Car Dheko - Used Car Price Prediction

Project Overview

Car Dheko is a Streamlit application designed to predict the price of used cars based on various features such as mileage, power, engine size, and more. This tool leverages machine learning models to provide accurate price estimates for users looking to buy or sell used cars.

Dependencies

The application relies on the following Python libraries:

- streamlit
- pandas
- numpy
- pickle
- matplotlib
- scikit-learn

Features

- **Input Fields**: Includes fields for mileage, max power, kilometres driven, engine size, wheel size, number of seats, ownership, fuel type, body type, transmission, OEM, insurance validity, and city.
- **Price Prediction**: Displays the predicted price of the used car based on input features.
- **Data Handling**: Cleans and preprocesses input data to align with the trained model requirements.

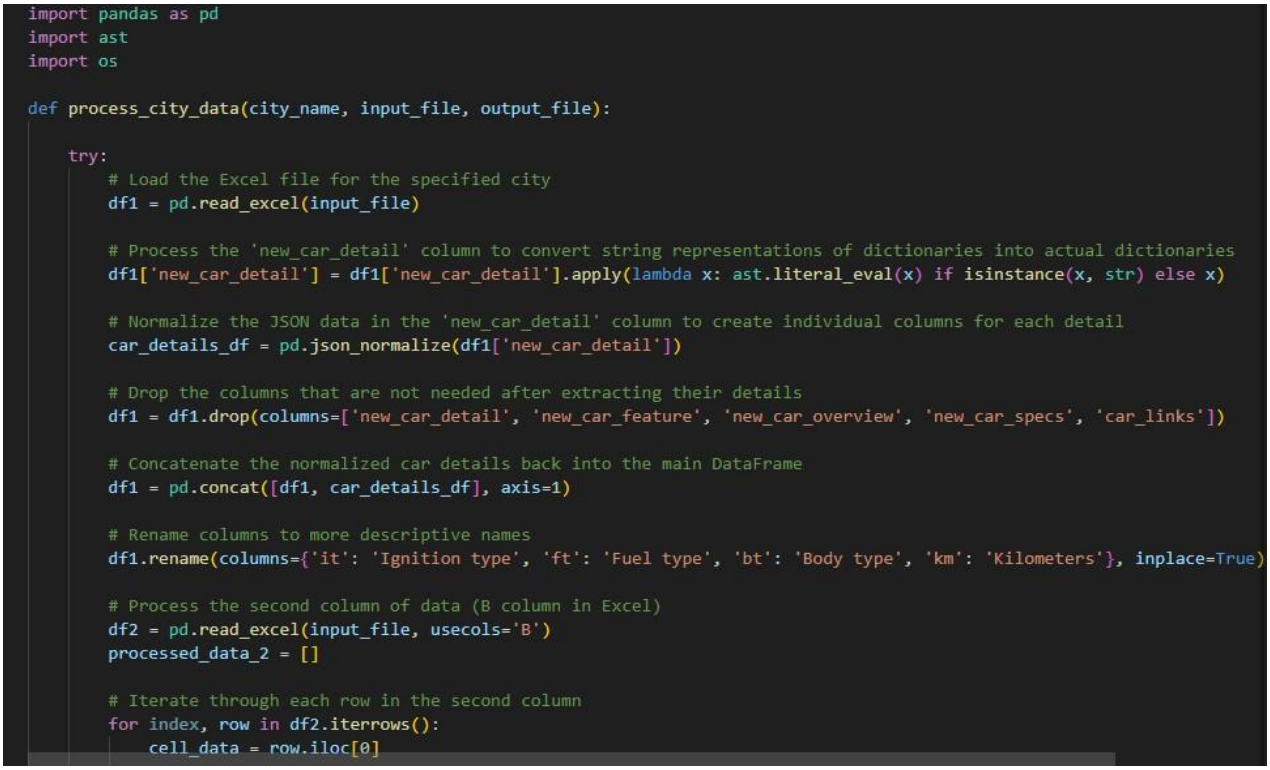
Unstructured to structured data format:

Overview

- This script processes car data from multiple Excel files, transforms it into a structured format, and saves it as CSV files. The code is divided into a function that handles the data processing and a section that iterates over multiple cities to apply this function**.**

- This is efficiently processes and structures data from multiple Excel files, transforming nested dictionary data into a clean, structured format suitable for further analysis. It includes robust error handling and supports data processing for multiple cities, saving the results in a specified directory

- To process the new_car_detail column by converting any string representations of dictionaries into actual dictionary objects. This conversion is necessary for later steps, such as flattening the nested structure or performing other operations that require the data to be in dictionary form.
- By using ast.literal_eval(), the code ensures that only safe expressions (like dictionaries) are evaluated, avoiding potential security risks associated with using the more general eval() function.
- Evaluates strings to convert them into actual dictionaries.
- Flattens nested dictionaries into a structured format using pd. json_normalize ().
- Merges these flattened details back into the original Data Frame.
- To transform complex, nested data structures stored in Excel into a simple, structured tabular format that is easier to analyse and manipulate.



```python
import pandas as pd
import ast
import os

def process_city_data(city_name, input_file, output_file):

    try:
        # Load the Excel file for the specified city
        df1 = pd.read_excel(input_file)

        # Process the 'new_car_detail' column to convert string representations of dictionaries into actual dictionaries
        df1['new_car_detail'] = df1['new_car_detail'].apply(lambda x: ast.literal_eval(x) if isinstance(x, str) else x)

        # Normalize the JSON data in the 'new_car_detail' column to create individual columns for each detail
        car_details_df = pd.json_normalize(df1['new_car_detail'])

        # Drop the columns that are not needed after extracting their details
        df1 = df1.drop(columns=['new_car_detail', 'new_car_feature', 'new_car_overview', 'new_car_specs', 'car_links'])

        # Concatenate the normalized car details back into the main DataFrame
        df1 = pd.concat([df1, car_details_df], axis=1)

        # Rename columns to more descriptive names
        df1.rename(columns={'it': 'Ignition type', 'ft': 'Fuel type', 'bt': 'Body type', 'km': 'Kilometers'}, inplace=True)

        # Process the second column of data (B column in Excel)
        df2 = pd.read_excel(input_file, usecols='B')
        processed_data_2 = []

        # Iterate through each row in the second column
        for index, row in df2.iterrows():
            cell_data = row.iloc[0]
```

```
        # Convert string representations of dictionaries into actual dictionaries
        if isinstance(cell_data, str):
            try:
                cell_data = ast.literal_eval(cell_data)
            except ValueError:
                continue

        # Extract relevant data if it is a dictionary and contains a 'top' key
        if isinstance(cell_data, dict) and 'top' in cell_data:
            flat_data = {item['key']: item['value'] for item in cell_data['top']}
            processed_data_2.append(flat_data)

    # Convert the processed data into a DataFrame
    final_df2 = pd.DataFrame(processed_data_2)

    # Process the third column of data (C column in Excel)
    df3 = pd.read_excel(input_file, usecols='C')
    processed_data_3 = []

    # Iterate through each row in the third column
    for index, row in df3.iterrows():
        cell_data = row.iloc[0]

        # Convert string representations of dictionaries into actual dictionaries
        if isinstance(cell_data, str):
            try:
                cell_data = ast.literal_eval(cell_data)
            except ValueError:
                continue
```

```
        # Extract relevant data if it is a dictionary and contains a 'top' key
        if isinstance(cell_data, dict) and 'top' in cell_data:
            flat_data = {item['value'] for item in cell_data['top']}
            processed_data_3.append(flat_data)

    # Convert the processed data into a DataFrame
    final_df3 = pd.DataFrame(processed_data_3)

    # Assign names to the columns of the third DataFrame
    final_df3.columns = ['Feature1', 'Feature2', 'Feature3', 'Feature4', 'Feature5', 'Feature6', 'Feature7', 'Feature8'

    # Process the fourth column of data (D column in Excel)
    df4 = pd.read_excel(input_file, usecols='D')
    processed_data_4 = []

    # Iterate through each row in the fourth column
    for index, row in df4.iterrows():
        cell_data = row.iloc[0]

        # Convert string representations of dictionaries into actual dictionaries
        if isinstance(cell_data, str):
            try:
                cell_data = ast.literal_eval(cell_data)
            except ValueError:
                continue

        # Extract relevant data if it is a dictionary and contains a 'top' key
        if isinstance(cell_data, dict) and 'top' in cell_data:
            flat_data = {item['key']: item['value'] for item in cell_data['top']}
            processed_data_4.append(flat_data)
```

```
        # Convert the processed data into a DataFrame
        final_df4 = pd.DataFrame(processed_data_4)

        # Process the fifth column of data (E column in Excel)
        df5 = pd.read_excel(input_file, usecols='E')

        # Concatenate all the processed DataFrames column-wise to form the final structured DataFrame
        city_df = pd.concat([df1, final_df2, final_df3, final_df4, df5], axis=1)

        # Add a new column to the DataFrame indicating the city name
        city_df['City'] = city_name

        # Save the final DataFrame to a CSV file
        city_df.to_csv(output_file, index=False)

        # Display a preview of the processed DataFrame for verification
        print(f"Processed data for {city_name}:")
        print(city_df.head())

        return city_df

    except Exception as e:
        # Catch any exceptions that occur during the processing and print an error message
        print(f"Error processing data for {city_name}: {e}")
        return None
```

```
# Define the input file paths and corresponding cities
cities = {
    'Bangalore': r"C:\Users\DELL\Desktop\car_details\Dataset\bangalore_cars.xlsx",
    'Chennai': r"C:\Users\DELL\Desktop\car_details\Dataset\chennai_cars.xlsx",
    'Delhi': r"C:\Users\DELL\Desktop\car_details\Dataset\delhi_cars.xlsx",
    'Hyderabad': r"C:\Users\DELL\Desktop\car_details\Dataset\hyderabad_cars.xlsx",
    'Jaipur': r"C:\Users\DELL\Desktop\car_details\Dataset\jaipur_cars.xlsx",
    'Kolkata': r"C:\Users\DELL\Desktop\car_details\Dataset\kolkata_cars.xlsx"
}

# Define the directory where the output CSV files will be saved
output_dir = r"C:\Users\DELL\Desktop\car_details\structured_data"

# Process the data for each city by iterating through the defined cities and file paths
for city, input_file in cities.items():
    # Generate the output file path for each city
    output_file = os.path.join(output_dir, f"structured_data_{city.lower()}.csv")

    # Call the function to process the city's data and save it to a CSV file
    process_city_data(city, input_file, output_file)
```

EDA Process:

Overview

This script processes a car dataset, handles categorical variables, normalizes features, visualizes data distributions, identifies and filters outliers, and calculates correlations with the car prices. It provides a comprehensive exploratory data analysis (EDA) to help understand the relationships between different features and car prices.

- Encoding categorical variables.
- Normalizing numerical features.
- Filtering outliers.
- Visualizing feature distributions.
- Log-transforming skewed data.
- Calculating and classifying kurtosis.
- Displaying feature correlations.
- This EDA process is critical for understanding the data before applying machine learning models, ensuring that the data is clean, well-prepared, and ready for further analysis or modeling.

```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy import stats
from sklearn.preprocessing import StandardScaler

def process_car_data(file_path):
    # Load the dataset
    df = pd.read_csv(file_path)

    # Drop the 'Torque' column if it exists
    if 'Torque' in df.columns:
        df.drop(['Torque'], axis=1, inplace=True)

    # One-hot encode categorical variables
    df = pd.get_dummies(df, columns=[
        'Fuel Type', 'Body type', 'transmission', 'Insurance Validity', 'oem',
        'Power Window Front', 'City', 'Head Light', 'Rear View Mirror', 'Air conditioner',
        'Power Steering', 'Heater', 'Child Lock', 'Power Window Front',
        'Central Locking', 'Cd Player&Radio', 'Anti Lock Braking System', 'Fog Lights Front'
    ], dtype='int')

    # Label encode the 'RTO' column
    label_encoder = LabelEncoder()
    df['RTO'] = label_encoder.fit_transform(df['RTO'])

    # Display box plots for various features
    features_to_plot = ['price', 'modelYear', 'Kms Driven', 'Year of Manufacture',
                        'Mileage', 'Engine']
```

```python
# Include 'Torque' in the plots if it exists
if 'Torque' in df.columns:
    features_to_plot.append('Torque')

for feature in features_to_plot:
    sns.boxplot(df[feature])
    plt.show()

# Normalize and filter outliers for each feature
def normalize_and_filter(df, column):
    df[column] = (df[column] - df[column].mean()) / df[column].std()
    return df[(df[column] > -2) & (df[column] < 2)]

df = normalize_and_filter(df, 'price')
sns.boxplot(df['price'])
plt.show()

df = normalize_and_filter(df, 'modelYear')
sns.boxplot(df['modelYear'])
plt.show()

df = normalize_and_filter(df, 'Kms Driven')
sns.boxplot(df['Kms Driven'])
plt.show()

df = normalize_and_filter(df, 'Year of Manufacture')
sns.boxplot(df['Year of Manufacture'])
plt.show()
```

```
    df = normalize_and_filter(df, 'Mileage')
    sns.boxplot(df['Mileage'])
    plt.show()

    df = normalize_and_filter(df, 'Engine')
    sns.boxplot(df['Engine'])
    plt.show()

    if 'Torque' in df.columns:
        df = normalize_and_filter(df, 'Torque')
        sns.boxplot(df['Torque'])
        plt.show()

    # Log-transform the 'price' column and plot the histogram
    skew = np.log(df['price'])
    sns.histplot(skew)
    plt.show()

    # Calculate kurtosis and determine the type of kurtosis
    def get_kurtosis_type(value):
        if value > 3:
            return "Leptokurtic"
        elif value < 3:
            return "Platykurtic"
        else:
            return "Mesokurtic"

    data = df['price']
    kurtosis_value = stats.kurtosis(data)
    kurtosis_type = get_kurtosis_type(kurtosis_value)
```

```
    print(f"Kurtosis Type: {kurtosis_type}")
    print(f"Kurtosis Value: {kurtosis_value}")

    # Display the correlation heatmap
    plt.figure(figsize=(40, 40))
    correlation_matrix = df.corr()
    sns.heatmap(correlation_matrix, annot=True)
    plt.show()

    # Return the correlation of features with the 'price' column
    price_correlation = correlation_matrix['price'].sort_values(ascending=False)
    return price_correlation

file_path = r"C:\Users\DELL\Desktop\car_details\structured_data\preprocessed_data.csv"
price_corr = process_car_data(file_path)
print(price_corr)
```

Model Training and Testing:

Overview

This script processes and cleans a car dataset, followed by training and evaluating multiple regression models to predict car prices. The script is structured into three main functions: clean_and_preprocess_data, train_and_evaluate_models, and main. It also includes saving the best-performing model (Random Forest) using the pickle module.

Import Libraries

- pandas: Used for data manipulation and analysis.
- re: Provides regular expressions for string matching and manipulation.
- numpy: Provides support for numerical operations and data structures.
- sklearn: Provides tools for model training, evaluation, and preprocessing.
- pickle: Used for saving and loading trained machine learning models.

Define the Models

- A dictionary of different regression models is defined, including linear models (Linear Regression, Lasso, Ridge), tree-based models (Decision Tree, Random Forest, Gradient Boosting), and a K-Nearest Neighbors model.

Model Training and Evaluation

- Each model is trained on the training data (x_train, y_train).
- Predictions are made on both the training and testing data.
- **Steps:**
    - **Split Data:** Splits the dataset into training and testing sets.
    - **Define Hyperparameter Grids:** Specifies different sets of hyperparameters to try for each model.
    - **Model Tuning and Evaluation:** Uses GridSearchCV to find the best hyperparameters for each model, evaluates their performance, and saves the best models.

Hyperparameter Tuning with GridSearchCV

GridSearchCV Overview:

- GridSearchCV is a technique for hyperparameter optimization, used to find the best combination of hyperparameters for a given model. It exhaustively searches through a specified grid of hyperparameters and evaluates the model's performance using cross-validation.

Evaluation Metrics

- **Mean Squared Error (MSE)**: Measures the average squared difference between the predicted and actual values.
- **Mean Absolute Error (MAE)**: Measures the average absolute difference between the predicted and actual values.
- **R-squared (R^2)**: Indicates the proportion of variance in the dependent variable that is predictable from the independent variables.

This code is designed to

1. **Clean and preprocess** a car dataset, converting relevant columns to numeric types, handling missing values, and one-hot encoding categorical variables.
2. **Train and evaluate** various regression models on the pre-processed data, comparing their performance using metrics like MSE, MAE, and R^2.
3. **Save** the best-performing model (Random Forest) for future use.
4. This approach is useful for building a robust car price prediction model by leveraging multiple regression techniques.

Codes are below:

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import pickle

def clean_and_preprocess_data(file_path):
    # Load the dataset
    df = pd.read_csv(file_path)

    # Drop unnecessary columns
    columns_to_drop = [
        'priceSaving', 'priceFixedText', 'trendingText.imgUrl', 'trendingText.heading',
        'trendingText.desc', 'car_links', 'Seats.1', 'Ignition type', 'owner', 'Feature1',
        'Feature2', 'Feature3', 'Feature4', 'Feature5', 'Feature6', 'Feature7', 'Feature8',
        'Feature9', 'Engine Displacement', 'variantName', 'priceActual', 'Kilometers',
        'model', 'ownerNo', 'centralVariantId', 'Registration Year', 'Fuel Type', 'RTO',
        'Engine Displacement', 'Transmission', 'Year of Manufacture', 'Torque',
        'Head Light', 'Rear View Mirror', 'Air conditioner', 'Power Steering', 'Heater',
        'Child Lock', 'Central Locking', 'Cd Player&Radio', 'Anti Lock Braking System',
        'Fog Lights Front'
    ]
    df = df.drop(columns=columns_to_drop)
```

```python
    # Convert and clean the 'price' column
    df = df.dropna(subset=['price'])

    def convert_price(price):
        price = price.replace('â‚¹', '').replace('₹', '').replace(',', '').strip()
        if 'Lakh' in price:
            price = price.replace('Lakh', '').strip()
            return float(price) * 100000
        elif 'Crore' in price:
            price = price.replace('Crore', '').strip()
            return float(price) * 10000000
        return float(price)

    df['price'] = df['price'].apply(convert_price)

    # Clean and preprocess other columns
    # Extract numeric values from the 'Mileage' column and convert them to float
    df['Mileage'] = df['Mileage'].str.extract(r'(\d+\.?\d*)').astype(float)
    # Fill missing values in the 'Mileage' column with the mean mileage
    df['Mileage'] = df['Mileage'].fillna(df['Mileage'].mean())

    # Extract numeric values from the 'Max Power' column and convert them to float
    df['Max Power'] = df['Max Power'].str.extract(r'(\d+\.?\d*)')[0].astype(float).fillna(
        # Fill missing values with the mode (most frequent value) of 'Max Power'
        df['Max Power'].str.extract(r'(\d+\.?\d*)')[0].mode()[0]
    )

    # Clean and convert 'Kms Driven' column to integer
    df['Kms Driven'] = df['Kms Driven'].str.replace(',', '').str.replace('Kms', '').fillna(0).astype(int)
```

```
    # Clean and convert 'Engine' column to integer
    df['Engine'] = df['Engine'].str.replace(',', '').str.replace('CC', '').fillna(0).astype(int)

    # Remove 'R' from 'Wheel Size' column and fill missing values with the mode
    df['Wheel Size'] = df['Wheel Size'].str.replace('R', '').fillna(df['Wheel Size'].mode()[0])

    # Clean and convert 'Seats' column to integer
    df['Seats'] = df['Seats'].str.replace('Seats', '').str.strip()
    df['Seats'] = df['Seats'].fillna(df['Seats'].mode()[0])
    df['Seats'] = df['Seats'].astype(int)

    # Fill missing values in 'Ownership' column with the mode
    df['Ownership'] = df['Ownership'].fillna(df['Ownership'].mode()[0])

    # One-hot encode categorical variables
    df = pd.get_dummies(df, columns=[
        'Fuel type', 'Body type', 'transmission', 'oem',
        'Insurance Validity', 'City', 'Power Window Front', 'Ownership'
    ], dtype='int')

    X = df.drop(['price'], axis=1)
    y = df['price']

    return X, y

def train_and_evaluate_models(X, y):
    # Split the data into training and testing sets
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
    # Define hyperparameter grids for GridSearchCV
    param_grids = {
        'Linear Regression': {},
        'Lasso': {'alpha': [0.1, 0.5, 1.0, 1.5, 2.0]},
        'Ridge': {'alpha': [0.1, 0.5, 1.0, 1.5, 2.0]},
        'Decision Tree': {
            'max_depth': [3, 5, 7, 10],
            'min_samples_split': [2, 5, 10]
        },
        'Random Forest': {
            'n_estimators': [50, 100, 200],
            'max_depth': [3, 5, 7, 10],
            'min_samples_split': [2, 5, 10]
        },
        'Gradient Boosting': {
            'n_estimators': [50, 100, 200],
            'learning_rate': [0.01, 0.05, 0.1, 0.2],
            'max_depth': [3, 5, 7]
        },
        'K-Neighbors Regressor': {
            'n_neighbors': [3, 5, 7, 10],
            'weights': ['uniform', 'distance']
        }
    }

    best_models = {}
```

```
    # Perform hyperparameter tuning and evaluation
    for name, model in models.items():
        if param_grids[name]:
            grid_search = GridSearchCV(model, param_grids[name], cv=5, scoring='r2', n_jobs=-1, verbose=2)
            grid_search.fit(x_train, y_train)
            best_model = grid_search.best_estimator_
            print(f"--- {name} ---")
            print(f"Best Parameters: {grid_search.best_params_}")

            # Evaluate on test data
            test_pred = best_model.predict(x_test)
            print(f"MSE Test: {mean_squared_error(y_test, test_pred)}")
            print(f"MAE Test: {mean_absolute_error(y_test, test_pred)}")
            print(f"R2 Test: {r2_score(y_test, test_pred)}\n")

            best_models[name] = best_model
        else:
            model.fit(x_train, y_train)
            train_pred = model.predict(x_train)
            test_pred = model.predict(x_test)
            print(f"--- {name} ---")
            print(f"MSE Train: {mean_squared_error(y_train, train_pred)}")
            print(f"MSE Test: {mean_squared_error(y_test, test_pred)}")
            print(f"MAE Train: {mean_absolute_error(y_train, train_pred)}")
            print(f"MAE Test: {mean_absolute_error(y_test, test_pred)}")
            print(f"R2 Train: {r2_score(y_train, train_pred)}")
            print(f"R2 Test: {r2_score(y_test, test_pred)}\n")
```

```
            # Save the final model
            if name == 'Random Forest':
                with open('random_forest.pkl', mode='wb+') as file:
                    pickle.dump(model, file)

    # Save the best model from each type
    for name, model in best_models.items():
        with open(f'{name.replace(" ", "_").lower()}_best_model.pkl', mode='wb+') as file:
            pickle.dump(model, file)
        print(f"Best model saved: {name}")

def main():
    # File path to the dataset
    file_path = r"C:\Users\DELL\Desktop\car_details\structured_data\updated_cars_features.csv"

    # Clean and preprocess the data
    X, y = clean_and_preprocess_data(file_path)
```

```
    # Define the models
    global models
    models = {
        'Linear Regression': LinearRegression(),
        'Lasso': Lasso(),
        'Ridge': Ridge(),
        'Decision Tree': DecisionTreeRegressor(),
        'Random Forest': RandomForestRegressor(),
        'Gradient Boosting': GradientBoostingRegressor(),
        'K-Neighbors Regressor': KNeighborsRegressor()
    }

    # Train and evaluate the models
    train_and_evaluate_models(X, y)

if __name__ == "__main__":
    main()
```

Streamlit:

Overview

This code implements a Streamlit application for predicting the price of a used car. The app uses a machine learning model (Random Forest) that has been trained on a dataset of car features and prices.

1. Imports and Setup

- **Libraries**:
    - pandas for data manipulation.
    - pickle for loading the trained model.
    - numpy for numerical operations.
    - streamlit for creating the web application interface.
- The application is designed to run within Streamlit, which is a framework for creating interactive web apps in Python.

2. Data Loading and Preprocessing (load_data)

- **Loading the Data**:
    - The function loads the dataset from a specified file path using pandas.read_csv.
- **Dropping Unnecessary Columns**:
    - Certain columns in the dataset that are not relevant for the prediction task are dropped.
- **Cleaning and Formatting**:
    - The price column is cleaned by removing non-numeric characters and converting values (e.g., converting "Lakh" and "Crore" to numeric values).
    - Other columns such as Mileage, Max Power, Kms Driven, and Engine are also cleaned and formatted for consistency.
- **One-Hot Encoding**:
    - Categorical variables are converted into binary columns (one-hot encoding) to be used by the machine learning model.
- The function returns the cleaned and processed DataFrame.

3. Model Loading (load_model)

- **Loading the Trained Model**:
    - The function loads a pre-trained Random Forest model from a pickle file (random_forest_best_model.pkl).
- The model is then returned for use in predictions.

4. Input Data Preprocessing (preprocess_input_data)

- **Input Formatting**:
    - The function converts user inputs into a DataFrame that matches the structure expected by the model.
- **One-Hot Encoding**:
    - Similar to the dataset preprocessing, the input data is one-hot encoded to align with the model's training.

- **Handling Missing Columns**:
    - The function ensures that any columns expected by the model but missing in the input data are added with a default value of 0.
- The function returns the processed input DataFrame, ready for prediction.

5. Price Formatting (format_price)

- **Price Display**:
    - This function converts a numeric price into a more human-readable format, displaying the price in "Lakhs" or "Crores" depending on its value.

6. Streamlit Application Interface (main)

- **Title and Description**:
    - The app displays a title ("Car Dheko Used Car Price Prediction") and a brief description.
- **Image Display**:
    - An image (e.g., a logo) is displayed at the top of the app.
- **Sidebar for Input**:
    - The app uses the sidebar to collect input from the user about the car's features (e.g., Mileage, Max Power, Kms Driven, Engine, etc.).
    - The user selects various options from dropdown menus (e.g., Fuel type, Body type, transmission, etc.).
- **Prediction Button**:
    - When the user clicks the "Predict Price" button, the app processes the input data, uses the loaded model to predict the price, and displays the result in a formatted manner.

7. Execution

- The main function is executed when the script runs, setting up the Streamlit interface and handling the prediction logic.

The app provides an interactive interface where users can input the features of a used car, and it predicts the price based on a pre-trained Random Forest model. The interface is user-friendly, and the code handles both data preprocessing and prediction, ensuring that the input data is correctly formatted for the model.

```python
import streamlit as st
import pandas as pd
import pickle
import numpy as np

#Load and preprocess the dataset
def load_data():
    df = pd.read_csv(
        r"C:\Users\DELL\Desktop\car_details\structured_data\updated_cars_features.csv"
    )

    # Drop unnecessary columns
    df = df.drop([
        'priceSaving', 'priceFixedText', 'trendingText.imgUrl', 'trendingText.heading',
        'trendingText.desc', 'car_links', 'Seats.1', 'Ignition type', 'owner', 'Feature1',
        'Feature2', 'Feature3', 'Feature4', 'Feature5', 'Feature6', 'Feature7', 'Feature8',
        'Feature9', 'Engine Displacement', 'variantName', 'priceActual', 'Kilometers',
        'model', 'ownerNo', 'centralVariantId', 'Registration Year', 'Fuel Type', 'RTO',
        'Engine Displacement', 'Transmission', 'Year of Manufacture', 'Torque',
        'Head Light', 'Rear View Mirror', 'Air conditioner', 'Power Steering', 'Heater',
        'Child Lock', 'Central Locking', 'Cd Player&Radio', 'Anti Lock Braking System',
        'Fog Lights Front'
    ], axis=1)

    # Convert and clean the 'price' column
    df = df.dropna(subset=['price'])
```

```
    #Convert price string to numeric value
    def convert_price(price):
        price = price.replace('â,¹', '').replace('₹', '').replace(',', '').strip()
        if 'Lakh' in price:
            price = price.replace('Lakh', '').strip()
            return float(price) * 100000
        elif 'Crore' in price:
            price = price.replace('Crore', '').strip()
            return float(price) * 10000000
        return float(price)

    df['price'] = df['price'].apply(convert_price)

    # Clean and preprocess other columns
    # Extract numeric values from the 'Mileage' column and convert them to float
    df['Mileage'] = df['Mileage'].str.extract(r'(\d+\.?\d*)').astype(float)
    # Fill missing values in the 'Mileage' column with the mean mileage
    df['Mileage'] = df['Mileage'].fillna(df['Mileage'].mean())
    # Extract numeric values from the 'Max Power' column and convert them to float
    df['Max Power'] = df['Max Power'].str.extract(r'(\d+\.?\d*)')[0].astype(float).fillna(
        # Fill missing values with the mode (most frequent value) of 'Max Power'
        df['Max Power'].str.extract(r'(\d+\.?\d*)')[0].mode()[0]
    )
    # Clean and convert 'Kms Driven' column to integer
    df['Kms Driven'] = df['Kms Driven'].str.replace(',', '').str.replace('Kms', '').fillna(0).astype(int)
    # Clean and convert 'Engine' column to integer
    df['Engine'] = df['Engine'].str.replace(',', '').str.replace('CC', '').fillna(0).astype(int)
    # Remove 'R' from 'Wheel Size' column and fill missing values with the mode
    df['Wheel Size'] = df['Wheel Size'].str.replace('R', '').fillna(df['Wheel Size'].mode()[0])
```

```
    # Clean and convert 'Seats' column to integer
    df['Seats'] = df['Seats'].str.replace('Seats', '').str.strip()
    df['Seats'] = df['Seats'].fillna(df['Seats'].mode()[0]).astype(int)
    # Fill missing values in 'Ownership' column with the mode
    df['Ownership'] = df['Ownership'].fillna(df['Ownership'].mode()[0])

    # One-hot encode categorical variables
    df = pd.get_dummies(df, columns=[
        'Fuel type', 'Body type', 'transmission', 'oem', 'Insurance Validity', 'City',
        'Power Window Front', 'Ownership'
    ], dtype='int')

    return df

#Load the trained model
def load_model():
    with open('random_forest_best_model.pkl', mode='rb+') as file:
        model = pickle.load(file)
    return model

#Preprocess input data and align it with the model's expected features
def preprocess_input_data(input_data, feature_columns):
    input_df = pd.DataFrame([input_data])

    # One-hot encode categorical variables
    input_df = pd.get_dummies(input_df, columns=[
        'Fuel type', 'Body type', 'transmission', 'oem', 'Insurance Validity', 'City', 'Ownership'
    ], dtype='int')
```

```
    # Handle missing columns in the new data
    missing_cols = set(feature_columns) - set(input_df.columns)
    for col in missing_cols:
        input_df[col] = 0
    input_df = input_df[feature_columns]

    return input_df

#Format price with Lakhs and Crores
def format_price(price):
    if price >= 10000000:  # 1 Crore
        price_in_crores = price / 10000000
        return f'₹{price_in_crores:,.2f} Crores'
    else:
        price_in_lakhs = price / 100000
        return f'₹{price_in_lakhs:,.2f} Lakhs'

# Main function to run the Streamlit app
def main():
    st.title('Car Dheko Used Car Price Prediction')
    st.write("This app predicts the price of a used car based on the input features.")

    # Add and center the image
    st.image(r'C:\Projects\project_3\Cardekho.jpg', caption='Car Dheko Logo', use_column_width=True)

    # Load data and model
    df = load_data()
    model = load_model()
    feature_columns = df.columns[df.columns != 'price']  # Features used by the model
```

```
  # Input fields for user
  st.sidebar.header('Car Details')
  Mileage = st.sidebar.number_input('Mileage (kmpl)', min_value=0.0)
  Max_power = st.sidebar.number_input('Max Power (bhp)', min_value=0.0)
  Kms_Driven = st.sidebar.number_input('Kms Driven', min_value=0)
  Engine = st.sidebar.number_input('Engine (CC)', min_value=0)
  Wheel_Size = st.sidebar.number_input('Wheel Size', min_value=0)
  Seats = st.sidebar.number_input('Seats', min_value=1)
  Ownership = st.sidebar.selectbox('Ownership', options=[
      'First Owner', 'Second Owner', 'Third Owner', 'Fourth & Above Owner'
  ])
  Fuel_type = st.sidebar.selectbox('Fuel Type', options=['Petrol', 'Diesel', 'CNG', 'LPG'])
  Body_type = st.sidebar.selectbox('Body Type', options=[
      'Hatchback', 'Sedan', 'SUV', 'MUV', 'Convertible', 'Wagon'
  ])
  transmission = st.sidebar.selectbox('Transmission', options=['Manual', 'Automatic'])
  oem = st.sidebar.selectbox('OEM', options=[
      'Maruti', 'Hyundai', 'Honda', 'Toyota', 'Ford', 'BMW', 'Mercedes', 'Audi'
  ])
  Insurance_Validity = st.sidebar.selectbox('Insurance Validity', options=['Yes', 'No'])
  car_age=st.sidebar.number_input('car age',min_value=0)
  City = st.sidebar.selectbox('City Name', options=['Delhi', 'Mumbai', 'Bangalore', 'Chennai', 'Kolkata', 'Jaipur'])
```

```
    # Input data for prediction
    input_data = {
        'Mileage': Mileage,
        'Max Power': Max_power,
        'Kms Driven': Kms_Driven,
        'Engine': Engine,
        'Wheel Size': Wheel_Size,
        'Seats': Seats,
        'Ownership': Ownership,
        'Fuel type': Fuel_type,
        'Body type': Body_type,
        'transmission': transmission,
        'oem': oem,
        'Insurance Validity': Insurance_Validity,
        'City': City,
        'car age':car_age
    }

    # Preprocess input data
    input_df = preprocess_input_data(input_data, feature_columns)

    # Prediction
    if st.button('Predict Price'):
        prediction = model.predict(input_df)
        formatted_price = format_price(prediction[0])
        st.write(f'Predicted Car Price: {formatted_price}')

if __name__ == "__main__":
    main()
```

GitHub Link:

https://github.com/vishnupriya08-hub/All-Capston-Repo/tree/main