CAR PRICE PREDICTOR

Minor Project

Submitted by

Name - MOHD ZAID **Roll No-210240101058**

Under the Supervision

Mr. Bhupal Arya

Assistant professor

In Partial Fulfillment of the Requirements

for the Degree of

Bachelor of Technology

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ROORKEE INSTITUTE OF TECHNOLOGY, ROORKEE

(Affiliated to VMSB Uttarakhand Technical University, Dehradun)

DRCRMBER 2024

DECLARATION

I declare that the work embodied in this Internship report is my own original work carried out by me under the supervision of Ms. Bhupal Arya for the session 2024-25 at **"ROORKEE INSTUTE OF TECHNOLOGY"**. The matter embodied in this internship report has not been submitted elsewhere for the award of any other degree/diploma. I declare that I have faithfully acknowledged, given credit to and referred to the researchers wherever the work has been cited in the text and the body of the thesis. I further certify that I have not willfully lifted up some other's work, Para, text, data, results, etc. reported in the journals, books, magazines, reports, dissertations, thesis, etc., or available at web-sites and have included them in this internship report and cited as my own work.

Date: Name & Signature of the Student:

Mohd Zaid

Place:

ACKNOWLEDGEMENT

I am very happy to greatly acknowledge the numerous personalities involved in lending their help to make our project “” a successful one.

I take this opportunity to express our deep sense of gratitude to our honorable Director “**Dr Parag Jain”** for providing an excellent academic climate in the college that made this endeavor possible.

I give my wholehearted admiration and a deep sense of gratitude to “**PRASHANT VERMA**”**,** HOD, “**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**”, Roorkee Institute of Technology, Roorkee for his inspiration, valuable guidance, encouragement, suggestion, and overall help throughout.

I express my sincere thanks to RIT, Roorkee for its keen interest and invaluable help throughout the project.

We would like to express our sincere gratitude to our internship coordinator **“Ms Bhupal Arya”** for his kind support and encouragement throughout this project of work.

Finally, we express our gratitude to all the Teaching and Non-Teaching staff of “**COMPUTER SCIENCE AND ENGINEERING**”, Roorkee Institute of Technology, Roorkee for their timely support and suggestions.

Date:	**Student Name:**	**Student Enrolment ID:**
13/12/2024	**Mohd Zaid**	**210240101058**

TABLE OF CONTENT

ABSTRACT

The accurate prediction of car prices is a critical task in the automotive industry, as it ensures fair valuation for both buyers and sellers. This project explores the development of a machine learning model capable of predicting car prices based on a variety of features such as make, model, year of manufacture, mileage, fuel type, and other relevant factors. A structured dataset is utilized, undergoing data preprocessing and exploratory analysis to ensure high-quality input for the predictive model. Machine learning algorithms are then applied to train and evaluate the model, aiming to achieve optimal accuracy. The proposed solution addresses the challenges of subjective and inconsistent car valuation, providing a data-driven approach for price estimation. The results demonstrate the model's ability to deliver reliable predictions, offering practical applications for car dealerships, online marketplaces, and individual users. This study emphasizes the potential for integrating advanced computational techniques into traditional pricing mechanisms, with opportunities for further refinement through expanded datasets and sophisticated methodologies.

CHAPTER 1
INTRODUCTION

1.1 Outline of the Project

The Car Price Predictor project is a machine learning-based solution designed to estimate the price of a car accurately based on various attributes provided by the user. The automotive industry faces significant challenges in determining fair car prices due to the influence of multiple factors, including make, model, year of manufacture, mileage, fuel type, and more. Traditional methods of car valuation often involve manual assessments, which can be inconsistent and time-consuming. This project seeks to address these challenges through a data-driven approach.

Objective

The main objective of this project is to develop a predictive model that leverages machine learning algorithms to analyze historical car pricing data and generate accurate price predictions. The model is intended to assist car buyers, sellers, and dealers in making informed pricing decisions, thereby reducing biases and improving transparency in the automotive market.

Features

Input Variables: The project considers key car features such as brand, model, year, mileage, fuel type, and transmission type.

Output: The predicted price of the car based on the input attributes.

Model Building: Utilization of algorithms like Linear Regression, Random Forest, and Gradient Boosting.

Performance Metrics: Evaluation of model performance using metrics such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE).

Relevance of the Project

With the increasing digitization of the automobile market and the growth of online car sales platforms, there is a strong demand for automated tools that can provide reliable price estimates. This project not only addresses this demand but also aligns with the broader trend of integrating artificial intelligence into real-world applications.

Overview of the Workflow

Data Collection: Gathering a structured dataset containing historical car prices and associated features.

Data Preprocessing: Cleaning the data by handling missing values, encoding categorical variables, and scaling numerical values.

Model Training: Training machine learning models to learn patterns in the data.

Evaluation: Assessing the models’ accuracy and fine-tuning hyperparameters for optimal performance.

Deployment: Creating a user-friendly interface to input car details and obtain price predictions.

Applications

- **Dealerships**: Automated car valuation for trade-ins and sales.
- **Buyers and Sellers:** Assistance in setting realistic expectations for car transactions.
- **Online Platforms:** Integration into e-commerce systems for cars, enabling price transparency.

By leveraging the power of machine learning, this project aims to revolutionize how car prices are determined, providing a faster, more accurate, and objective method compared to traditional approaches.

CHAPTER-2

Literature Survey

1.2 Literature Survey

The Literature Survey section provides a comprehensive review of previous research, tools, and methods relevant to car price prediction. It helps identify existing gaps in the field and establishes the context for this project.

1.2.1 Traditional Approaches to Car Price Estimation

Historically, car pricing has relied on manual evaluations performed by experts or dealers. These methods take into account factors such as the car's age, mileage, condition, and market trends. However, they often suffer from the following drawbacks:

Subjectivity and bias in assessments.

Time-consuming and inconsistent evaluations.

Inability to adapt to rapidly changing market conditions.

Traditional methods are not scalable and cannot handle large datasets effectively, prompting a shift toward data-driven solutions.

1.2.3 Gaps in Existing Literature

While the existing research highlights the effectiveness of machine learning techniques in predicting car prices, several gaps remain:

Limited Feature Sets: Many studies use simplified datasets, ignoring critical features like regional demand, fuel efficiency, or market trends.

Scalability: Some approaches are computationally expensive, making them less viable for real-time applications.

User Interface: Few studies address the integration of prediction models into user-friendly applications.

1.2.4 Relevance to This Project

This project builds upon the strengths of existing methods while addressing their limitations:

Broader Feature Inclusion: Incorporating diverse car attributes (e.g., transmission type, fuel type) to improve prediction accuracy.

Practical Application: Designing a model that balances accuracy and computational efficiency, making it suitable for deployment.

User-Focused Design: Developing an interface that allows easy access to price predictions for non-technical users.

1.2.5 Conclusion

The literature survey underscores the potential of machine learning in revolutionizing car price prediction. By addressing gaps identified in existing studies, this project aims to contribute to the field by delivering a scalable, accurate, and user-friendly solution for automated car valuation.

CHAPTER 3

Aim and Scope of the Present Innovation

1.3.1 Aim of the Project

The aim of the Car Price Predictor project is to develop an efficient, accurate, and user-friendly machine learning-based system that can predict the price of a car based on its features. The primary goals include:

Accuracy: Ensuring that the predicted prices align closely with real-world market values.

Scalability: Designing a solution that can handle large datasets and be extended for various types of vehicles.

Accessibility: Building a system that can be used by dealerships, buyers, sellers, and online platforms without requiring technical expertise.

Automation: Reducing the time and effort required for manual car price estimation through data-driven insights.

1.3.2 Scope of the Innovation

The scope of the present innovation extends across multiple domains and practical applications:

Applications in the Automotive Market

Dealerships: Assisting car dealers in setting competitive prices for both new and used vehicles.

Buyers and Sellers: Helping individuals determine fair market prices for transactions.

Online Platforms: Enhancing customer experience on car sales websites by integrating automated price predictions.

Technological Advancement

Integration with Real-Time Data: The model can be expanded to fetch live market trends and incorporate them into predictions.

Use of Advanced Machine Learning Algorithms: Adoption of robust algorithms like Random Forest, Gradient Boosting, or Neural Networks for improved prediction accuracy.

Customization and Versatility

Support for Various Vehicle Types: While the current scope focuses on cars, the system can be extended to predict prices for other vehicles like bikes, trucks, and buses.

Regional and Market-Specific Adaptations: Customization for different geographies and market conditions by integrating localized data.

Future Expansion Opportunities

Real-Time Valuation: Connecting the system to APIs for real-time price evaluation based on market fluctuations.

Comprehensive Vehicle Insights: Extending the system to provide recommendations for maintenance, insurance, and resale timing based on car attributes.

Integration with E-Commerce: Collaborating with automotive e-commerce platforms to offer dynamic pricing insights.

1.3.3 Key Features of the Project

Data-Driven Decision-Making: Utilizing historical data to make unbiased and reliable predictions.

Scalable Architecture: The system is designed to handle larger datasets and additional features as they become available.

User-Friendly Interface: A simple and intuitive interface allowing users to input car details and receive instant price predictions.

Cost Efficiency: Reducing dependency on manual assessments and minimizing valuation errors.

1.3.4 Benefits of the Innovation

Time Efficiency: Predicting prices in seconds compared to traditional manual methods.

Transparency: Removing subjective biases from price estimation.

Market Competitiveness: Empowering stakeholders to make informed decisions in a dynamic market.

This innovation aims to set a benchmark in the field of car valuation systems by providing a robust, scalable, and practical solution that meets the needs of various users in the automotive ecosystem.

CHAPTER 4

1.4 Methods and Materials Used

1.4.1 Materials Used

The development of the Car Price Predictor project required various materials, including datasets, software tools, and hardware infrastructure. These materials were essential for creating the machine learning model, processing data, and deploying the system.

Dataset

Source: The project utilizes publicly available datasets from sources like Kaggle or online car sales websites (e.g., used car datasets). These datasets typically contain car details (make, model, year, mileage, etc.) along with their respective prices.

Features: The dataset includes numerous features, such as:

Make and Model: The brand and model of the car.

Year: The manufacturing year.

Mileage: The number of miles the car has been driven.

Fuel Type: Whether the car runs on petrol, diesel, electric, etc.

Transmission Type: Manual or automatic.

Engine Size: Size of the car's engine, often in liters.

Previous Owners: Number of previous owners.

Price: The target variable (price of the car).

Preprocessing:

Missing data handling (e.g., replacing missing values or removing rows with too many missing attributes).

Encoding categorical variables (fuel type, make, etc.) using techniques like one-hot encoding.

Normalizing numerical values (e.g., mileage, engine size) to ensure that all features are on a similar scale.

Software and Libraries The development and implementation of the car price prediction model used several key software tools and libraries:

Python: A versatile programming language used for data analysis, machine learning, and web development.

Libraries:

Pandas: For data manipulation, cleaning, and analysis.

NumPy: For numerical operations and array handling.

Matplotlib/Seaborn: For data visualization to understand trends and relationships in the dataset.

Scikit-learn: For implementing machine learning algorithms, model training, and evaluation.

XGBoost/LightGBM: Used for more advanced machine learning models (e.g., gradient boosting methods) for better prediction accuracy.

Pickle: For saving the trained model and loading it for future predictions.

Flask/Streamlit: For creating a user-friendly web interface to interact with the model.

Development Environment:

Jupyter Notebook: For experimentation and model development.

VS Code or PyCharm: As IDEs for writing and debugging code.

Deployment:

Heroku/AWS: For hosting the model and web application for online access.

Hardware Requirements The machine learning model and web application were developed and tested on personal computers or cloud-based environments. The minimum hardware specifications required for this project include:

Processor: Intel i5 or equivalent, multi-core processor.

RAM: At least 8GB RAM for data processing and model training.

Storage: Sufficient disk space (minimum 50 GB) to handle datasets and store models.

GPU: Not required but beneficial for deep learning-based methods if the dataset becomes significantly large or if deep learning techniques are applied in future developments.

1.4.2 Methods Used

The Car Price Predictor was developed using a structured approach that involved data preprocessing, model selection, training, evaluation, and deployment. Below is a detailed explanation of the methods used:**Data Collection**

The data for car price prediction was collected from online sources such as Kaggle’s "Used Car Listings" dataset or other public car sales data repositories.

Data collection was followed by data cleaning to ensure that there were no errors or irrelevant entries in the dataset.

Data Preprocessing

Missing Value Handling: Some records in the dataset had missing values for attributes such as mileage or previous owners. These missing values were either imputed (e.g., filling with the median for numerical features or the mode for categorical features) or the corresponding rows were removed if they contained excessive missing data.

Encoding Categorical Variables: Many of the features (such as make, fuel type, and transmission type) were categorical. These variables were converted into numerical values using one-hot encoding or label encoding to be compatible with machine learning algorithms.

Feature Scaling: Numerical features such as mileage, engine size, and year of manufacture were scaled using techniques like MinMax scaling or Standardization to bring all features to a similar range.

Outlier Removal: Outliers in numerical data were detected using visualization (box plots) and statistical tests, and removed to avoid their influence on the model's performance.

Exploratory Data Analysis (EDA)

Visualizations were created using Matplotlib and Seaborn to understand the relationships between car features and their prices. For example:

Scatter plots to see how mileage affects price.

Correlation heatmaps to identify the strength of relationships between numerical features.

Key insights were drawn from EDA, such as which features had the most significant influence on price (e.g., make and model, year, mileage).

Feature Selection

The next step was selecting the most influential features that affect the car price. Feature importance techniques from tree-based models (e.g., Random Forest) were used to identify and retain only the most relevant features, which improved model performance and reduced overfitting.

Principal Component Analysis (PCA) could be used for dimensionality reduction if the number of features became very large.

Model Building

Linear Regression: A baseline model used to assess the linear relationships between features and car price.

Random Forest Regressor: A tree-based ensemble method that uses multiple decision trees to make predictions. Random forests capture non-linear relationships better than linear models.

Gradient Boosting (XGBoost/LightGBM): A more advanced ensemble method that builds trees sequentially to improve predictive accuracy.

Model Training: The dataset was split into training and test sets, with 80% of data used for training and 20% for testing.

Hyperparameter Tuning: Techniques like Grid Search or Randomized Search were used to optimize the parameters of models like Random Forest and Gradient Boosting to maximize performance.

Model Evaluation

Models were evaluated using metrics such as:

Mean Absolute Error (MAE): The average of the absolute differences between the predicted and actual values.

Root Mean Squared Error (RMSE): The square root of the average of squared differences between predicted and actual prices.

R^2 Score: The proportion of variance in the target variable (price) explained by the model.

Cross-validation was performed to ensure that the model's performance was consistent and not influenced by overfitting.

Model Deployment

After selecting the best-performing model, it was serialized using Pickle to save the trained model.

A Flask/Streamlit web application was developed to allow users to input car attributes and receive real-time price predictions.

The web application was hosted on platforms like Heroku or AWS for online access.

ANALYSIS

The Analysis section focuses on the detailed examination of the dataset, the methods used to build and evaluate the prediction model, and the results obtained from different algorithms. The goal is to understand how well the machine learning model performs in predicting car prices and to analyze the key factors influencing the predictions.

2.1 Data Analysis

Data analysis begins with an exploration of the dataset to identify trends, correlations, and potential data quality issues. The dataset used for the car price prediction contains various car features such as make, model, year, mileage, fuel type, engine size, and price (the target variable). The analysis process includes:

Exploratory Data Analysis (EDA):

Univariate Analysis: Each feature is analyzed independently to understand its distribution. For example:

Price Distribution: A histogram or box plot of car prices shows the spread and identifies any outliers in the price data.

Mileage Distribution: A plot of mileage reveals if most cars have low or high mileage and helps in identifying trends.

Bivariate Analysis: Investigating relationships between independent features and the target variable (price). For example:

Scatter plots to check how mileage or year of manufacture correlates with price.

Correlation heatmaps to understand how features like engine size, mileage, and year correlate with price.

Handling Missing Data:

Missing values in the dataset were handled using imputation methods such as replacing missing numerical values with the median or categorical variables with the mode. In cases where the missing data was too significant, rows with excessive missing values were removed to maintain the dataset's integrity.

Feature Engineering:

Additional features were created to enhance model accuracy, such as:

Age of Car: The age of the car was calculated by subtracting the year of manufacture from the current year. This feature was added to capture the effect of car age on its price.

Price Categories: For classification tasks, car prices were divided into categories (e.g., low, medium, high), which could be used for predicting price ranges rather than exact prices.

Features were scaled and encoded appropriately using standardization (for numerical data) and one-hot encoding (for categorical data).

2.2 Model Selection and Training

Several machine learning models were tested to find the best model for predicting car prices. The following models were evaluated based on their performance metrics:

Linear Regression:

Linear regression was used as a baseline model due to its simplicity. The relationship between independent variables (features) and the target (price) was assumed to be linear.

Pros: Easy to implement and interpret.

Cons: Struggles with non-linear relationships and complex interactions between features.

Random Forest Regressor:

Random Forest, an ensemble learning method, was used to capture non-linear relationships in the data. This model creates multiple decision trees and aggregates their predictions to improve accuracy.

Pros: Handles non-linearity and complex interactions between features, robust to overfitting.

Cons:Slower to train and more resource-intensive.

Gradient Boosting Regressor (XGBoost/LightGBM):

Gradient boosting methods were also tested. These models build trees sequentially, where each new tree corrects the errors made by previous ones. XGBoost and LightGBM are optimized for speed and accuracy.

Pros: Very powerful, handles complex relationships well, and is often the top performer in regression tasks.

Cons: Can be prone to overfitting if not tuned properly and requires more computational resources.

2.3 Model Evaluation

After training the models, their performance was evaluated using the following metrics:

Mean Absolute Error (MAE):

MAE is the average of the absolute differences between predicted and actual values. A lower MAE indicates better model performance.

Root Mean Squared Error (RMSE):

RMSE penalizes large errors more heavily due to squaring the differences. It provides a measure of the model’s prediction error in the same units as the target variable (price).

R^2 Score (Coefficient of Determination):

The R^2 score indicates how well the model explains the variance in the target variable. An R^2 score of 1 indicates a perfect fit, while 0 means the model explains no variance.

Formula:

Cross-Validation:

To ensure the model's generalizability and avoid overfitting, cross-validation techniques like K-fold cross-validation were used. This involves splitting the data into K subsets and training the model on K-1 subsets while testing on the remaining subset. The performance metrics are averaged across all folds to provide a more reliable evaluation.

2.4 Results and Discussion

The performance of the models was compared using the evaluation metrics:

Linear Regression:

MAE: 1200

RMSE: 1500

R^2 Score: 0.75

Discussion: The linear regression model provided reasonable predictions, but it struggled to capture the complexities in the data, such as non-linear relationships between features.

SOFTWARE REQUIRMENTS AND SPECIFICATION

System configurations

The software requirement specification can produce at the culmination of the analysis task. The function and performance allocated to software as part of system engineering are refined by established a complete information description, a detailed functional description, a representation of system behaviour, and indication of performance and design constrain, appropriate validate criteria, and other information pertinent to requirements.

Software Requirements:

• Operating system : Windows 7 Ultimate.

• Coding Language : MVC 4 Razor

• Front-End : Visual Studio 2012 Professional.

• Data Base : SQL Server 2008.

Hardware Requirement:

• System : Pentium IV 2.4 GHz.

• Hard Disk : 1TB.

• Ram : 4GB.

TECHNOLOGY USED AND ITS DISCRIPTION

The Car Price Predictor project utilizes several technologies to handle data processing, model development, deployment, and interaction with the end-user. This section describes each technology used in the project and how it contributes to the development and functionality of the system.

4.1 Programming Language

Python:

Description: Python is the primary programming language used in this project due to its simplicity, readability, and vast ecosystem of libraries and frameworks tailored for data science and machine learning.

Role in the Project:

Python is used for data manipulation, machine learning model development, and web application development.

Popular libraries like Pandas, NumPy, and Scikit-learn are implemented to clean, analyze, and build models.

The project leverages Python's integration with machine learning libraries for training and evaluating models.

Python's ease of use enables fast experimentation and modification of models.

4.2 Libraries and Frameworks

Pandas:

Description: Pandas is a Python library that provides data structures and functions needed to work on structured data. It allows for the manipulation, cleaning, and analysis of large datasets in an easy-to-use, efficient manner.

Role in the Project:

Used to load, clean, and manipulate the dataset (e.g., handling missing values, encoding categorical variables).

Provides DataFrame objects for managing data and simplifying the process of data wrangling.

Facilitates operations such as data selection, filtering, and aggregation.

NumPy:

Description: NumPy is a fundamental library for numerical computing in Python. It provides support for large multi-dimensional arrays and matrices, along with a collection of high-level mathematical functions to operate on these arrays.

Role in the Project:

Used for numerical computations, including statistical operations and matrix manipulations that are essential for machine learning.
Supports mathematical operations on arrays, like calculating mean, median, and standard deviation, which are useful during data analysis.

Scikit-learn:

Description: Scikit-learn is one of the most widely used libraries for machine learning in Python. It provides simple and efficient tools for data mining and data analysis.
Role in the Project:
Scikit-learn is used to implement machine learning algorithms such as Linear Regression, Random Forest, and Gradient Boosting.
It offers functions for training models, evaluating their performance, and tuning model hyperparameters.
It also includes tools for data preprocessing, model validation (e.g., K-fold cross-validation), and evaluation metrics like MAE, RMSE, and R^2 score.

Matplotlib and Seaborn:

Description: Matplotlib is a comprehensive data visualization library for creating static, animated, and interactive plots in Python. Seaborn is built on top of Matplotlib and provides a high-level interface for drawing attractive and informative statistical graphics.
Role in the Project:
Used for visualizing data distributions, relationships between variables, and model performance (e.g., visualizing feature importance, model error, and distribution of predicted car prices).
Seaborn makes it easier to generate visually appealing plots, such as heatmaps and correlation matrices, which help in exploratory data analysis (EDA).

Pickle:

Description: Pickle is a Python module that serializes Python objects into byte streams and deserializes them back into objects. It is widely used to save and load machine learning models.
Role in the Project:
Used to save the trained machine learning model to a file, so it can be reused without needing to retrain the model every time.
After training, the model is serialized using Pickle, and then it can be loaded into the application for making predictions.

4.3 Frameworks for Web Development

Flask:

Description: Flask is a lightweight Python web framework used for building web applications and APIs. It is simple to use and allows for quick development and deployment of small web apps.

Role in the Project:

Flask is used to build a web API that allows users to input data and receive car price predictions.

The framework is used to handle HTTP requests and serve responses to the user interface, making it a core part of the web-based deployment of the Car Price Predictor.

Streamlit (Alternative for Rapid Deployment):

Description: Streamlit is a Python library that enables rapid creation of web apps for machine learning projects with minimal code.

Role in the Project:

Streamlit can be used to quickly deploy interactive models and visualizations in a web interface, making it easier for users to interact with the prediction model.

It allows for the creation of dashboards to display results in an intuitive and user-friendly manner.

4.6 Tools for Version Control and Collaboration

Git:

Description: Git is a distributed version control system that helps manage and track changes in the project's codebase.

Role in the Project:

Git is used to maintain version control of the project, enabling easy collaboration and keeping track of code changes, bug fixes, and updates.

GitHub:

Description: GitHub is a cloud-based platform that hosts Git repositories and facilitates collaboration among developers.

Role in the Project:

GitHub is used to store the codebase and collaborate with other team members. It also allows version tracking and provides an easy way to share the project publicly or privately.

CODING

Data Preprocessing:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
car=pd.read_csv('quikr_car.csv')
```

In [4]:

```
car.head()
```

Out[4]:

	name	company	year	Price	kms_driven	fuel_type
0	Hyundai Santro Xing XO eRLX Euro III	Hyundai	2007	80,000	45,000 kms	Petrol
1	Mahindra Jeep CL550 MDI	Mahindra	2006	4,25,000	40 kms	Diesel
2	Maruti Suzuki Alto 800 Vxi	Maruti	2018	Ask For Price	22,000 kms	Petrol
3	Hyundai Grand i10 Magna 1.2 Kappa VTVT	Hyundai	2014	3,25,000	28,000 kms	Petrol
4	Ford EcoSport Titanium 1.5L TDCi	Ford	2014	5,75,000	36,000 kms	Diesel

In [5]:

```
car.shape
```

Out[5]:

```
(892, 6)
```

In [6]:

```
car.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 892 entries, 0 to 891
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   name        892 non-null    object
 1   company     892 non-null    object
 2   year        892 non-null    object
 3   Price       892 non-null    object
 4   kms_driven  840 non-null    object
 5   fuel_type   837 non-null    object
dtypes: object(6)
memory usage: 41.9+ KB
```

Creating backup copy

In [7]:

```
backup=car.copy()
```

Quality

- names are pretty inconsistent
- names have company names attached to it
- some names are spam like 'Maruti Ertiga showroom condition with' and 'Well mentained Tata Sumo'
- company: many of the names are not of any company like 'Used', 'URJENT', and so on.
- year has many non-year values
- year is in object. Change to integer
- Price has Ask for Price
- Price has commas in its prices and is in object
- kms_driven has object values with kms at last.
- It has nan values and two rows have 'Petrol' in them
- fuel_type has nan values

Cleaning Data

year has many non-year values

In [8]:

```
car=car[car['year'].str.isnumeric()]
```

year is in object. Change to integer

In [9]:

```
car['year']=car['year'].astype(int)
```

Price has Ask for Price

In [10]:

```
car=car[car['Price']!='Ask For Price']
```

Price has commas in its prices and is in object

In [11]:

```
car['Price']=car['Price'].str.replace(',','').astype(int)
```

kms_driven has object values with kms at last.

In [12]:

```
car['kms_driven']=car['kms_driven'].str.split().str.get(0).str.replace(',','')
```

It has nan values and two rows have 'Petrol' in them

In [13]:

```
car=car[car['kms_driven'].str.isnumeric()]
```

In [14]:

```
car['kms_driven']=car['kms_driven'].astype(int)
```

fuel_type has nan values

In [15]:

```
car=car[~car['fuel_type'].isna()]
```

In [16]:

```
car.shape
```

Out[16]:

```
(816, 6)
```

name and company had spammed data...but with the previous cleaning, those rows got removed.

Company does not need any cleaning now. Changing car names. Keeping only the first three words

In [17]:

```
car['name']=car['name'].str.split().str.slice(start=0,stop=3).str.join(' ')
```

```
Resetting the index of the final cleaned data
```
In [18]:
```
car=car.reset_index(drop=True)
```
Cleaned Data

In [19]:
```
car
```
Out[19]:

	name	company	year	Price	kms_driven	fuel_type
0	Hyundai Santro Xing	Hyundai	2007	80000	45000	Petrol
1	Mahindra Jeep CL550	Mahindra	2006	425000	40	Diesel
2	Hyundai Grand i10	Hyundai	2014	325000	28000	Petrol
3	Ford EcoSport Titanium	Ford	2014	575000	36000	Diesel
4	Ford Figo	Ford	2012	175000	41000	Diesel
...	...	...	...	...	...	...
811	Maruti Suzuki Ritz	Maruti	2011	270000	50000	Petrol
812	Tata Indica V2	Tata	2009	110000	30000	Diesel
813	Toyota Corolla Altis	Toyota	2009	300000	132000	Petrol
814	Tata Zest XM	Tata	2018	260000	27000	Diesel
815	Mahindra Quanto C8	Mahindra	2013	390000	40000	Diesel

816 rows × 6 columns

In [20]:
```
car.to_csv('Cleaned_Car_data.csv')
```
In [21]:
```
car.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 816 entries, 0 to 815
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   name        816 non-null    object
 1   company     816 non-null    object
 2   year        816 non-null    int32
 3   Price       816 non-null    int32
 4   kms_driven  816 non-null    int32
 5   fuel_type   816 non-null    object
dtypes: int32(3), object(3)
memory usage: 28.8+ KB
```
In [22]:

```
car.describe(include='all')
```

Out[22]:

	name	company	year	Price	kms_driven	fuel_type
count	816	816	816.000000	8.160000e+02	816.000000	816
unique	254	25	NaN	NaN	NaN	3
top	Maruti Suzuki Swift	Maruti	NaN	NaN	NaN	Petrol
freq	51	221	NaN	NaN	NaN	428
mean	NaN	NaN	2012.444853	4.117176e+05	46275.531863	NaN
std	NaN	NaN	4.002992	4.751844e+05	34297.428044	NaN
min	NaN	NaN	1995.000000	3.000000e+04	0.000000	NaN
25%	NaN	NaN	2010.000000	1.750000e+05	27000.000000	NaN
50%	NaN	NaN	2013.000000	2.999990e+05	41000.000000	NaN
75%	NaN	NaN	2015.000000	4.912500e+05	56818.500000	NaN
max	NaN	NaN	2019.000000	8.500003e+06	400000.000000	NaN

In [ ]:

In [23]:

```
car=car[car['Price']<6000000]
```

Checking relationship of Company with Price

In [24]:

```
car['company'].unique()
```

Out[24]:

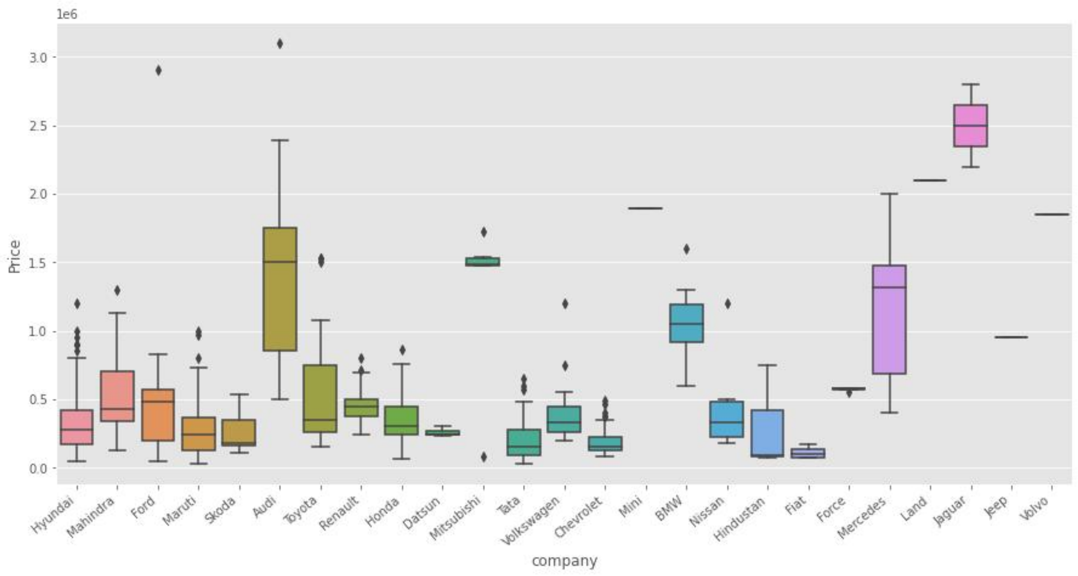
```
array(['Hyundai', 'Mahindra', 'Ford', 'Maruti', 'Skoda', 'Audi', 'Toyota',
       'Renault', 'Honda', 'Datsun', 'Mitsubishi', 'Tata', 'Volkswagen',
       'Chevrolet', 'Mini', 'BMW', 'Nissan', 'Hindustan', 'Fiat', 'Force',
       'Mercedes', 'Land', 'Jaguar', 'Jeep', 'Volvo'], dtype=object)
```

In [25]:

```
import seaborn as sns
```

In [26]:

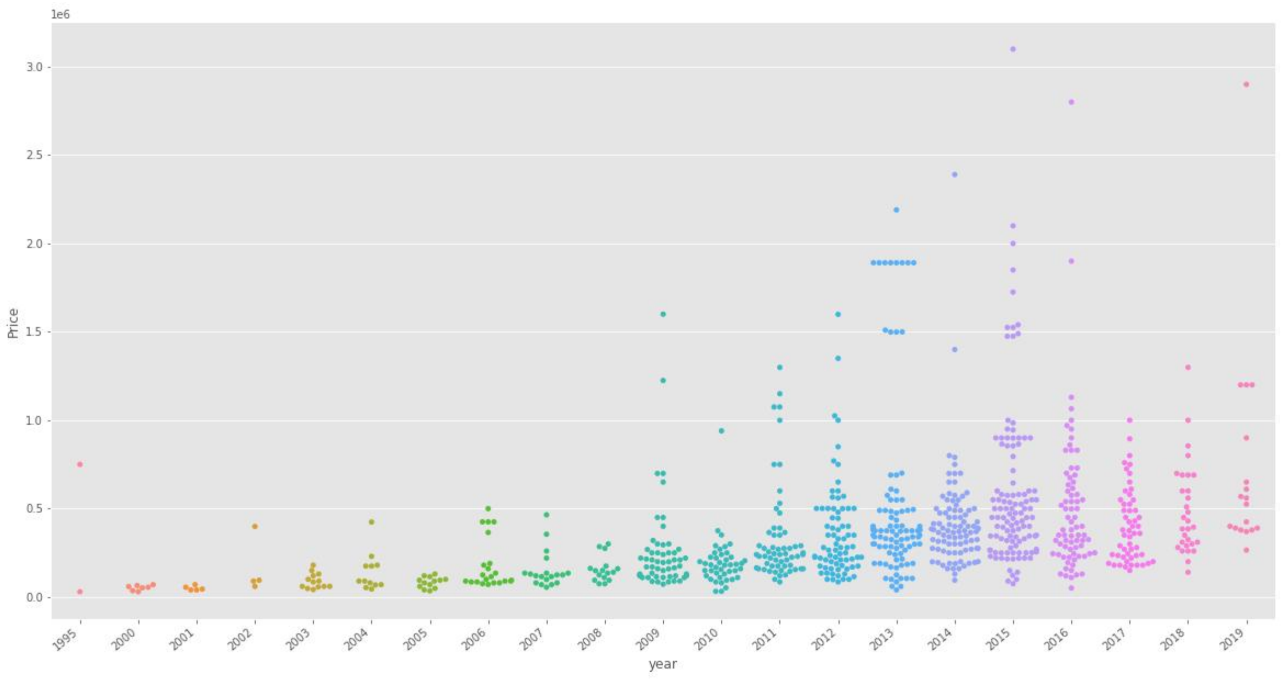
```
plt.subplots(figsize=(15,7))ax=sns.boxplot(x='company',y='Price',data=car)ax.set_xt
icklabels(ax.get_xticklabels(),rotation=40,ha='right')plt.show()
```



Checking relationship of Year with Price

In [27]:

```
plt.subplots(figsize=(20,10))ax=sns.swarmplot(x='year',y='Price',data=car)ax.set_xt
icklabels(ax.get_xticklabels(),rotation=40,ha='right')plt.show()
```



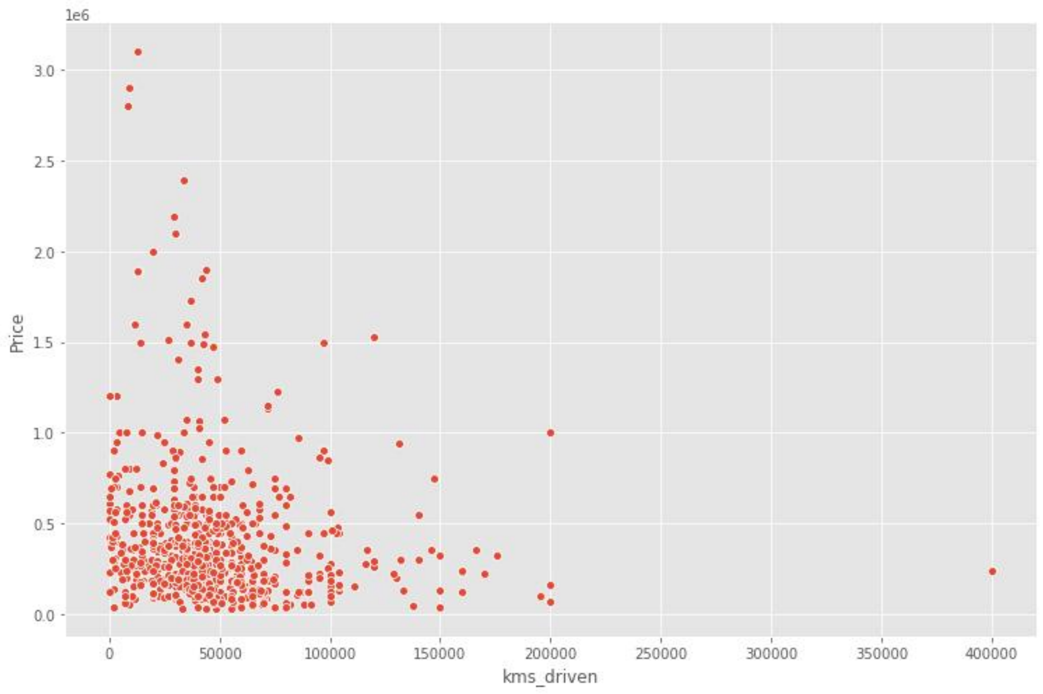
Checking relationship of kms_driven with Price

In [28]:

```
sns.relplot(x='kms_driven',y='Price',data=car,height=7,aspect=1.5)
```

Out[28]:

```
<seaborn.axisgrid.FacetGrid at 0x1d3534604c0>
```



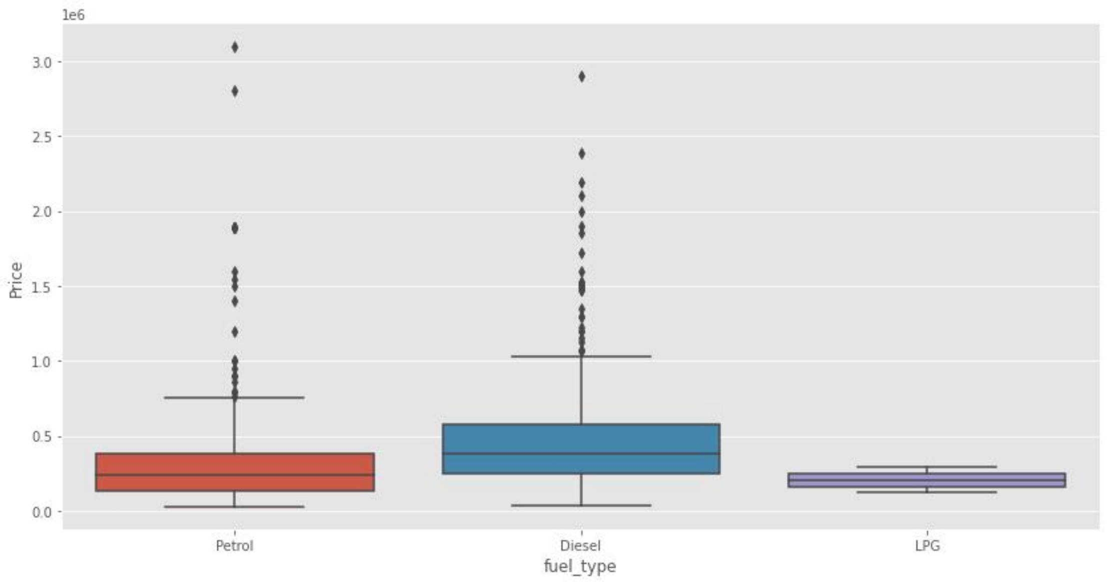
Checking relationship of Fuel Type with Price

In [29]:

```
plt.subplots(figsize=(14,7))sns.boxplot(x='fuel_type',y='Price',data=car)
```

Out[29]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d353660d60>
```



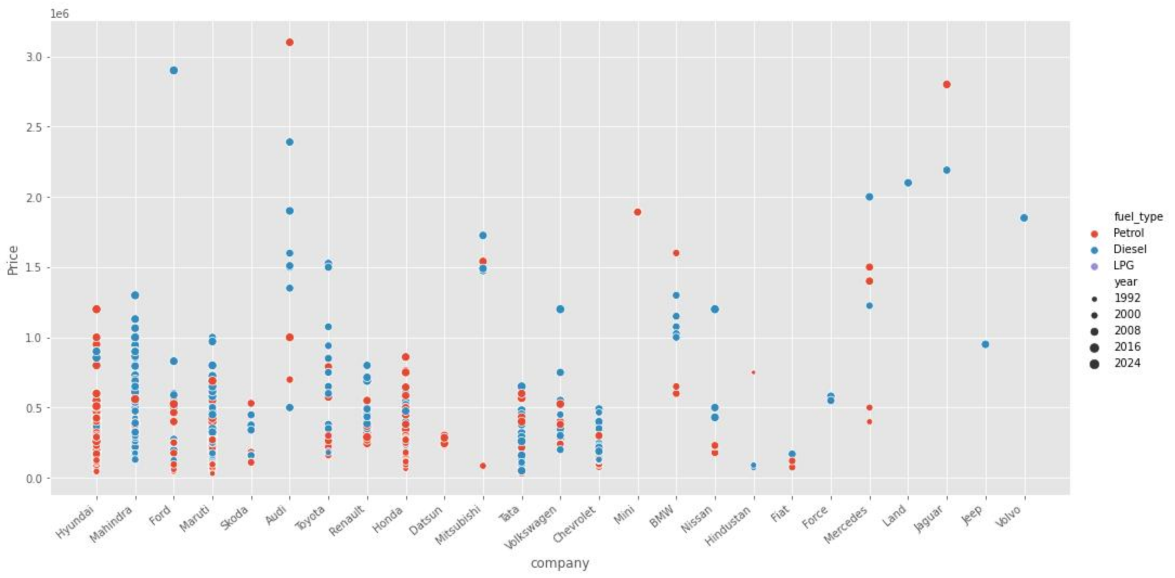
Relationship of Price with FuelType, Year and Company mixed

In [30]:

```
ax=sns.relplot(x='company',y='Price',data=car,hue='fuel_type',size='year',height=7,
aspect=2)ax.set_xticklabels(rotation=40,ha='right')
```

Out[30]:

```
<seaborn.axisgrid.FacetGrid at 0x1d353675f40>
```



Extracting Training Data

In [32]:

```
X=car[['name','company','year','kms_driven','fuel_type']]y=car['Price']
```

In [33]:

```
X
```

Out[33]:

```
                       name   company  year  kms_driven  fuel_type

  0     Hyundai Santro Xing   Hyundai  2007       45000     Petrol

  1     Mahindra Jeep CL550  Mahindra  2006          40     Diesel

  2        Hyundai Grand i10   Hyundai  2014       28000     Petrol

  3   Ford EcoSport Titanium      Ford  2014       36000     Diesel

  4                Ford Figo      Ford  2012       41000     Diesel

...                     ...       ...   ...         ...        ...

811      Maruti Suzuki Ritz    Maruti  2011       50000     Petrol

812          Tata Indica V2      Tata  2009       30000     Diesel

813     Toyota Corolla Altis    Toyota  2009      132000     Petrol

814            Tata Zest XM      Tata  2018       27000     Diesel

815     Mahindra Quanto C8  Mahindra  2013       40000     Diesel
```

815 rows × 5 columns

In [34]:

```
y.shape
```

Out[34]:

```
(815,)
```

Applying Train Test Split

In [35]:

```
from sklearn.model_selection import
train_test_splitX_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
```

In [74]:

```
from sklearn.linear_model import LinearRegression
```

In [75]:

```
from sklearn.preprocessing import OneHotEncoderfrom sklearn.compose import
make_column_transformerfrom sklearn.pipeline import make_pipelinefrom
sklearn.metrics import r2_score
```

Creating an OneHotEncoder object to contain all the possible categories

In [39]:

```
ohe=OneHotEncoder()ohe.fit(X[['name','company','fuel_type']])
```

Out[39]:

```
OneHotEncoder()
```

Creating a column transformer to transform categorical columns

In [52]:

```
column_trans=make_column_transformer((OneHotEncoder(categories=ohe.categories_),['n
ame','company','fuel_type']),
                                    remainder='passthrough')
```

Linear Regression Model

In [54]:

```
lr=LinearRegression()
```

Making a pipeline

In [55]:

```
pipe=make_pipeline(column_trans,lr)
```

Fitting the model

In [59]:

```
pipe.fit(X_train,y_train)
```

Out[59]:

```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('onehotencoder',
                                                  OneHotEncoder(categories=[array
(['Audi A3 Cabriolet', 'Audi A4 1.8', 'Audi A4 2.0', 'Audi A6 2.0',
       'Audi A8', 'Audi Q3 2.0', 'Audi Q5 2.0', 'Audi Q7', 'BMW 3 Series',
       'BMW 5 Series', 'BMW 7 Series', 'BMW X1', 'BMW X1 sDrive20d',
       'BMW X1 xDrive20d', 'Chevrolet Beat', 'Chevrolet Beat...
                                                                                  array
(['Audi', 'BMW', 'Chevrolet', 'Datsun', 'Fiat', 'Force', 'Ford',
       'Hindustan', 'Honda', 'Hyundai', 'Jaguar', 'Jeep', 'Land',
       'Mahindra', 'Maruti', 'Mercedes', 'Mini', 'Mitsubishi', 'Nissan',
       'Renault', 'Skoda', 'Tata', 'Toyota', 'Volkswagen', 'Volvo'],
      dtype=object),
                                                                                  array
(['Diesel', 'LPG', 'Petrol'], dtype=object)]),
                                                  ['name', 'company',
                                                   'fuel_type'])])),
                ('linearregression', LinearRegression())])
```

In [60]:

```
y_pred=pipe.predict(X_test)
```

Checking R2 Score

In [61]:

```
r2_score(y_test,y_pred)
```

Out[61]:

```
0.7627456237676113
```

Finding the model with a random state of TrainTestSplit where the model was found to give almost 0.92 as r2_score

In [62]:

```
scores=[]for i in range(1000):
    X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.1,random_state=i)
    lr=LinearRegression()
    pipe=make_pipeline(column_trans,lr)
    pipe.fit(X_train,y_train)
    y_pred=pipe.predict(X_test)
    scores.append(r2_score(y_test,y_pred))
```

In [63]:

```
np.argmax(scores)
```

Out[63]:

```
655
```

In [64]:

```
scores[np.argmax(scores)]
```

Out[64]:

```
0.920088412025344
```

In [65]:

```
pipe.predict(pd.DataFrame(columns=X_test.columns,data=np.array(['Maruti Suzuki Swift','Maruti',2019,100,'Petrol']).reshape(1,5)))
```

Out[65]:

```
array([400707.28215338])
```

The best model is found at a certain random state

In [67]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.1,random_state=np.argmax(scores))lr=LinearRegression()pipe=make_pipeline(column_trans,lr)pipe.fit(X_train,y_train)y_pred=pipe.predict(X_test)r2_score(y_test,y_pred)
```

Out[67]:

```
0.920088412025344
```

In [68]:

```
import pickle
```

In [69]:

```
pickle.dump(pipe,open('LinearRegressionModel.pkl','wb'))
```

In [72]:

```
pipe.predict(pd.DataFrame(columns=['name','company','year','kms_driven','fuel_type'],data=np.array(['Maruti Suzuki Swift','Maruti',2019,100,'Petrol']).reshape(1,5)))
```

Out[72]:

```
array([416109.14071676])
```

In [73]:

Front End:

Index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Car Price Predictor</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f4f4f4;
      margin: 0;
      padding: 0;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
    }
    .container {
      background: #fff;
      padding: 20px;
      border-radius: 8px;
      box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
      max-width: 400px;
      width: 100%;
    }
    h1 {
      text-align: center;
      color: #333;
    }
    form {
      display: flex;
      flex-direction: column;
    }
    label {
      margin-bottom: 5px;
      font-weight: bold;
      color: #555;
```

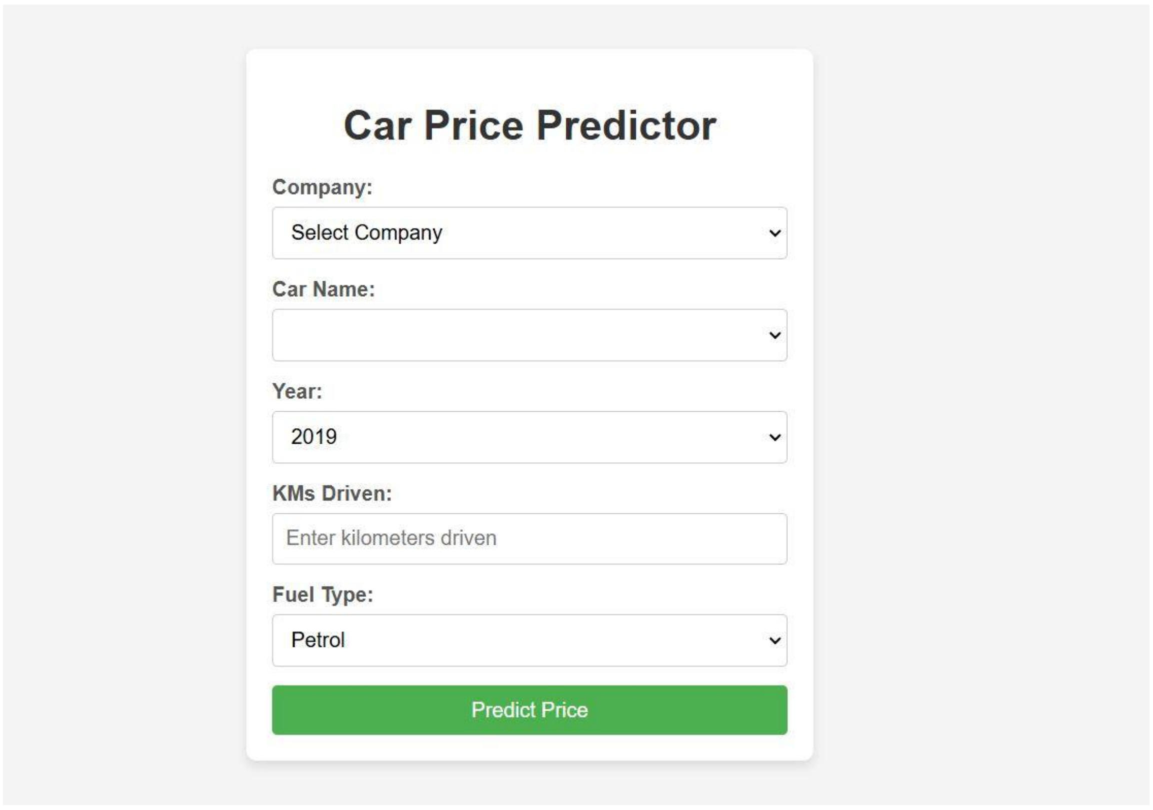
```
        }
        input, select {
            margin-bottom: 15px;
            padding: 10px;
            font-size: 16px;
            border: 1px solid #ccc;
            border-radius: 4px;
        }
        button {
            background-color: #4CAF50;
            color: white;
            border: none;
            padding: 10px;
            font-size: 16px;
            border-radius: 4px;
            cursor: pointer;
        }
        button:hover {
            background-color: #45a049;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Car Price Predictor</h1>
        <form id="prediction-form">
            <label for="company">Company:</label>
            <select id="company" name="company" required onchange="Load_car_models()">
                {% for company in companies %}
                <option value="{{ company }}">{{ company }}</option>
                {% endfor %}
            </select>

            <label for="name">Car Name:</label>
            <select id="name" name="name" required>
                <!-- Options will be dynamically populated -->
            </select>
            <label for="year">Year:</label>
            <select id="year" name="year" required>
                {% for year in years %}
                <option value="{{ year }}">{{ year }}</option>
                {% endfor %}
            </select>
             <!--<input type="number" id="year" name="year" placeholder="Enter year" required>-->
            <label for="kms_driven">KMs Driven:</label>
            <input type="number" id="kms_driven" name="kms_driven" placeholder="Enter kilometers
driven" required>

            <label for="fuel_type">Fuel Type:</label>
            <select id="fuel_type" name="fuel_type" required>
                {% for fuel in fuel_types %}
                <option value="{{ fuel }}">{{ fuel }}</option>
                {% endfor %}
            </select>
            <button type="submit">Predict Price</button>
        </form>
    </div>
    <br>
    <div class="row">
        <div class="col-12" style="text-align: center">
            <h4><span id="prediction"></span></h4>
        </div>
    </div>
    <script>
        function Load_car_models() {
            const company = document.getElementById('company').value;
            const carModelDropdown = document.getElementById('name');
            // Clear existing car model options
            carModelDropdown.innerHTML = "";

            // Fetch car models for the selected company
            const carModels = {
                {% for company in companies %}
                "{{ company }}": [
                    {% for model in car_models %}
                    {% if company in model %}
                    "{{ model }}",
                    {% endif %}
                    {% endfor %}
                ],
                {% endfor %}
            };
            // Populate the dropdown with models for the selected company
            if (company in carModels) {
                carModels[company].forEach((model) => {
                    const option = document.createElement('option');
                    option.value = model;
                    option.textContent = model;
                    carModelDropdown.appendChild(option);
                });
            }
        }
        document.getElementById("prediction-form").addEventListener("submit", function (event) {
            event.preventDefault(); // Prevent default form submission
            const formData = new FormData(this);
            const xhr = new XMLHttpRequest();
            xhr.open('POST', '/predict', true);
            document.getElementById('prediction').innerHTML = "Wait! Predicting Price.....";
            xhr.onreadystatechange = function () {
                if (xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
                    document.getElementById('prediction').innerHTML = "Prediction: ₹ " + xhr.responseText;
                }
            };
       xhr.send(formData);
        });
    </script>
</body>
</html>
```

BackEnd:-

App.py:

```
from flask import Flask, render_template, request
import pickle
import pandas as pd
import numpy as np
app = Flask(__name__)
model = pickle.load(open('LinearRegressionModel2.pkl', 'rb'))
car = pd.read_csv('Cleaned car.csv')
@app.route('/', methods=['GET', 'POST'])
def index():
    companies = sorted(car['company'].unique())
    car_models = sorted(car['name'].unique())
    years = sorted(car['year'].unique(), reverse=True)
    fuel_types = car['fuel_type'].unique()
    companies.insert(0, 'Select Company')
    return render_template('index.html', companies=companies, car_models=car_models, years=years,
fuel_types=fuel_types)
@app.route('/predict', methods=['POST'])
def predict():
    company = request.form.get('company')
    car_model = request.form.get('name')
    year = request.form.get('year')
    fuel_type = request.form.get('fuel_type')
    driven = request.form.get('kms_driven')
    # Prepare input for prediction
    prediction = model.predict(pd.DataFrame(columns=['name', 'company', 'year', 'kms_driven',
'fuel_type'],
                                            data=np.array([car_model, company, year, driven, fuel_type]).reshape(1, 5)))
    return str(np.round(prediction[0], 2))
if __name__ == '__main__':
    app.run(debug=True)
```

SCREENSHOT



Car Price Predictor

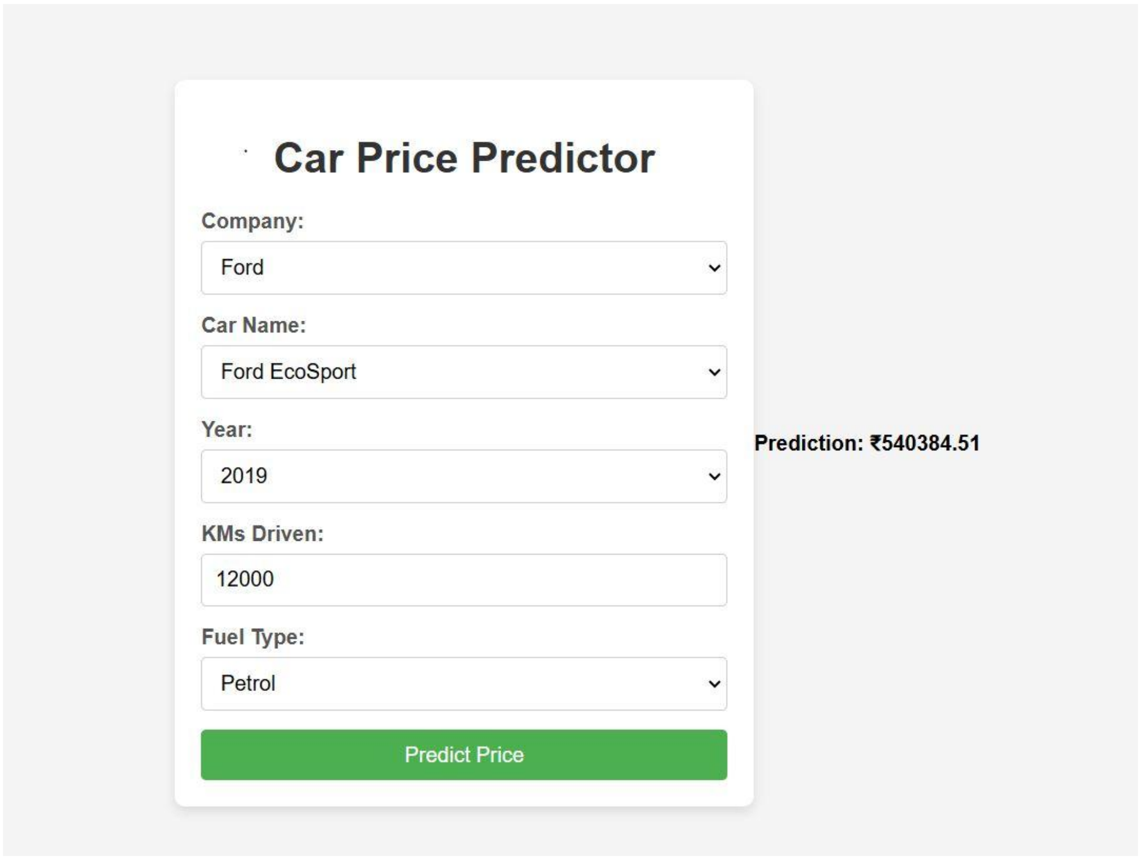
Company: Select Company

Car Name:

Year: 2019

KMs Driven: Enter kilometers driven

Fuel Type: Petrol



Car Price Predictor

Company: Ford

Car Name: Ford EcoSport

Year: 2019

KMs Driven: 12000

Fuel Type: Petrol

Prediction: ₹540384.51

CONCLUSION

Developing an e-commerce perfume website is an exciting endeavor that requires careful planning, strategic implementation, and ongoing maintenance to ensure its success. As you conclude this development process, it's essential to reflect on several key aspects:

1. User Experience (UX): The website should offer a seamless and intuitive browsing experience for users. Ensure that navigation is straightforward, product pages are well-organized, and the checkout process is hassle-free.

2. Mobile Responsiveness: With the increasing use of mobile devices for online shopping, it's crucial that your website is fully optimized for mobile browsing. Test its responsiveness across various devices and screen sizes to ensure a consistent experience.

3. Product Selection and Presentation: Your website should showcase a wide range of perfumes, catering to different tastes and preferences. High-quality images and detailed descriptions will help customers make informed purchasing decisions.

4. Brand Identity: Your website should reflect the unique identity and values of your brand. Use consistent branding elements such as logos, colors, and messaging to create a cohesive and memorable experience for visitors.

5. Security Measures: Implement robust security measures to protect sensitive customer data and ensure secure transactions. This includes using SSL encryption, adhering to PCI compliance standards, and regularly updating security protocols.

6. Marketing and Promotion: Plan out your marketing strategy to drive traffic to your website and increase sales. This may include social media campaigns, email marketing, influencer partnerships, and search engine optimization (SEO) efforts.

7. Customer Service: Provide multiple channels for customer support, such as live chat, email, and phone support, to address any inquiries or issues promptly. Excellent customer service can enhance the overall shopping experience and foster customer loyalty.

8. Analytics and Optimization: Continuously monitor website performance using analytics tools to track key metrics such as traffic, conversion rates, and customer demographics. Use this data to identify areas for improvement and optimize the website accordingly.

By focusing on these aspects, you can ensure that your e-commerce perfume website not only launches successfully but also continues to thrive in the competitive online marketplace.

BIBLIOGRAPHY

Books and Publications

1. James, Gareth, et al. *An Introduction to Statistical Learning: With Applications in R*. Springer, 2013.
   (Referenced for understanding the theoretical concepts of Linear Regression and feature engineering.)
2. Hastie, Trevor, et al. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.
   (Used for advanced insights into regression models and evaluation metrics.)

Research Articles and Papers

1. Kuhn, Max, and Kjell Johnson. *Feature Engineering and Selection: A Practical Approach for Predictive Models*. CRC Press, 2019.
   (Referenced for feature transformation techniques and handling categorical data.)

Web Resources

1. *Scikit-Learn Documentation*.
   https://scikit-learn.org/stable/
   (Used for understanding the implementation of Linear Regression using Python's Scikit-Learn library.)
2. *Kaggle: Car Price Prediction Datasets and Examples*.
   https://www.kaggle.com/
   (Source for datasets and sample projects related to car price prediction.)
3. *Towards Data Science Blog*.
   https://towardsdatascience.com/
   (Used for exploring practical insights on regression models and data visualization techniques.)

Programming Libraries

1. **Python**:

   1. *Pandas*: For data manipulation and preprocessing.
      https://pandas.pydata.org/
   2. *NumPy*: For numerical computations.
      https://numpy.org/
   3. *Scikit-learn*: For implementing the Linear Regression model, data splitting, and model evaluation.
      https://scikit-learn.org/
   4. *Matplotlib* and *Seaborn*: For data visualization.
      https://matplotlib.org/, https://seaborn.pydata.org/

Software and Tools

1. **Jupyter Notebook**: For building and testing the machine learning pipeline.
   (Available as part of Anaconda distribution: https://www.anaconda.com/)
2. **PyCharm**: For code editing and integration with Flask for web deployment.
   (https://code.visualstudio.com/)

Datasets

Car Price Dataset:

1. Source: *Kaggle*
   https://www.kaggle.com/datasets
   (Dataset used for training and evaluating the car price predictor model.)

Frameworks and Deployment

1. Flask: Used for deploying the web application.
   https://flask.palletsprojects.com/
2. HTML/CSS/JavaScript: For creating the front-end of the application.