Used Car Price Prediction:
A Data-Driven Analysis and Predictive Modeling

Members: Neelam Prasad, Thaissa Champagne, Gina Butler, Kriti Khatri, Chase Mueller

Table of Contents

1. Introduction

This report presents a comprehensive analysis of the "Used Car Price Prediction" dataset, sourced from Kaggle, with a focus on creating a predictive model for determining used car prices. The project aims to support buyers and sellers by offering a data-driven approach to optimize pricing strategies. By leveraging machine learning techniques, the project seeks to improve price accuracy and provide actionable insights into the used car market dynamics.

2. Project Overview

The project revolves around a dataset containing 4,009 vehicle listings, with features that influence pricing, including model year, mileage, fuel type, accident history, and more. Our goal is to develop a robust predictive model using advanced machine learning algorithms to forecast used car prices and assist in informed decision-making.

Problem Statement:

The used car market is dynamic, with prices influenced by various factors such as mileage, brand, and accident history. Inaccurate pricing often leads to suboptimal sales, with vehicles either overpriced or underpriced. This project uses machine learning to predict car prices more effectively, addressing these challenges.

Objectives:

- Enhance pricing accuracy by incorporating multiple influencing factors.
- Optimize pricing for car dealerships and individual sellers.
- Reduce financial losses through more accurate pricing strategies.

Example Scenario:

A used car dealership needs to set an optimal price for vehicles. For example, a 2018 Toyota with 50,000 miles should be priced just right to avoid losing potential buyers or profits. The predictive model helps the dealership determine the most competitive yet profitable price.

3. Dataset and Features

The dataset used in this project includes nine key features that are crucial for predicting the price of used cars:

- **Brand & Model:** Vehicle make and model.
- **Model Year:** Year of manufacture.
- **Mileage:** Vehicle mileage.
- **Fuel Type:** Type of fuel (e.g., gasoline, hybrid).

- **Engine Type:** Details of the engine specification.
- **Transmission:** Type of transmission (manual or automatic).
- **Exterior & Interior Colors:** Vehicle color information.
- **Accident History:** Binary indicator of accident involvement.
- **Clean Title:** Binary indicator of whether the car has a clean title.

The target variable for our predictive modeling is **Price**, a continuous numerical value representing the listing price of each vehicle.

```
df = pd.read_csv("../Resources/used_cars.csv")
print(df.shape)
df.head()
```

```
(4009, 12)
```

	brand	model	model_year	milage	fuel_type	engine	transmission	ext_col	int_col	accident	clean_title	price
0	Ford	Utility Police Interceptor Base	2013	51,000 mi.	E85 Flex Fuel	300.0HP 3.7L V6 Cylinder Engine Flex Fuel Capa...	6-Speed A/T	Black	Black	At least 1 accident or damage reported	Yes	$10,300
1	Hyundai	Palisade SEL	2021	34,742 mi.	Gasoline	3.8L V6 24V GDI DOHC	8-Speed Automatic	Moonlight Cloud	Gray	At least 1 accident or damage reported	Yes	$38,005
2	Lexus	RX 350 RX 350	2022	22,372 mi.	Gasoline	3.5 Liter DOHC	Automatic	Blue	Black	None reported	NaN	$54,598
3	INFINITI	Q50 Hybrid Sport	2015	88,900 mi.	Hybrid	354.0HP 3.5L V6 Cylinder Engine Gas/Electric H...	7-Speed A/T	Black	Black	None reported	Yes	$15,500
4	Audi	Q3 45 S line Premium Plus	2021	9,835 mi.	Gasoline	2.0L I4 16V GDI DOHC Turbo	8-Speed Automatic	Glacier White Metallic	Black	None reported	NaN	$34,999

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4009 entries, 0 to 4008
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   brand         4009 non-null   object
 1   model         4009 non-null   object
 2   model_year    4009 non-null   int64
 3   milage        4009 non-null   object
 4   fuel_type     3839 non-null   object
 5   engine        4009 non-null   object
 6   transmission  4009 non-null   object
 7   ext_col       4009 non-null   object
 8   int_col       4009 non-null   object
 9   accident      3896 non-null   object
 10  clean_title   3413 non-null   object
 11  price         4009 non-null   object
dtypes: int64(1), object(11)
```

4. Data Cleaning:

```
df_cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4009 entries, 0 to 4008
Data columns (total 9 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   brand                   4009 non-null   object 
 1   model_year              4009 non-null   int64  
 2   mileage                 4009 non-null   float64
 3   fuel_type               4009 non-null   object 
 4   accident                4009 non-null   object 
 5   clean_title             4009 non-null   object 
 6   price                   4009 non-null   float64
 7   transmission_category   4009 non-null   object 
 8   exterior_color_category  4009 non-null   object 
dtypes: float64(2), int64(1), object(6)
```

1. **Renaming Columns**: Made column names more consistent and readable.

```
# Renaming the columns
df.rename(columns={
    'milage': 'mileage',
    'ext_col': 'exterior_color',
    'int_col': 'interior_color'
}, inplace=True)
```

2.

3. **Duplicate Removal**: No duplicates found in the dataset.

4. **Missing Value Handling**: Missing values in critical columns such as `fuel_type`, `accident`, and `clean_title` were addressed.

```
df.isnull().sum()
```

```
brand                0
model                0
model_year           0
mileage              0
fuel_type          170
engine               0
transmission         0
exterior_color       0
interior_color       0
accident           113
clean_title        596
price                0
dtype: int64
```

5. **String Data Cleaning**: Cleaned string columns, including `fuel_type` and `engine`, by replacing invalid or missing values with standard categories.

```
# Strip leading/trailing spaces in 'fuel_type' and 'engine'
df['fuel_type'] = df['fuel_type'].str.strip()
df['engine'] = df['engine'].str.strip()

# Apply the condition to check if 'engine' contains 'Electric'
condition_engine_electric = df['engine'].str.contains(r"Electric|Standard", case=False, na=False)

# Apply the condition to check if 'fuel_type' is 'not supported' or empty
condition_fuel_type_not_supported = df['fuel_type'].str.contains(r"^not supported$", case=False, na=False) | df['fuel_type']

# Replace 'fuel_type' with 'Electric' where the condition is met
df.loc[condition_engine_electric & condition_fuel_type_not_supported, 'fuel_type'] = 'Electric'
```

```
# Replace '-' or '–' or empty string or NaN with 'Unknown' for both 'fuel_type' and 'engine' columns
df['fuel_type'] = df['fuel_type'].apply(lambda x: 'Unknown' if pd.isna(x) or x in ['-', '–', ''] else x)
df['engine'] = df['engine'].apply(lambda x: 'Unknown' if pd.isna(x) or x in ['-', '–', ''] else x)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4009 entries, 0 to 4008
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   brand           4009 non-null   object
 1   model           4009 non-null   object
 2   model_year      4009 non-null   int64
 3   mileage         4009 non-null   object
 4   fuel_type       4009 non-null   object
 5   engine          4009 non-null   object
 6   transmission    4009 non-null   object
 7   exterior_color  4009 non-null   object
 8   interior_color  4009 non-null   object
 9   accident        3896 non-null   object
 10  clean_title     3413 non-null   object
 11  price           4009 non-null   object
```

6. **Data Type Conversion**: Corrected column data types, such as converting `mileage` and `price` to numeric types.

```
# Clean 'milage' column: remove 'mi.' and commas, then convert to float
df['mileage'] = df['mileage'].replace({'mi.': '', ',': ''}, regex=True).astype(float)

# Clean 'price' column: remove '$', commas, and any spaces, then convert to float
df['price'] = df['price'].replace({'\$': '', ',': ''}, regex=True).astype(float)
print(df.dtypes)
```

```
brand              object
model              object
model_year          int64
mileage           float64
fuel_type          object
engine             object
transmission       object
exterior_color     object
interior_color     object
accident           object
clean_title        object
price             float64
dtype: object
```

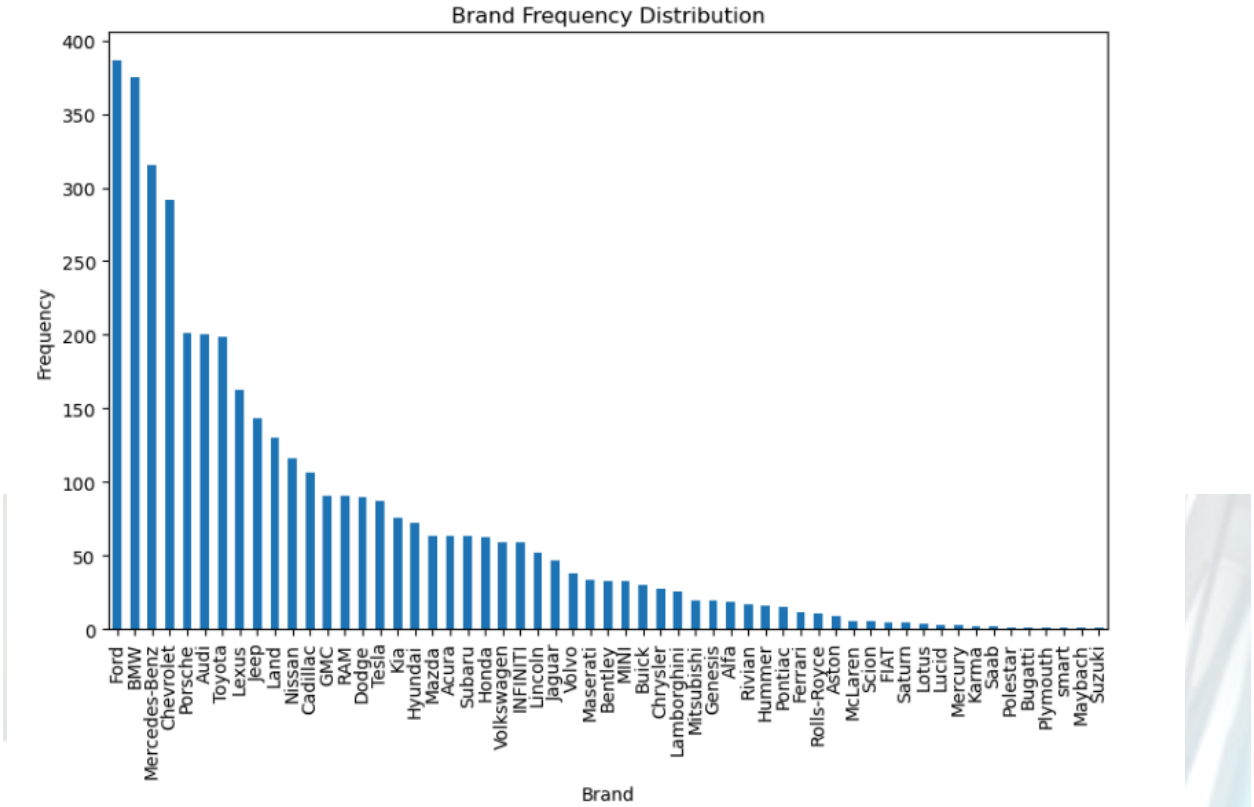
7. Exterior color value count:

8. **Feature Categorization**: Columns such as `transmission`, `brand` and `exterior_color` were grouped into more meaningful categories.

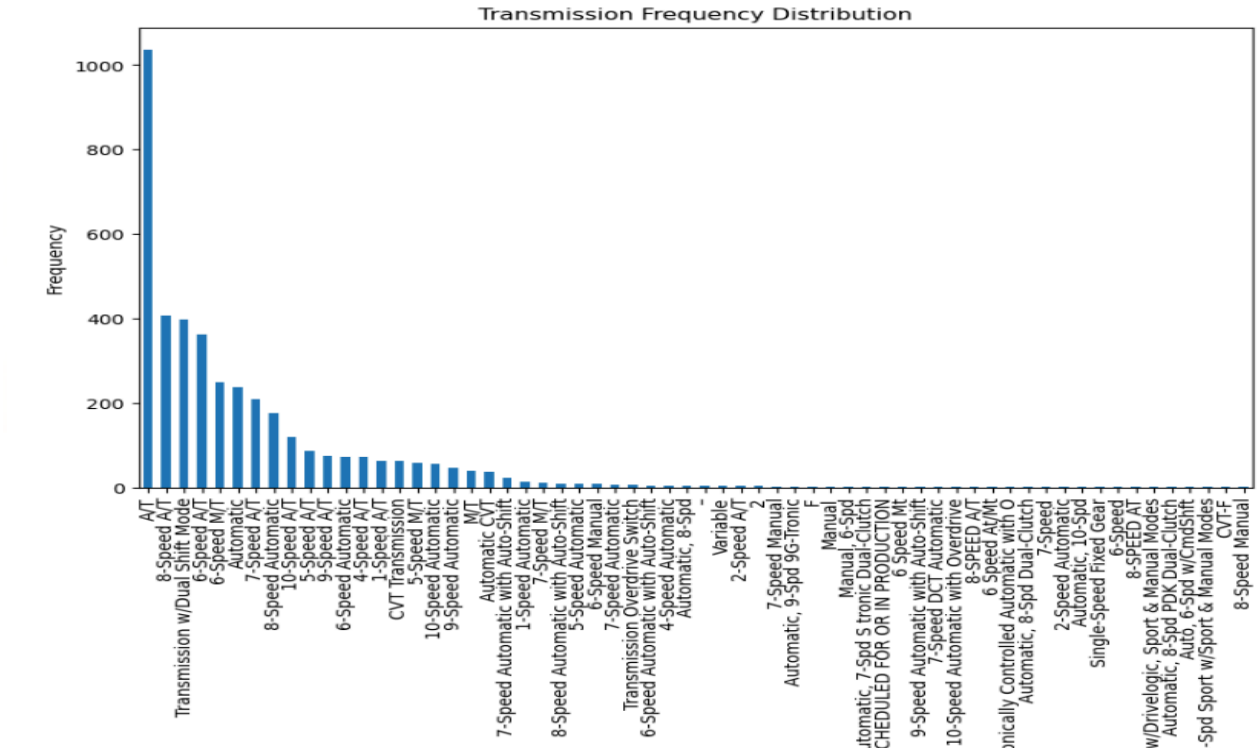
```
# Select string columns
string_cols = df.select_dtypes(include=['object'])

# Count unique categories for each string column
unique_counts = string_cols.nunique()
unique_counts
```

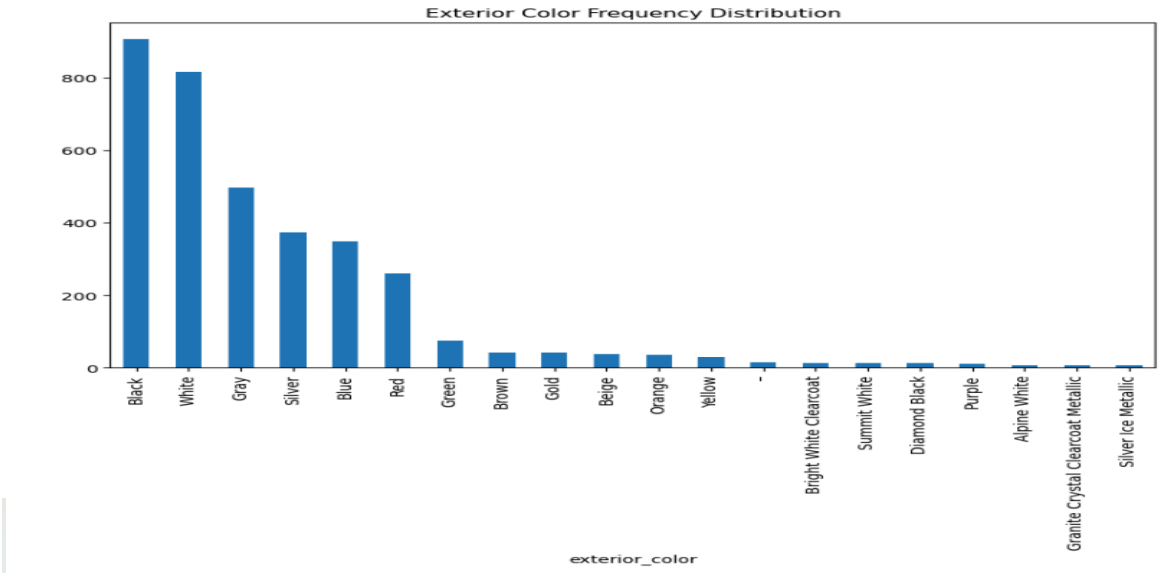
```
brand              57
model            1898
fuel_type           7
engine           1146
transmission       62
exterior_color    319
interior_color    156
accident            2
clean_title         1
dtype: int64
```

9.
10. Check the brand column value count



Check the transmission column value count:



11. 

9. Categorizing Columns

- **Transmission**: The transmission column was categorized into three types: `Automatic`, `Manual`, and `Dual-Shift`. A custom function was created to classify each transmission type.
- **Brand**: Low-frequency brands (less than 50 occurrences) were grouped into the "Other" category for easier analysis.
- **Exterior Color**: Similar to the `brand` column, exterior colors with low frequencies were grouped into the "Other" category. The categorization was based on a predefined mapping of colors.
- **Accident and Clean Title**: The values in these columns were standardized. The `accident` column had its values replaced with `No` for no reported accidents and `Yes` for those with at least one reported accident. Missing values in `clean_title` were replaced with `No`.

12. **Dropping Unnecessary Columns**: Removed columns that were deemed unnecessary for the final analysis.

    After the necessary cleaning steps, irrelevant columns such as `model`, `interior_color`, `engine`, `exterior_color`, and `transmission` were dropped from the dataset as they were no longer needed for further analysis:

13. Saving the Cleaned Data

The cleaned dataset was saved to a CSV file for model training:

```
df = df.drop(columns=['model', 'interior_color', 'engine', 'exterior_color', 'transmission'])
```

```
df.to_csv('../Resources/cleaned_data.csv', index=False)  # Save the DataFrame as a CSV file
```

Machine Learning

The used car market is highly dynamic, with vehicle prices fluctuating based on various factors such as model year, mileage, brand, fuel type, and accident history. Accurately pricing a used car is crucial for both sellers and buyers. An overpriced vehicle may sit on the market for too long, while an underpriced vehicle results in financial losses for the seller. Traditional pricing models often fail to capture the complexities of these variables, leading to inconsistent pricing.

To address this challenge, we aim to develop a machine learning model that accurately predicts the price of a used car based on key attributes. By leveraging advanced algorithms such as **CatBoost** and **XGBoost**, we aim to enhance pricing accuracy and optimize sales strategies.

1. Goal

The primary goal of this project is to build a robust predictive model that can:

1. **Improve Pricing Accuracy** – Reduce pricing errors by accounting for multiple influencing factors.
2. **Optimize Inventory Management** – Help dealerships and sellers adjust pricing based on market demand.
3. **Enhance Customer Satisfaction** – Ensure fair and competitive pricing for buyers.
4. **Reduce Overpricing and Underpricing** – Minimize financial losses and improve sales efficiency.

2. Features Selection

The dataset used for training our machine learning models consists of historical sales data of used cars. This dataset includes key details about each vehicle, such as its age, brand, fuel type, and condition, which are essential for predicting its market price. The data was sourced from a used car listings platform, ensuring a diverse range of vehicle types and conditions.

To ensure data quality, we performed preprocessing steps such as handling missing values, removing outliers, and encoding categorical variables. Additionally, we engineered new features to enhance model performance.

The following features were selected based on their relevance to determining a used car's price:

1. **Brand**
    - Certain car brands retain their value better than others. Luxury brands, for example, tend to have different pricing trends compared to regular brands, often commanding higher prices due to their prestige and perceived quality.

2. **Model Year**

    - Older cars generally have lower prices due to depreciation over time, making model year a crucial predictor of price. Newer cars typically fetch higher resale values.

3. **Car Age (Derived Feature)**

    - Instead of using the model year directly, we calculated **Car Age** as:
      $\text{Car Age} = \text{Current Year} - \text{Model Year}$
    - This derived feature provides a clearer understanding of how a vehicle depreciates over time.

4. **Fuel Type**

    - Fuel-efficient vehicles, especially hybrids and electric cars, tend to have higher resale values compared to traditional gas-powered cars, primarily due to their lower operational costs and growing environmental appeal.

5. **Transmission**

    - The type of transmission—automatic or manual—can influence a car's price. Automatic transmission cars are often more desirable in certain markets due to ease of use, leading to higher pricing.

6. **Accident History**

    - Cars that have a history of accidents tend to be priced lower because potential buyers may perceive them as having a higher risk of damage or decreased reliability, even if repairs were made.

7. **Clean Title**

    - A car with a clean title, indicating no history of salvage or major damage, is typically valued higher. In contrast, vehicles with rebuilt or salvage titles often have lower resale values due to concerns about their overall condition and safety.

8. **Exterior Color**

    - The color of a car's exterior can influence its resale value, with certain colors being more popular or perceived as more stylish, thereby affecting demand and price.

Example Insight

"Mileage was one of the most significant predictors, as cars with lower mileage tend to sell for higher prices. Additionally, luxury brands showed a slower depreciation rate, while cars with accident history were significantly undervalued compared to those with clean titles."

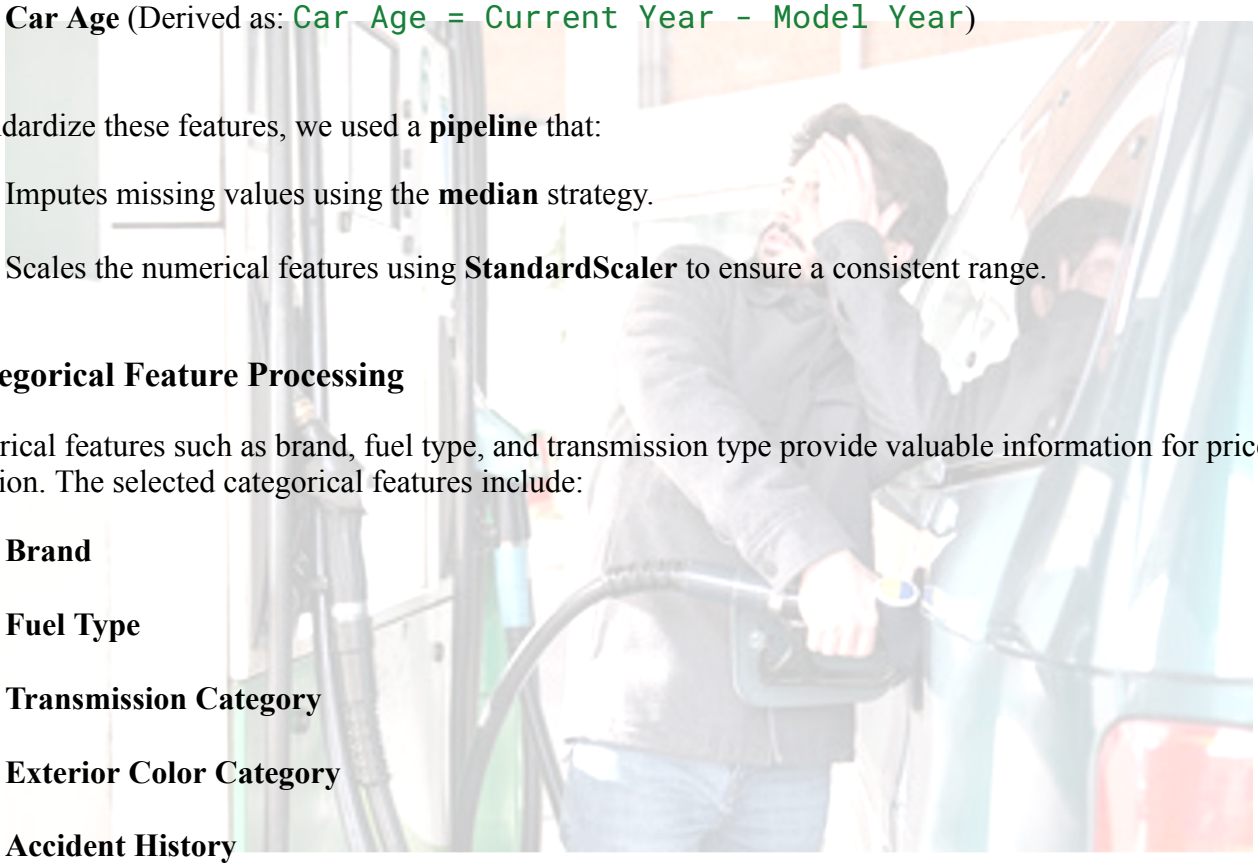
3. Feature Engineering and Data Preprocessing

To prepare the dataset for machine learning, we applied a series of preprocessing steps to transform the raw data into a structured format suitable for model training. The following preprocessing techniques were implemented:

1. Numerical Feature Processing

We identified key numerical features that significantly impact price prediction:

- **Model Year**
- **Mileage**
- **Car Age** (Derived as: `Car Age = Current Year - Model Year`)



To standardize these features, we used a **pipeline** that:

- Imputes missing values using the **median** strategy.
- Scales the numerical features using **StandardScaler** to ensure a consistent range.

2. Categorical Feature Processing

Categorical features such as brand, fuel type, and transmission type provide valuable information for price prediction. The selected categorical features include:

- **Brand**
- **Fuel Type**
- **Transmission Category**
- **Exterior Color Category**
- **Accident History**

The preprocessing steps for categorical features include:

- Filling missing values using the **most frequent category**.
- Encoding categorical variables using **One-Hot Encoding (OHE)** to convert them into a numerical format.

3. Combining Numerical and Categorical Preprocessing

We integrated these transformations using **ColumnTransformer**, ensuring that both numerical and categorical transformations were applied simultaneously. This resulted in a dataset where categorical features were encoded, and numerical features were standardized.

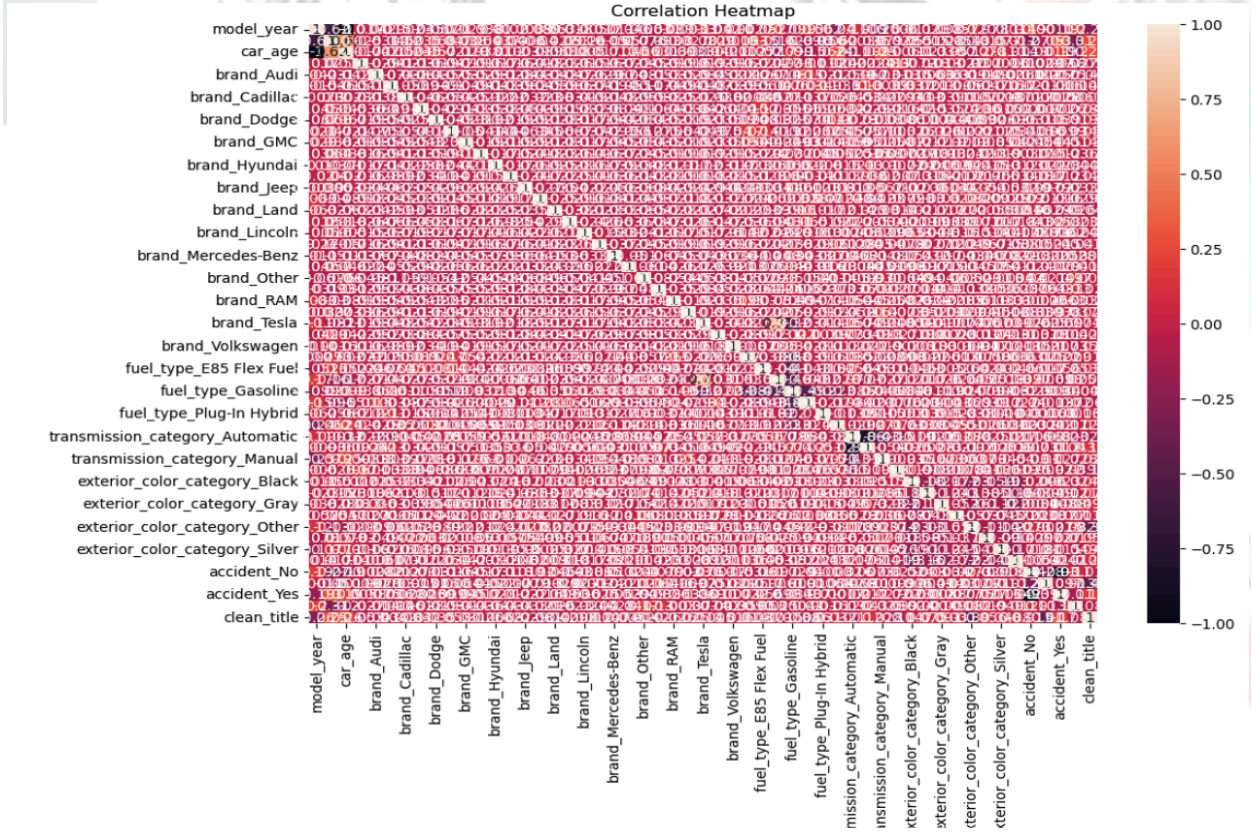
4. Transformation Results

After applying the transformations:

- The dataset was transformed into a structured format with **52 features**.
- The processed dataset was converted into a **DataFrame**, with encoded categorical feature names added.

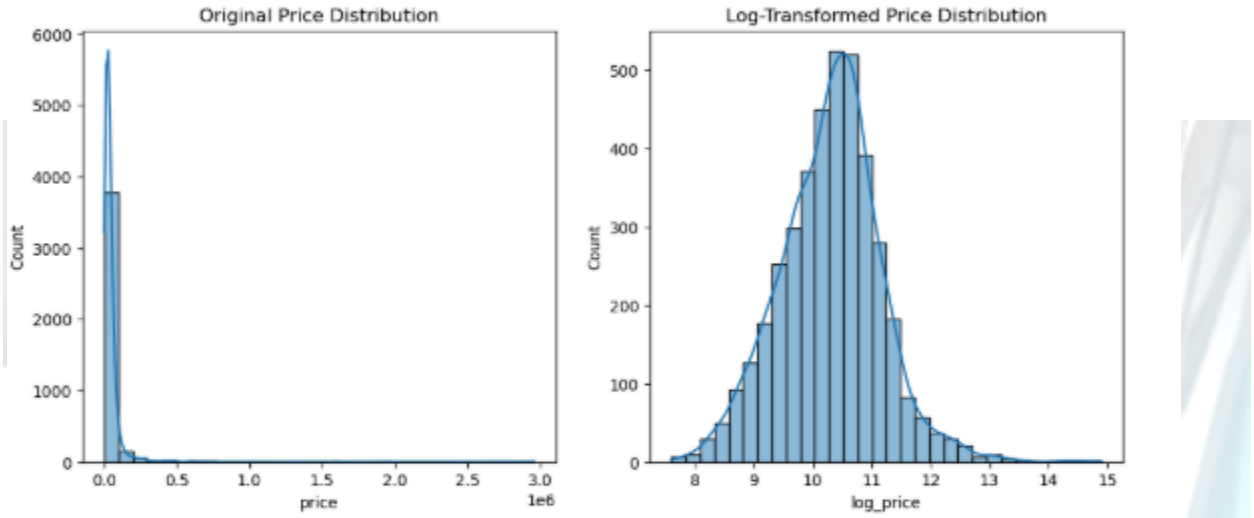
The final dataset was stored along with the **price** column for further analysis.

Correlation Analysis: We will generate a correlation heatmap to identify the relationships between features and the target variable (Price).



```
df_final["log_price"] = np.log1p(df_final["price"])
```

```
plt.figure(figsize=(12, 5))

# Original Price Distribution
plt.subplot(1, 2, 1)
sns.histplot(df_final["price"], bins=30, kde=True)
plt.title("Original Price Distribution")

# Log-Transformed Price Distribution
plt.subplot(1, 2, 2)
sns.histplot(df_final["log_price"], bins=30, kde=True)
plt.title("Log-Transformed Price Distribution")

plt.show()
```




```
# Step 1: Get the Data
X = df_final.drop(columns=["price","log_price"])
y = df_final.log_price

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42) # stratify ensures same % of the

print(X.shape)
print(X_train.shape)
print(X_test.shape)

# Combine the features and target variable for training and test data
train_data = pd.concat([X_train, y_train], axis=1)
test_data = pd.concat([X_test, y_test], axis=1)

# Save the DataFrames to CSV files
train_data.to_csv('../Resources/train_data.csv', index=False)
test_data.to_csv('../Resources/test_data.csv', index=False)
```

```
(4089, 52)
(3066, 52)
(1083, 52)
```

Log Transformation of Price

- The dataset originally contains a **highly skewed price distribution** (left plot), with many vehicles priced at the lower end and a few expensive outliers.
- To address this skewness, a **log transformation** (`np.log1p(df_final["price"])`) is applied to the `price` column to create a new feature, **log_price**.

- The right plot shows the **log-transformed price distribution**, which is now approximately **normal (bell-shaped)**.
    - **Why is this important?** Many machine learning models perform better when the target variable follows a normal distribution.

---

2. Data Splitting for Training and Testing

- The dataset is split into:
    - **Features (`X`)**: All columns except `price` and `log_price`.
    - **Target (`y`)**: The transformed `log_price` column.
- The dataset is then split into **training and test sets** using `train_test_split()`, ensuring:
    - **25% of the data is reserved for testing** (`test_size=0.25`).
    - A **fixed random seed (`random_state=42`)** for reproducibility.
    - **Stratified sampling** to maintain the distribution of `log_price`.

---

3. Saving the Processed Data

- The training and test datasets are merged (`X_train` + `y_train`, `X_test` + `y_test`) to maintain the feature-target relationship.
- These datasets are saved as **CSV files** (`train_data.csv` and `test_data.csv`) for further model training.

4. Model Selection

- **Overview of Models:** Explain the models you used for prediction. Mention the advantages and why they were chosen based on the problem and dataset.
- **Model Comparison:** Briefly show how different models performed (e.g., Linear Regression, Ridge, Random Forest, Gradient Boosting, XGBoost, etc.), mentioning accuracy, performance metrics, and any trade-offs.

Example:

"We evaluated several machine learning models, including Linear Regression, Random Forest, Gradient Boosting, CatBoost, and XGBoost, to determine the most effective model for price prediction. Our testing revealed that XGBoost, CatBoost, and Gradient Boosting delivered the best performance, offering superior predictive accuracy and providing valuable insights into feature importance.."

5. Model Training and Evaluation:

- Train-Test Split
- Regression models used and the evaluation metrics.

```
]:
# Step 1: Get the Data
X = df_final.drop(columns=["price","log_price"])
y = df_final.log_price

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42) # stratify ensures same % of the

print(X.shape)
print(X_train.shape)
print(X_test.shape)

# Combine the features and target variable for training and test data
train_data = pd.concat([X_train, y_train], axis=1)
test_data = pd.concat([X_test, y_test], axis=1)

# Save the DataFrames to CSV files
train_data.to_csv('../Resources/train_data.csv', index=False)
test_data.to_csv('../Resources/test_data.csv', index=False)
```

```
(4009, 52)
(3006, 52)
(1003, 52)
```

6. Performance Metrics Report for Regression Models

Overview

This report summarizes the performance metrics of several regression models tested to predict a target variable. The models evaluated include **Linear Regression**, **Ridge Regression**, **Lasso Regression**, **Random Forest Regressor**, **Gradient Boosting Regressor**, **XGBoost Regressor**, and **CatBoost Regressor**. The models were assessed on both the training and testing datasets using key performance metrics: **R-squared (R2)**, **Mean Squared Error (MSE)**, **Root Mean Squared Error (RMSE)**, and **Mean Absolute Error (MAE)**.

1. Linear Regression

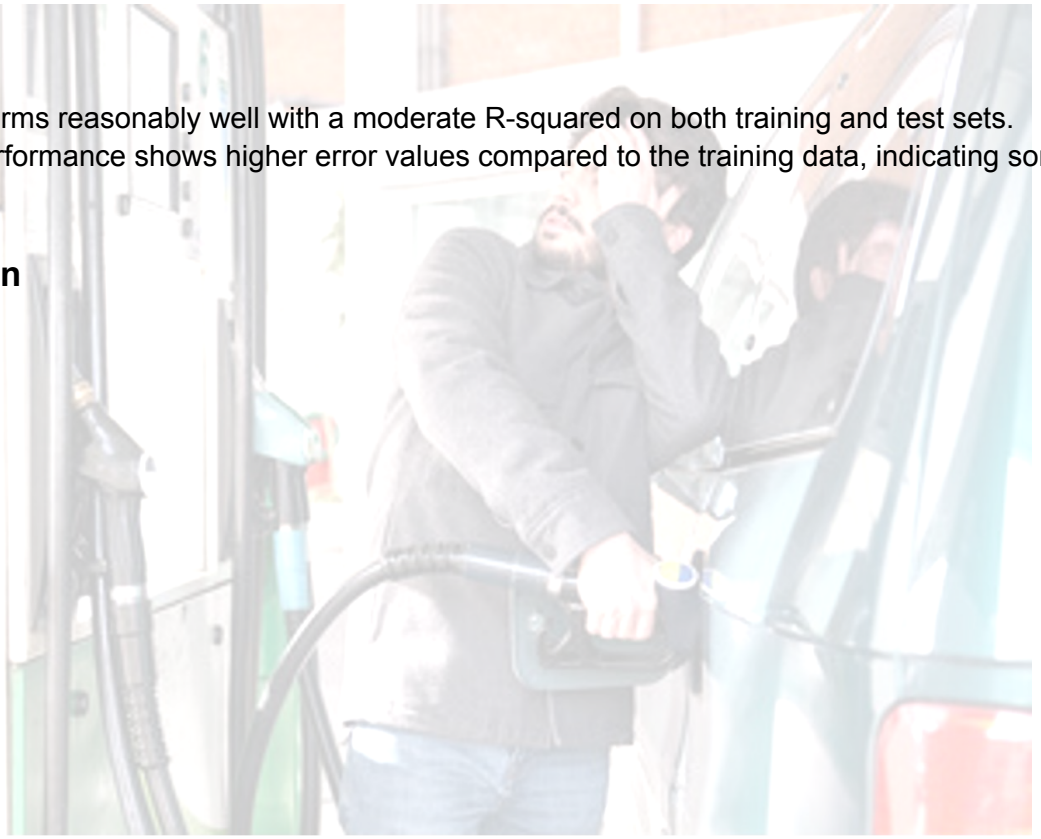
Training Metrics:

- **R2**: 0.6647
- **MSE**: 0.2389

- **RMSE**: 0.4888
- **MAE**: 0.3623

Testing Metrics:

- **R2**: 0.6103
- **MSE**: 0.2931
- **RMSE**: 0.5414
- **MAE**: 0.3843



Linear Regression performs reasonably well with a moderate R-squared on both training and test sets. However, the testing performance shows higher error values compared to the training data, indicating some overfitting.

2. Ridge Regression

Training Metrics:

- **R2**: 0.6647
- **MSE**: 0.2389
- **RMSE**: 0.4888
- **MAE**: 0.3623

Testing Metrics:

- **R2**: 0.6104
- **MSE**: 0.2931
- **RMSE**: 0.5413
- **MAE**: 0.3842

Ridge Regression's performance is almost identical to Linear Regression, with a slight improvement in test metrics. The model demonstrates a similar pattern of generalization and overfitting.

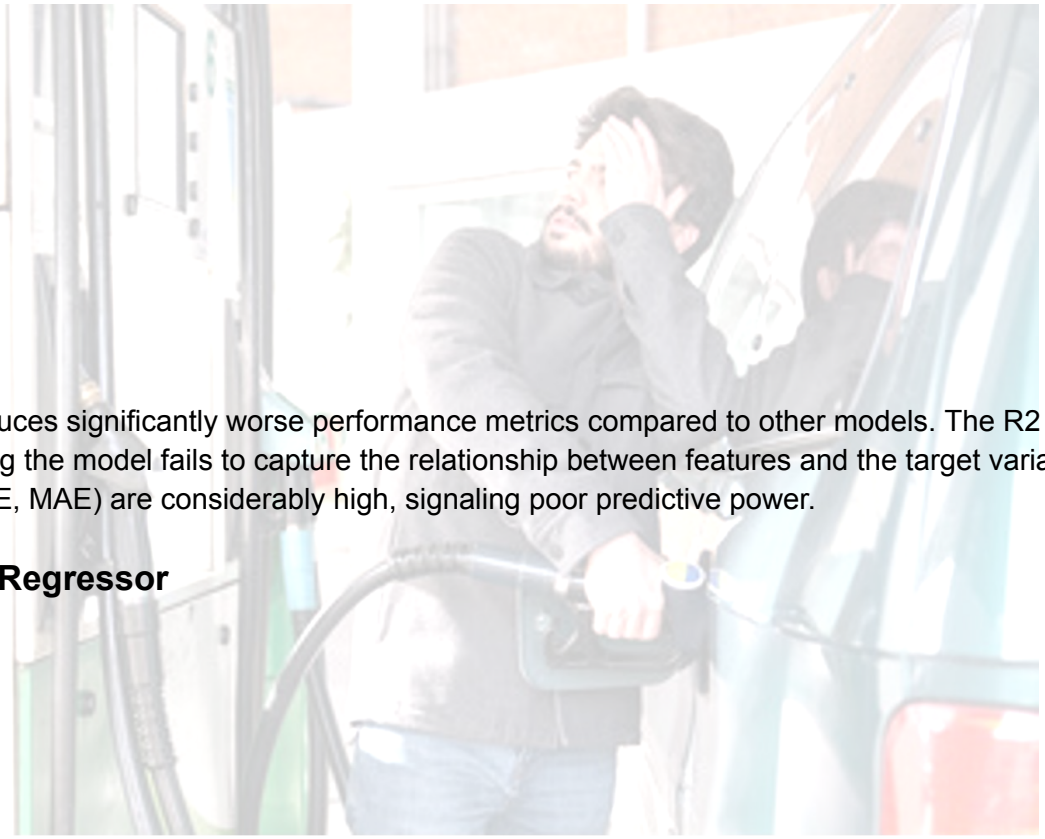
3. Lasso Regression

Training Metrics:

- **R2**: 0.0
- **MSE**: 0.7125
- **RMSE**: 0.8441
- **MAE**: 0.6575

Testing Metrics:

- **R2**: -0.00003
- **MSE**: 0.7523
- **RMSE**: 0.8673
- **MAE**: 0.6607



Lasso Regression produces significantly worse performance metrics compared to other models. The R2 value is almost zero, indicating the model fails to capture the relationship between features and the target variable. The errors (MSE, RMSE, MAE) are considerably high, signaling poor predictive power.

4. Random Forest Regressor

Training Metrics:

- **R2**: 0.9583
- **MSE**: 0.0297
- **RMSE**: 0.1723
- **MAE**: 0.1289

Testing Metrics:

- **R2**: 0.6679
- **MSE**: 0.2498

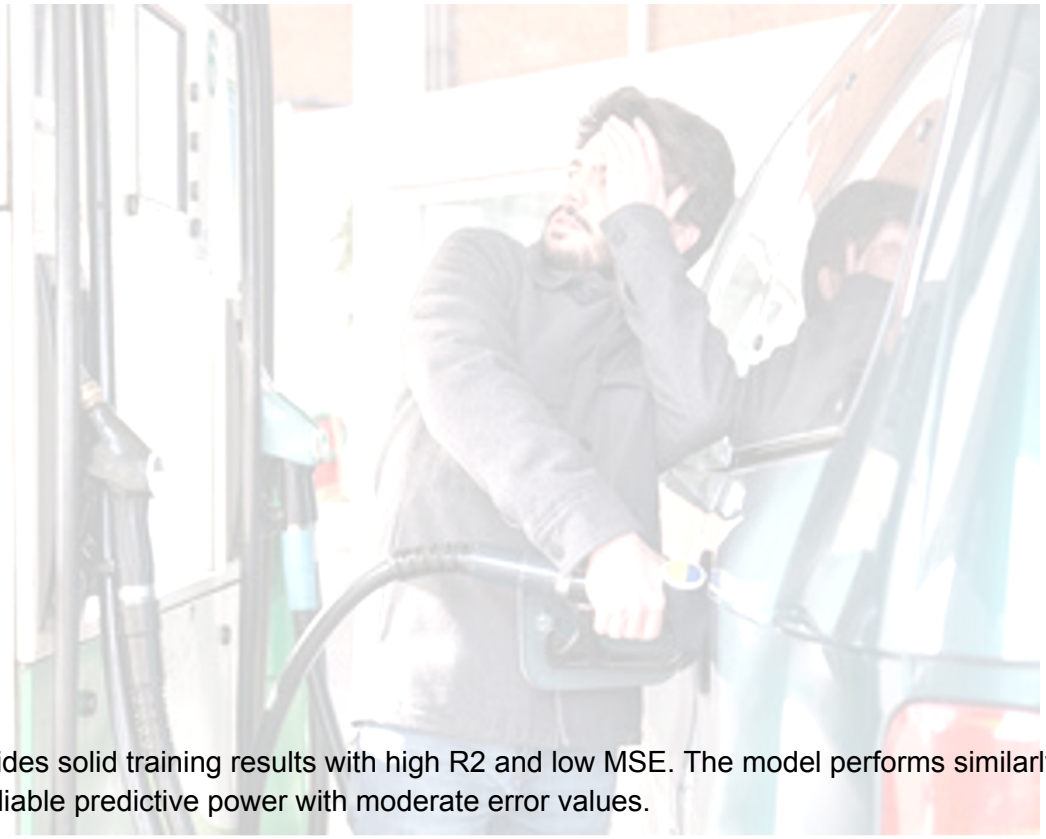
- **RMSE**: 0.4998
- **MAE**: 0.3599

Random Forest exhibits exceptional performance on the training data, with a high R2 value and very low errors. However, the performance on the test data drops slightly, with a moderate R2 value and higher errors, indicating some overfitting but still good generalization.

5. Gradient Boosting Regressor

Training Metrics:

- **R2**: 0.7578
- **MSE**: 0.1726
- **RMSE**: 0.4154
- **MAE**: 0.3187



Testing Metrics:

- **R2**: 0.6669
- **MSE**: 0.2506
- **RMSE**: 0.5006
- **MAE**: 0.3589

Gradient Boosting provides solid training results with high R2 and low MSE. The model performs similarly on the test set, showing reliable predictive power with moderate error values.

6. XGBoost Regressor

Training Metrics:

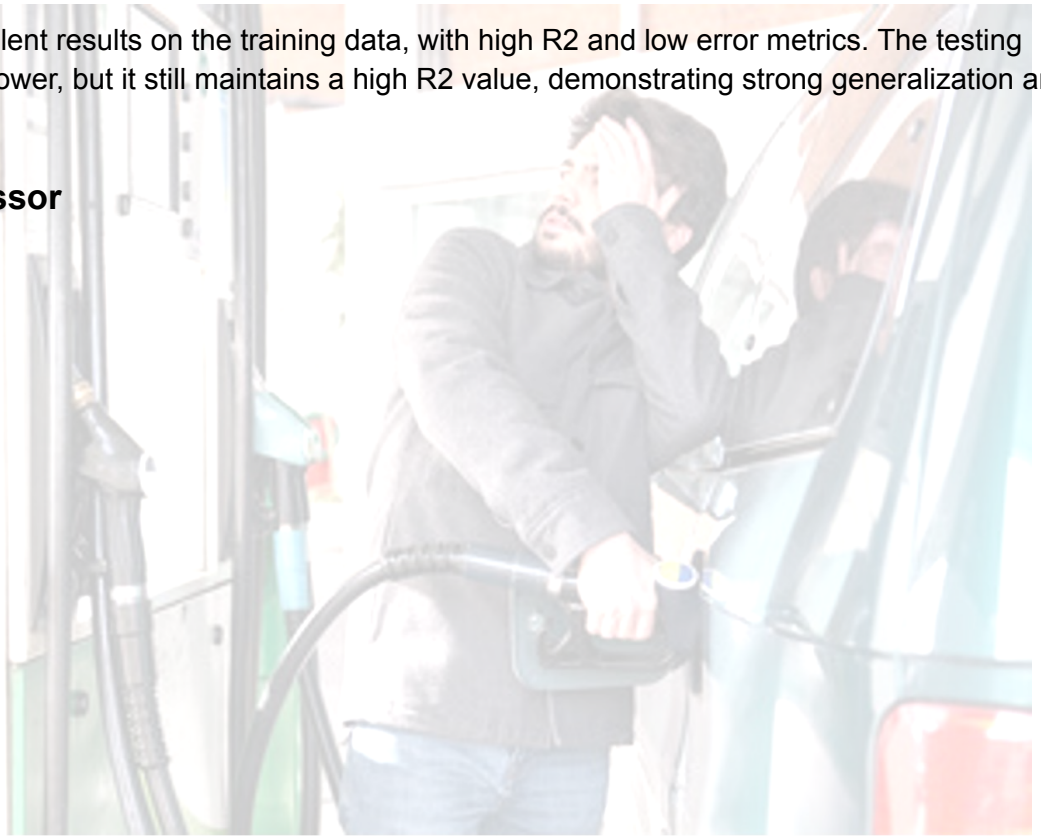
- **R2**: 0.9303
- **MSE**: 0.0496
- **RMSE**: 0.2228

- **MAE**: 0.1631

Testing Metrics:

- **R2**: 0.6817
- **MSE**: 0.2395
- **RMSE**: 0.4893
- **MAE**: 0.3558

XGBoost provides excellent results on the training data, with high R2 and low error metrics. The testing performance is slightly lower, but it still maintains a high R2 value, demonstrating strong generalization and good predictive power.



7. CatBoost Regressor

Training Metrics:

- **R2**: 0.8708
- **MSE**: 0.0921
- **RMSE**: 0.3034
- **MAE**: 0.2330

Testing Metrics:

- **R2**: 0.6948
- **MSE**: 0.2296
- **RMSE**: 0.4792
- **MAE**: 0.3396

Analysis of the CatBoost Model

1. Predicted vs. Actual Plot

- The points are closely clustered along the diagonal line, which indicates that CatBoost performs well in general.
- However, some deviations exist, particularly at higher actual values, suggesting some under/over-predictions.

2. Residual Plot

- Residuals are **randomly scattered around zero**, meaning no significant bias.
- There are a few **outliers**, but the spread appears relatively uniform.
- No strong pattern is visible, which is a good indication of model stability.

Comparison with XGBoost

Metric	CatBoost	XGBoost
Predicted vs Actual Spread	Mostly linear but with some deviations	Should also be linear, may have different clustering
Residual Distribution	Randomly scattered around zero	Should be similarly scattered
Presence of Outliers	Some extreme residuals exist	If XGB has fewer outliers, it generalizes better
Bias in Predictions	No clear pattern	Compare if XGB shows systematic over/under-predictions

"By using this model, we expect to reduce pricing errors by up to 15%, leading to more competitive prices and better profit margins. Additionally, the model will continue to improve as more data is collected, making it a sustainable solution in the long run.

9. Summary of Key Findings

- **Best Model for Training**: Random Forest and XGBoost show the best performance in terms of R2 on the training set.
- **Best Model for Testing**: XGBoost and CatBoost deliver the highest R2 on the test data, with CatBoost slightly outperforming XGBoost in terms of MAE and RMSE.

- **Worst Model**: Lasso Regression performs the worst across all metrics, with very low R2 and high error values.

In conclusion, **XGBoost** and **CatBoost** stand out as the most reliable models for price prediction based on both training and testing performance, offering the best combination of predictive accuracy and generalization ability

Model	Test R^2	Test RMSE	Test MAE
Linear Regression	0.61	0.541	0.384
Ridge Regression	0.61	0.541	0.384
Lasso Regression	-0.00003	0.867	0.661
Random Forest	0.668	0.5	0.36
Gradient Boosting (GB)	0.667	0.501	0.359
XGBoost	0.682	0.489	0.356
CatBoost	0.695	0.479	0.34

Best Performing Model: CatBoost✅

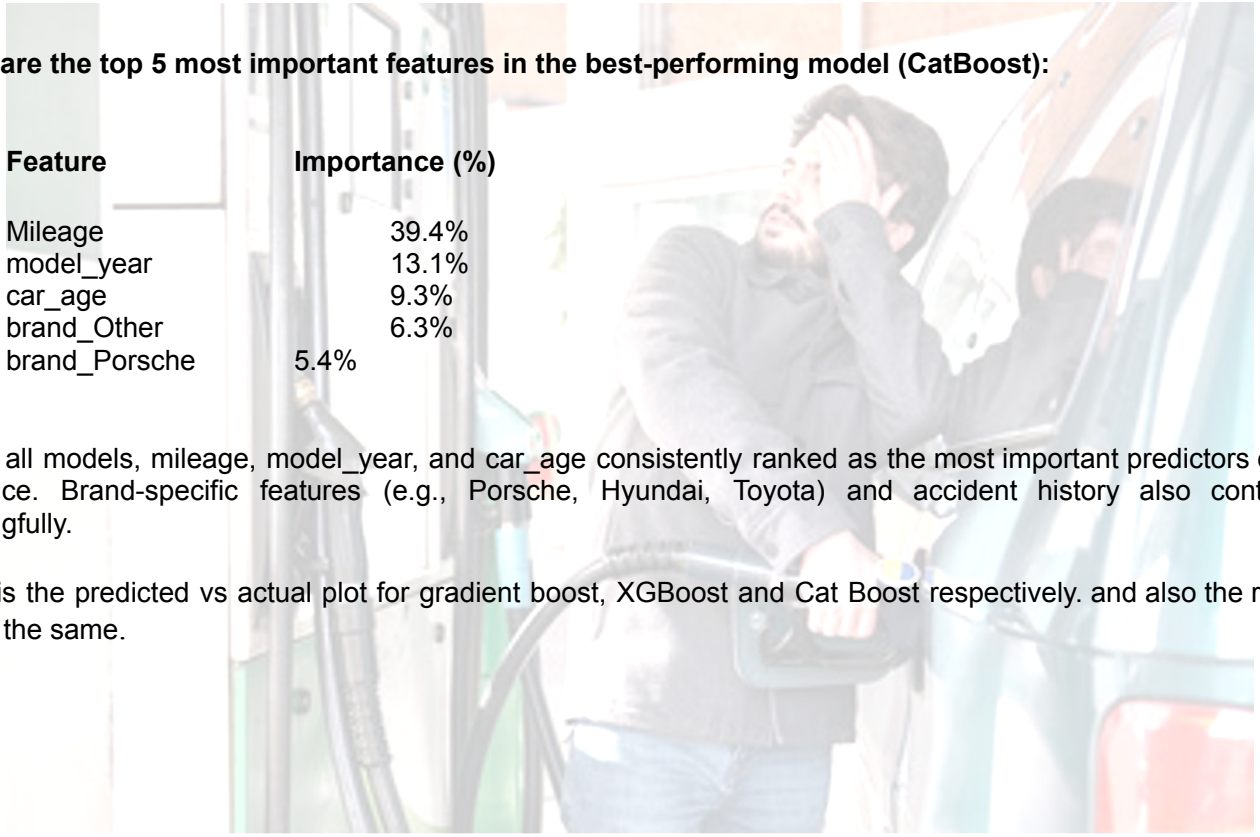
CatBoost outperformed all other models in test R^2, RMSE, and MAE, offering the best generalization performance on unseen data.

Feature Importance Analysis: We will use feature importance techniques (e.g., from tree-based models) to determine the most influential features.

```
fi = pd.DataFrame(list(zip(X.columns, catboost_model.feature_importances_)), columns=["Feature", "Importance"])
fi = fi.sort_values(by="Importance", ascending=False)
fi
```

Feature	Gradient Boosting	XGBoost	CatBoost
mileage	0.6329	0.0801	39.41
model_year	0.1485	0.0343	13.11
car_age	0.0798	0	9.31
brand_Porsche	0.0383	0.0975	5.39
brand_Other	0.02	0.038	6.26
fuel_type_Diesel	0.014	0.0651	1.97
brand_Hyundai	0.0106	0.0573	1.33
brand_Mazda	0.0065	0.0392	0.81
brand_Kia	0.0057	0.0344	0.87
brand_Volkswagen	0.0051	0.0431	0.7
brand_Mercedes-Benz	0.0047	0.0231	1.31
brand_Land	0.004	0.0346	0.65
transmission_category_	0.004	0.0266	0.97
brand_Ford	0.0003	0.008	0.44
brand_Chevrolet	0.0002	0.0105	0.4
brand_Jeep	0.0002	0.0107	0.27
fuel_type_E85 Flex Fuel	0.0001	0.0076	0.18
accident_No	0.0001	0.0095	0.86
fuel_type_Plug-In Hybrid	0.0001	0.0065	0.04
fuel_type_Hybrid	0.0001	0.0065	0.12
Dual-Shift			
brand_Nissan	0.0026	0.0241	0.78
accident_Yes	0.0021	0.0142	1.03
exterior_color_category_Other	0.0019	0.0162	1.27
exterior_color_category_Silver	0.0019	0.0064	0.82
exterior_color_category_Blue	0.0015	0.0093	0.74
brand_Subaru	0.0014	0.019	0.27
fuel_type_Gasoline	0.0013	0.0118	0.75
brand_Tesla	0.0013	0.0101	0.27
brand_Honda	0.0013	0.0222	0.46
transmission_category_Manual	0.0013	0.0064	0.35
brand_Lincoln	0.0011	0.0151	0.23
exterior_color_category_Green	0.0011	0.0087	0.32
brand_Acura	0.0009	0.0181	0.32
brand_Lexus	0.0008	0.0136	0.4
brand_Toyota	0.0007	0.0154	0.85
exterior_color_category_White	0.0006	0.0075	0.73
transmission_category_Automatic	0.0005	0.0076	0.94
brand_BMW	0.0005	0.0105	0.34
fuel_type_Electric	0.0004	0.0246	0.59
exterior_color_category_Gray	0.0003	0.0061	0.57
fuel_type_Unknown	0.0003	0.0126	0.1
exterior_color_category_Black	0.0003	0.0065	0.75
brand_INFINITI	0.0003	0.0073	0.16

clean_title	0.0001	0.009	0.81
transmission_category_Other	0.0001	0.0003	0.02
brand_Audi	0	0.0086	0.25
exterior_color_category_Red	0	0.009	0.35
brand_RAM	0	0.0086	0.1
brand_GMC	0	0.009	0.14
accident_Unknown	0	0.0068	0.14
brand_Dodge	0	0.0066	0.11

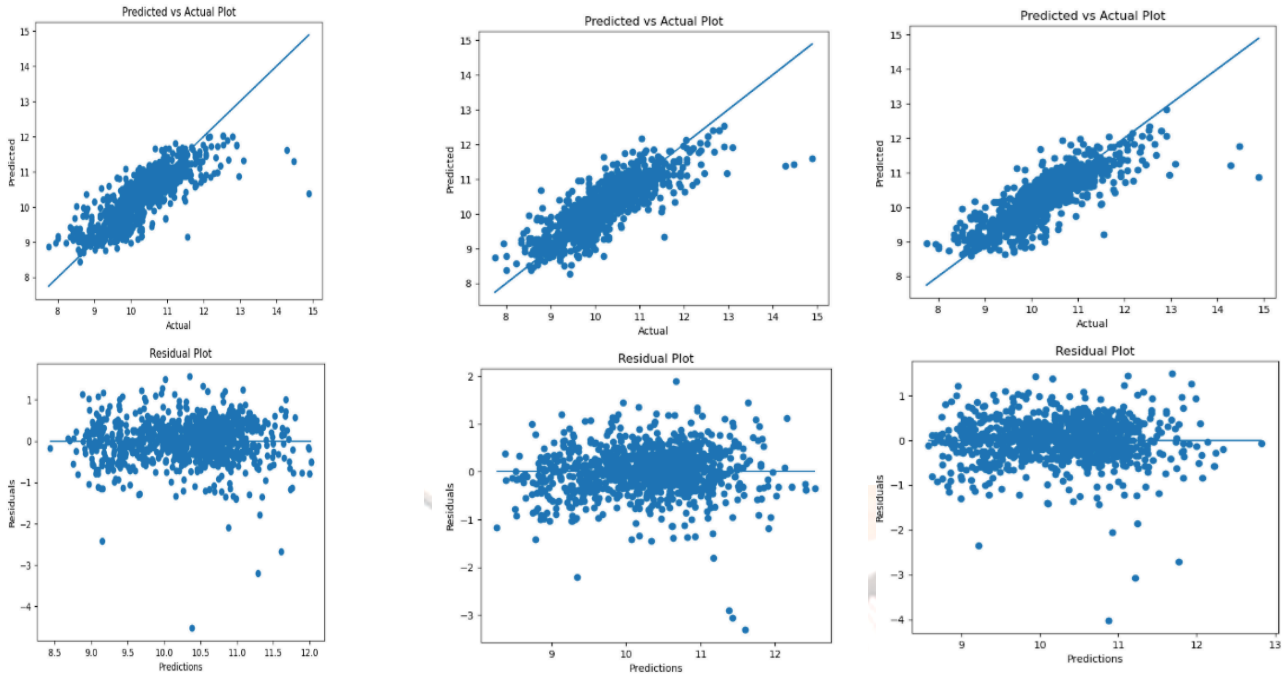


Below are the top 5 most important features in the best-performing model (CatBoost):

Feature	**Importance (%)**
Mileage	39.4%
model_year	13.1%
car_age	9.3%
brand_Other	6.3%
brand_Porsche	5.4%

Across all models, mileage, model_year, and car_age consistently ranked as the most important predictors of used car price. Brand-specific features (e.g., Porsche, Hyundai, Toyota) and accident history also contributed meaningfully.

Below is the predicted vs actual plot for gradient boost, XGBoost and Cat Boost respectively. and also the residual plot for the same.



Hyperparameter Tuning & Deployment

Although XGBoost and CatBoost were the top models, CatBoost edged slightly ahead in performance metrics. We serialized the XGBoost model (due to deployment constraints on PythonAnywhere) using pickle, as it offered a good balance between performance and compatibility.

pickle.dump(xgb, open("model_pipeline.pkl", "wb"))

```
# Save model
model = pickle.dump(xgb, open("model_pipeline.pkl", 'wb'))
# wb = write binary for the h5 file
```

Research Questions and Visualizations

The project addressed several key research questions to understand the dynamics of used car pricing:

- **What factors most significantly affect the price of a used car?**
  - **Analysis:** Correlation analysis and scatter plots were used to identify key predictors.
  - **Findings:**
    - Mileage: A strong negative correlation with price, indicating that higher mileage generally leads to lower prices.
    - Model Year: A positive correlation, suggesting that newer cars command higher prices.
    - Brand and Model: Significant impact due to brand reputation and model features.
    - Accident History: Cars with accident history tend to have a lower price.
  - **Visualization:** Correlation heatmap, Price vs. Mileage, Price vs. Model Year scatter plot.

- **Can we predict the price of a used car based on its attributes?**
    - **Analysis:** Regression models (Linear Regression, Random Forest, Gradient Boosting) were employed.
    - **Findings:** Although XGBoost and CatBoost were the top models, CatBoost edged slightly ahead in performance metrics. **Visualization:** Actual vs. Predicted Price scatter plot, Regression line chart.

- **How does the model year of a car affect its price?**
    - **Analysis:** Average prices were calculated and compared across different model years.
    - **Findings:** Newer model years consistently resulted in higher average prices, reflecting depreciation patterns.
    - **Visualization:** Line chart or bar chart showing average prices by model year.

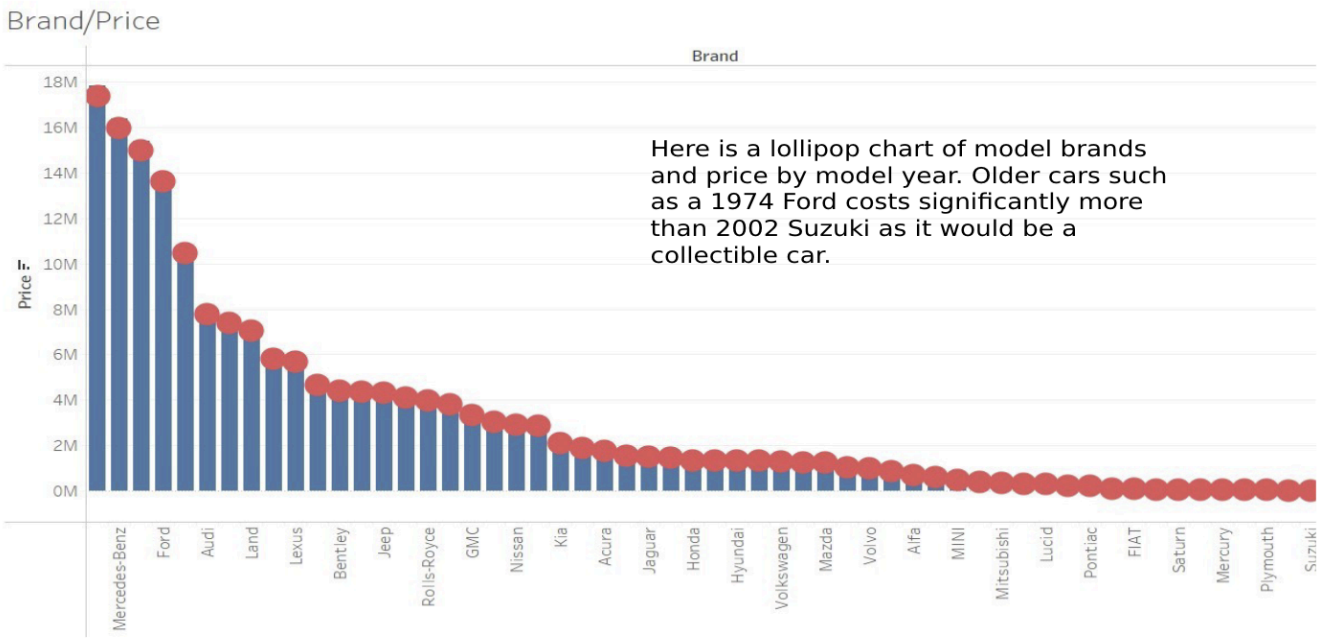
- **Do accident history and clean title status impact the price of a used car?**
    - **Analysis:** Plots and bar charts were used to compare prices based on accident history and clean title status.
    - **Findings:** Cars with accident history had significantly lower prices, and clean title status positively impacted resale value.
    - **Visualization:** Bar chart comparing prices based on clean title status.

- **Which fuel type (gasoline, diesel, electric, hybrid) has the highest average price?**
    - **Analysis:** Average prices were calculated and compared for each fuel type.
    - **Findings:** Electric and hybrid vehicles tended to have higher average prices due to their advanced technology and fuel efficiency.
    - **Visualization:** Bar chart comparing average prices by fuel type.

- **How does engine type influence the price of used cars?**
    - **Analysis:** Box plots and bar charts were used to compare prices based on engine type.
    - **Findings:** Engine types like V8 and electric engines generally led to higher prices due to their performance and technology.
    - **Visualization:** bar chart comparing prices based on engine type.

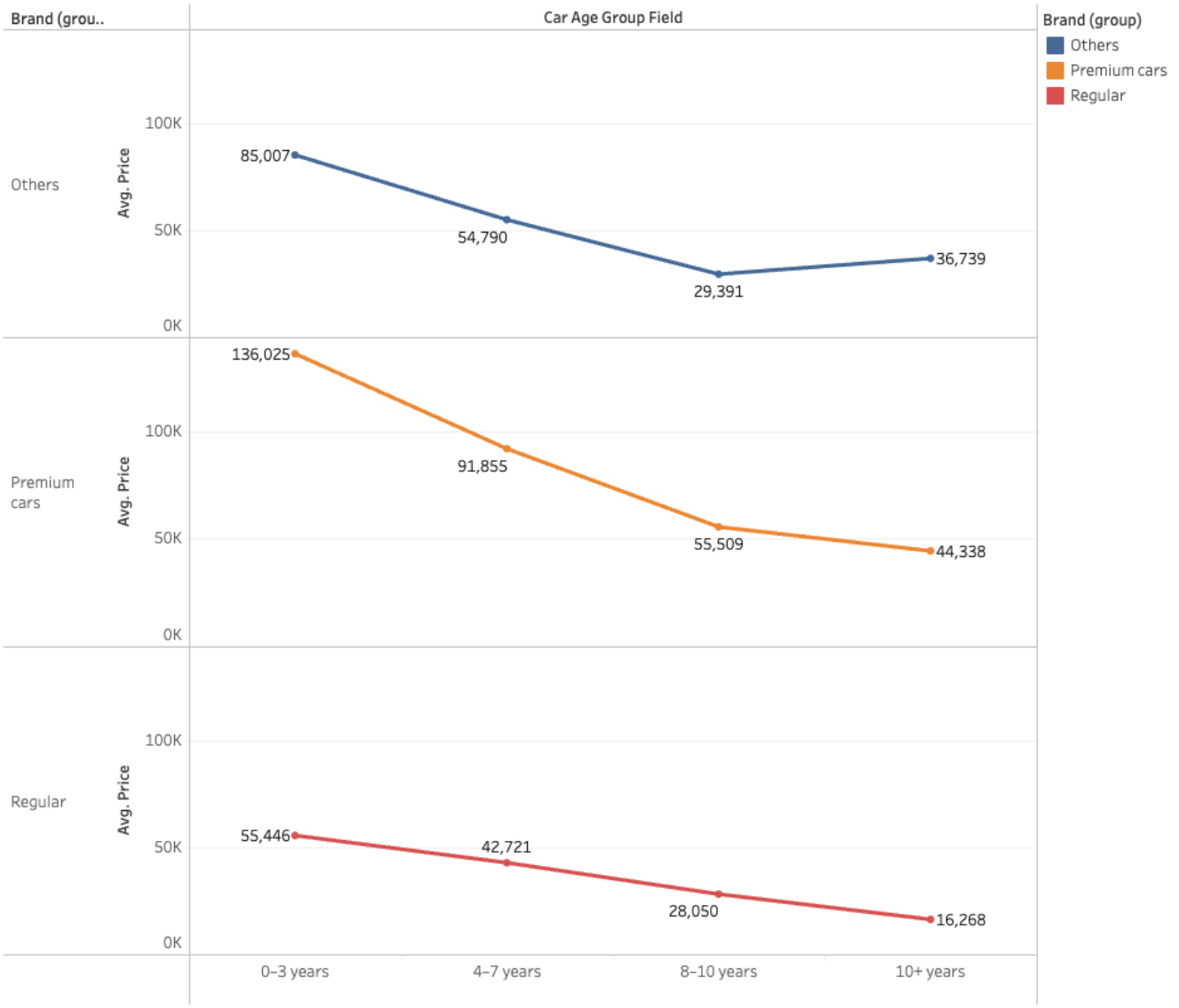
Tableau Dashboards

- **Dashboard 1: Overview of Car Prices:**



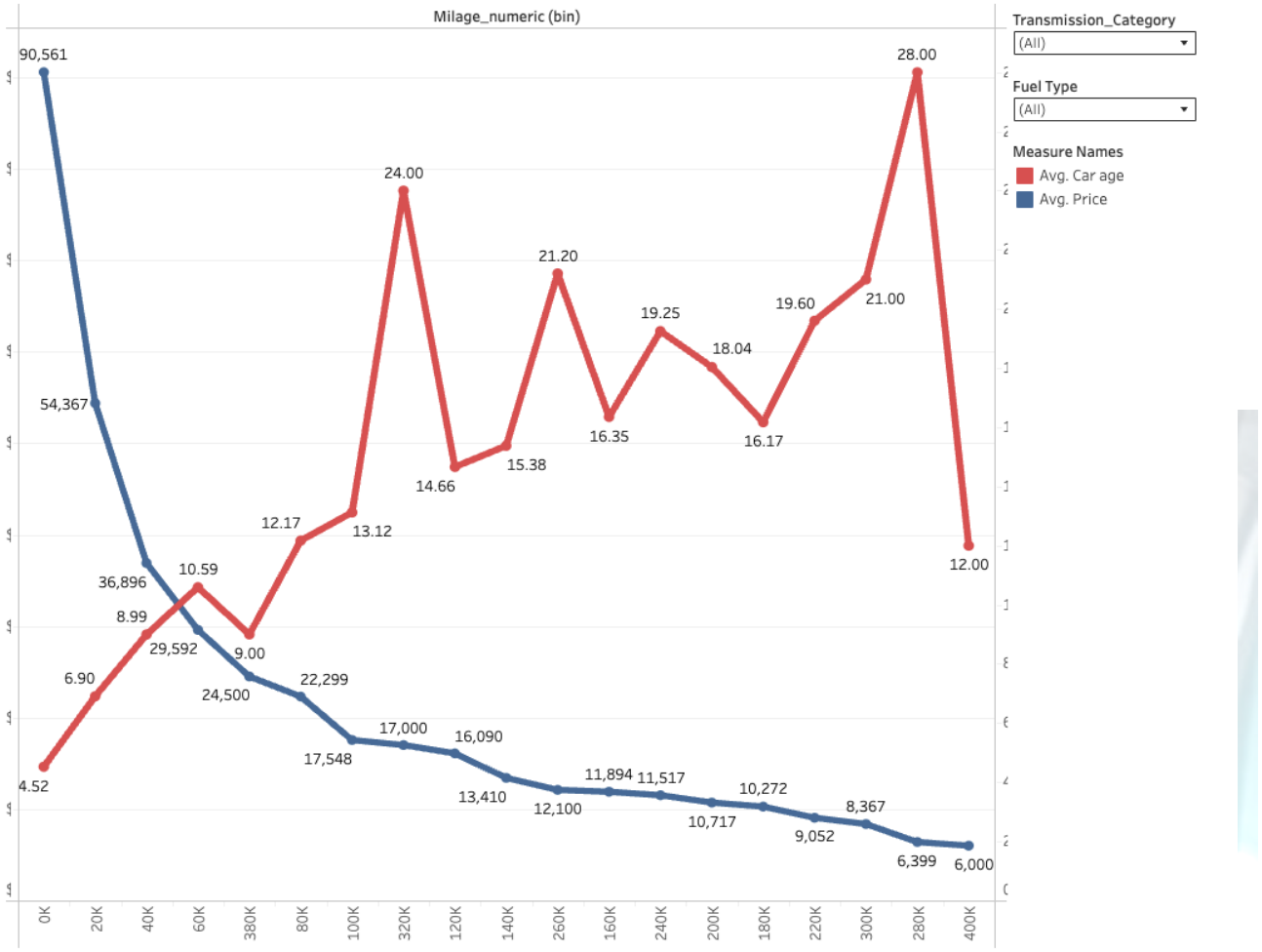
- **Dashboard 2: Predictive Model Dashboard:**
  - Describe the visualizations and model performance metrics.

Premium and regular used car brands price by age



Price of used cars based on exterior colors

Used cars price based on car age and mileage filtered based on engine transmission



Model Persistence

- How the model was saved using pickle:
  - The proposal mentions that the machine learning models (such as Linear Regression, Random Forest, or Gradient Boosting) will be built to predict used car prices. Once the best performing model is identified, it needs to be saved for later use without retraining. This is where `pickle` comes in.
- Why **pickle** is Used:
  - **Simplicity:** It's straightforward to use for saving and loading Python objects.
  - **Preserves Object Structure:** It preserves the state and structure of the Python object, including the trained model's parameters.

Deliverables

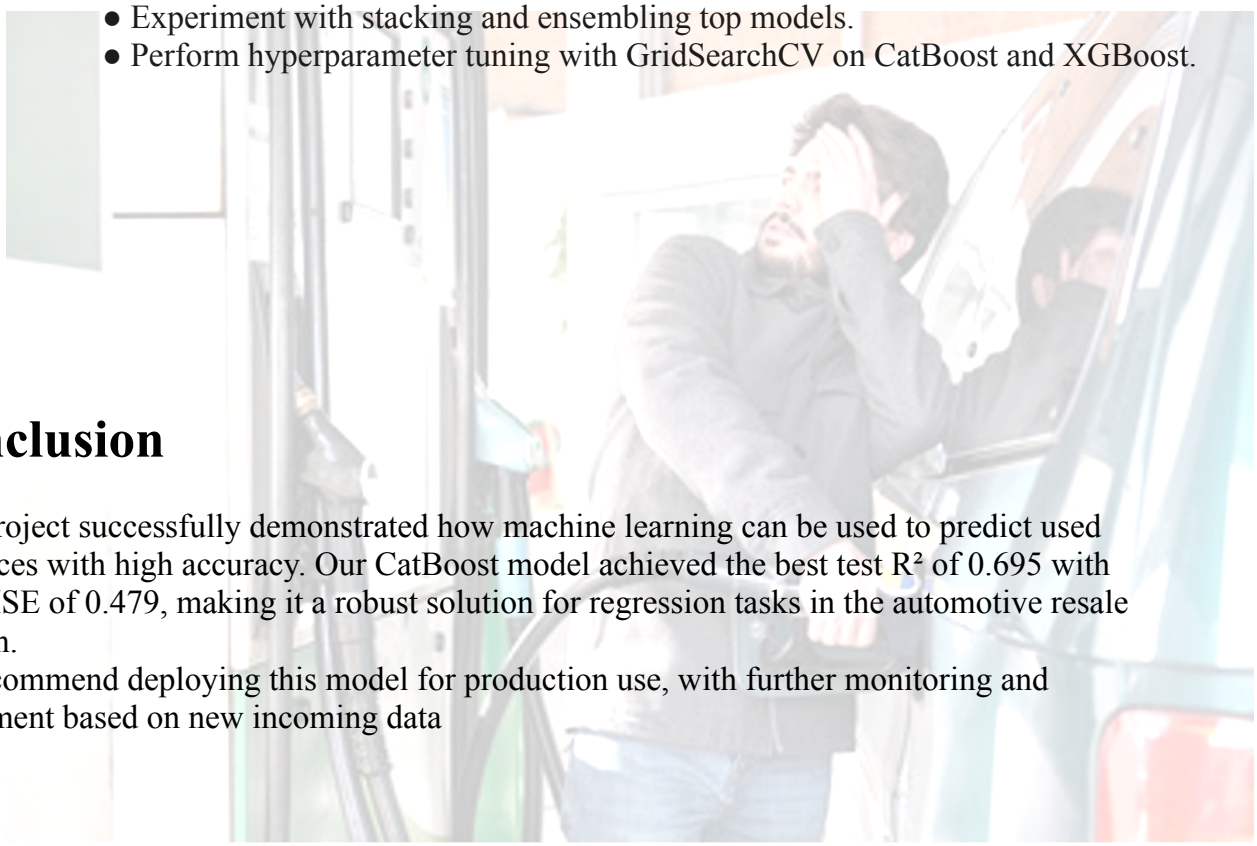
- **Tableau Dashboards:**
  - Dashboard 1: Overview of Car Prices (Price Distribution, Price vs. Mileage, etc.).
  - Dashboard 2: Predictive Model Dashboard (Actual vs. Predicted Prices, Feature Importance, etc.).
- **Python Notebooks:**
  - Jupyter notebook notebooks for data analysis and model building.
- **Project Report (PDF):**
  - This document, detailing the project's methodology, findings, and visualizations.
- **GitHub Repository:**
  - Complete project code, documentation, and README.





Limitations & Future Work

- Limitations:
    - No detailed car condition or service history.
    - Some categorical levels may have been over- or under-represented.
    - Feature imbalance in certain brands and fuel types.
- Future improvements:
    - Include car condition scores or inspection ratings.
    - Experiment with stacking and ensembling top models.
    - Perform hyperparameter tuning with GridSearchCV on CatBoost and XGBoost.



Conclusion

This project successfully demonstrated how machine learning can be used to predict used car prices with high accuracy. Our CatBoost model achieved the best test R^2 of 0.695 with an RMSE of 0.479, making it a robust solution for regression tasks in the automotive resale domain.
We recommend deploying this model for production use, with further monitoring and refinement based on new incoming data