Documentation Report: Used Car Price Prediction Application

Introduction:

This application is designed to predict the price of a used car based on specific input features provided by the user. The prediction model is built using a linear regression algorithm, and the application interface is developed with Streamlit. The primary goal of this application is to offer a simple, user-friendly experience for estimating the price of a car using essential attributes.

Key Features:

1.User-Centered Input Interface:

- **All inputs are positioned in the center of the page for a focused and streamlined user experience.**
- **The inputs include car make, fuel type, registration year, engine power, kilometers driven, and city.**

2.Prediction Model:

- **The model uses Linear Regression to predict the price of a used car.**
- **Essential features for prediction include Kilometers_Driven, Registration_Year, Mileage, and Engine_Power.**

Convert the complex unstructured json dataset into structured data:

- **Package used:pandas and json**

Model code:

```
import pandas as pd
import ast

# Load the dataset with the correct engine
file_path = r'C:\Users\moham\vs code\guvi\project-3\hyderabad_cars.xlsx'
```

```
data = pd.read_excel(file_path, sheet_name='hyderabad_cars.csv',
engine='openpyxl')

# Convert string representations of dictionaries into actual dictionaries
data['new_car_detail'] = data['new_car_detail'].apply(ast.literal_eval)
data['new_car_overview'] = data['new_car_overview'].apply(ast.literal_eval)
data['new_car_feature'] = data['new_car_feature'].apply(ast.literal_eval)
data['new_car_specs'] = data['new_car_specs'].apply(ast.literal_eval)

# Function to extract data from nested dictionaries
def extract_details(row):
        details = row.get('new_car_detail', {})
        overview = row.get('new_car_overview', {})
        features = row.get('new_car_feature', {})
        specs = row.get('new_car_specs', {})

        # Extracting fields from details
        fuel_type = details.get('ft')
        body_type = details.get('bt')
        kilometers = details.get('km')
        price = details.get('price')
        car_model = details.get('model')
        car_age = details.get('modelYear')


        # Extracting fields from overview (like registration year, model, etc.)
        overview_dict = {item['key']: item['value'] for item in overview.get('top',
[])}

        # Extracting features (listing them as comma-separated values)
        feature_list = [feature['value'] for feature in features.get('top', [])]
        feature_str = ', '.join(feature_list)

        # Extracting specifications (like engine power, mileage, etc.)
        specs_dict = {item['key']: item['value'] for item in specs.get('top', [])}

        # Return a flattened structure
        return pd.Series({
        'Fuel_Type': fuel_type,
        'Body_Type': body_type,
        'Kilometers_Driven': kilometers,
        'Registration_Year': overview_dict.get('Registration Year'),
        'Car_Model': car_model,
        'Ownership': overview_dict.get('Ownership'),
```

```
        'Features': feature_str,
        'Engine_Power': specs_dict.get('Max Power'),
        'Mileage': specs_dict.get('Mileage'),
        'Car_Link': row.get('car_links'),
        'Price': price,
        'Car_Age': car_age

        })

# Apply the extraction function to each row
hyderabad_df = data.apply(extract_details, axis=1)

# Display the structured dataset
print(hyderabad_df.head())

# Save the structured data to a new Excel file (optional)
hyderabad_df.to_csv('hyderabad_cars.csv', index=False)
```

Data cleaning and null handling:

- **Combine all state data in a single dataset and save it as csv file**
- **I have named it as cleaned_combined_car.csv**
- **Drop the unwanted columns**
- **Encode the categorical columns to use as numerical features (did label encoding for owner ship and drop the excessing columns)**

Convert features of the car in a data set into star rating to use it for further usage.

- **Decide how many stars (e.g., 1 to 5) each feature contributes to the rating. For example:**
- **5 Stars: Premium features (e.g., Leather Seats, Sunroof)**
- **4 Stars: Essential features (e.g., Power Steering, Power Windows)**
- **3 Stars: Basic features (e.g., Air Conditioning)**
- **2 Stars: Minimal features**
- **1 Star: No features**
- **Map each feature to its corresponding star value.**
- **feature_star_mapping = {**
  'Leather Seats': 5,
  'Sunroof': 5,
  'Power Steering': 4,
  'Power Windows': 4,
  'Air Conditioning': 3,

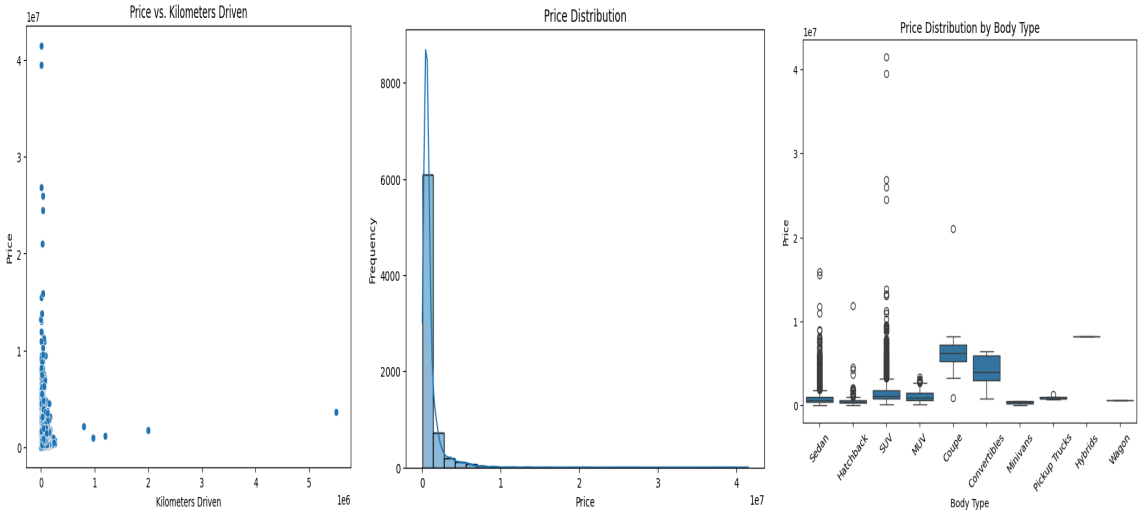
 # Add more features as needed
}

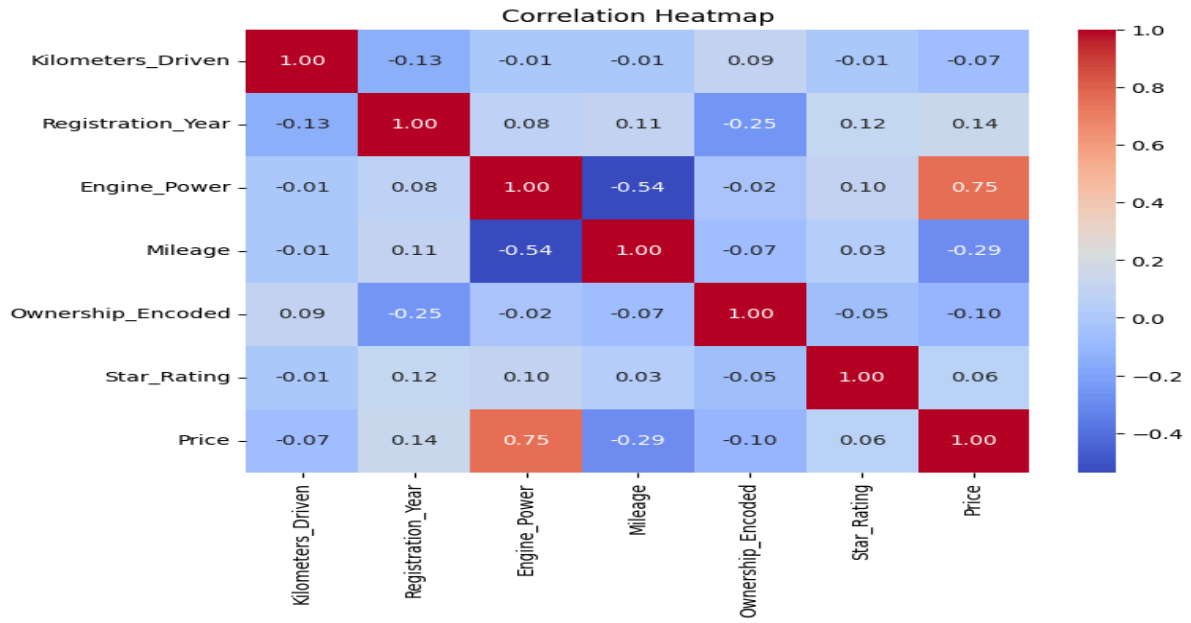
- **Create a function to calculate the star rating for each car based on its features.**
- **def calculate_star_rating(features):**
  stars = 0
  for feature in features.split(', '):
  stars += feature_star_mapping.get(feature, 0) # Default to 0 if feature not found
  return stars

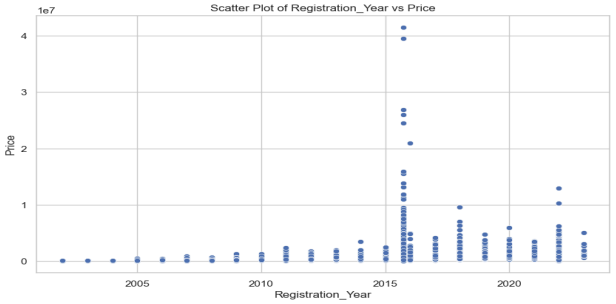
  # Apply the function to the Features column
  df['Star_Rating'] = df['Features'].apply(calculate_star_rating)
- **If you want the ratings to be on a standard scale (1 to 5), you can normalize them.**
- **# Normalize to a 5-star scale**
  max_stars = df['Star_Rating'].max()
  df['Star_Rating'] = (df['Star_Rating'] / max_stars) * 5
- **Check the updated DataFrame to see the star ratings.**
- **print(df[['Features', 'Star_Rating']].head())**
- **Did scalar and standardization for all the columns**
- **Save the handled data into csv file.**

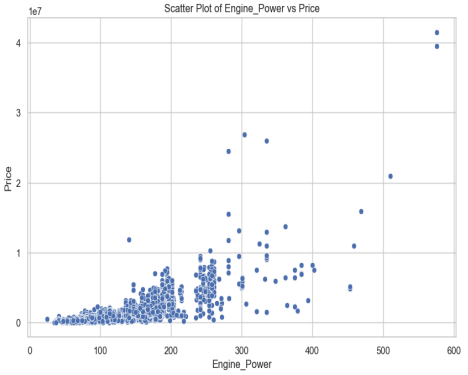
Exploratory Data Analysis (EDA):

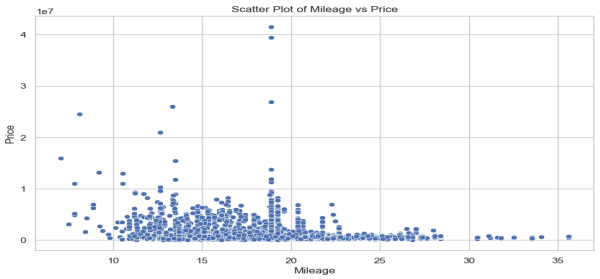
- **Did a descriptive statistic to measure (mean,median,mode)**
- **Did a data visualization for the data set to understand the distribution of the data.**

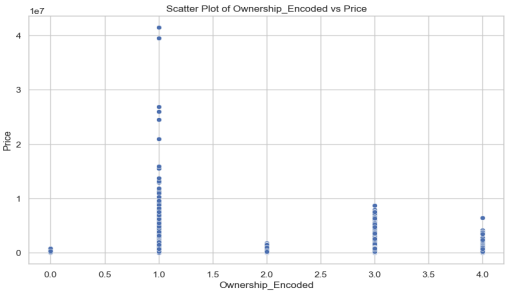


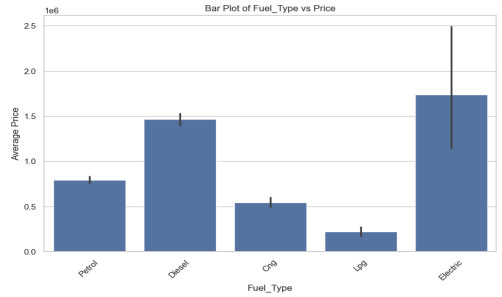

Correlation Heatmap
Kilometers_Driven
Registration_Year
Engine_Power
Mileage
Ownership_Encoded
Star_Rating
Price
1.00 -0.13 -0.01 -0.01 0.09 -0.01 -0.07
-0.13 1.00 0.08 0.11 -0.25 0.12 0.14
-0.01 0.08 1.00 -0.54 -0.02 0.10 0.75
-0.01 0.11 -0.54 1.00 -0.07 0.03 -0.29
0.09 -0.25 -0.02 -0.07 1.00 -0.05 -0.10
-0.01 0.12 0.10 0.03 -0.05 1.00 0.06
-0.07 0.14 0.75 -0.29 -0.10 0.06 1.00
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

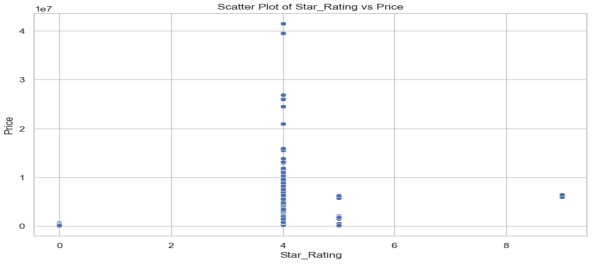

Scatter Plot of Registration_Year vs Price
1e7
Price
Registration_Year
2005
2010
2015
2020

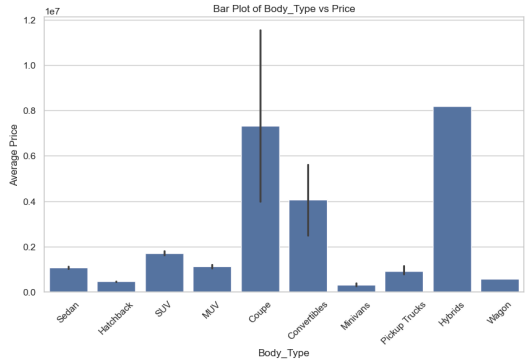

Scatter Plot of Engine_Power vs Price
1e7
Price
Engine_Power
0
100
200
300
400
500
600

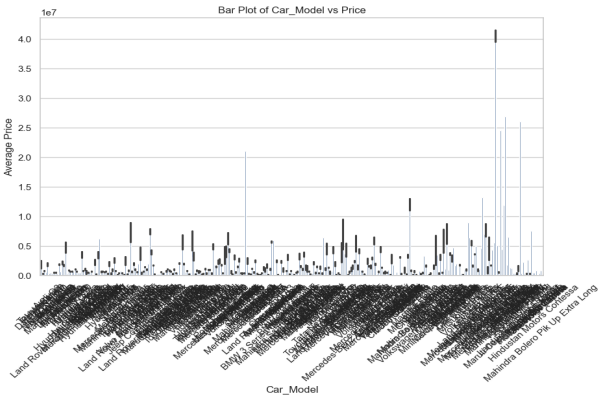

Scatter Plot of Mileage vs Price
1e7
Price
Mileage
10
15
20
25
30
35

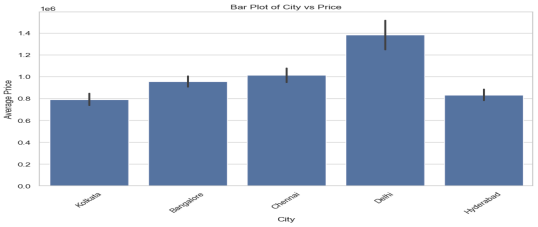

Scatter Plot of Ownership_Encoded vs Price
1e7
Price
Ownership_Encoded
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

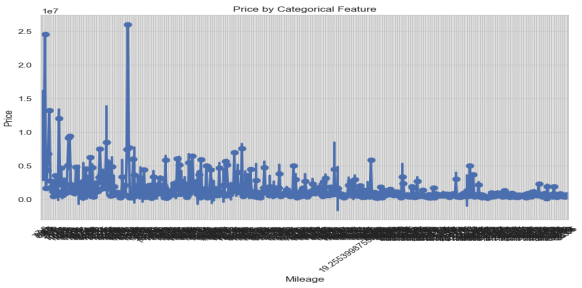

Bar Plot of Fuel_Type vs Price
1e6
Average Price
Fuel_Type
Petrol
Diesel
Cng
Lpg
Electric

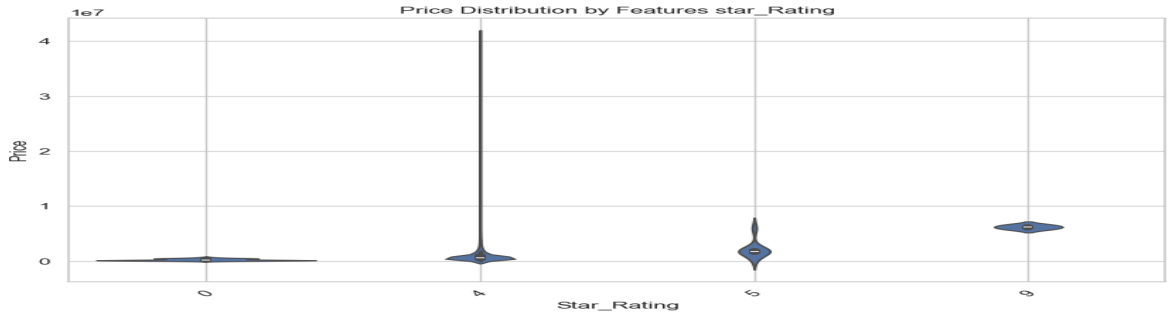

Scatter Plot of Star_Rating vs Price
1e7
Price
Star_Rating
0
2
4
6
8

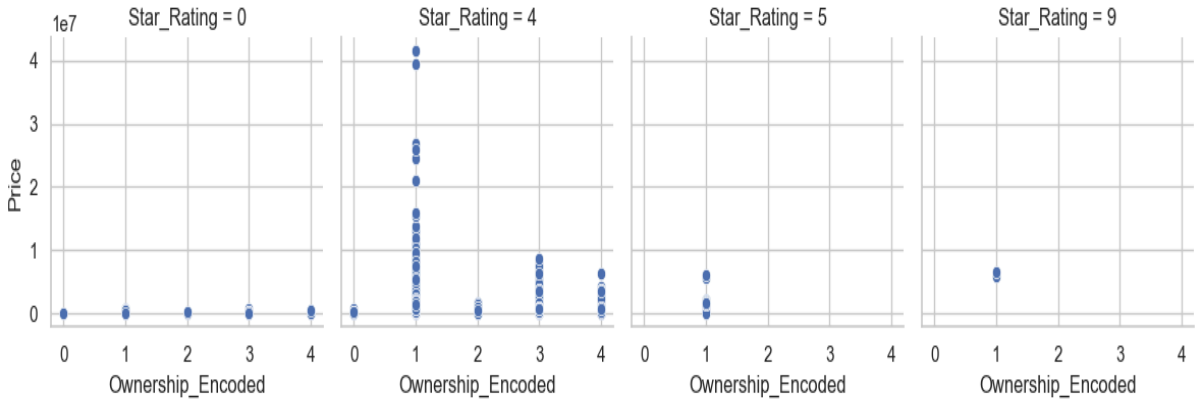












Did feature selection:

	feature	importance
2	Engine_Power	7.000078e-01

0	Kilometers_Driven	5.925044e-02
212	Car_Model_Mercedes-Benz AMG G 63	4.430159e-02
133	Car_Model_Land Rover Range Rover	2.974559e-02
221	Car_Model_Mercedes-Benz G	2.135470e-02
3	Mileage	1.753086e-02
391	Features_Power Steering, Power Windows Front, ...	1.339737e-02
309	Car_Model_Toyota Land Cruiser 300	1.266843e-02
5	Ownership_Encoded	1.076907e-02
4	Car_Age	7.301825e-03
1	Registration_Year	6.411828e-03
225	Car_Model_Mercedes-Benz GLC	4.576602e-03
396	Features_Power Steering, Power Windows Front, ...	4.333725e-03
19	Body_Type_SUV	3.578288e-03
20	Body_Type_Sedan	3.472503e-03
576	City_Kolkata	3.467339e-03
466	Features_Power Steering, Power Windows Front, ...	2.555994e-03
227	Car_Model_Mercedes-Benz GLE	2.273845e-03
394	Features_Power Steering, Power Windows Front, ...	2.131694e-03
573	City_Chennai	2.083202e-03
135	Car_Model_Land Rover Range Rover Sport	1.939022e-03
574	City_Delhi	1.931076e-03
116	Car_Model_Jaguar F-Pace	1.816045e-03
214	Car_Model_Mercedes-Benz AMG GT	1.664162e-03

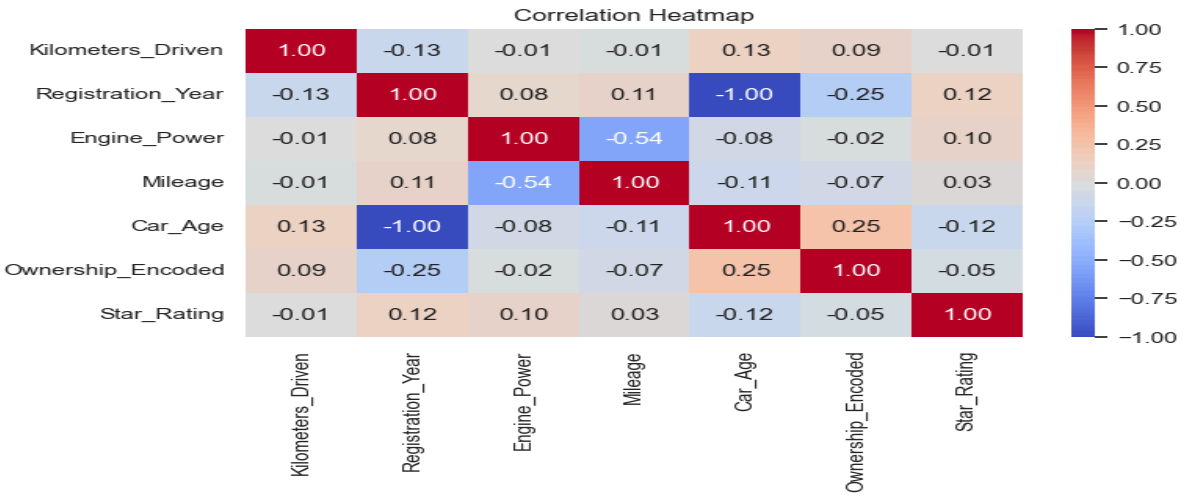
...

235 Car_Model_Mini Cooper Clubman 0.000000e+00

233 Car_Model_Mini 5 DOOR 0.000000e+00

531 Features_Power Steering, Remote Fuel Lid Opene... 0.000000e+00

532 Features_Power Steering, Remote Fuel Lid Opene... 0.000000e+00



Feature Importance
Importance
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Features

Model development:

- **Data preparation by converting categorical variables to numaric and drop rows with missing values.**
- **Train test split-(80-20)ratio**
- **Standardization by using standarscaling.**
- **Model training train various modes and evaluate them using cross-validation.**
- **Hyperparameter tuning by using grid search for random forest and random search for garient boosting to find the best hyperparameters**

  Cross validation score:

Model	CV Mean Score	CV Std Score	Test MSE	Justification
Linear Regression	5.41E+39	3.23E+39	9.39E+37	Poor fit: Extremely high scores suggest the model fails to capture complex relationships.
Decision Tree	494,717,648,540.41	172,665,389,845.16	326,319,492,704.22	Overfitting: Improved over Linear Regression, but high CV std shows instability across folds.
Random Forest	398,454,491,628.33	144,649,441,043.32	263,214,666,960.99	Better generalization: More stable than Decision Tree, showing less overfitting.
Gradient Boosting	372,827,194,351.88	144,005,191,296.86	327,419,450,476.65	Similar to Random Forest: Performance close to Random Forest, but slightly higher Test MSE.
XGBoost	337,817,787,939.64	130,205,768,488.70	266,294,509,600.02	Efficient optimization: Lower MSE than Random Forest and Gradient Boosting, showing better fit.
CatBoost	325,389,256,311.80	139,345,251,339.86	220,562,979,764.72	Best performance: Lowest Test MSE and more stable CV scores, handles categorical features well.

Explanation for Justification:

- **Linear Regression: The extremely high values indicate that the model doesn't suit the non-linear nature of the dataset.**

- **Decision Tree: It improves over Linear Regression, but due to high variability in CV scores, it seems to overfit in some cases.**
- **Random Forest: More stable than Decision Tree due to its ensemble nature, reducing overfitting and leading to better generalization.**
- **Gradient Boosting: Slightly more prone to overfitting than Random Forest, but still provides decent results.**
- **XGBoost: Optimized for non-linearity and provides better performance than both Random Forest and Gradient Boosting.**
- **CatBoost: Excels at handling categorical variables, offering the lowest MSE and the most consistent performance, making it the best choice.**

Model	Best Parameters	Best Score	Justification
Random Forest	{'max_depth': None, 'min_samples_split': 2, 'n_estimators': 50}	388,015,744,230.44	Random Forest works well by combining multiple decision trees, and tuning these parameters helps control tree complexity and accuracy.
Gradient Boosting	{'n_estimators': 200, 'max_depth': 7, 'learning_rate': 0.1}	332,860,870,887.31	Gradient Boosting optimizes weak learners, and the choice of n_estimators, max_depth, and learning_rate refines model performance.
XGBoost	{'colsample_bytree': 1, 'learning_rate': 0.5, 'max_depth': 3, 'n_estimators': 200, 'subsample': 1}	325,147,990,800.31	XGBoost boosts performance by sampling columns and trees; learning_rate and max_depth affect model speed and complexity.
CatBoost	{'depth': 4, 'iterations': 500, 'learning_rate': 0.1}	328,457,115,576.81	CatBoost works with categorical data efficiently, and tuning depth, iterations, and learning_rate enhances its learning process.

Decision Tree	{'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 10}	470,682,232,197.94	Decision Trees overfit without constraints; tuning max_depth and min_samples controls tree splitting and overfitting.
Ridge Regression	{'alpha': 100}	658,646,765,705.17	Ridge Regression adds L2 regularization to reduce overfitting, and the alpha parameter controls the regularization strength.

Random Forest:

- **Best Parameters:** {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 50}
- **Justification:** Allowing unrestricted depth (max_depth=None) helps capture intricate patterns. A minimal split (min_samples_split=2) ensures thorough exploration of data, while 50 estimators improve stability through averaging.
- **Gradient Boosting:**
  - **Best Parameters:** {'n_estimators': 200, 'max_depth': 7, 'learning_rate': 0.1}
  - **Justification:** A moderately deep tree (max_depth=7) balances complexity and overfitting, while the combination of 200 estimators and a learning rate of 0.1 ensures gradual, stable improvements.
- **XGBoost:**
  - **Best Parameters:** {'colsample_bytree': 1, 'learning_rate': 0.5, 'max_depth': 3, 'n_estimators': 200, 'subsample': 1}
  - **Justification:** A smaller tree depth (max_depth=3) prevents overfitting, while a high learning rate (learning_rate=0.5) speeds up convergence, and subsampling features (colsample_bytree=1) ensures all features are considered.
- **CatBoost:**
  - **Best Parameters:** {'depth': 4, 'iterations': 500, 'learning_rate': 0.1}
  - **Justification:** Shallow trees (depth=4) maintain simplicity, while a large number of iterations (500) and a learning rate of 0.1 guarantee a robust and gradual learning process.
- **Decision Tree:**
  - **Best Parameters:** {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 10}
  - **Justification:** Unrestricted depth ensures full data capture, while setting min_samples_leaf=2 and min_samples_split=10 prevents overfitting by requiring more samples to split nodes.

- **Ridge Regression:**
  - **Best Parameters: {'alpha': 100}**
  - **Justification: A higher alpha value (100) ensures stronger regularization, which helps in controlling overfitting and stabilizing the model in the presence of multicollinearity.**

Model evaluation:

Model	Test MSE	Test MAE	R^2 Score	Jestification
CatBoost:	Test MSE: 221014142376.61	Test MAE: 184409.85	R^2 Score: 0.85	Highest R^2 score, indicating strong predictive power. Good balance of MSE and MAE.
XGBoost:	Test MSE: 248640397125.70	Test MAE: 178325.66	R^2 Score: 0.83	Very close to CatBoost in performance, slightly lower MSE and MAE. Strong alternative.
Random Forest:	Test MSE: 262989441640.37	Test MAE: 169266.20	R^2 Score: 0.82	Good performance but not as high as the top two models.
Gradient Boosting:	Test MSE: 329219235695.23	Test MAE: 240202.57	R^2 Score: 0.78	Lower R^2 and higher errors indicate weaker performance compared to others.
Decision Tree:	Test MSE: 344554561341.00	Test MAE: 216658.16	R^2 Score: 0.77	Simplest model with the lowest performance metrics. May overfit.
Ridge Regression:	Test MSE: 621646041270.78	Test MAE: 466179.81	R^2 Score: 0.58	Worst performing model

Summary:

- **CatBoost stands out with the best performance, making it the preferred model for further tuning and application.**
- **XGBoost is a strong alternative, particularly if interpretability or specific features are prioritized.**
- **Random Forest, Gradient Boosting, and Decision Tree offer diminishing returns and may be less suitable for this dataset.**

This table provides a clear overview for decision-making based on model performance.

And did hyperparameter tuning for catboost modes to increase accuracy and achieved Best score for CatBoost: 0.86.

User Interface:

1. Input Fields

The application provides the following input fields:

•Car Make: Drop-down selection of available car models.

•Fuel Type: Drop-down selection of fuel types (e.g., Petrol, Diesel).

•Registration Year: Drop-down selection of the year the car was registered.

•Engine Power: Numeric input for the car's engine power in bhp.

•Kilometers Driven: Numeric input for the total kilometers the car has been driven.

•City: Drop-down selection of cities where the car is located.

2. Output

The predicted car price is prominently displayed in the center of the page, ensuring visibility and clarity. The output is formatted to appear professional and easily readable.

3. Model Performance Metrics

At the bottom of the page, the application provides the MSE and R2 scores, giving users a quick overview of the model's accuracy.

Conclusion:

This documentation outlines the functionality, data handling, and user interface of the Used Car Price Prediction Application. The application is designed for simplicity, accuracy, and user-friendliness, making it a valuable tool for estimating the price of used cars based on essential attributes. The model's performance metrics are included to assure users of the prediction's reliability.