**Used Car Price Prediction: A Comprehensive Machine Learning Project**

**Sai Teja Gurrapu**

saiteja.goud2729@gmail.com

**1. Introduction**

The automotive industry is a dynamic market where used car pricing plays a crucial role in sales, financing, and insurance evaluations. Pricing a used car accurately is a challenging problem due to the influence of multiple factors such as **mileage, age, engine performance, accidents, transmission type, and brand reputation**. Traditional valuation methods rely on historical sales trends and expert knowledge, but they often fail to capture the intricate relationships between different variables.

This project aims to develop a **machine learning model** to predict the price of a used car based on its features. The project includes **data preprocessing, feature engineering, exploratory data analysis (EDA), model selection with hyperparameter tuning**, and **deployment using Django**. By automating price estimation, the model provides insights for dealerships, individual buyers, and financial institutions.

**2. Problem Statement**

**Challenges in Used Car Price Prediction**

1. **Data Complexity** – The dataset consists of **both numerical and categorical features**, requiring **appropriate encoding and transformations**.
2. **Text Parsing** – Extracting structured information from unstructured fields like engine specifications (e.g., "420HP 5.9L V12 Gasoline") is non-trivial.
3. **Skewed Price Distribution** – Used car prices follow a skewed distribution, requiring **log transformations** to improve prediction accuracy.
4. **Feature Relationships** – High multicollinearity exists between variables such as **horsepower and engine displacement**.
5. **Outliers** – Extreme values in mileage, accident history, and price can affect model performance.

**Objective**

Build a robust machine learning model that takes in raw input data (as seen in real-world used car listings) and predicts an **accurate price** using advanced data preprocessing and model optimization techniques.

**3. Scope of the Project**

1. **Data Ingestion:** Load and clean the dataset from CSV files.
2. **Feature Engineering:** Extract relevant attributes, handle missing values, encode categorical variables, and normalize numerical data.
3. **Exploratory Data Analysis (EDA):** Identify patterns, correlations, and key factors influencing used car prices.
4. **Model Selection & Tuning:** Train and optimize multiple models using **GridSearchCV**.
5. **Evaluation:** Measure performance using **Mean Absolute Error (MAE), Mean Squared Error (MSE), and R² Score**.
6. **Deployment:** Deploy the final model using Django for real-time predictions via a web-based API.

**4. Use Cases**

1. **Car Dealerships:** Price used cars accurately to remain competitive.
2. **Buyers & Sellers:** Assess fair market value before making transactions.
3. **Financial Institutions:** Adjust loan amounts based on predicted car values.
4. **Insurance Companies:** Set appropriate premium rates based on vehicle condition.

**Dataset Description: Used Cars Data**

The initial dataset consists of 4,009 records with 12 columns, capturing key attributes that influence the pricing of used cars. The dataset includes both categorical and numerical features, some of which required preprocessing to be used effectively in a machine learning model.

**Dataset Overview**

| **Column Name** | **Data Type** | **Description** | **Issues & Observations** |
| --- | --- | --- | --- |
| **brand** | Object | The manufacturer of the car (e.g., Toyota, BMW, Ford). | Needs categorical encoding. |
| **model** | Object | The specific model of the car (e.g., Camry, Accord, X5). | Not used in ML model (high cardinality). |
| **model\_year** | Integer | The manufacturing year of the car. | Used to compute car age (2025 - model\_year). |
| **milage** | Object | Distance the car has traveled (in miles). | Contains text formatting (needs conversion to numeric). |
| **fuel\_type** | Object | The type of fuel used (Gasoline, Diesel, Hybrid, Electric). | 170 missing values (needs imputation). |
| **engine** | Object | Contains information like horsepower, engine size, and fuel type. | Needs feature extraction (horsepower, displacement). |
| **transmission** | Object | Transmission type (Manual, Automatic, Other). | Needs one-hot encoding. |
| **ext\_col** | Object | Exterior color of the car. | Standardized into basic colors. |
| **int\_col** | Object | Interior color of the car. | Standardized into basic colors. |
| **accident** | Object | Accident history (e.g., "No Accidents", "1+ Accident"). | 113 missing values (needs handling). |
| **clean\_title** | Object | Indicates if the car has a clean title (Yes/No). | 596 missing values (needs handling). |
| **price** | Object | The listing price of the car. | Needs conversion to numeric, highly skewed distribution. |

This dataset provides a diverse range of features influencing the price of used cars. Through careful preprocessing, feature engineering, and handling of missing values, we transformed this dataset into a structured format suitable for machine learning.

**A screenshot of a computer

AI-generated content may be incorrect.**

**5. Data Preprocessing Pipeline**

**5.1 Extracting Engine Features**

The dataset includes an **"Engine"** column with values like:

"420.0HP 5.9L 12 Cylinder Engine Gasoline Fuel."

Using **regular expressions (regex)**, we extract:

* **Horsepower** (420.0 HP)
* **Engine Displacement** (5.9 L)

import re

def extract\_engine\_att(engine\_str):

horsepower = re.search(r'(\d+\.\d+)HP|\d+\.\d+', engine\_str)

displacement = re.search(r'(\d+\.\d+L|\d+\.\d+ Liter)', engine\_str)

return horsepower.group(1) if horsepower else '',

displacement.group(1) if displacement else ''

**5.2 Handling Outliers Using the IQR Technique**

Outliers can significantly impact machine learning models by introducing extreme values that skew predictions. In this project, we used the Interquartile Range (IQR) method to detect and remove outliers from key numerical features.

**1️. What is the IQR Technique?**

The Interquartile Range (IQR) is the range between the 25th percentile (Q1) and the 75th percentile (Q3) of the data. Any value beyond 1.5 times the IQR is considered an outlier.

IQR=Q3−Q1IQR = Q3 - Q1

LowerBound = Q1−1.5×IQRLower Bound = Q1 - 1.5 \times IQR

UpperBound = Q3+1.5×IQRUpper Bound = Q3 + 1.5 \times IQR

Values below the lower bound or above the upper bound are treated as outliers and replaced with the nearest boundary value.

**Outlier Removal in the Dataset**

We applied IQR-based outlier removal on the following numerical features:

* Age (in years)
* Mileage
* Horsepower
* Engine Displacement
* Price (log-transformed)

**2️. Implementation in Python**

import numpy as np

# Define a function to remove outliers using IQR

def remove\_outliers\_iqr(df, column):

Q1 = df[column].quantile(0.25)

Q3 = df[column].quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

df[column] = np.where(df[column] < lower\_bound, lower\_bound, df[column])

df[column] = np.where(df[column] > upper\_bound, upper\_bound, df[column])

return df

# Apply IQR outlier removal on numerical columns

num\_cols = ['Age', 'milage', 'Horsepower', 'Engine\_Displacement', 'price']

for col in num\_cols:

df = remove\_outliers\_iqr(df, col)

**Why Use IQR for Outlier Detection?**

Better than Mean-Based Methods: Unlike Z-score (which assumes normal distribution), IQR works well for skewed data like car prices.  
Preserves Important Data: Unlike simple trimming, IQR modifies extreme values instead of removing entire rows.  
Robust for Regression Models: It prevents extreme values from dominating model training.

A graph of a bar graph

AI-generated content may be incorrect.

**5.3 Feature Engineering: Horsepower per Liter**

To improve feature relevance, we compute:

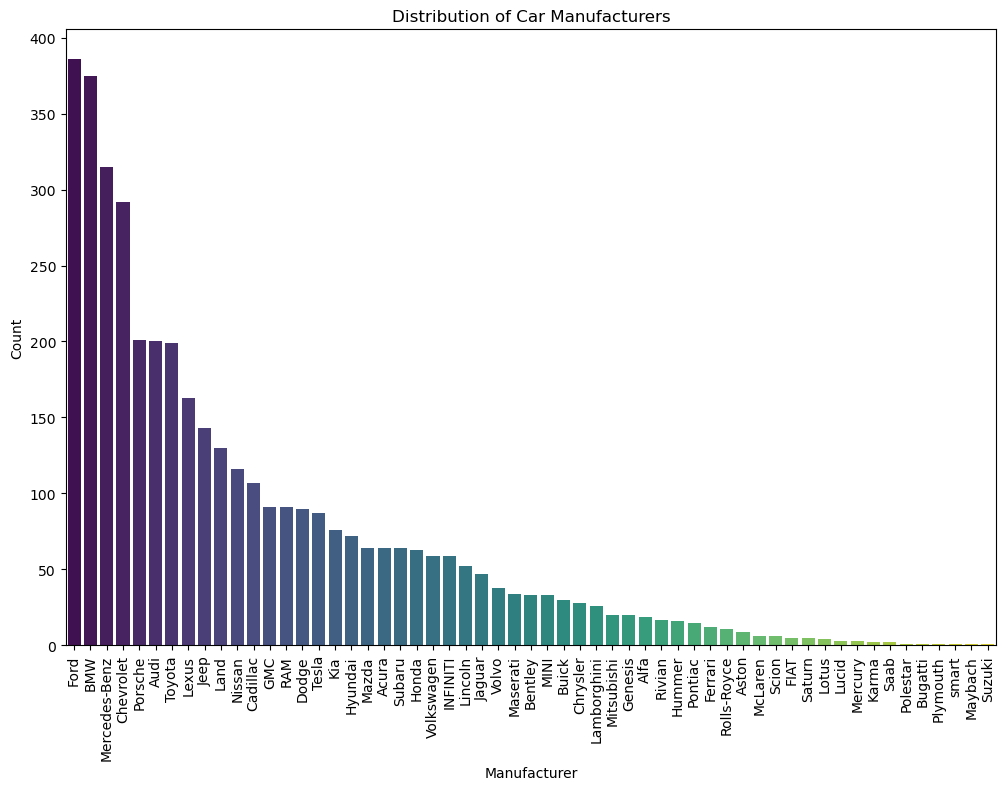
df['Horsepower\_per\_Liter'] = df['Horsepower'] / df['Engine\_Displacement']

df.drop(columns=['Horsepower', 'Engine\_Displacement'], inplace=True)

**5.4 Categorical to Numerical Values**

The dataset contained several categorical variables that needed to be converted into numerical format before being used in the machine learning model. Different encoding techniques were used based on the nature of each feature:

| **Feature** | **Encoding Method Used** | **Reason for Selection** |
| --- | --- | --- |
| **Brand** (brand) | **Target Encoding** | Avoids high-dimensionality, captures brand influence on price. |
| **Transmission** (transmission) | **One-Hot Encoding** | Limited unique values, maintains interpretability. |
| **Fuel Type** (fuel\_type) | **Label Encoding** | Preserves ordinal relationship in fuel categories. |
| **Accident History** (accident) | **One-Hot Encoding** | Clear distinction between accident categories. |
| **Exterior & Interior Color** (ext\_col, int\_col) | **Grouping & Simplification** | Reduces cardinality, maintains information. |



**Why Use Target Encoding for Brand Instead of One-Hot Encoding?**

**1️. One-Hot Encoding (OHE) - Issues with High Cardinality**

One-hot encoding **creates a new binary column for each unique category**. While this works well for low-cardinality features like **transmission**, it is problematic for **high-cardinality features like brand**.

**Example:**

* If the dataset contains **40+ unique car brands**, one-hot encoding would create **40+ additional columns**, leading to:
  + **Increased dimensionality** (sparse dataset).
  + **Higher memory usage and slower training**.
  + **Poor generalization on unseen brands**.

**2. Target Encoding - A Better Alternative**

Instead of creating multiple binary columns, **target encoding replaces each category with its mean target value (price in this case).**

**Example:**  
If the average price of **BMW** cars is $41,000 and for **Toyota** it is $30,000, we assign:

* brand\_encoded (BMW) = 41000
* brand\_encoded (Toyota) = 30000

**Advantages of Target Encoding:**

* **Reduces dimensionality** → Keeps dataset compact.
* **Preserves meaningful price relationships** → Captures statistical trends.
* **Handles unseen categories better** → Works well with out-of-sample brands.

**How Target Encoding Was Implemented**

import pandas as pd

# Compute target mean encoding for brand

brand\_encoded\_values = df.groupby("brand")["price"].mean()

# Map brand values to target encoding

df["brand\_encoded"] = df["brand"].map(brand\_encoded\_values)

By using target encoding, we significantly reduced feature space while preserving brand influence on car prices!

**5.5 Standard Scaling**

Only numerical columns (Age, milage, Horsepower\_per\_Liter) are scaled:

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df[['Age', 'milage', 'Horsepower\_per\_Liter']] = scaler.fit\_transform(df[['Age', 'milage', 'Horsepower\_per\_Liter']])

A graph of a price distribution

AI-generated content may be incorrect.

Distribution of price

A diagram of a distribution

AI-generated content may be incorrect.

Distribution of price logarithmic after standardisation

**6. Model Building & Evaluation**

**6.1 Ridge Regression (Baseline Model)**

* **Best hyperparameter (**``**)** found using GridSearchCV.
* Performance:
  + **MAE:** 0.1564
  + **MSE:** 0.0405
  + **R²:** 0.6945

**6.2 Random Forest Regression (Best Model)**

* **Hyperparameter tuning via GridSearchCV** optimized n\_estimators, max\_depth, etc.
* **Best Model Performance:**
  + **MAE:** 0.1225
  + **MSE:** 0.0280
  + **R²:** 0.7887

**6.3 Model Selection**

* **Random Forest outperformed Ridge Regression**, capturing non-linear relationships better.
* Predictions require reversing log transformation:

predicted\_price = 10 \*\* model.predict(X\_test)

**7. Model Deployment Using Django**

Django follows the MVT (Model-View-Template) architectural pattern, which is a variant of the MVC (Model-View-Controller) framework. It separates concerns into three distinct components:

1. Model (M) – Represents the database structure and handles data storage and retrieval.
2. View (V) – Contains the business logic and processes user requests, retrieves necessary data, and returns a response.
3. Template (T) – Defines the front-end (HTML, CSS, JS) structure that displays dynamic data to the user.

**How We Used MVT in Our Project**

1. Model (M) – models.py

* We defined a PredictionRecord model that stores user inputs and predicted prices.
* The database (SQLite) stores each prediction request along with the result for tracking purposes.
* Fields:
  + brand, model, year, transmission, fuel\_type, mileage, engine\_size, horsepower, predicted\_price, and timestamp.

2. View (V) – views.py

* Handles HTTP requests for the homepage and prediction feature.
* Loads the trained machine learning model and scaler to process input data.
* Converts user inputs into numerical values, applies feature scaling, and makes price predictions.
* Saves prediction results into the database.
* Returns JSON responses for API calls.

3. Template (T) – templates/predict.html

* Designed a user-friendly form where users enter car details (brand, model, year, mileage, etc.).
* Uses Django Template Language (DTL) to dynamically generate forms and display predicted prices.
* Displays error messages and results using front-end logic.

**MVT in Action: The User Flow in Our Application**

1. User visits the homepage (/)

* View (home function in views.py) loads home.html and renders the webpage.

2. User submits car details in the prediction form (/predict/)

* View (predict\_price function in views.py) processes input data, loads the machine learning model, and makes a prediction.
* Model (PredictionRecord in models.py) saves the user input and predicted price in the database.
* Template (predict.html) displays the predicted car price to the user.

**Benefits of Using Django’s MVT Architecture**

Separation of Concerns: MVT keeps data (models), logic (views), and UI (templates) separate, making the code modular and maintainable.

Reusability: The trained machine learning model is loaded only once and reused efficiently within views.

Scalability: The prediction API can be easily extended to support multiple users and integrate with external applications.

Security: Django provides built-in security features like CSRF protection and input validation to prevent attacks.

**Conclusion**

By leveraging Django’s MVT architecture, our project ensures a well-structured, modular, and scalable implementation for used car price prediction. The separation of models, views, and templates enables smooth development, easy maintenance, and future enhancements.

**8. Limitations & Future Work**

**8.1 Limitations**

* **Data Quality Issues:** Inconsistent and missing data may affect predictions.
* **Feature Extraction Challenges:** Parsing engine details may not generalize well.
* **Real-World Variability:** Used car prices fluctuate based on market demand, inflation, and interest rates.

**8.2 Future Enhancements**

* **Boosting Models:** Implement **XGBoost or LightGBM** for better performance.
* **Real-Time Updates:** Integrate with **live car listing data**.
* **Cloud Deployment:** Deploy on **AWS/GCP** for scalability.

**9. Conclusion**

This project demonstrates the effectiveness of **Random Forest Regression** in predicting used car prices based on historical data. The **deployed Django API** provides a scalable solution for real-time price estimation. Future enhancements can further improve accuracy and generalizability, making this a valuable tool for the automotive industry.