Car Dekho - Used Car Price Prediction

Machine Learning-Based Price Prediction for Used Cars

PRESENTED BY

NITHYA SRI.N

Project Overview

This project aims to predict the prices of used cars listed on Car Dekho using machine learning models. The goal is to create an interactive and user-friendly Streamlit web application that helps customers and sales representatives estimate car prices based on various features such as car make, model, year, fuel type, and more.

Domain: Automotive Industry

Skills & Tools

- **Data Cleaning and Preprocessing**
- Exploratory Data Analysis (EDA)
- Machine Learning Model Development
- Price Prediction Techniques
- Model Evaluation and Optimization
- Streamlit Application Development

Libraries Used

- Pandas: For data manipulation and analysis.
- Matplotlib: For data visualization.
- Streamlit: For built web application
- Seaborn: For statistical data visualization.
- Scikit-learn: For machine learning algorithms, model evaluation, and preprocessing.
- Pickle: For saving and loading Python objects

Problem Statement

The goal of the project is to improve the customer experience by building a machine learning model that accurately predicts the prices of used cars. The model will be integrated into a web-based application using Streamlit, allowing users to input car details and receive price predictions in real time.

Objective

- Develop a machine learning model to predict used car prices based on various car features.
- Integrate the model into a Streamlit application to provide real-time price predictions

Project Scope

<u>Data Processing :</u>

- **Import and Concatenate:** Import datasets from different cities, convert unstructured data into structured format, and concatenate them into a single dataset with an additional 'City' column.
- **Handling Missing Values:** Use imputation techniques such as mean, median, or mode for numerical columns, and mode or a new category for categorical columns.
- **Standardizing Data Formats:** Ensure consistency by removing units (e.g., 'kms' from numerical columns) and converting all relevant fields to the correct data types.

<u>Encoding Categorical Variables :</u>

- Apply label encoding for ordinal categories (e.g., fuel types, transmission types).

Import and Concatenate:
convert unstructured data into structured format for all city data

new_car_detail (Column 1)

```
# Extract data from each structured column
car_details_df = pd.json_normalize(chennai_df['new_car_detail'].apply(lambda x: ast.literal_eval(x) if isinstance(x, str) else x))
```

<u>new_car_overview (Column 2) Convert to structure format</u>

```python
# Function to extract 'top' items from the car overview, including icons
def extract_overview_data(overview):
    if isinstance(overview, str):
        overview = ast.literal_eval(overview)

    # Extract key, value, and icon
    extracted_data = {}
    for item in overview.get('top', []):
        key = item['key']
        extracted_data[f"{key}_value"] = item['value']
        extracted_data[f"{key}_icon"] = item.get('icon', '')  # Use empty string if icon is missing

    return extracted_data

# Apply the extraction function to the 'new_car_overview' column
car_overview_df = chennai_df['new_car_overview'].apply(extract_overview_data).apply(pd.Series)

# Display the extracted DataFrame
print(car_overview_df.columns)
```

new_car_feature (Column 3) Convert to structure format

```python
# Function to extract features from the car feature data
def extract_features(features):
    if isinstance(features, str):
        features = ast.literal_eval(features)

    top_features = [item['value'] for item in features.get('top', [])]
    common_icon = features.get('commonIcon', '')

    return top_features, common_icon

# Apply the function and create a DataFrame
car_features_df = chennai_df['new_car_feature'].apply(extract_features).apply(pd.Series)
car_features_df.columns = ['Top_Features', 'Common_Icon']

# Display the extracted DataFrame
print(car_features_df.columns)
```

new_car_specs (Column 4) Convert to structure format

```
# Function to extract specifications from a structured dictionary
def extract_specifications(specifications):
    if isinstance(specifications, str):
        specifications = ast.literal_eval(specifications)

    top_specs = {item['key']: item['value'] for item in specifications.get('top', [])}

    detailed_specs = {}
    for category in specifications.get('data', []):
        for item in category.get('list', []):
            detailed_specs[item['key']] = item['value']

    return {**top_specs, **detailed_specs, 'Common_Icon': specifications.get('commonIcon', '')}

# **top_specs unpacks all key-value pairs from the top_specs dictionary.
# **detailed_specs unpacks all key-value pairs from the detailed_specs dictionary.

# Apply the extraction function to the 'new_car_specifications' column
car_specs_df = chennai_df['new_car_specs'].apply(extract_specifications).apply(pd.Series)

# Display the extracted DataFrame
print(car_specs_df.columns)
```

```python
# Combine all DataFrames into one
chennai_combined_df = pd.concat([
    car_details_df.reset_index(drop=True),
    car_overview_df.reset_index(drop=True),
    car_features_df.reset_index(drop=True),
    car_specs_df.reset_index(drop=True),
    car_links_df], axis=1)

# Add a new column named 'city' with the value 'chennai'
chennai_combined_df.insert(0, 'city', 'chennai')

# Save the final combined DataFrame to a CSV file
output_file = 'chennai_cars_final.csv'
chennai_combined_df.to_csv(output_file, index=False)

print(f"Structured data saved to {output_file}")
```

```python
# Reset the index for each DataFrame to ensure unique index
chennai_combined_df = chennai_combined_df.reset_index(drop=True)
Bangalore_combined_df = Bangalore_combined_df.reset_index(drop=True)
Delhi_combined_df = Delhi_combined_df.reset_index(drop=True)
Hyderabad_combined_df = Hyderabad_combined_df.reset_index(drop=True)
Jaipur_combined_df = Jaipur_combined_df.reset_index(drop=True)
Kolkata_combined_df = Kolkata_combined_df.reset_index(drop=True)

# Ensure that all DataFrames have the same columns
common_columns = list(set(chennai_combined_df.columns).intersection(
    Bangalore_combined_df.columns,
    Delhi_combined_df.columns,
    Hyderabad_combined_df.columns,
    Jaipur_combined_df.columns,
    Kolkata_combined_df.columns
))

# Filter each DataFrame to include only the common columns
chennai_combined_df = chennai_combined_df[common_columns]
Bangalore_combined_df = Bangalore_combined_df[common_columns]
Delhi_combined_df = Delhi_combined_df[common_columns]
Hyderabad_combined_df = Hyderabad_combined_df[common_columns]
Jaipur_combined_df = Jaipur_combined_df[common_columns]
Kolkata_combined_df = Kolkata_combined_df[common_columns]

# Concatenate all city DataFrames
all_city_cars_df = pd.concat([chennai_combined_df, Bangalore_combined_df, Delhi_combined_df, Hyderabad_combined_df,
                              Jaipur_combined_df, Kolkata_combined_df], ignore_index=True)

# Save the combined DataFrame to a CSV file
all_city_cars_df.to_csv('all_city_cars.csv', index=False)

print("Data for all cities saved to 'all_city_cars.csv'")
```

Exploratory Data Analysis (EDA)

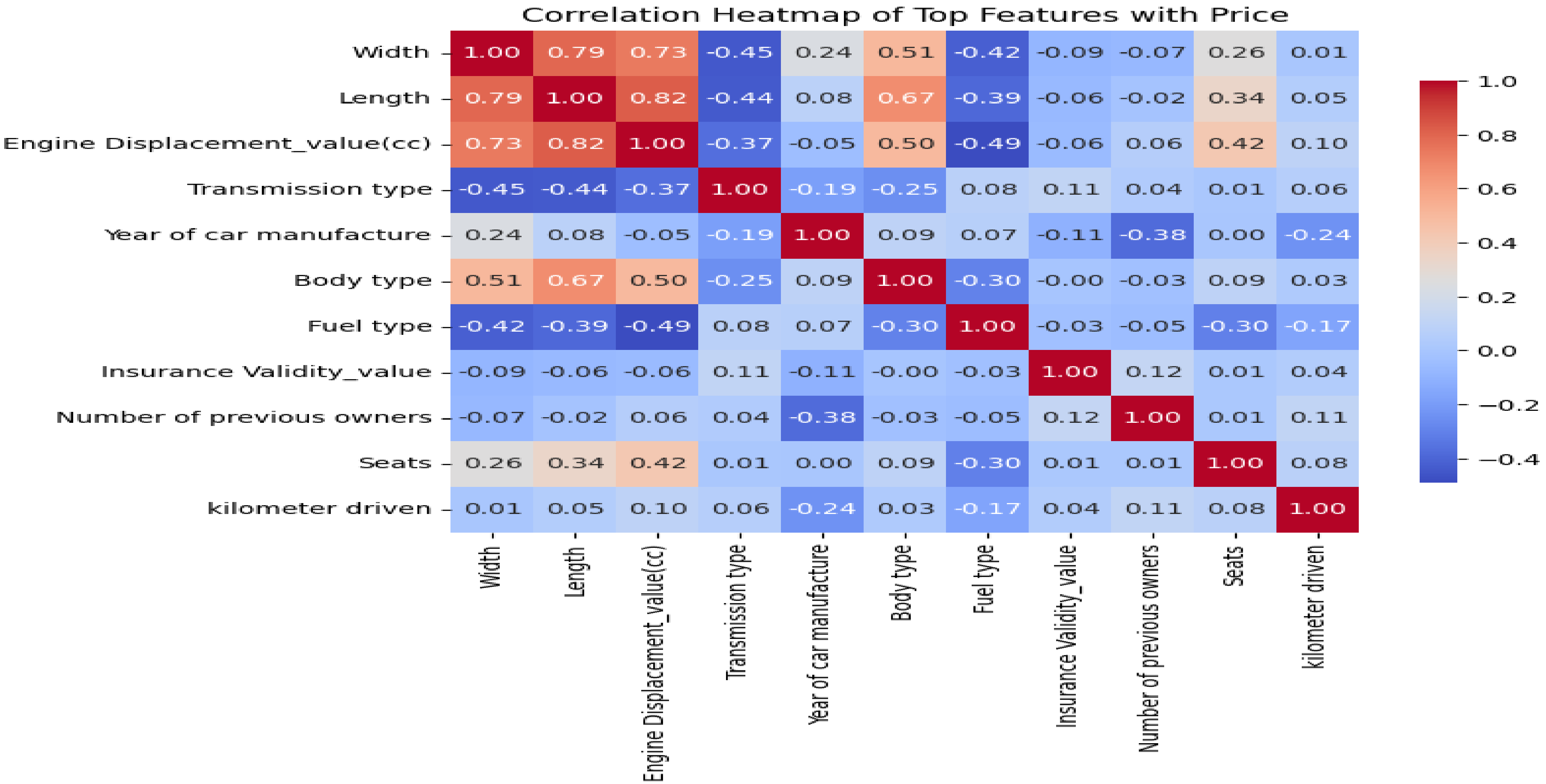
Perform descriptive statistics (mean, median, mode, etc.) to understand the distribution of the data.

Use visualizations such as scatter plots, histograms, box plots, and correlation heatmaps to identify patterns and relationships.

Encoding Categorical Variables:

```
# Create a backup DataFrame for original categorical values
original_values_df = all_city_cars_df.select_dtypes(include=['object']).copy()

# Initialize a dictionary to store label encoders for each column
label_encoders = {}

# Loop through each categorical column
for col in all_city_cars_df.select_dtypes(include=['object']).columns:
    label_encoder = LabelEncoder()

    # Fill missing values with the mode (most frequent value)
    mode_value = all_city_cars_df[col].mode()[0]
    all_city_cars_df[col] = all_city_cars_df[col].fillna(mode_value)

    # Fit and transform the column using label encoding
    all_city_cars_df[col] = label_encoder.fit_transform(all_city_cars_df[col])

    # Store the label encoder for future use if needed
    label_encoders[col] = label_encoder

# Display the encoded DataFrame and original values
print(all_city_cars_df.head())
print(original_values_df.head())  # Original values saved here
```

```
# Save the label encoders to a pickle file
with open('label_encoders.pkl', 'wb') as f:
    pickle.dump(label_encoders, f)
```


Correlation Heatmap of Top Features with Price
Width
Length
Engine Displacement_value(cc)
Transmission type
Year of car manufacture
Body type
Fuel type
Insurance Validity_value
Number of previous owners
Seats
kilometer driven
1.00 0.79 0.73 -0.45 0.24 0.51 -0.42 -0.09 -0.07 0.26 0.01
0.79 1.00 0.82 -0.44 0.08 0.67 -0.39 -0.06 -0.02 0.34 0.05
0.73 0.82 1.00 -0.37 -0.05 0.50 -0.49 -0.06 0.06 0.42 0.10
-0.45 -0.44 -0.37 1.00 -0.19 -0.25 0.08 0.11 0.04 0.01 0.06
0.24 0.08 -0.05 -0.19 1.00 0.09 0.07 -0.11 -0.38 0.00 -0.24
0.51 0.67 0.50 -0.25 0.09 1.00 -0.30 -0.00 -0.03 0.09 0.03
-0.42 -0.39 -0.49 0.08 0.07 -0.30 1.00 -0.03 -0.05 -0.30 -0.17
-0.09 -0.06 -0.06 0.11 -0.11 -0.00 -0.03 1.00 0.12 0.01 0.04
-0.07 -0.02 0.06 0.04 -0.38 -0.03 -0.05 0.12 1.00 0.01 0.11
0.26 0.34 0.42 0.01 0.00 0.09 -0.30 0.01 0.01 1.00 0.08
0.01 0.05 0.10 0.06 -0.24 0.03 -0.17 0.04 0.11 0.08 1.00
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

Model Development

- **Train-Test Split:**

  Split the dataset into training and testing sets (70-30 split).

- **Model Selection:**

  Evaluate various models, including Linear Regression, Decision Trees, Random Forests, and Gradient Boosting.

- **Model Training:**

  Train the selected models and use cross-validation to ensure robust performance.

```python
X = final_df.drop(columns=['price'])  # Features (excluding the target variable)
y = final_df['price']  # Target variable

# Split the dataset into training and testing sets
# Using an 70-30 split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


print(f'Training feature set shape: {X_train.shape}')
print(f'Testing feature set shape: {X_test.shape}')
print(f'Training target set shape: {y_train.shape}')
print(f'Testing target set shape: {y_test.shape}')
```

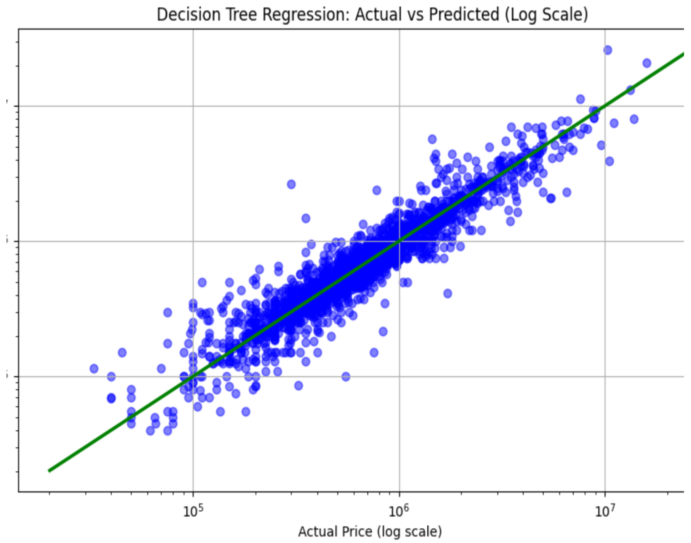
```python
from sklearn.ensemble import GradientBoostingRegressor

# Gradient Boosting model
gb_model = GradientBoostingRegressor(n_estimators=300, random_state=0)
gb_model.fit(X_train, y_train)

# Predictions
y_pred_gb = gb_model.predict(X_test)

# Evaluation
mse_gb = mean_squared_error(y_test, y_pred_gb)
mae_gb = mean_absolute_error(y_test, y_pred_gb)
r2_gb = r2_score(y_test, y_pred_gb)

# Printing evaluation metrics
print(f'Gradient Boosting - Mean Squared Error: {mse_gb:.2f}')
print(f'Gradient Boosting - Mean Absolute Error: {mae_gb:.2f}')
print(f'Gradient Boosting - R-squared: {r2_gb:.2f}')

# Plotting Actual vs Predicted values on log scale
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_gb, color='blue', alpha=0.5)
plt.plot([y.min(), y.max()], [y.min(), y.max()], color='green', linewidth=2.5)
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Actual Price (log scale)')
plt.ylabel('Predicted Price (log scale)')
plt.title('Gradient Boosting Regression: Actual vs Predicted (Log Scale)')
plt.grid(True)
plt.show()
```

Model Evaluation

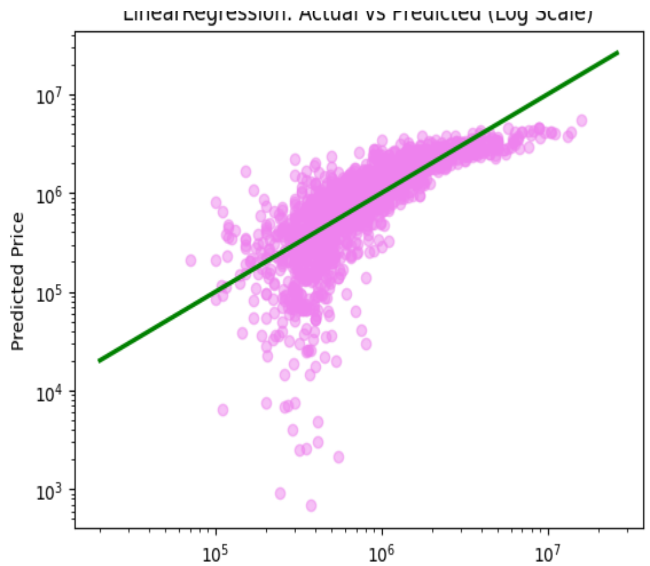
Use metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared (R^2) to evaluate model performance.

Compare the performance of different models and select the best-performing one.

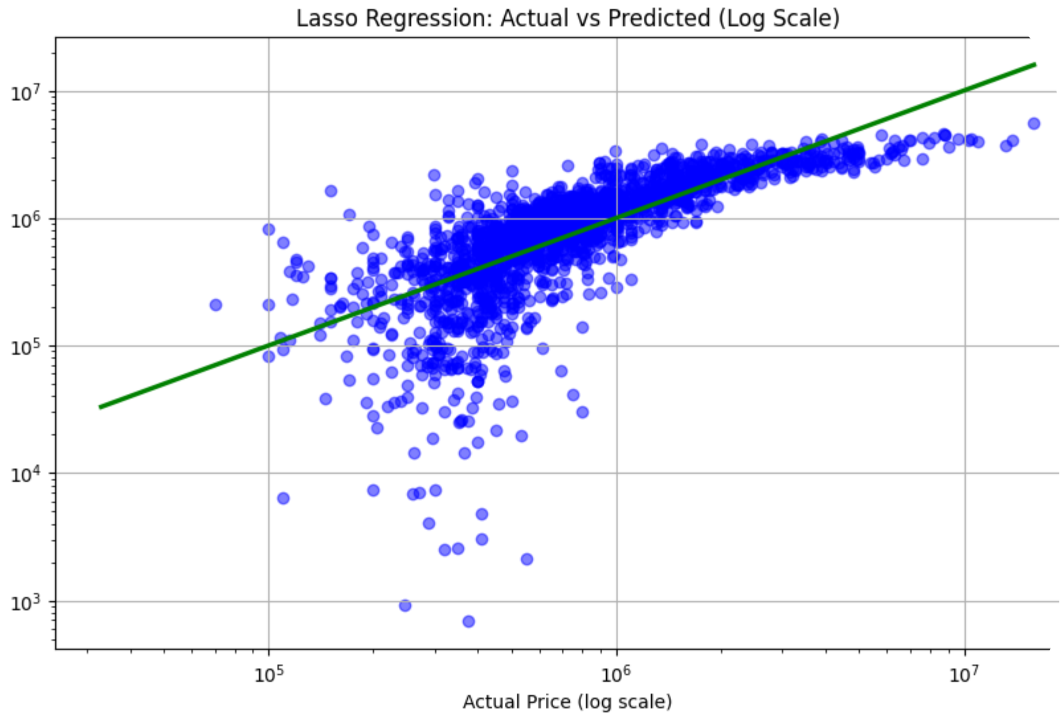
Model Performance Summary

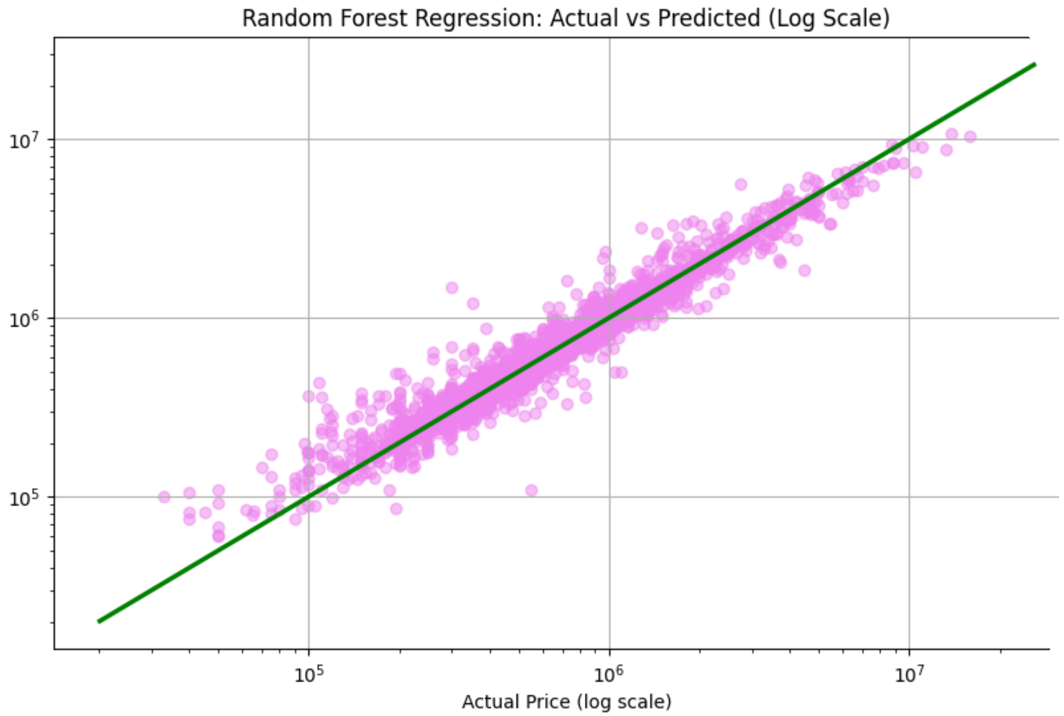


Decision Tree: Moderate performance with an R-squared value of 0.78.



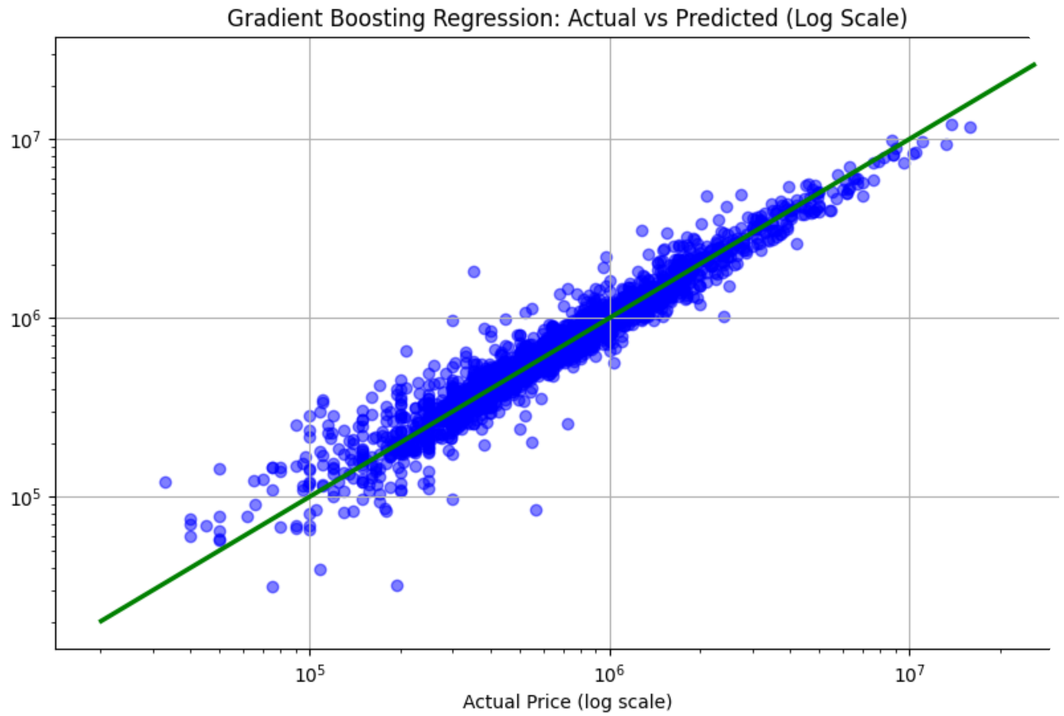
Linear Regression: Weakest model with an R-squared value of 0.59.

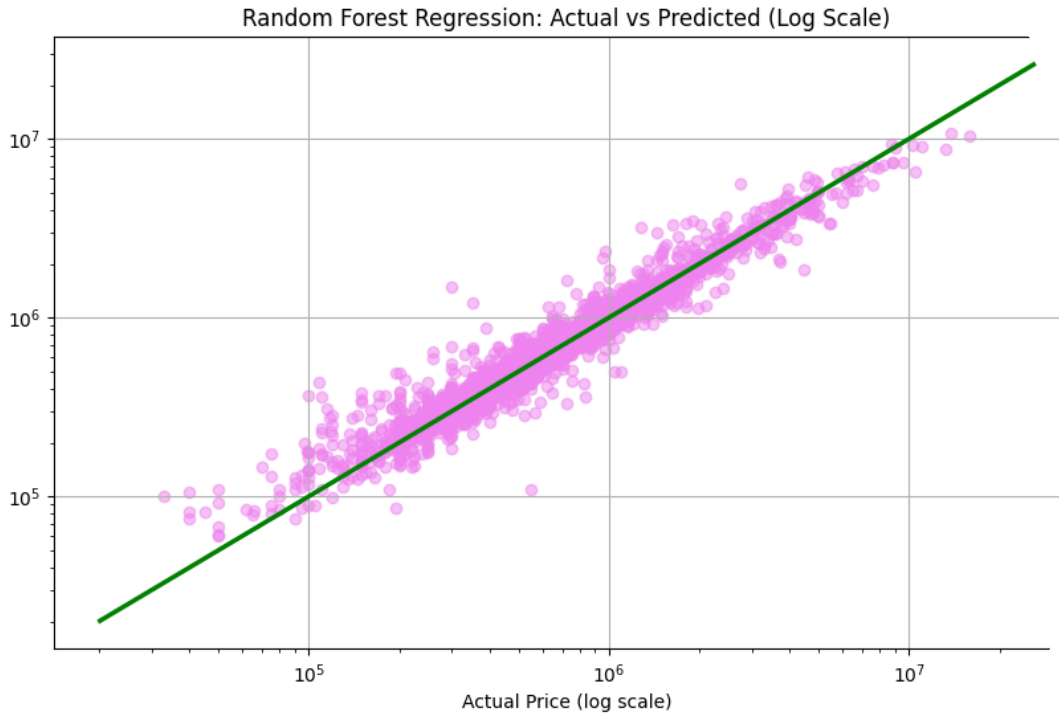




Lasso Regression: Weakest model with an R-squared value of 0.59.

Ridge Regression : Weakest model with an R-squared value of 0.59.





Gradient Boosting: Best performer with an R-squared value of 0.94, demonstrating high accuracy.

Random Forest: Strong performance with an R-squared value of 0.93.

Model	R-Squared (R^2)	Performance
Gradient Boosting	0.94	Best
Random Forest	0.93	Strong
Decision Tree	0.78	Moderate
Ridge Regression	0.59	Low
Linear Regression	0.59	Low
Lasso Regression	0.59	Low

Deployment

Deploy the model using Streamlit to create an intuitive web application.

Enable users to input car features and obtain real-time price predictions.

Streamlit Application

How It Works:

User inputs car details (e.g., make, model, year, mileage) in the sidebar.

The machine learning model predicts the price based on input features.

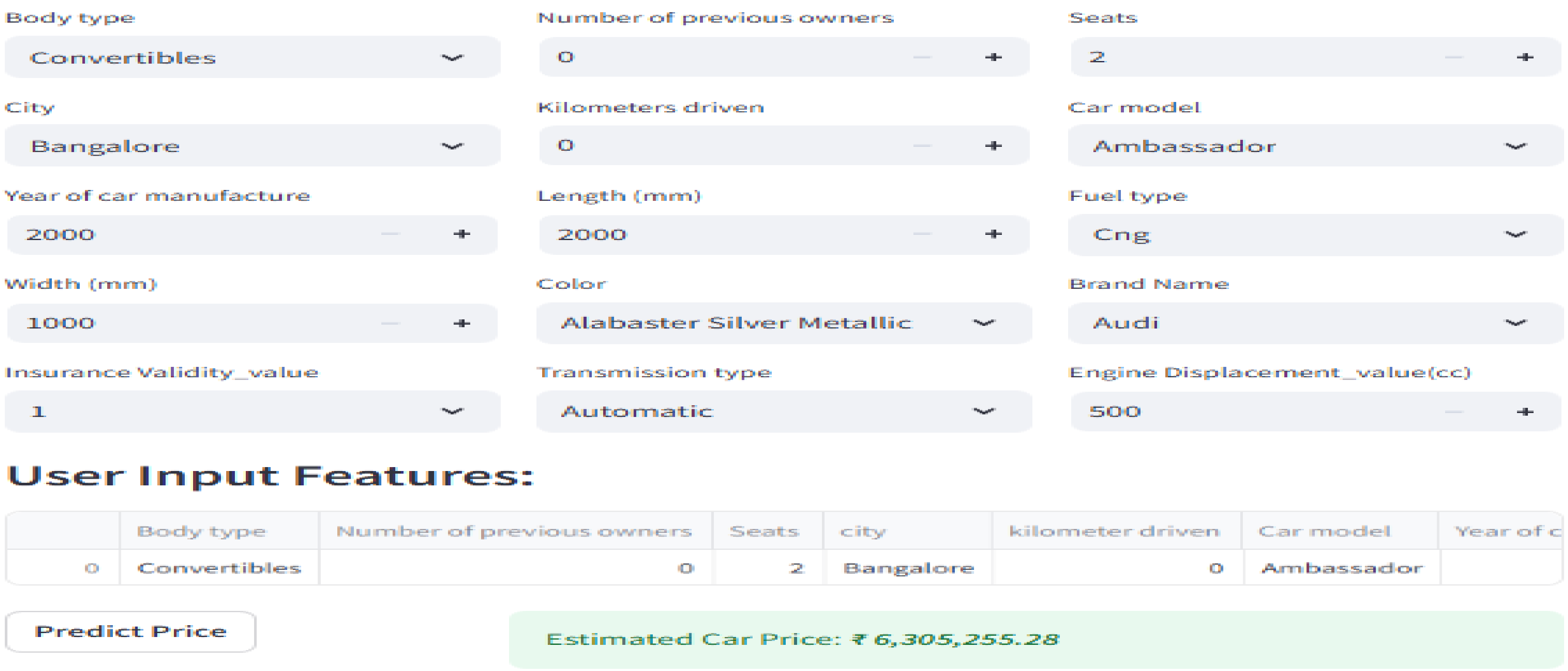
The app displays the estimated car price.

User Interface:

Sidebar with drop-down menus and input fields.

Real-time price prediction displayed on the main screen.

STREAMLIT WEB APP



THANK YOU