DEBRE BIRHAN UNIVERSITY

COLLEGE OF COMPUTING

DEPARTMENT OF SOFTWARE ENGINEERING

Individual Project

Course title: fundamental of machine learning

Course code: SEng4091

Used Car Price Prediction

Submitted by: Betel Yemanebirhan

Id:1400991

Submitted to: Derebew F. (msc)

Submitted date:02/06/2017E.c

Table of Contents

1. Project Overview

Objective

The goal of this project is to develop a machine learning model that predicts the price of used cars based on various attributes. The model follows the complete machine learning lifecycle, from data acquisition and preprocessing to training, evaluation, and deployment as an API.

Introduction

Predicting the price of used cars is a challenging yet essential task in the automotive market. It provides buyers and sellers with an estimate of a car's value based on its characteristics, which can influence the decision-making process. In this project, we aim to develop a predictive model that estimates the price of used cars based on various features such as brand, model, year, age, kilometers driven, transmission type, number of owners, fuel type, and additional information. This can help individual...

Deployment

The trained model has been deployed as a REST API using FastAPI and Render. The API endpoint can be accessed at: **Used Car Price Prediction API**

2. Dataset Description

Data Source

A self-sourced dataset containing used car listings was used for model training.

The dataset used in this project was downloaded from **Kaggle** (Used Car Dataset- https://www.kaggle.com/datasets/mohitkumar282/used-car-dataset)

It is licensed under: CC0: Public Domain

licence link:https://creativecommons.org/publicdomain/zero/1.0/

The dataset consists of the following features:

Feature	**Description**
Brand	The manufacturer of the car (e.g., Toyota, Honda)
Model	The specific model of the car
Year	Year of manufacture
Age	Age of the car (calculated as Current Year - Year)
kmDriven	Total kilometers driven
Transmission	Type of transmission (Manual/Automatic)
Owner	Ownership history (First, Second, Third, etc.)
FuelType	Type of fuel used (Petrol/Diesel/CNG/Electric)
PostedDate	Date when the car was listed
AdditionInfo	Additional descriptive information
AskPrice	The listed price of the car (Target variable)

.

Below is a detailed breakdown of each feature:

1. **Brand:** The make of the car, such as Toyota, Ford, etc. It is a categorical variable and is expected to influence the price.
2. **Model:** The specific model of the car. Like the brand, it is categorical but with more granularity.
3. **Year:** The year of manufacture of the car, typically associated with a car's age and condition.

Newer cars are usually priced higher.
4. **Age:** The difference between the current year and the year of manufacture. Age often negatively correlates with the price, as older cars tend to depreciate.
5. **KmDriven:** The total distance the car has been driven, measured in kilometers. Cars with higher mileage tend to have lower prices.
6. **Transmission:** Whether the car has an automatic or manual transmission. This is categorical and could affect price, as some buyers have preferences for one over the other.
7. **Owner:** The number of previous owners. More owners often indicate a higher chance of wear and tear, potentially decreasing the car's value.
8. **FuelType:** The type of fuel the car uses (e.g., petrol, diesel, electric). This can influence the price based on market demand and fuel efficiency.
9. **PostedDate:** The date the listing was created. Newer listings might indicate more recent cars or a greater demand for that particular model.
10. **AdditionInfo:** Extra details provided by the seller, such as condition or modifications. These can affect the car's perceived value.
11. **AskPrice:** The target variable, representing the price the seller is asking for the car.

3. The code with comment explanation

For visualization I use this code to tarin each line explanation using comment is below

1. import necessary that to work in the code first

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import joblib
import warnings
```

2 suppress warnings for missing fonts

```
warnings.filterwarnings("ignore", category=UserWarning)
```

3 set a font that support Devanagari script

For window

```
matplotlib.rcParams['font.family'] = 'Nirmala UI
```

If you have Linux

```
matplotlib.rcParams['font.family'] = 'Lohit Devanagari'
```

Step 1 load the dataset

To load the data set I saved the data set in name used_car_dataset.csv I download from kaggle.

```
df = pd.read_csv('used_car_dataset.csv')
```

Step 2 Data preprocessing:

```
    print("Initial Data Info:")# print the info fist

print(df.info())
print("Missing Values:") # print the missing value
print(df.isnull().sum())
```

Clean the ‘AskPrice’ colomn to remove currency symbols and commas and convert it to numeric

```
df['AskPrice'] = df['AskPrice'].replace({'₹': '', ',': ''}, regex=True).astype(float)
```

Drop rows with missing target values (assuming 'AskPrice' is the target column)

```
df = df.dropna(subset=['AskPrice'])
```

Fill missing categorical values with mode

```
for col in df.select_dtypes(include=['object']).columns:
   df[col] = df[col].fillna(df[col].mode()[0])
```

Fill missing categorical values with median

```
for col in df.select_dtypes(include=['int64', 'float64']).columns:
   df[col] = df[col].fillna(df[col].median())
```

Rename columns to remove non-Ascii charactes

```
df.columns = [col.encode('ascii', 'ignore').decode('utf-8') for col in df.columns]
```

Step3: Encode categorical variables,execluding ‘AskPrice’ from the encoding

```
categorical_cols = df.select_dtypes(include=['object']).columns
categorical_cols = categorical_cols[categorical_cols != 'AskPrice']  # Exclude 'AskPrice'
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
```

Step4:Feature selection

```
X = df.drop(columns=['AskPrice'])  # Features
y = df['AskPrice']  # Target
```

Step 5:train-test split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 6 train the model

```
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Step 7 evaluate the model

```
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Model Performance:\nMSE: {mse:.2f}\nR² Score: {r2:.2f}")
```

Step 8 save the model

```
joblib.dump(model, "used_car_price_model.pkl")
print("Model saved as used_car_price_model.pkl")
```

Step 9 model performance visualization

```
residuals = y_test - y_pred
plt.figure(figsize=(8, 5))
sns.histplot(residuals, bins=30, kde=True)
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.title("Residual Distribution")
plt.show()
```

```
# Actual vs Predicted Plot
plt.figure(figsize=(8, 5))
sns.scatterplot(x=y_test, y=y_pred, alpha=0.5)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted Prices")
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--')
plt.show()
```

the detail explanation and how and which part of code do what work is explained in the below of document written by their work name.

4. Exploratory Data Analysis (EDA)

Key Observations

- Some columns contained missing values, particularly kmDriven.
- The AskPrice column had currency symbols and commas, which needed to be removed.
- Categorical variables such as Brand, Transmission, and FuelType needed encoding for model training.
- Year was converted into Age to improve interpretability.

5. Data Preprocessing

Steps Taken

✓ **Cleaning Numeric Columns**:

- kmDriven values contained commas and text (e.g., "98,000 km"), which were cleaned and converted to integers.
- AskPrice values contained currency symbols (₹), which were removed and converted to numerical format.

✓ **Handling Missing Values**:

We examined the dataset for any missing values. If any feature had missing data, it was either imputed with a reasonable estimate (e.g., using the mean or median) or removed, depending on the feature's importance.

Example code for handling missing values:

```python
import pandas as pd

# Load the dataset
df = pd.read_csv('used_cars.csv')

# Check for missing values
missing_values = df.isnull().sum()

# Impute missing numerical values with the median
df['kmDriven'].fillna(df['kmDriven'].median(), inplace=True)
df['Owner'].fillna(df['Owner'].median(), inplace=True)

# Remove rows with missing categorical values
df.dropna(subset=['Brand', 'Model'], inplace=True)

print("Missing values handled.")
```

✓ Encoding Categorical Variables

The dataset includes categorical variables like brand, model, transmission, fuel type, and others. To convert these into a form that can be used by machine learning models, we applied one-hot encoding or label encoding, depending on the variable type and number of unique values.

Example code for encoding categorical variables:

```python

```

```python
# One-hot encoding for categorical columns

df = pd.get_dummies(df, columns=['Brand', 'Model', 'Transmission', 'FuelType'],
drop_first=True)

print("Categorical variables encoded.")
```

✓ Feature Scaling

Numerical features such as ‘Age’, ‘KmDriven’, and ‘Owner’ were scaled using standardization (zero mean and unit variance) to ensure all features contribute equally to the model.

Example code for feature scaling:

~~~python

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df[['Age', 'kmDriven', 'Owner']] = scaler.fit_transform(df[['Age', 'kmDriven', 'Owner']])

print("Features scaled.")
~~~

✅ Outlier Detection:

We analyzed the distribution of numerical features and identified any extreme outliers that could disproportionately affect model performance. Some of these outliers were removed or capped

✅ Feature Engineering:

New features were created from existing ones. For instance, we calculated the car's age by subtracting the year of manufacture from the current year. We also created a combined feature of ‘Brand-Model’ to represent a more specific vehicle type.

~~~python

```python

# Feature Engineering for Age

df['Age'] = 2025 - df['Year']

# Combining Brand and Model for a more specific feature
df['Brand-Model'] = df['Brand'] + '-' + df['Model']

print("New features created.")
~~~

5. Model Implementation & Training

Model Choice

A **RandomForestRegressor** model was chosen

- due to its ability to handle non-linear relationships and categorical variables effectively.
- for its robustness and ability to handle both categorical and numerical data without the need for extensive preprocessing.
- are also less prone to overfitting compared to individual decision trees, making them ideal for predicting complex relationships such as the ones in the used car dataset.

Train-Test Split

We split the data into training and testing sets and used the **RandomForestRegressor** for training. Hyperparameters such as the number of trees, maximum depth, and minimum samples split were tuned to achieve better results.

The dataset was split into training and testing sets:

- **80% Training Data**
- **20% Testing Data**

Training Process

The RandomForestRegressor model was trained using **100 estimators** (trees) and a fixed random state for reproducibility. This ensures the model learns relationships between car features and price effectively.

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Evaluation Metrics

- **Mean Squared Error (MSE)**: Measures the average squared difference between actual and predicted values.
- **R² Score**: This indicates the proportion of variance in the target variable (price) explained by the model. A higher R² score suggests that the model does a good job of predicting car prices.

```
• from sklearn.metrics import mean_squared_error, r2_score
• y_pred = model.predict(X_test)
• mse = mean_squared_error(y_test, y_pred)
• r2 = r2_score(y_test, y_pred)
• print(f"Model Performance:\nMSE: {mse:.2f}\nR² Score: {r2:.2f}")
```

Results

- **MSE**: 549237739701.47
- **R² Score**: 0.80

6. Interpretation of results

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import joblib
import warnings

# Suppress warnings for missing fonts (optional)
warnings.filterwarnings("ignore", category=UserWarning)

# Set a font that supports Devanagari script (if needed)
matplotlib.rcParams['font.family'] = 'Nirmala UI'  # Works on Windows
# matplotlib.rcParams['font.family'] = 'Lohit Devanagari'  # Use for
Linux users

# Step 1: Load the Dataset
```

```
df = pd.read_csv('used_car_dataset.csv')

# Step 2: Data Preprocessing
print("Initial Data Info:")
print(df.info())
print("Missing Values:")
print(df.isnull().sum())

# Clean the 'AskPrice' column to remove currency symbols and commas,
and convert it to numeric
df['AskPrice'] = df['AskPrice'].replace({'₹': '', ',': ''},
regex=True).astype(float)

# Drop rows with missing target values (assuming 'AskPrice' is the
target column)
df = df.dropna(subset=['AskPrice'])

# Fill missing categorical values with mode
for col in df.select_dtypes(include=['object']).columns:
    df[col] = df[col].fillna(df[col].mode()[0])

# Fill missing numerical values with median
for col in df.select_dtypes(include=['int64', 'float64']).columns:
    df[col] = df[col].fillna(df[col].median())

# Rename columns to remove non-ASCII characters (if necessary)
df.columns = [col.encode('ascii', 'ignore').decode('utf-8') for col in
df.columns]

# Step 3: Encode categorical variables, excluding 'AskPrice' from the
encoding
categorical_cols = df.select_dtypes(include=['object']).columns
categorical_cols = categorical_cols[categorical_cols != 'AskPrice']  #
Exclude 'AskPrice'
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)

# Step 4: Feature Selection
X = df.drop(columns=['AskPrice'])  # Features
y = df['AskPrice']  # Target
```

```
# Step 5: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Step 6: Model Training
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Step 7: Model Evaluation
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Model Performance:\nMSE: {mse:.2f}\nR² Score: {r2:.2f}")

# Step 8: Save the Model
joblib.dump(model, "used_car_price_model.pkl")
print("Model saved as used_car_price_model.pkl")
```

the out put is

```
Initial Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9582 entries, 0 to 9581
Data columns (total 11 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------  -----
 0   Brand         9582 non-null   object
 1   model         9582 non-null   object
 2   Year          9582 non-null   int64
 3   Age           9582 non-null   int64
 4   kmDriven      9535 non-null   object
 5   Transmission   9582 non-null   object
 6   Owner         9582 non-null   object
 7   FuelType      9582 non-null   object
 8   PostedDate    9582 non-null   object
 9   AdditionInfo   9582 non-null   object
 10   AskPrice       9582 non-null   object
dtypes: int64(2), object(9)
memory usage: 823.6+ KB
None
Missing Values:
Brand            0
model            0
Year             0
Age              0
kmDriven        47
Transmission     0
Owner            0
FuelType         0
PostedDate       0
AdditionInfo     0
AskPrice         0
dtype: int64
Model Performance:
MSE: 549237739701.47
R² Score: 0.80
Model saved as used_car_price_model.pkl
```

7. Model Deployment

Saving and Deploying the Model

The trained model was saved using joblib for later use:

```
import joblib
joblib.dump(model, "used_car_price_model.pkl")
```

The model was then integrated into a FastAPI-based backend for real-time price prediction.

API Endpoint & Usage

Base URL: https://used-car-price-prediction-1-b7n2.onrender.com

Example API Request (POST):

```
{
"Brand": "Toyota",
  "model": "Innova",
  "Year": 2009,
  "Age": 15,
  "kmDriven": 190000,
  "Transmission": "Manual",
  "Owner": "second",
  "FuelType": "Diesel"
}
```

API Response:

```
{
  "predicted_price": 332920
}
```

Backend API Implementation (main.py)

The backend API was implemented using FastAPI:

```
from fastapi import FastAPI, HTTPException
import pandas as pd
import joblib
import uvicorn

app = FastAPI()

# Load trained model
model = joblib.load("used_car_price_model.pkl")
```

This initializes FastAPI and loads the trained model.

```
@app.post("/predict")
def predict(data: dict):
    try:
        input_data = pd.DataFrame([data])
        input_data["kmDriven"] = input_data["kmDriven"].astype(int)
        prediction = model.predict(input_data)
        return {"predicted_price": prediction[0]}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

This function handles incoming POST requests, processes the input, and returns a predicted price.

8. Visualizing Model Performance

To analyze model performance, we plotted residuals and actual vs. predicted prices:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Residual Plot
residuals = y_test - y_pred
plt.figure(figsize=(8, 5))
sns.histplot(residuals, bins=30, kde=True)
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.title("Residual Distribution")
plt.show()
```

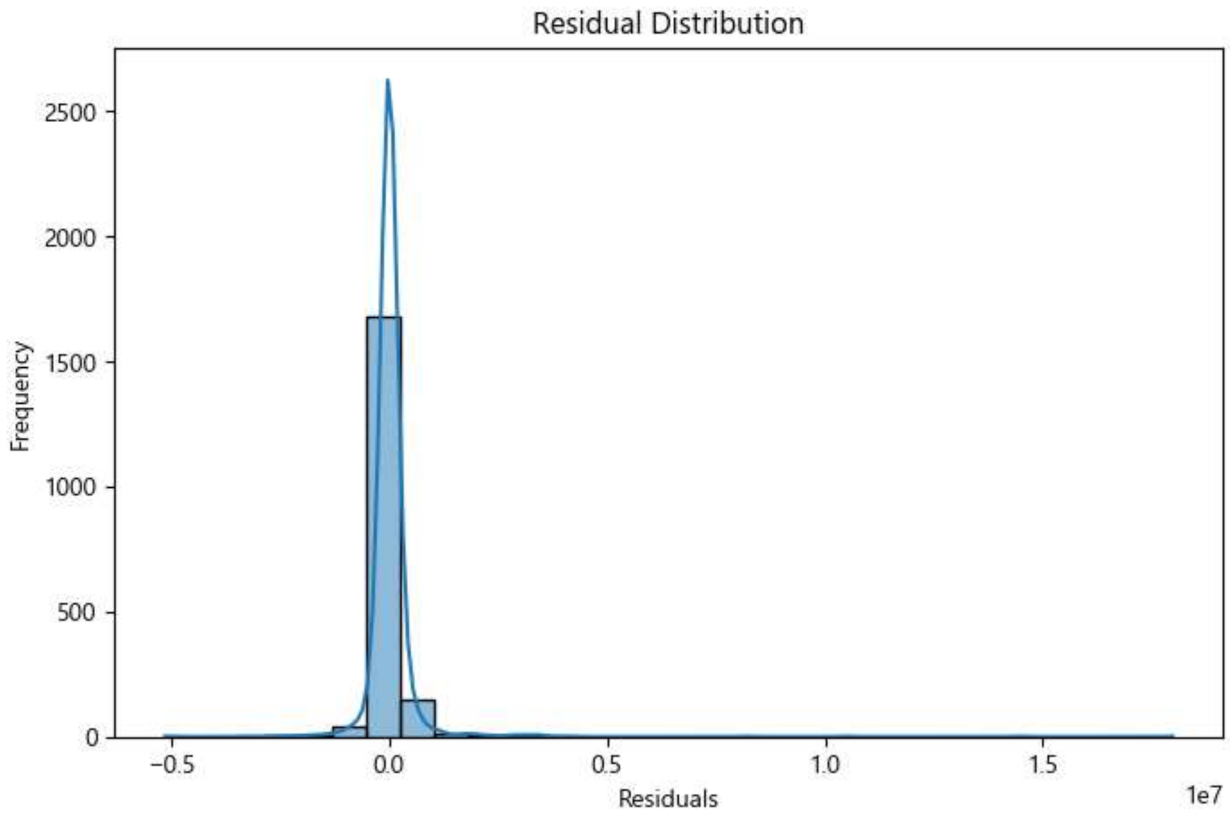

Figure 1

The residual plot shows how well the model predicts prices. Ideally, residuals should be normally distributed around zero.

```
# Actual vs Predicted Plot
plt.figure(figsize=(8, 5))
sns.scatterplot(x=y_test, y=y_pred, alpha=0.5)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted Prices")
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--')
plt.show()
```

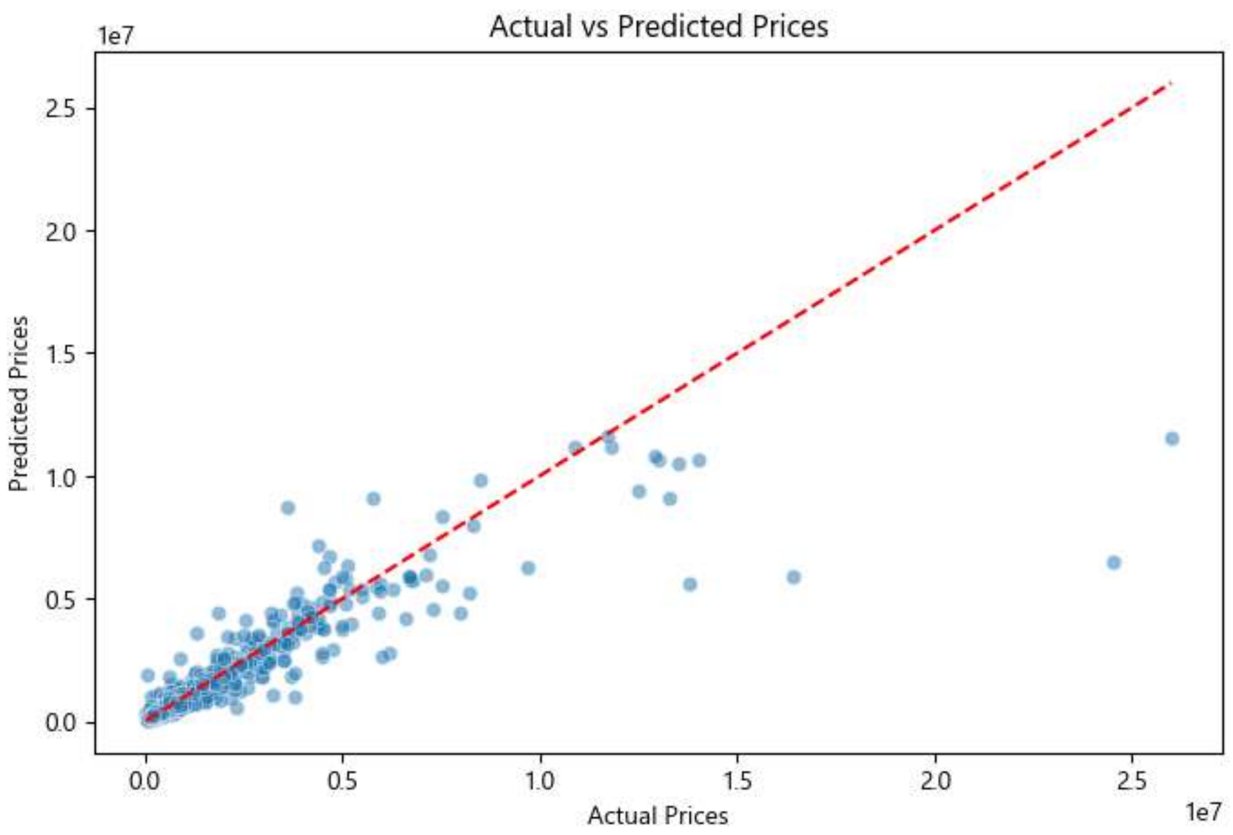

Figure 2

This scatterplot helps visualize the accuracy of predictions.

9. Challenges and Learnings

During the course of the project, several challenges were encountered:

1. **Data Quality Issues:** Some features had missing or inconsistent data, especially in the "AdditionInfo" column. Handling this missing data required thoughtful imputation strategies to avoid introducing bias.

2. **Model Overfitting:** Initial tests with the RandomForestRegressor resulted in overfitting, where the model performed well on the training data but poorly on unseen data. This was mitigated by using cross-validation and adjusting the model's hyperparameters to prevent it from memorizing the training data too closely.

3. **Feature Importance:** Determining the most important features for predicting car prices required analyzing the model's feature importance scores. This helped in understanding which factors (e.g., brand, kmDriven) played the most significant role in determining the price.

4. **Computational Efficiency:** As the dataset grew larger with additional features, training time increased significantly. To speed up the process, we optimized the RandomForestRegressor settings, including reducing the number of trees or using parallel processing during training.

5. **Model Performance:** The model performed well on test data but had a few discrepancies in certain edge cases, where cars with unusual features (e.g., rare models, new brands) were incorrectly priced. A more refined dataset or additional data collection could help.

10. Future Improvements

While the current model performs well, there are several areas where it could be improved:

1. **Additional Features:** Adding more features, such as vehicle condition, geographical location, and seller reputation, could further improve the model's predictions.

2. **Model Selection:** Exploring other machine learning models, such as **Gradient Boosting** or XGBoost, could yield better results, especially for regression tasks where the relationships between features and the target variable are non-linear.

3. **Advanced Feature Engineering:** Using techniques like "principal component analysis

(PCA)” for dimensionality reduction or creating interaction features between different variables could further enhance the model’s performance.

4. **Ensemble Methods:** Combining multiple models (e.g., Random Forest and Gradient Boosting) in an ensemble approach might improve accuracy by reducing bias and variance in the predictions.

5. **Deployment and Real-time Predictions:** While the project is deployed as an API, real-time predictions using a web-based interface could be improved by implementing caching for faster response times and adding real-time updates from external sources (e.g., car pricing websites).

6. **Cross-industry Applications:** The techniques used in this project could be applied to other domains such as real estate, electronics pricing, and more, where similar predictive modeling tasks are required.

11. Conclusion

In this project, we successfully developed a machine learning model to predict the price of used cars based on various features such as brand, model, year, mileage, transmission type, and fuel type. By following the complete machine learning pipeline—including data preprocessing, exploratory data analysis, model selection, and performance evaluation—we ensured that our model is robust and provides reasonable predictions.

The RandomForestRegressor model performed well, achieving an R^2 score of 0.80, indicating a strong correlation between the predicted and actual prices. The model was then deployed as a REST API using FastAPI and Render, making it accessible for real-time predictions.

Despite its success, the project encountered challenges such as handling missing data, addressing outliers, and mitigating overfitting. These challenges provided valuable learning opportunities, reinforcing the importance of data cleaning, feature engineering, and hyperparameter tuning.

For future improvements, incorporating additional features like vehicle condition, seller reputation, and location-based pricing could enhance model accuracy. Furthermore, experimenting with advanced models such as XGBoost and implementing an ensemble approach may yield better performance. Finally, integrating the model with a user-friendly web application would provide an interactive experience for users to get instant price estimates.

This project demonstrates the practical application of machine learning in the automotive industry and can be extended to other domains requiring predictive pricing models.

Links

Render: https://used-car-price-prediction-1-b7n2.onrender.com/docs

Github: https://github.com/betel09/used-car-price-prediction.git