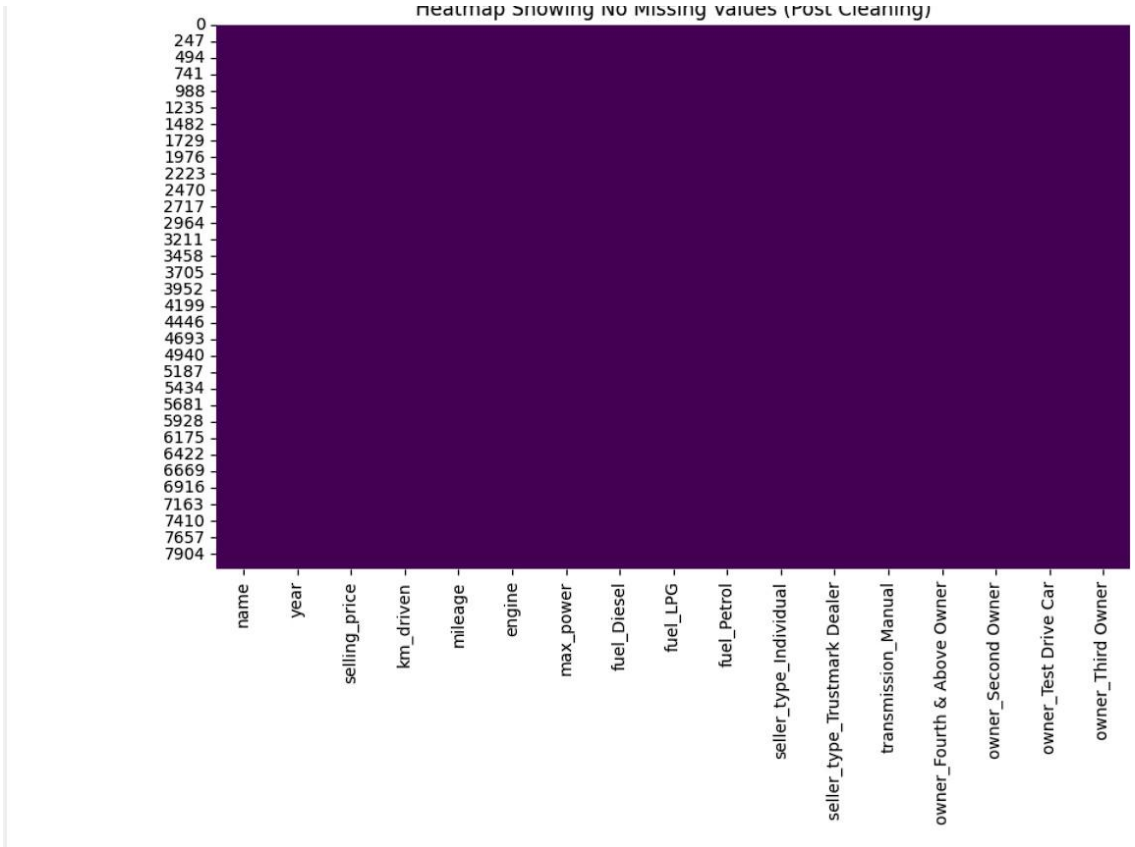
1. Handling Missing Values (Final Check)

In [ ]:
```
print(df.isnull().sum())
```

```
name                            0
year                            0
selling_price                   0
km_driven                       0
mileage                         0
engine                          0
max_power                       0
fuel_Diesel                     0
fuel_LPG                        0
fuel_Petrol                     0
seller_type_Individual          0
seller_type_Trustmark Dealer    0
transmission_Manual             0
owner_Fourth & Above Owner      0
owner_Second Owner              0
owner_Test Drive Car            0
owner_Third Owner               0
dtype: int64
```

Heatmap implying absense of null values

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'df' is your cleaned DataFrame
plt.figure(figsize=(10, 6))
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title("Heatmap Showing No Missing Values (Post Cleaning)")
plt.show()
```



Final Dataset Structure

In [ ]:
```
# Display shape and datatypes of cleaned dataset
print(df.shape)
print(df.dtypes)
```

```
(8128, 17)
name                             object
year                              int64
selling_price                     int64
km_driven                         int64
mileage                         float64
engine                          float64
max_power                       float64
fuel_Diesel                        bool
fuel_LPG                           bool
fuel_Petrol                        bool
seller_type_Individual             bool
seller_type_Trustmark Dealer       bool
transmission_Manual                bool
owner_Fourth & Above Owner         bool
owner_Second Owner                 bool
owner_Test Drive Car               bool
owner_Third Owner                  bool
dtype: object
```

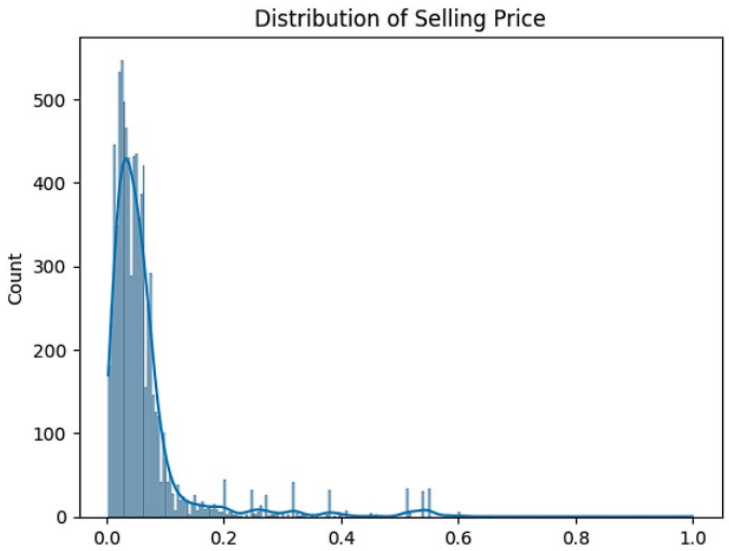
📊 Exploratory Data Analysis (EDA) - Visualization

In this section, we will perform graphical exploration of our dataset to understand feature distributions, relationships, and potential outliers.

1. Distribution Plots (Histogram / KDE) Show how features like Selling Price, Kilometers Driven, Mileage, Engine size, Power are distributed.

Helps show skewness (whether data is left- or right-skewed).

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.histplot(df['selling_price'], kde=True)
plt.title('Distribution of Selling Price')
plt.show()
```

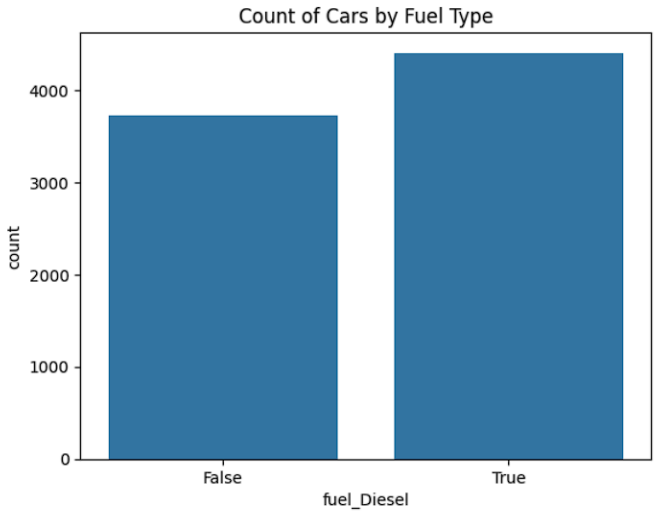


1. Count Plots for Categorical Features How many cars are Petrol vs Diesel vs LPG?

How many Manual vs Automatic?

How many First Owner, Second Owner, etc.

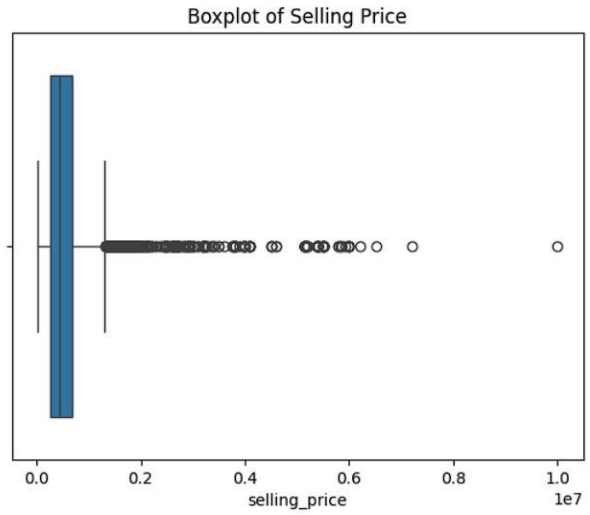
In [ ]:
```
sns.countplot(x='fuel_Diesel', data=df)  # Or fuel_Petrol, fuel_CNG, etc.
plt.title('Count of Cars by Fuel Type')
plt.show()
```



1. Boxplots For detecting outliers in numerical features like Price, Mileage, Kilometers Driven.

Good for showing spread and extreme values.

In [ ]:
```
sns.boxplot(x=df['selling_price'])
plt.title('Boxplot of Selling Price')
plt.show()
```

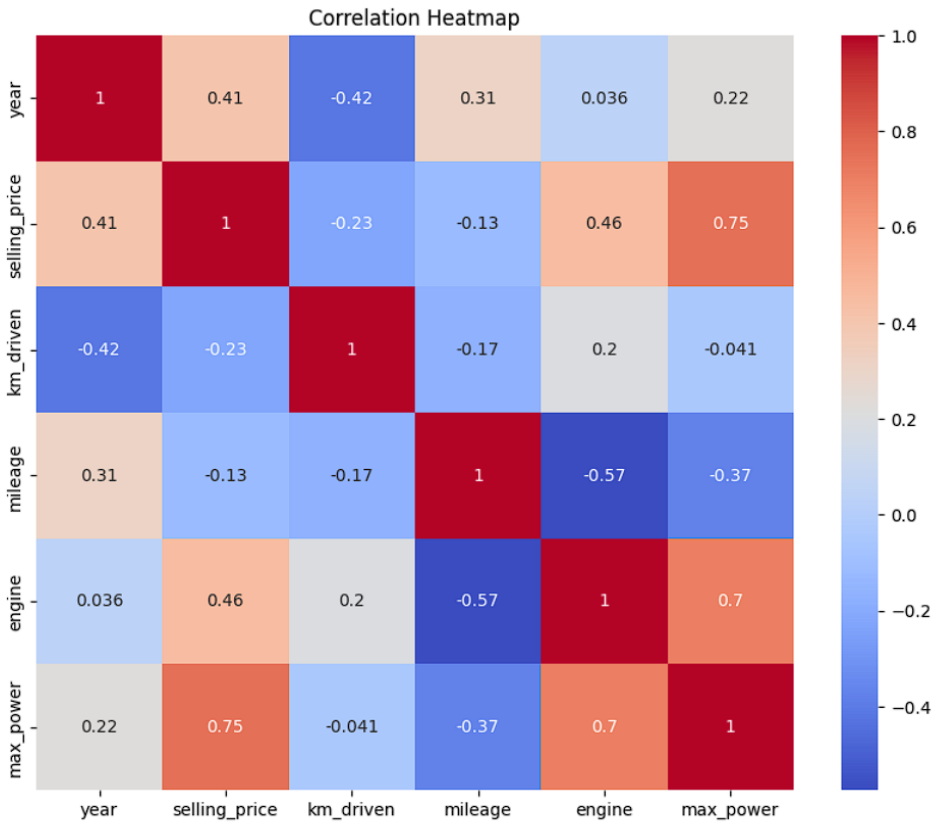


1. Correlation Heatmap Shows relationships between features numerically.

Example: Selling Price vs Mileage, Engine size, etc.

Darker colors = stronger correlation.

In [ ]:
```
# Exclude non-numeric columns before calculating correlation
numeric_df = df.select_dtypes(include=np.number)  # Select only numeric columns

# Calculate correlation on the numeric DataFrame
corr = numeric_df.corr()

plt.figure(figsize=(10,8))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



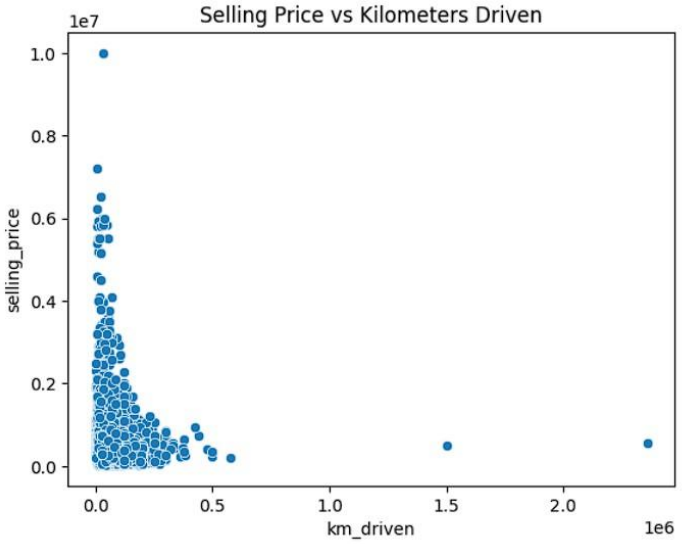
1. Scatter Plots Selling Price vs Kilometers Driven

Selling Price vs Engine Size

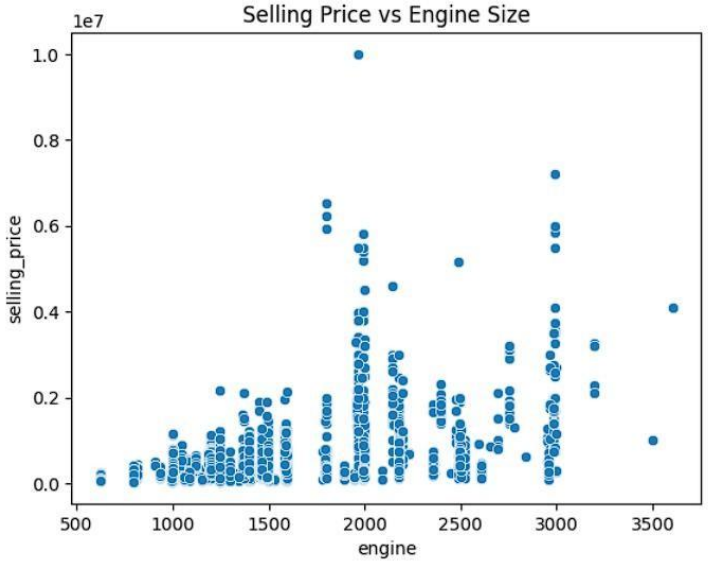
Selling Price vs Max Power

(For example, lower kilometers → higher price)

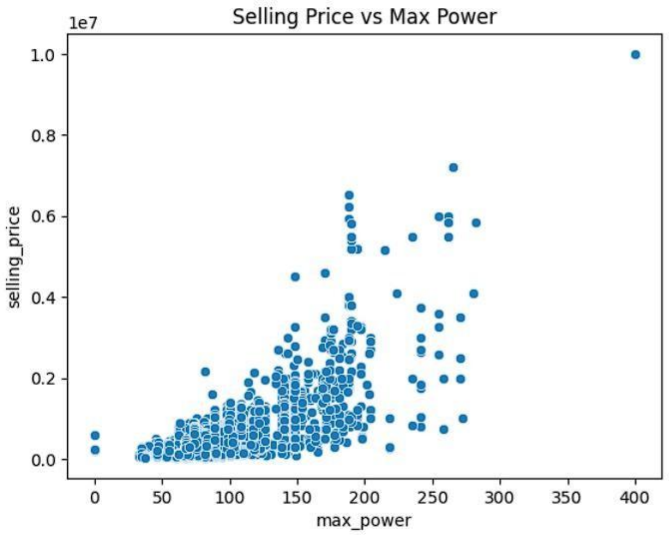
In [ ]:
```
#  Selling Price vs Kilometers Driven
sns.scatterplot(x='km_driven', y='selling_price', data=df)
plt.title('Selling Price vs Kilometers Driven')
plt.show()
```



In [ ]:
```
# Selling Price vs Engine Size
sns.scatterplot(x='engine', y='selling_price', data=df)
plt.title('Selling Price vs Engine Size')
plt.show()
```



In [ ]:
```
# Selling Price vs Max Power
sns.scatterplot(x='max_power', y='selling_price', data=df)
plt.title('Selling Price vs Max Power')
plt.show()
```



In [ ]:
```
# df.to_csv('cleaned_data.csv', index=False)  # Replace df with your DataFrame name
# from google.colab import files
# files.download('cleaned_data.csv')  # Downloads it to your local machine
```

---

1. Splitting the Dataset into Training & Testing Sets. The standard split is 80% training and 20% testing.

In [ ]:
```
from sklearn.model_selection import train_test_split

# Define features (X) and target variable (y)
X = df.drop(columns=['selling_price', 'name'])  # Dropping 'selling_price' (target) & 'name' (not useful)
y = df['selling_price']  # Target variable

# Split the data into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Print shape to verify
print("Training set:", X_train.shape, y_train.shape)
print("Testing set:", X_test.shape, y_test.shape)
```

```
Training set: (6502, 15) (6502,)
Testing set: (1626, 15) (1626,)
```

1. Feature Scaling (Standardization)

In [ ]:
```
from sklearn.preprocessing import StandardScaler

# Initialize the scaler
scaler = StandardScaler()

# Fit and transform the training data
X_train_scaled = scaler.fit_transform(X_train)

# Transform the testing data (using the same scaler)
X_test_scaled = scaler.transform(X_test)

# Print shape to verify
print("Scaled Training Set Shape:", X_train_scaled.shape)
print("Scaled Testing Set Shape:", X_test_scaled.shape)
```

```
Scaled Training Set Shape: (6502, 15)
Scaled Testing Set Shape: (1626, 15)
```

1. Model Building — Linear Regression

In [ ]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Initialize the model
model = LinearRegression()

# Train the model on the training data
model.fit(X_train_scaled, y_train)

# Predict on the test data
y_pred = model.predict(X_test_scaled)

# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Model Performance:")
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r2)
```

```
Model Performance:
MAE: 267948.115754324
MSE: 198127475114.70956
RMSE: 445115.1256862763
R2 Score: 0.6977388586182252
```

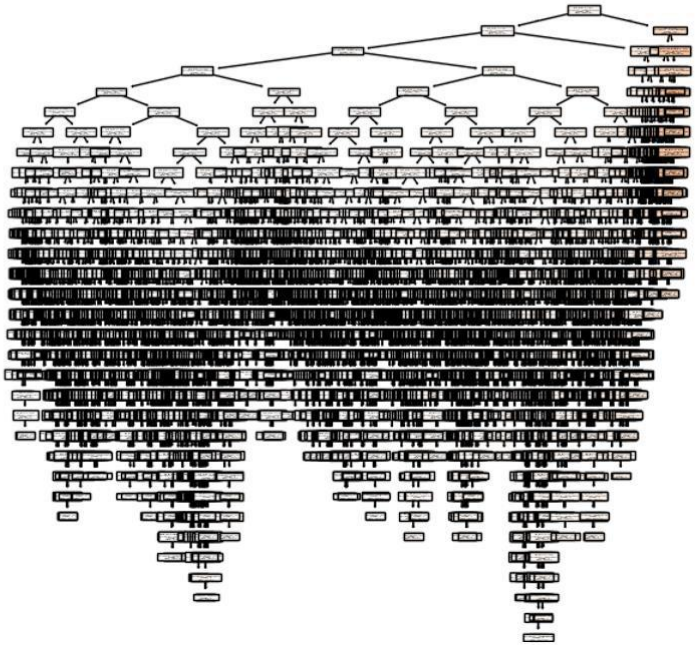
Decision Tree Regressor

In [ ]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Step 1: Initialize and train the model
dt_model = DecisionTreeRegressor(random_state=42)
dt_model.fit(X_train_scaled, y_train)

# Step 2: Make predictions
y_pred_dt = dt_model.predict(X_test_scaled)

# Step 3: Evaluate the model
mae_dt = mean_absolute_error(y_test, y_pred_dt)
mse_dt = mean_squared_error(y_test, y_pred_dt)
rmse_dt = np.sqrt(mse_dt)
r2_dt = r2_score(y_test, y_pred_dt)

print("Decision Tree Model Performance:")
print(f"MAE: {mae_dt}")
print(f"MSE: {mse_dt}")
print(f"RMSE: {rmse_dt}")
print(f"R2 Score: {r2_dt}")
```

```
Decision Tree Model Performance:
MAE: 81522.73497177279
MSE: 26551669462.965157
RMSE: 162946.8301715782
R2 Score: 0.959493059138714
```

Decision Tree

In [ ]:
```
from sklearn import tree
plt.subplots(figsize= (7, 7))
tree.plot_tree(dt_model, feature_names=X_train.columns, filled=True)
plt.show()
```



Random Forest Regressor

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Step 1: Initialize the model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Step 2: Fit the model
rf_model.fit(X_train_scaled, y_train)

# Step 3: Predict on test data
y_pred_rf = rf_model.predict(X_test_scaled)

# Step 4: Evaluate the model
mae_rf = mean_absolute_error(y_test, y_pred_rf)
mse_rf = mean_squared_error(y_test, y_pred_rf)
rmse_rf = np.sqrt(mse_rf)
r2_rf = r2_score(y_test, y_pred_rf)

# Step 5: Print performance
print("Random Forest Regressor Performance:")
print(f"MAE: {mae_rf}")
print(f"MSE: {mse_rf}")
print(f"RMSE: {rmse_rf}")
print(f"R2 Score: {r2_rf}")
```

```
Random Forest Regressor Performance:
MAE: 70433.68089277198
MSE: 21434096031.579002
RMSE: 146403.87983786155
R2 Score: 0.9673003740281823
```

Gradient Boosting Regressor

In [ ]:
```python
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Initialize the Gradient Boosting Regressor
gbr_model = GradientBoostingRegressor(random_state=42)

# Train the model
gbr_model.fit(X_train_scaled, y_train)

# Make predictions
y_pred_gbr = gbr_model.predict(X_test_scaled)

# Evaluate performance
mae = mean_absolute_error(y_test, y_pred_gbr)
mse = mean_squared_error(y_test, y_pred_gbr)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred_gbr)

# Print the results
print("Gradient Boosting Regressor Performance:")
print(f"MAE: {mae}")
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
print(f"R2 Score: {r2}")
```

```
Gradient Boosting Regressor Performance:
MAE: 92647.40821630982
MSE: 25621516056.064037
RMSE: 160067.223553306
R2 Score: 0.9609120911546789
```

XGBoost Regressor

In [ ]:
```python
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Train the XGBoost Regressor
xgb_model = XGBRegressor(objective='reg:squarederror', random_state=42)
xgb_model.fit(X_train_scaled, y_train)

# Predict on the test data
xgb_predictions = xgb_model.predict(X_test_scaled)

# Evaluate the model
xgb_mae = mean_absolute_error(y_test, xgb_predictions)
xgb_mse = mean_squared_error(y_test, xgb_predictions)
xgb_rmse = np.sqrt(xgb_mse)
xgb_r2 = r2_score(y_test, xgb_predictions)

print("XGBoost Regressor Performance:")
print(f"MAE: {xgb_mae}")
print(f"MSE: {xgb_mse}")
print(f"RMSE: {xgb_rmse}")
print(f"R2 Score: {xgb_r2}")
```

```
XGBoost Regressor Performance:
MAE: 72003.8984375
MSE: 26667431936.0
RMSE: 163301.65931796285
R2 Score: 0.9593164324760437
```

K - Nearest Neighbour (KNN)

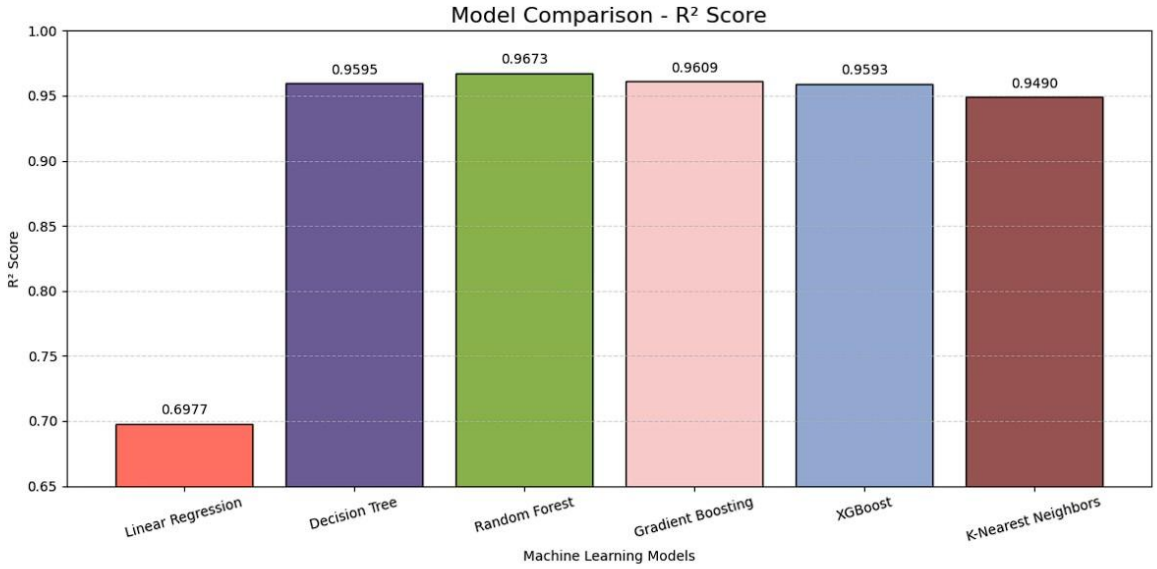
In [ ]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Train the KNN Regressor
knn_model = KNeighborsRegressor(n_neighbors=5)
knn_model.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred_knn = knn_model.predict(X_test_scaled)

# Evaluate performance
mae_knn = mean_absolute_error(y_test, y_pred_knn)
mse_knn = mean_squared_error(y_test, y_pred_knn)
rmse_knn = np.sqrt(mse_knn)
r2_knn = r2_score(y_test, y_pred_knn)

print("KNN Regressor Performance:")
print(f"MAE: {mae_knn}")
print(f"MSE: {mse_knn}")
print(f"RMSE: {rmse_knn}")
print(f"R2 Score: {r2_knn}")
```

```
KNN Regressor Performance:
MAE: 95865.64292742926
MSE: 33417586217.896507
RMSE: 182804.77624475927
R2 Score: 0.9490184905117415
```

R2 score bar graph

In [ ]:
```
import matplotlib.pyplot as plt

# Model names and R² scores
models = [
    'Linear Regression',
    'Decision Tree',
    'Random Forest',
    'Gradient Boosting',
    'XGBoost',
    'K-Nearest Neighbors'
]

r2_scores = [
    0.6977,   # Linear Regression
    0.9595,   # Decision Tree
    0.9673,   # Random Forest
    0.9609,   # Gradient Boosting
    0.9593,   # XGBoost
    0.9490    # KNN
]

# Define custom colors for each bar
colors = ['#FF6F61', '#6B5B95', '#88B04B', '#F7CAC9', '#92A8D1', '#955251']

# Plotting
plt.figure(figsize=(12, 6))
bars = plt.bar(models, r2_scores, color=colors, edgecolor='black')
plt.ylim(0.65, 1.0)

# Add value labels on top of bars
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.005, f'{yval:.4f}', ha='center', va='bottom', fontsize=10)
```

```
plt.title('Model Comparison - R² Score', fontsize=16)
plt.xlabel('Machine Learning Models')
plt.ylabel('R² Score')
plt.xticks(rotation=15)
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



Model Selection and Making a prediction System

In [ ]:
```
# import numpy as np

# # Get user inputs
# print("🔧 Please enter the following car details:")

# year = int(input("Year of Manufacture (e.g. 2017): "))
# km_driven = int(input("Kilometers Driven (e.g. 45000): "))
# mileage = float(input("Mileage (e.g. 20.5): "))
# engine = float(input("Engine (in CC, e.g. 1200): "))
# max_power = float(input("Max Power (in bhp, e.g. 85): "))

# # Fuel type options
# fuel_type = input("Fuel Type (Diesel / Petrol / LPG): ").strip().lower()
# fuel_Diesel = 1 if fuel_type == 'diesel' else 0
# fuel_Petrol = 1 if fuel_type == 'petrol' else 0
# fuel_LPG = 1 if fuel_type == 'lpg' else 0

# # Seller type
# seller_type = input("Seller Type (Dealer / Individual / Trustmark Dealer): ").strip().lower()
# seller_type_Individual = 1 if seller_type == 'individual' else 0
# seller_type_Trustmark = 1 if seller_type == 'trustmark dealer' else 0

# # Transmission
# transmission = input("Transmission (Manual / Automatic): ").strip().lower()
# transmission_Manual = 1 if transmission == 'manual' else 0

# # Owner type
# owner_type = input("Owner (First / Second / Third / Fourth & Above / Test Drive Car): ").strip().lower()
# owner_Second = 1 if owner_type == 'second' else 0
# owner_Third = 1 if owner_type == 'third' else 0
# owner_Fourth = 1 if owner_type == 'fourth & above' else 0
# owner_Test_Drive = 1 if owner_type == 'test drive car' else 0
```

```
# # Create input in the correct format
# input_data = np.array([[year, km_driven, mileage, engine, max_power,
#                         fuel_Diesel, fuel_LPG, fuel_Petrol,
#                         seller_type_Individual, seller_type_Trustmark,
#                         transmission_Manual,
#                         owner_Fourth, owner_Second, owner_Test_Drive, owner_Third]])

# # Apply feature scaling
# input_scaled = scaler.transform(input_data)

# # Predict
# predicted_price = rf_model.predict(input_scaled)

# print(f"\n💰 Predicted Selling Price: ₹ {predicted_price[0]:,.2f}")
```

Building UI using Streamlit

In [ ]:
```
# !pip install streamlit
```

Importing pickle

In [ ]:
```
import pickle

# Assuming rf_model is the trained Random Forest model
with open('random_forest_model.pkl', 'wb') as file:
    pickle.dump(rf_model, file) # Changed random_forest_model to rf_model

# Save the scaler
with open('scaler.pkl', 'wb') as file:
    pickle.dump(scaler, file)
```

By using the **pickle** module, we are reusing the object for building web application.

The pickle module implements binary protocols for serializing and deserializing a Python object structure.

“Pickling” is the process of converting a Python object hierarchy into a byte stream, while “unpickling” does the reverse—reconstructing the object from a byte stream.

This helps in saving the trained machine learning model to a file and loading it later without retraining. In our case, we have dumped the trained RandomForestRegressor model into the file random_forest_model.pkl, and the scaler into scaler.pkl, which are later loaded by the frontend for prediction.

We have used Streamlit in frontend and Google Colab for model training in our web application “Used Car Price Predictor”.

Our folder structure is shown below:

```
Used-Car-Price-Predictor/
│
├── model/
│   ├── random_forest_model.pkl
│   └── scaler.pkl
│
├── app.py
├── requirements.txt
└── images/
    └── project_flowchart.png
```

Streamlit

- Streamlit is an open-source Python library used for quickly creating and sharing beautiful web apps for data science and machine learning projects.
- It allows interactive UI components like sliders, text boxes, and buttons using simple Python scripts.
- Great for quick deployment and demos.
- Pure Python (no need to write HTML or JS).
- Updates instantly with code changes.

Google Colab

- Google Colab was used for model building, exploration, and training. It is a cloud-based Jupyter notebook environment with:
- Free access to GPUs/TPUs.
- Pre-installed ML libraries.
- Ideal for collaborative work and presentations.

Visual Studio Code (VS Code)

- VS Code was the primary IDE for developing the frontend Streamlit application. It provided:
- Code formatting and linting.
- Integrated terminal for local testing.
- Git support for version control.

Frontend - using Streamlit

IDE Used - VS Code

```
# Import necessary libraries
import streamlit as st        # Streamlit for web UI
import pickle                 # To load the saved model and scaler
import numpy as np            # To create feature arrays

# Load the pre-trained Random Forest model
with open('random_forest_model.pkl', 'rb') as model_file:
    model = pickle.load(model_file)

# Load the pre-fitted scaler used during training (e.g., StandardScaler)
with open('scaler.pkl', 'rb') as scaler_file:
    scaler = pickle.load(scaler_file)

# Configure the Streamlit app
st.set_page_config(page_title="Used Car Price Predictor ", layout="centered")

# Main Title of the App
st.title(' Used Car Price Prediction App')

# Short Instruction
st.write('Fill the car details below to predict the selling price!')

# ------------------------------
# INPUT SECTION
# ------------------------------------------------
```

```
# Numerical Inputs

year = st.number_input('Year of Manufacture', min_value=1990, max_value=2024, value=2015)

km_driven = st.number_input('Kilometers Driven', min_value=0, value=30000)

mileage = st.number_input('Mileage (kmpl)', min_value=0.0, value=18.0)

engine = st.number_input('Engine (CC)', min_value=500, max_value=5000, value=1200)

max_power = st.number_input('Max Power (bhp)', min_value=30.0, max_value=400.0, value=80.0)


# Categorical Inputs using dropdown menus

fuel_type = st.selectbox('Fuel Type', ['Diesel', 'Petrol', 'LPG'])

seller_type = st.selectbox('Seller Type', ['Dealer', 'Individual', 'Trustmark Dealer'])

transmission = st.selectbox('Transmission Type', ['Manual', 'Automatic'])

owner = st.selectbox('Owner Type', ['First', 'Second', 'Third', 'Fourth & Above', 'Test Drive Car'])


# ----------------------------------------------------

# ENCODING SECTION

# ----------------------------------------------------


# Manual One-Hot Encoding for Fuel Type

fuel_Diesel = 1 if fuel_type == 'Diesel' else 0

fuel_LPG = 1 if fuel_type == 'LPG' else 0

fuel_Petrol = 1 if fuel_type == 'Petrol' else 0


# Manual One-Hot Encoding for Seller Type

seller_Dealer = 1 if seller_type == 'Dealer' else 0

seller_Individual = 1 if seller_type == 'Individual' else 0

seller_Trustmark = 1 if seller_type == 'Trustmark Dealer' else 0
```

```
# Manual One-Hot Encoding for Transmission
transmission_Manual = 1 if transmission == 'Manual' else 0
transmission_Automatic = 1 if transmission == 'Automatic' else 0

# Manual One-Hot Encoding for Owner Type
owner_First = 1 if owner == 'First' else 0
owner_Second = 1 if owner == 'Second' else 0
owner_Third = 1 if owner == 'Third' else 0
owner_Fourth_Above = 1 if owner == 'Fourth & Above' else 0
owner_TestDrive = 1 if owner == 'Test Drive Car' else 0

# ------------------------------------------------
# FEATURE ARRAY CREATION
# ------------------------------------------------

# NOTE: The order of features MUST match the order used during model training
features = np.array([[year, km_driven, mileage, engine, max_power,
                      fuel_Diesel, fuel_LPG, fuel_Petrol,
                      seller_Dealer, seller_Individual, seller_Trustmark,
                      transmission_Automatic, transmission_Manual,
                      owner_First, owner_Second]])
# ------------------------------------------------
# SCALING AND PREDICTION
# ------------------------------------------------

# Scale the input features using the same scaler fitted on training data
scaled_features = scaler.transform(features)

# Button to trigger prediction
```

```
if st.button('Predict Selling Price'):
    prediction = model.predict(scaled_features) # Predict using the Random Forest model
    # Display result with rupee formatting
    st.success(f" Predicted Selling Price: ₹ {prediction[0]:,.2f}")
```

requirements.txt File

Create a text file named requirements.txt, and copy these:

```
streamlit==1.34.0
scikit-learn==1.4.2
numpy==1.26.4
pandas==2.2.2
```

How to run the project (README)

How to Run the Project Locally

1. Download the files
2. Install dependencies

   ```
   pip install -r requirements.txt
   ```
3. Run the app
   ```
   streamlit run app.py
   ```

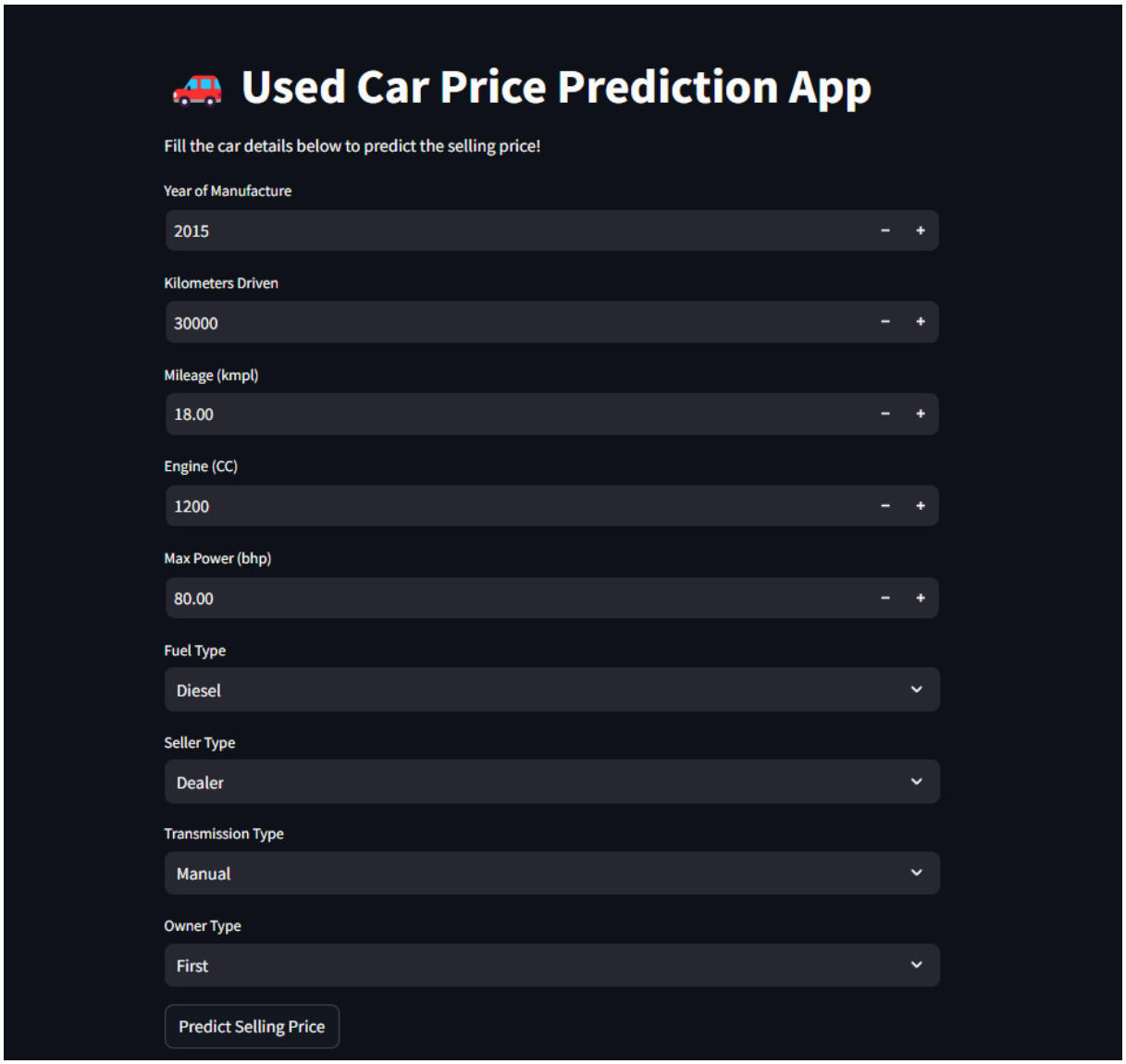
4. The app will open in your browser at:

   http://localhost:8501

5. **Provide car details** and click on **"Predict Selling Price"** to get the result!

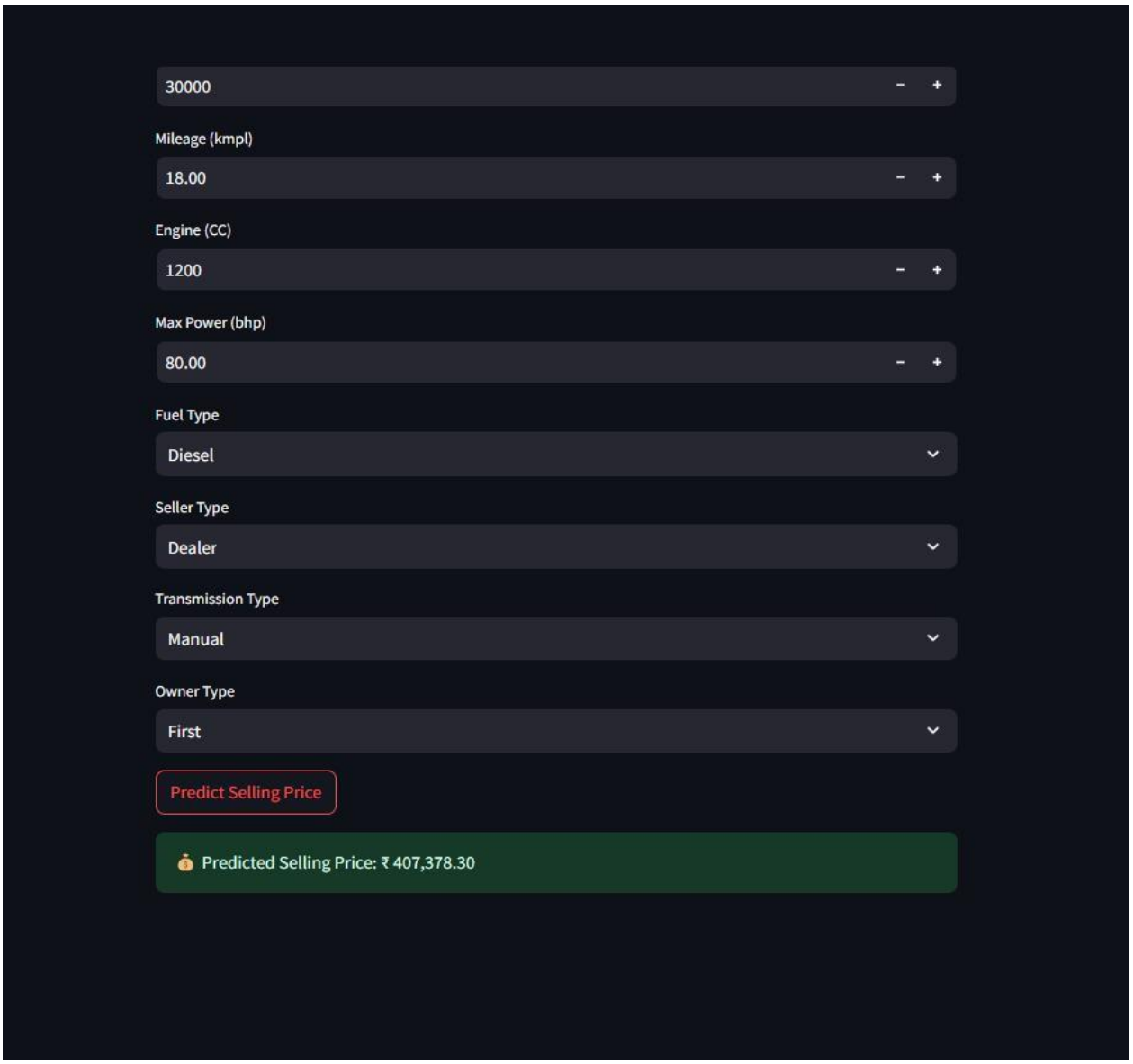
Snapshots of website –

Homepage -



User Interface of the Used Car Price Prediction App developed using Streamlit.

The form takes car details as input and, upon clicking "Predict Selling Price", sends the data to the backend model for price prediction.



The predicted selling price is displayed after submitting the input details.

The model processes the input features and returns the estimated price instantly.

This interface offers a simple and interactive way to estimate used car prices using machine learning.

FUTURE SCOPE OF IMPROVEMENT

While the current model delivers reliable predictions based on available features, there is significant potential to enhance its accuracy and usability in the future. Below are some key areas of improvement:

- **Include More Features**
  Adding additional attributes like insurance status, service history, accident reports, brand reputation, and resale demand could lead to more precise predictions.

- **Model Optimization**
  Experimenting with advanced models like CatBoost or ensemble stacking could improve predictive performance.

- **Real-time Market Data**
  Connecting the app to live databases or APIs to fetch current car prices and market listings could keep predictions more up to date.

- **User Feedback Loop**
  Adding a system where users can rate the prediction accuracy would allow the model to learn and improve over time.

- **Mobile App Integration**
  Extending the platform to Android/iOS would increase accessibility and usability.

- **Deployment**
  The application can be deployed on platforms like Heroku, Render, or AWS to make it publicly accessible and usable by a wider audience.

CERTIFICATE

This is to certify that **Mr. Aritra Banerjee** of Asansol Engineering College, Registration Number: **10871023012,** has successfully completed a project on **Car Price Prediction Model** using Machine Learning with Python under the guidance of **Dr. Arnab Chakraborty.**

Dr. Arnab Chakraborty

Globsyn Finishing School

CERTIFICATE

This is to certify that **Mr. Shouvik Ghanty** of Asansol Engineering College, Registration Number: **10871023045,** has successfully completed a project on **Car Price Prediction Model** using Machine Learning with Python under the guidance of **Dr. Arnab Chakraborty.**

Dr. Arnab Chakraborty

Globsyn Finishing School

CERTIFICATE

This is to certify that **Mr. Nilesh Maji** of Asansol Engineering College, Registration Number: **10871023030,** has successfully completed a project on **Car Price Prediction Model** using Machine Learning with Python under the guidance of **Dr. Arnab Chakraborty.**

Dr. Arnab Chakraborty

Globsyn Finishing School

CERTIFICATE

This is to certify that **Mr. Sujan Banerjee** of Asansol Engineering College, Registration Number: **10871023055,** has successfully completed a project on **Car Price Prediction Model** using Machine Learning with Python under the guidance of **Dr. Arnab Chakraborty.**

Dr. Arnab Chakraborty

Globsyn Finishing School

CERTIFICATE

This is to certify that **Mr. Md. Musharraf** of Asansol Engineering College, Registration Number: **10871023027,** has successfully completed a project on **Car Price Prediction Model** using Machine Learning with Python under the guidance of **Dr. Arnab Chakraborty.**

Dr. Arnab Chakraborty

Globsyn Finishing School

CERTIFICATE

This is to certify that **Mr. Sounak Kundu** of Asansol Engineering College, Registration Number: **10871023051,** has successfully completed a project on **Car Price Prediction Model** using Machine Learning with Python under the guidance of **Dr. Arnab Chakraborty.**

Dr. Arnab Chakraborty

Globsyn Finishing School