In [ ]:
```
%config IPCompleter.greedy=True
```

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```

```

In [ ]:
```
# df=pd.read_csv('Machine_learning//car_purchasing.csv')
dataset= pd.read_csv("C://Users//ALWAYSRAMESH//Downloads//cardata_new.csv", encodin
```

In [ ]:
```
dataset.head(2)
```

In [ ]:
```
dataset.isnull().sum()
```

In [ ]:
```
dataset.info()
```

Car_name

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
```

In [ ]:
```
Car_Name_le=LabelEncoder()
dataset['Car_Name']=Car_Name_le.fit_transform(dataset['Car_Name'])
```

In [ ]:
```
dataset
```

Fuel_Type

In [ ]:
```
dataset['Fuel_Type'].unique()
```

In [ ]:
```
Fuel_Type_le=LabelEncoder()
dataset['Fuel_Type']=Fuel_Type_le.fit_transform(dataset['Fuel_Type'])
```

Seller_Type

In [ ]:
```
Seller_Type=LabelEncoder()
dataset['Seller_Type']=Seller_Type.fit_transform(dataset['Seller_Type'])
```

In [ ]:
```
Transmission=LabelEncoder()
dataset['Transmission']=Transmission.fit_transform(dataset['Transmission'])
```

In [ ]:
```
input_data=dataset[['Car_Name','Year',  'Present_Price','Odometer',     'Fuel_Type'
output_data=dataset["Selling_Price"]
```

In [ ]:
```
sns.heatmap(data=dataset.corr(),annot=True)
plt.show()
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
ss=StandardScaler()
ss.fit_transform(input_data)
```

In [ ]:
```
#import numpy as np
```

In [ ]:
```
ss=StandardScaler()
input_data=pd.DataFrame(ss.fit_transform(input_data),columns=input_data.columns)
```

In [ ]:
```

```

In [ ]:
```
from sklearn.model_selection import train_test_split
```

In [ ]:
```
x_train,x_test,y_train,y_test=train_test_split(input_data,output_data,test_size=0.2
```

In [ ]:
```
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
```

In [ ]:
```
from sklearn.metrics import mean_squared_error,mean_absolute_error
```

In [ ]:
```
lr=LinearRegression()
lr.fit(x_train,y_train)
lr.score(x_train,y_train)*100 , lr.score(x_test,y_test)*100
```

In [ ]:
```
mean_squared_error(y_test,lr.predict(x_test)), mean_absolute_error(y_test,lr.predic
```

In [ ]:
```
lr1=Lasso(alpha=0.05)
lr1.fit(x_train,y_train)
lr1.score(x_train,y_train)*100 , lr1.score(x_test,y_test)*100
```

In [ ]:
```
lr2=Ridge(alpha=0.5)
lr2.fit(x_train,y_train)
lr2.score(x_train,y_train)*100 , lr2.score(x_test,y_test)*100
```

In [ ]:
```
lr3=ElasticNet(alpha=4)
lr3.fit(x_train,y_train)
lr3.score(x_train,y_train)*100 , lr3.score(x_test,y_test)*100
```

In [ ]:
```
dt=DecisionTreeRegressor()
dt.fit(x_train,y_train)
dt.score(x_train,y_train)*100 , dt.score(x_test,y_test)*100
```

In [ ]:
```
mean_squared_error(y_test,dt.predict(x_test))
```

In [ ]:
```
mean_squared_error(y_test,dt.predict(x_test)), mean_absolute_error(y_test,dt.predic
```

In [ ]:
```
# rf=RandomForestRegressor(n_estimators=10)
rf=RandomForestRegressor()
```

```
rf.fit(x_train,y_train)
rf.score(x_train,y_train)*100 , rf.score(x_test,y_test)*100
```

In [ ]:
```
mean_squared_error(y_test,rf.predict(x_test)), mean_absolute_error(y_test,rf.predic
```

In [ ]:
```
sv=SVR()
sv.fit(x_train,y_train)
sv.score(x_train,y_train)*100 , sv.score(x_test,y_test)*100
```

In [ ]:
```
knn=KNeighborsRegressor(n_neighbors=5)
knn.fit(x_train,y_train)
knn.score(x_train,y_train)*100 , knn.score(x_test,y_test)*100
```

In [ ]:
```
mean_squared_error(y_test,knn.predict(x_test)), mean_absolute_error(y_test,knn.pred
```

In [ ]:
```
y_test
```

In [ ]:
```
rf.predict([[-1.275759,0.821718,-0.817924,      -0.333500,      0.500183,       1.3
```

In [ ]:
```

```

In [ ]:
```
x_test
```

In [ ]:
```
print(x_train.shape)
```

In [ ]:
```
new_data=pd.DataFrame([["ritz", 2014,5.50, 27000, "Petrol", "Dealer", "Manual", 0]]
```

In [ ]:
```
Car_Name_le.transform(new_data["Car_Name"])
```

In [ ]:
```
new_data
```

In [ ]:
```
#new_data['Car_Name']=Car_Name_le.transform(new_data["Car_Name"])
new_data['Car_Name']=Car_Name_le.transform(new_data["Car_Name"])
```

In [ ]:
```
new_data["Fuel_Type"]=Fuel_Type_le.transform(new_data["Fuel_Type"])
```

In [ ]:
```
new_data["Seller_Type"]=Seller_Type.transform(new_data["Seller_Type"])
```

In [ ]:
```
new_data["Transmission"]=Transmission.transform(new_data["Transmission"])
```

In [ ]:
```
new_data=pd.DataFrame(ss.transform(new_data),columns=new_data.columns)
```

In [ ]:
```
new_data
```

In [ ]:
```
dt.predict(new_data)
```

In [ ]:
```
model =RandomForestRegressor()
model.fit(x_train, y_train)
```

In [ ]:
```
model.predict(new_data)
```

In [ ]:
```
!python app.py
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
import pickle

# Load the model
model = pickle.load(open("model.pkl", "rb"))

# Check expected input features
if hasattr(model, "feature_names_in_"):
    print("Model expects features:", model.feature_names_in_)
else:
    print("Feature names not found. Check dataset preprocessing.")
```

In [ ]:
```
from flask import Flask, request, render_template_string
import pickle
import numpy as np
import os
import pandas as pd

# Flask App Initialization
app = Flask(__name__)

# Load Machine Learning Model
model = pickle.load(open("model.pkl", "rb"))

html_code = '''
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Car Price Prediction</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            background: linear-gradient(to right, #1F1C2C, #928DAB);
            background-size: cover;
            color: white;
            padding: 50px;
        }
        form {
            background: rgba(0, 0, 0, 0.7);
            padding: 30px;
            border-radius: 10px;
            display: inline-block;
```

```
        }
        input, select {
            padding: 10px;
            margin: 10px;
            width: 300px;
            border-radius: 5px;
            border: none;
        }
        button {
            padding: 10px 20px;
            background-color: #ffcc00;
            border: none;
            cursor: pointer;
        }
    </style>
</head>
<body>
    <h1>🚗 Car Price Prediction</h1>
    <form action="/predict" method="post">
        <label>CarName:</label>
        <input type="text" name="Car_Name" placeholder="Car Name" required><br>

        <label>Manufacturing Year:</label>
        <input type="number" name="Year" placeholder="Year" required><br>

        <label>Present Price (in Lakhs):</label>
        <input type="number" step="0.01" name="Present_Price" placeholder="Price" r

        <label>KM Driven:</label>
        <input type="number" name="Odometer" placeholder="KM Driven" required><br>

        <label>Fuel Type:</label>
        <select name="Fuel_Type">
            <option value="0">Petrol</option>
            <option value="1">Diesel</option>
        </select><br>

        <label>Seller Type:</label>
        <select name="Seller_Type">
            <option value="0">Dealer</option>
            <option value="1">Individual</option>
        </select><br>

        <label>Transmission:</label>
        <select name="Transmission">
            <option value="0">Manual</option>
            <option value="1">Automatic</option>
        </select><br>

        <label>Owner:</label>
        <select name="Owner">
            <option value="0">First Owner</option>
            <option value="1">Second Owner</option>
            <option value="3">Third Owner</option>
        </select><br>
```

```
        <button type="submit">Predict Price</button>
    </form>
    {% if prediction_text %}
    <h2>{{ prediction_text }}</h2>
    {% endif %}
</body>
</html>
'''

@app.route("/")
def home():
    return render_template_string(html_code)

@app.route("/predict", methods=["POST"])
def predict():
    try:
        Car_Name = request.form["Car_Name"]
        Year = int(request.form["Year"])
        Present_Price = float(request.form["Present_Price"])
        Odometer = int(request.form["Odometer"])
        Fuel_Type = int(request.form["Fuel_Type"])
        Seller_Type = int(request.form["Seller_Type"])
        Transmission = int(request.form["Transmission"])
        Owner = int(request.form["Owner"])

        input_data = np.array([[Car_Name, Year, Present_Price, Odometer, Fuel_Type,
        prediction = model.predict(input_data)

        return render_template_string(html_code, prediction_text=f"🚗 Predicted Car
    except Exception as e:
        return render_template_string(html_code, prediction_text=f"❌ Error: {str(e

if __name__ == "__main__":
    app.run(debug=True)
```

if **name** == "**main**": app.run(debug=True, port=5001)

In [ ]:
```
if __name__ == "__main__":
    app.run(debug=False)
```

In [ ]:
```

```

In [ ]:
```

```