Mã nguồn Bài tập lớn Cuối kỳ

Học phần: Hệ hỗ trợ quyết định (MI4216)

Sinh viên: Nguyễn Trung Kiên 20227180

Mã lớp học: 163653 (2025.1)

GVHD: TS. Trần Ngọc Thăng

CHƯƠNG 2: KHAI PHÁ VÀ TIỀN XỬ LÝ DỮ LIỆU (CAR PRICE PREDICTION)

Sinh viên: Nguyễn Trung Kiên - 20227180
Mục tiêu:

1. Thống kê mô tả và đánh giá chất lượng dữ liệu.
2. Xử lý nhiễu, dữ liệu thiếu và các vấn đề về phân phối (Imbalance/Skewness).
3. Mã hóa và chuẩn hóa dữ liệu để sẵn sàng cho huấn luyện mô hình.

```
# ================================================================================
# 1. KHỞI TẠO VÀ TẢI DỮ LIỆU
# ================================================================================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, StandardScaler
from scipy import stats
import warnings

warnings.filterwarnings('ignore') # Tắt cảnh báo để output sạch hơn

# Cấu hình giao diện biểu đồ
sns.set(style="whitegrid")
plt.rcParams['figure.figsize'] = (12, 6)

# [TODO]: Kiên hãy thay đổi đường dẫn file csv của bạn tại đây
# Nếu chạy trên Colab, hãy upload file vào mục Files bên trái
file_path = 'CarsData.csv'

try:
    df = pd.read_csv(file_path)
    print("✅ Đã tải dữ liệu thành công!")
    print(f"Kích thước bộ dữ liệu: {df.shape[0]} dòng, {df.shape[1]} cột")
except FileNotFoundError:
    print("❌ Lỗi: Không tìm thấy file. Vui lòng kiểm tra lại đường dẫn.")
    # Tạo dữ liệu giả lập để demo code chạy (Nếu không có file)
    data = {
        'model': ['Fiesta', 'Focus', 'Kuga', 'Golf', 'Polo'] * 200,
        'year': np.random.randint(2015, 2024, 1000),
        'price': np.random.randint(5000, 30000, 1000),
        'transmission': ['Manual', 'Automatic', 'Semi-Auto'] * 333 + ['Manual'],
        'mileage': np.random.randint(1000, 100000, 1000),
        'fuelType': ['Petrol', 'Diesel'] * 500,
        'tax': np.random.choice([0, 20, 145, 160], 1000),
        'mpg': np.random.uniform(30, 80, 1000),
        'engineSize': np.random.choice([1.0, 1.5, 2.0, 3.0], 1000),
        'Manufacturer': ['Ford', 'Volkswagen', 'BMW'] * 333 + ['Ford']
    }
    df = pd.DataFrame(data)
    print("⚠️ Đã tạo dữ liệu giả lập để demo.")
```

```
# ==============================================================================
# 2. MÔ TẢ DỮ LIỆU (DESCRIPTIVE STATISTICS)
# ==============================================================================
print("\n--- 2.1. Xem trước dữ liệu ---")
display(df.head())

print("\n--- 2.2. Thông tin kiểu dữ liệu và giá trị thiếu ---")
print(df.info())

print("\n--- 2.3. Thống kê mô tả (Biến số) ---")
display(df.describe())

print("\n--- 2.4. Thống kê biến phân loại ---")
display(df.describe(include=['O']))

# ==============================================================================
# 3. XỬ LÝ DỮ LIỆU (CLEANING & IMBALANCE HANDLING)
# ==============================================================================

# 3.1. Xử lý dữ liệu thiếu (Missing Values)
# Chiến lược: Điền Median cho biến số (tránh ảnh hưởng bởi outliers) và Mode cho biến phân loại
print("\n--- 3.1. Xử lý Missing Values ---")
numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = df.select_dtypes(include=['object']).columns

for col in numeric_cols:
    if df[col].isnull().sum() > 0:
        df[col].fillna(df[col].median(), inplace=True)

for col in categorical_cols:
    if df[col].isnull().sum() > 0:
        df[col].fillna(df[col].mode()[0], inplace=True)

print("Đã xử lý xong giá trị thiếu.")

# 3.2. Xử lý Nhiễu (Outliers) cho biến mục tiêu 'price'
# Chiến lược: Sử dụng IQR (Interquartile Range) để lọc các xe có giá quá cao hoặc quá thấp bất thường
print("\n--- 3.2. Xử lý Outliers (Biến Price) ---")
Q1 = df['price'].quantile(0.25)
Q3 = df['price'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Lọc bỏ outliers
df_clean = df[(df['price'] >= lower_bound) & (df['price'] <= upper_bound)]
print(f"Số lượng bản ghi trước khi lọc: {len(df)}")
print(f"Số lượng bản ghi sau khi lọc IQR: {len(df_clean)}")
print(f"Đã loại bỏ {len(df) - len(df_clean)} bản ghi nhiễu.")

# 3.3. Phân tích phân phối và Xử lý "Mất cân bằng" (Skewness)
# Trong hồi quy, "mất cân bằng" thường là phân phối lệch. Ta dùng Log Transform.
print("\n--- 3.3. Kiểm tra phân phối Price ---")
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.histplot(df_clean['price'], kde=True, color='blue')
plt.title('Phân phối Giá xe (Gốc)')

plt.subplot(1, 2, 2)
sns.histplot(np.log1p(df_clean['price']), kde=True, color='green')
plt.title('Phân phối Giá xe (Log Transformed)')
plt.show()

# ==============================================================================
# 4. TRÍCH CHỌN ĐẶC TRƯNG & TIỀN XỬ LÝ (FEATURE ENGINEERING)
# ==============================================================================
print("\n--- 4.1. Feature Engineering ---")
# Tạo biến 'CarAge' (Tuổi xe) thay vì dùng 'year' (Năm SX)
current_year = 2025
df_clean['CarAge'] = current_year - df_clean['year']
```

```
print("Đã tạo đặc trưng mới: 'CarAge'")

# 4.2. Mã hóa biến phân loại (Encoding)
print("\n--- 4.2. Mã hóa dữ liệu ---")
# One-Hot Encoding cho các biến ít giá trị (Transmission, FuelType)
df_final = pd.get_dummies(df_clean, columns=['transmission', 'fuelType'], drop_first=True)

# Label Encoding cho các biến nhiều giá trị (Make, Model) để dùng cho CatBoost/Tree-based models
# Lưu ý: Nếu dùng Linear Regression thuần, nên dùng Target Encoding thay vì Label Encoding
le = LabelEncoder()
df_final['Manufacturer_Code'] = le.fit_transform(df_final['Manufacturer'])
df_final['Model_Code'] = le.fit_transform(df_final['model'])

# 4.3. Chuẩn hóa dữ liệu (Scaling)
# Chỉ chuẩn hóa các biến số liên tục để mô hình hội tụ nhanh hơn (quan trọng với Linear/KNN/Neural Net)
scaler = MinMaxScaler()
cols_to_scale = ['mileage', 'tax', 'mpg', 'engineSize', 'CarAge']
df_final[cols_to_scale] = scaler.fit_transform(df_final[cols_to_scale])

print("\n--- 5. DỮ LIỆU MẪU SAU KHI TIỀN XỬ LÝ ---")
display(df_final.head())
# --- Bổ sung: Vẽ Ma trận tương quan ---
print("\n--- Vẽ Ma trận tương quan (Correlation Matrix) ---")

# Chỉ lấy các cột số để tính tương quan
df_corr = df_final.select_dtypes(include=[np.number])

# Tính ma trận tương quan
corr_matrix = df_corr.corr()

# Vẽ Heatmap
plt.figure(figsize=(12, 10))
# cmap='coolwarm': Màu đỏ là tương quan dương, xanh là âm
# annot=True: Hiển thị số
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', linewidths=0.5)
plt.title('Ma trận hệ số tương quan (Correlation Heatmap)')
plt.tight_layout()

# Lưu ảnh
plt.savefig('correlation_heatmap.png')
print("Đã lưu ảnh: correlation_heatmap.png")
plt.show()
# ==============================================================================
# 5. LƯU FILE KẾT QUẢ
# ==============================================================================
df_final.to_csv('CarsData_Processed.csv', index=False)
print("Đã lưu file: CarsData_Processed.csv")
```

✅ Đã tải dữ liệu thành công!
Kích thước bộ dữ liệu: 97712 dòng, 10 cột

--- 2.1. Xem trước dữ liệu ---

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize	Manufacturer
0	I10	2017	7495	Manual	11630	Petrol	145	60.1	1.0	hyundi
1	Polo	2017	10989	Manual	9200	Petrol	145	58.9	1.0	volkswagen
2	2 Series	2019	27990	Semi-Auto	1614	Diesel	145	49.6	2.0	BMW
3	Yeti Outdoor	2017	12495	Manual	30960	Diesel	150	62.8	2.0	skoda
4	Fiesta	2017	7999	Manual	19353	Petrol	125	54.3	1.2	ford

--- 2.2. Thông tin kiểu dữ liệu và giá trị thiếu ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 97712 entries, 0 to 97711
Data columns (total 10 columns):
 #   Column   Non-Null Count Dtype
--- ------   -------------- -----
 0   model   97712 non-null object
 1   year   97712 non-null int64
 2   price   97712 non-null int64
 3   transmission 97712 non-null object
 4   mileage   97712 non-null int64
 5   fuelType   97712 non-null object
 6   tax   97712 non-null int64
 7   mpg   97712 non-null float64
 8   engineSize   97712 non-null float64
 9   Manufacturer 97712 non-null object
dtypes: float64(2), int64(4), object(4)
memory usage: 7.5+ MB
None

--- 2.3. Thống kê mô tả (Biến số) ---

	year	price	mileage	tax	mpg	engineSize
count	97712.000000	97712.000000	97712.000000	97712.000000	97712.000000	97712.000000
mean	2017.066502	16773.487555	23219.475499	120.142408	55.205623	1.664913
std	2.118661	9868.552222	21060.882301	63.357250	16.181659	0.558574
min	1970.000000	450.000000	1.000000	0.000000	0.300000	0.000000
25%	2016.000000	9999.000000	7673.000000	125.000000	47.100000	1.200000
50%	2017.000000	14470.000000	17682.500000	145.000000	54.300000	1.600000
75%	2019.000000	20750.000000	32500.000000	145.000000	62.800000	2.000000
max	2024.000000	159999.000000	323000.000000	580.000000	470.800000	6.600000

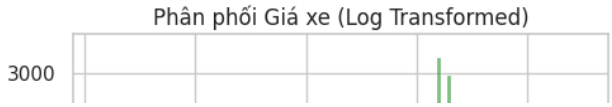
--- 2.4. Thống kê biến phân loại ---

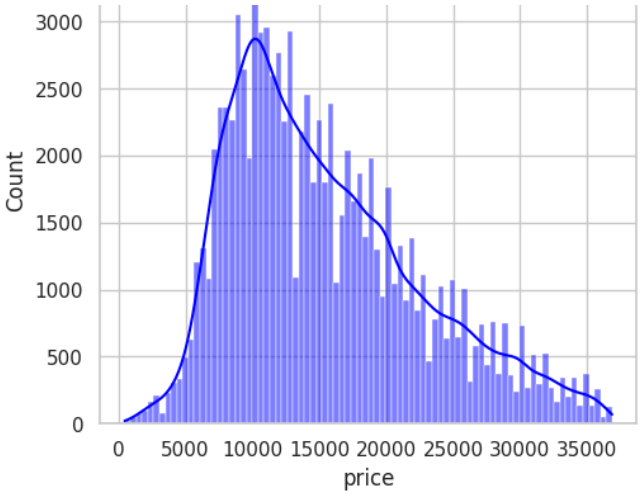
	model	transmission	fuelType	Manufacturer
count	97712	97712	97712	97712
unique	196	4	5	9
top	Fiesta	Manual	Petrol	ford
freq	6509	55502	53982	17811

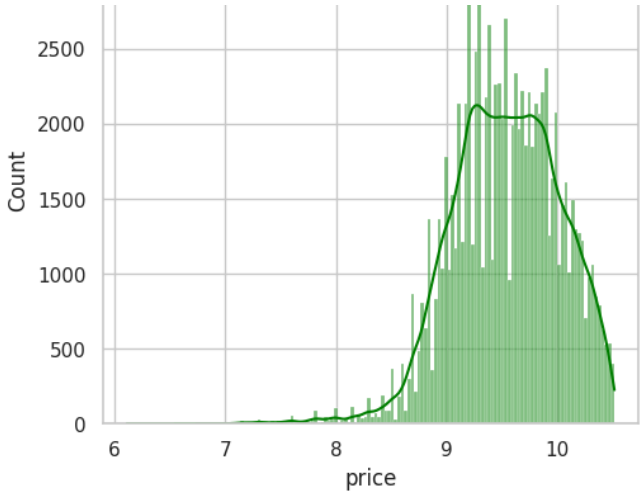
--- 3.1. Xử lý Missing Values ---
Đã xử lý xong giá trị thiếu.

--- 3.2. Xử lý Outliers (Biến Price) ---
Số lượng bản ghi trước khi lọc: 97712
Số lượng bản ghi sau khi lọc IQR: 93887
Đã loại bỏ 3825 bản ghi nhiễu.

--- 3.3. Kiểm tra phân phối Price ---



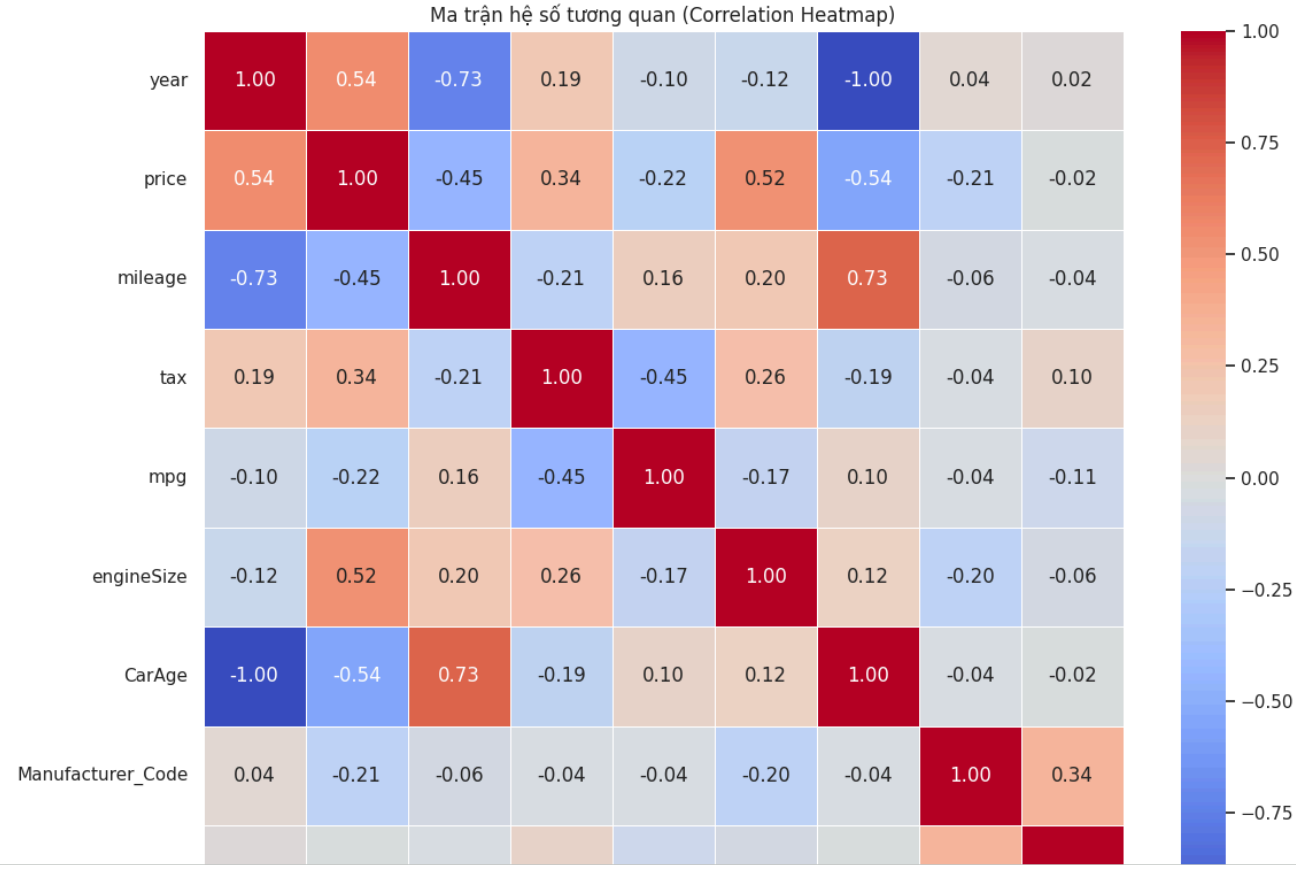


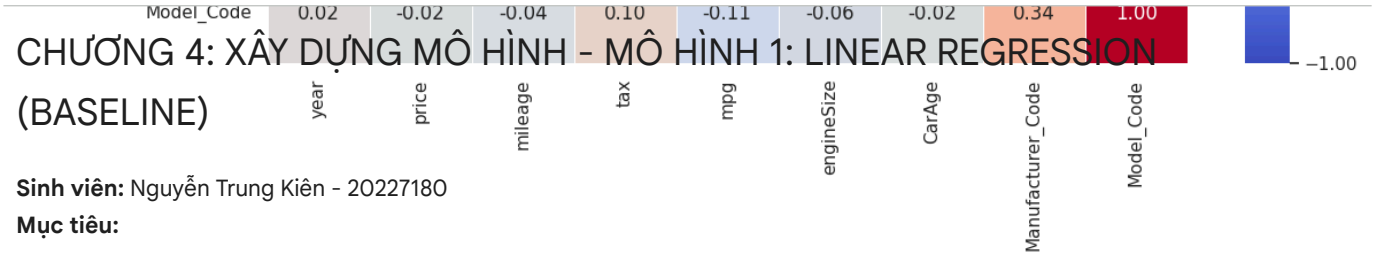



```
--- 4.1. Feature Engineering ---
Đã tạo đặc trưng mới: 'CarAge'

--- 4.2. Mã hóa dữ liệu ---

--- 5. DỮ LIỆU MẪU SAU KHI TIỀN XỬ LÝ ---
```

	model	year	price	mileage	tax	mpg	engineSize	Manufacturer	CarAge	transmission_Manual	transmissi
0	I10	2017	7495	0.036003	0.250000	0.127099	0.158730	hyundi	0.129630	True	
1	Polo	2017	10989	0.028480	0.250000	0.124548	0.158730	volkswagen	0.129630	True	
2	2 Series	2019	27990	0.004994	0.250000	0.104782	0.317460	BMW	0.092593	False	
3	Yeti Outdoor	2017	12495	0.095849	0.258621	0.132837	0.317460	skoda	0.129630	True	
4	Fiesta	2017	7999	0.059913	0.215517	0.114772	0.190476	ford	0.129630	True	

```
--- Vẽ Ma trận tương quan (Correlation Matrix) ---
Đã lưu ảnh: correlation_heatmap.png
```





CHƯƠNG 4: XÂY DỰNG MÔ HÌNH - MÔ HÌNH 1: LINEAR REGRESSION (BASELINE)

Sinh viên: Nguyễn Trung Kiên - 20227180

Mục tiêu:

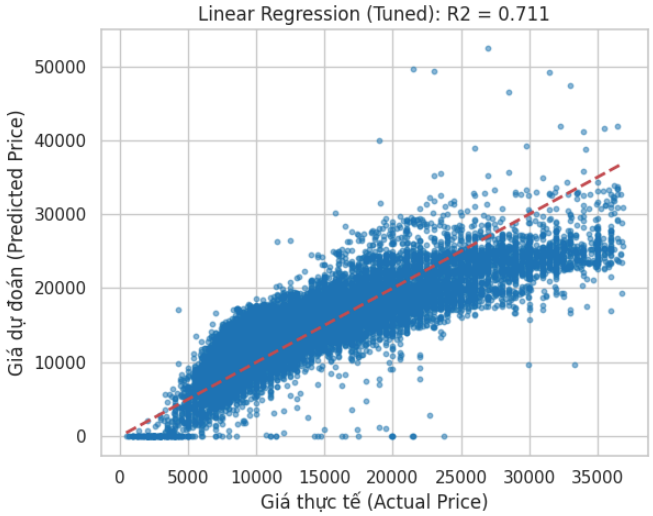
1. Huấn luyện mô hình Hồi quy tuyến tính nguyên bản (không Log Transform, không Feature Eng. phức tạp).
2. Tìm kiếm siêu tham số tối ưu (Hyperparameter Tuning) bằng GridSearchCV.
3. Đánh giá hiệu năng toàn diện với bộ chỉ số đầy đủ và trực quan hóa kết quả.

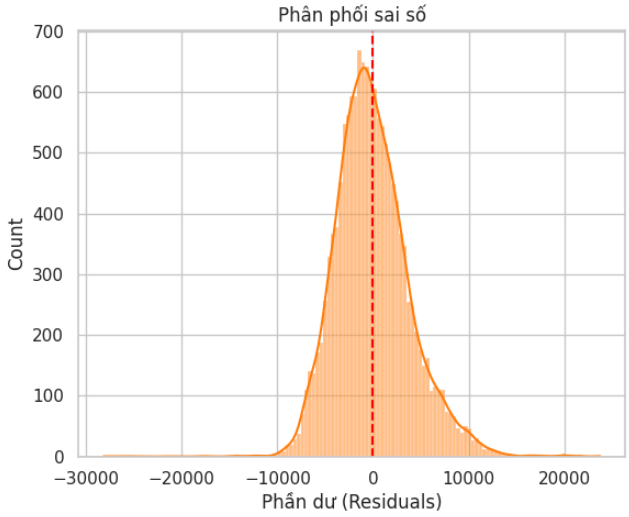
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, median_absolute_error, mean_absolute_pe
import joblib
import time

# Cấu hình giao diện
sns.set(style="whitegrid")
plt.rcParams['figure.figsize'] = (10, 6)

# ==============================================================================
# 1. TẢI DỮ LIỆU
# ==============================================================================
try:
    # Dữ liệu đã được làm sạch, encode và scale ở Chương 2
    df = pd.read_csv('CarsData_Processed.csv')
    print("✅ Đã tải dữ liệu 'CarsData_Processed.csv' thành công!")
except FileNotFoundError:
    print("❌ Lỗi: Không tìm thấy file 'CarsData_Processed.csv'. Hãy chạy code Chương 2 trước.")
    raise SystemExit

# ==============================================================================
# 2. CHIA TẬP DỮ LIỆU (70-15-15)
# ==============================================================================
print("\n--- 2. Thiết lập dữ liệu ---")

# Đảm bảo chỉ dùng các cột số (đã được encode)
# Linear Regression nguyên bản không chạy được với string, nên ta chỉ lấy numeric
df_numeric = df.select_dtypes(include=[np.number])

# Loại bỏ biến mục tiêu 'price' khỏi tập đặc trưng
# Lưu ý: Ở đây ta KHÔNG Log Transform biến price để giữ nguyên bản chất dữ liệu gốc
X = df_numeric.drop(columns=['price'])
y = df_numeric['price']

print(f"Số lượng đặc trưng (Features): {X.shape[1]}")
print(f"Danh sách đặc trưng: {X.columns.tolist()}")

# Chia tập dữ liệu: Train (70%) - Validation (15%) - Test (15%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

print(f"Train set: {X_train.shape[0]} bản ghi")
print(f"Val set:   {X_val.shape[0]} bản ghi")
print(f"Test set:  {X_test.shape[0]} bản ghi")

# ==============================================================================
# 3. HUẤN LUYỆN & TÌM SIÊU THAM SỐ (GRID SEARCH)
# ==============================================================================
print("\n--- 3. Huấn luyện Linear Regression (Tuning) ---")

start_train_time = time.time()
```

```
# Định nghĩa không gian tham số cho Linear Regression
# fit_intercept: Có tính hệ số chặn hay không (True/False)
# positive: Có ép buộc các hệ số phải dương hay không (True/False)
param_grid = {
    'fit_intercept': [True, False],
    'positive': [True, False]
}

# Khởi tạo mô hình và GridSearch
lr = LinearRegression()
grid_search = GridSearchCV(estimator=lr, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)

# Huấn luyện trên tập Train (kết hợp Cross-Validation nội bộ)
grid_search.fit(X_train, y_train)

end_train_time = time.time()
training_time = end_train_time - start_train_time

print("✅ Huấn luyện và Tuning hoàn tất.")
print(f"Thời gian huấn luyện: {training_time:.4f} giây")
print(f"Tham số tối ưu (Best Params): {grid_search.best_params_}")

# Lấy mô hình tốt nhất
best_model = grid_search.best_estimator_
if hasattr(best_model, 'intercept_'):
    print(f"Hệ số chặn (Intercept): {best_model.intercept_:.2f}")
else:
    print("Mô hình không sử dụng hệ số chặn.")

# ==============================================================================
# 4. ĐÁNH GIÁ MÔ HÌNH (TOÀN DIỆN)
# ==============================================================================
print("\n--- 4. Đánh giá trên tập Test ---")

start_pred_time = time.time()
y_pred = best_model.predict(X_test)
end_pred_time = time.time()
inference_time = (end_pred_time - start_pred_time) / len(X_test) # Thời gian suy diễn trung bình per sample

# Xử lý giá trị âm (nếu có, dù hiếm khi dùng positive=True)
y_pred = np.maximum(y_pred, 0)

# Tính các chỉ số đánh giá đầy đủ
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
medae = median_absolute_error(y_test, y_pred) # Sai số trung vị (ít nhạy cảm với outlier hơn MAE)
mape = mean_absolute_percentage_error(y_test, y_pred)

print(f"MAE  (Mean Absolute Error):        {mae:,.2f}")
print(f"RMSE (Root Mean Squared Error):    {rmse:,.2f}")
print(f"MedAE (Median Absolute Error):     {medae:,.2f}")
print(f"MAPE (Mean Abs. Percentage Error): {mape:.2%}")
print(f"R² Score:                          {r2:.4f}")
print(f"Inference Time (per sample):       {inference_time:.6f} s")

# ==============================================================================
# 5. TRỰC QUAN HÓA KẾT QUẢ
# ==============================================================================
print("\n--- 5. Trực quan hóa ---")

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

# 5.1. Scatter Plot: Thực tế vs Dự đoán
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred, alpha=0.5, color='#1f77b4', s=10)
# Vẽ đường chéo y=x (Dự đoán hoàn hảo)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', lw=2)
plt.xlabel('Giá thực tế (Actual Price)')
plt.ylabel('Giá dự đoán (Predicted Price)')
plt.title(f'Linear Regression (Tuned): R2 = {r2:.3f}')
```

```
# 5.2. Residual Plot: Phân phối phần dư
residuals = y_test - y_pred
plt.subplot(1, 2, 2)
sns.histplot(residuals, kde=True, color='#ff7f0e')
plt.xlabel('Phần dư (Residuals)')
plt.title('Phân phối sai số')
plt.axvline(x=0, color='red', linestyle='--')

plt.tight_layout()
plt.show()

# 5.3. Phân tích lỗi cực đoan (Top Errors)
print("\n--- Top 5 trường hợp sai lệch lớn nhất ---")
df_compare = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df_compare['Error'] = df_compare['Actual'] - df_compare['Predicted']
df_compare['Abs_Error'] = df_compare['Error'].abs()
display(df_compare.sort_values(by='Abs_Error', ascending=False).head(5))

# ================================================================================
# 6. LƯU KẾT QUẢ
# ================================================================================
# Lưu mô hình
joblib.dump(best_model, 'linear_regression_baseline.pkl')
print("\nĐã lưu mô hình: 'linear_regression_baseline.pkl'")

# Lưu kết quả metrics để so sánh sau này
results = pd.DataFrame({
    'Model': ['Linear Regression'],
    'Params': [str(grid_search.best_params_)],
    'MAE': [mae],
    'RMSE': [rmse],
    'MedAE': [medae],
    'MAPE': [mape],
    'R2': [r2],
    'Train_Time': [training_time],
    'Inference_Time': [inference_time]
})
results.to_csv('model_1_results.csv', index=False)
print("Đã lưu kết quả đánh giá: 'model_1_results.csv'")
```

```
✅ Đã tải dữ liệu 'CarsData_Processed.csv' thành công!

--- 2. Thiết lập dữ liệu ---
Số lượng đặc trưng (Features): 8
Danh sách đặc trưng: ['year', 'mileage', 'tax', 'mpg', 'engineSize', 'CarAge', 'Manufacturer_Code', 'Model_Code']
Train set: 65720 bản ghi
Val set:   14083 bản ghi
Test set:  14084 bản ghi

--- 3. Huấn luyện Linear Regression (Tuning) ---
✅ Huấn luyện và Tuning hoàn tất.
Thời gian huấn luyện: 3.4508 giây
Tham số tối ưu (Best Params): {'fit_intercept': False, 'positive': False}
Hệ số chặn (Intercept): 0.00

--- 4. Đánh giá trên tập Test ---
MAE  (Mean Absolute Error):        2,983.33
RMSE (Root Mean Squared Error):    3,865.78
MedAE (Median Absolute Error):     2,441.12
MAPE (Mean Abs. Percentage Error): 22.59%
R² Score:                          0.7114
Inference Time (per sample):       0.000000 s

--- 5. Trực quan hóa ---
```






```
--- Top 5 trường hợp sai lệch lớn nhất ---
```

	Actual	Predicted	Error	Abs_Error
61697	21495	49677.047930	-28182.047930	28182.047930
74665	23000	49355.017983	-26355.017983	26355.017983
58815	26998	52398.472547	-25400.472547	25400.472547
53522	23751	0.000000	23751.000000	23751.000000
40666	33333	9683.804193	23649.195807	23649.195807

```
Đã lưu mô hình: 'linear_regression_baseline.pkl'
Đã lưu kết quả đánh giá: 'model_1_results.csv'
```

CHƯƠNG 4: XÂY DỰNG MÔ HÌNH - MÔ HÌNH 2: RIDGE REGRESSION

Sinh viên: Nguyễn Trung Kiên - 20227180
Mục tiêu:

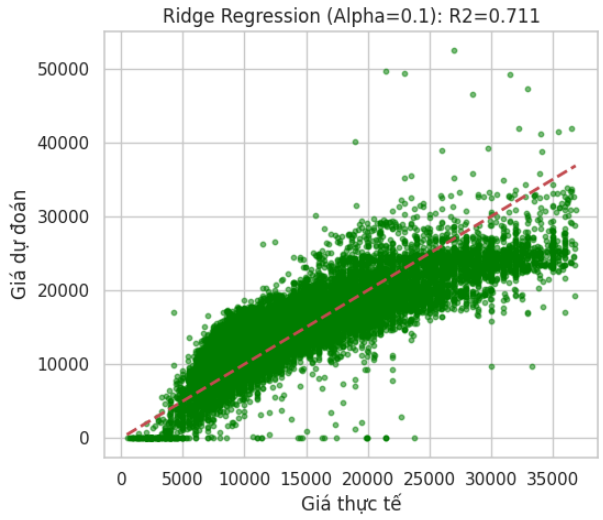
1. Huấn luyện mô hình Ridge Regression (Linear Regression + L2 Regularization).
2. Tối ưu hóa siêu tham số Alpha bằng GridSearchCV.
3. So sánh hiệu năng với Baseline (Linear Regression). """

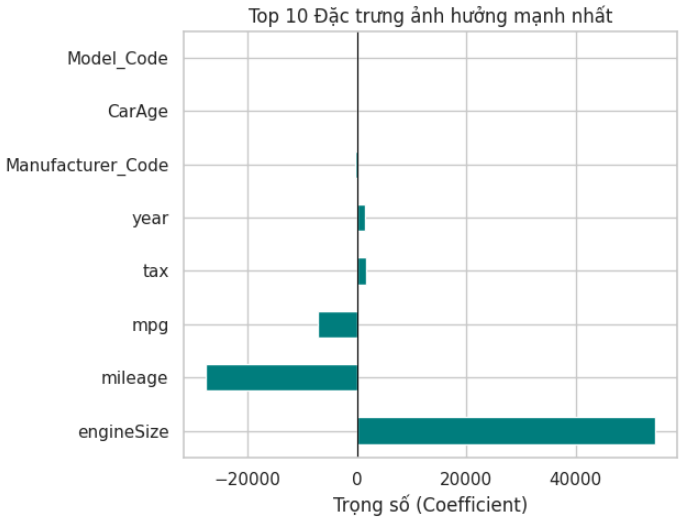
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, median_absolute_error, mean_absolute_pe
import joblib
import time

# Cấu hình giao diện
sns.set(style="whitegrid")
plt.rcParams['figure.figsize'] = (10, 6)

# ==============================================================================
# 1. TẢI DỮ LIỆU
# ==============================================================================
try:
    df = pd.read_csv('CarsData_Processed.csv')
    print("✅ Đã tải dữ liệu 'CarsData_Processed.csv' thành công!")
except FileNotFoundError:
    print("❌ Lỗi: Không tìm thấy file 'CarsData_Processed.csv'. Hãy chạy code Chương 2 trước.")
    raise SystemExit

# ==============================================================================
# 2. CHIA TẬP DỮ LIỆU (70-15-15)
# ==============================================================================
print("\n--- 2. Thiết lập dữ liệu ---")

# Chỉ lấy dữ liệu số (tương tự Model 1 để so sánh công bằng)
df_numeric = df.select_dtypes(include=[np.number])

X = df_numeric.drop(columns=['price'])
y = df_numeric['price']

print(f"Số lượng đặc trưng (Features): {X.shape[1]}")

# Chia tập dữ liệu (Giữ nguyên random_state=42 để đồng bộ với Model 1)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

print(f"Train set: {X_train.shape[0]} bản ghi")
print(f"Val set:   {X_val.shape[0]} bản ghi")
print(f"Test set:  {X_test.shape[0]} bản ghi")

# ==============================================================================
# 3. HUẤN LUYỆN & TÌM SIÊU THAM SỐ (GRID SEARCH)
# ==============================================================================
print("\n--- 3. Huấn luyện Ridge Regression (Tuning) ---")

start_train_time = time.time()

# Định nghĩa không gian tham số Alpha
# Alpha = 0: Tương đương Linear Regression
# Alpha càng lớn: Phạt càng mạnh, mô hình càng đơn giản (giảm variance, tăng bias)
param_grid = {
    'alpha': [0.01, 0.1, 1.0, 10.0, 100.0, 1000.0],
    'solver': ['auto', 'svd', 'cholesky', 'lsqr'] # Thử nghiệm các thuật toán giải khác nhau
}

ridge = Ridge(random_state=42)
grid_search = GridSearchCV(estimator=ridge, param_grid=param_grid, cv=5,
                           scoring='neg_mean_squared_error', n_jobs=-1)

# Huấn luyện
grid_search.fit(X_train, y_train)
```

```
end_train_time = time.time()
training_time = end_train_time - start_train_time

print("✅ Huấn luyện và Tuning hoàn tất.")
print(f"Thời gian huấn luyện: {training_time:.4f} giây")
print(f"Tham số tối ưu (Best Params): {grid_search.best_params_}")

# Lấy mô hình tốt nhất
best_model = grid_search.best_estimator_
print(f"Hệ số chặn (Intercept): {best_model.intercept_:.2f}")

# ================================================================================
# 4. ĐÁNH GIÁ MÔ HÌNH (TOÀN DIỆN)
# ================================================================================
print("\n--- 4. Đánh giá trên tập Test ---")

start_pred_time = time.time()
y_pred = best_model.predict(X_test)
end_pred_time = time.time()
inference_time = (end_pred_time - start_pred_time) / len(X_test)

# Xử lý giá trị âm
y_pred = np.maximum(y_pred, 0)

# Tính chỉ số
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
medae = median_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

print(f"MAE  (Mean Absolute Error):        {mae:,.2f}")
print(f"RMSE (Root Mean Squared Error):    {rmse:,.2f}")
print(f"MedAE (Median Absolute Error):     {medae:,.2f}")
print(f"MAPE (Mean Abs. Percentage Error): {mape:.2%}")
print(f"R² Score:                          {r2:.4f}")
print(f"Inference Time (per sample):       {inference_time:.6f} s")

# ================================================================================
# 5. SO SÁNH VỚI BASELINE (MODEL 1)
# ================================================================================
# Giả sử kết quả Model 1: R2 = 0.7114, RMSE = 3865.78
baseline_r2 = 0.7114
improvement = r2 - baseline_r2
status = "Cải thiện" if improvement > 0 else "Không đổi/Kém hơn"
print(f"\n-> So với Model 1 (Linear Regression): {status} ({improvement:+.4f})")

# ================================================================================
# 6. TRỰC QUAN HÓA & PHÂN TÍCH
# ================================================================================
print("\n--- 6. Trực quan hóa ---")

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

# 6.1. Scatter Plot
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred, alpha=0.5, color='green', s=10)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', lw=2)
plt.xlabel('Giá thực tế')
plt.ylabel('Giá dự đoán')
plt.title(f'Ridge Regression (Alpha={grid_search.best_params_["alpha"]}): R2={r2:.3f}')

# 6.2. Phân tích hệ số (Top Features)
plt.subplot(1, 2, 2)
coefs = pd.Series(best_model.coef_, index=X.columns)
# Lấy Top 10 đặc trưng có trọng số tuyệt đối lớn nhất
top_coefs = coefs.abs().sort_values(ascending=False).head(10)
# Vẽ giá trị thực (có âm dương) của top features này
coefs[top_coefs.index].plot(kind='barh', color='teal')
plt.title('Top 10 Đặc trưng ảnh hưởng mạnh nhất')
plt.xlabel('Trọng số (Coefficient)')
plt.axvline(x=0, color='black', linewidth=0.8)
```

```
plt.tight_layout()
plt.show()

# ==============================================================================
# 7. LƯU KẾT QUẢ
# ==============================================================================
joblib.dump(best_model, 'ridge_regression_model.pkl')
print("\nĐã lưu mô hình: 'ridge_regression_model.pkl'")

results = pd.DataFrame({
    'Model': ['Ridge Regression'],
    'Params': [str(grid_search.best_params_)],
    'MAE': [mae], 'RMSE': [rmse], 'MedAE': [medae], 'MAPE': [mape], 'R2': [r2],
    'Train_Time': [training_time], 'Inference_Time': [inference_time]
})
results.to_csv('model_2_results.csv', index=False)
print("Đã lưu kết quả đánh giá: 'model_2_results.csv'")
```

```
✅ Đã tải dữ liệu 'CarsData_Processed.csv' thành công!

--- 2. Thiết lập dữ liệu ---
Số lượng đặc trưng (Features): 8
Train set: 65720 bản ghi
Val set:   14083 bản ghi
Test set:  14084 bản ghi

--- 3. Huấn luyện Ridge Regression (Tuning) ---
✅ Huấn luyện và Tuning hoàn tất.
Thời gian huấn luyện: 4.9134 giây
Tham số tối ưu (Best Params): {'alpha': 0.1, 'solver': 'lsqr'}
Hệ số chặn (Intercept): -2890675.58

--- 4. Đánh giá trên tập Test ---
MAE  (Mean Absolute Error):        2,985.47
RMSE (Root Mean Squared Error):    3,867.28
MedAE (Median Absolute Error):     2,441.68
MAPE (Mean Abs. Percentage Error): 22.61%
R² Score:                          0.7112
Inference Time (per sample):       0.000000 s

-> So với Model 1 (Linear Regression): Không đổi/Kém hơn (-0.0002)

--- 6. Trực quan hóa ---
```






```
Đã lưu mô hình: 'ridge_regression_model.pkl'
Đã lưu kết quả đánh giá: 'model_2_results.csv'
```

CHƯƠNG 4: XÂY DỰNG MÔ HÌNH - MÔ HÌNH 3: LASSO REGRESSION (L1 REGULARIZATION)

Sinh viên: Nguyễn Trung Kiên - 20227180

Mục tiêu:

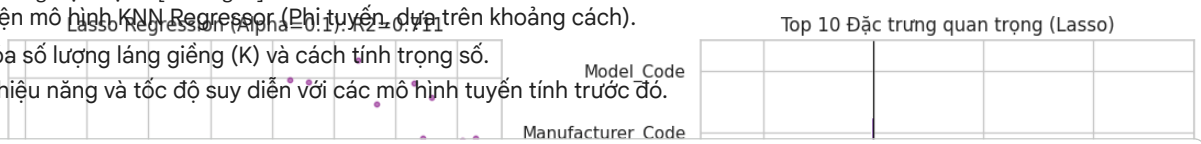
1. Huấn luyện mô hình Lasso Regression.
2. Tối ưu hóa siêu tham số Alpha bằng GridSearchCV.
3. Phân tích khả năng chọn lọc đặc trưng (Feature Selection) của Lasso.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, median_absolute_error, mean_absolute_per
import joblib
import time

# Cấu hình giao diện
sns.set(style="whitegrid")
plt.rcParams['figure.figsize'] = (10, 6)

# ==============================================================================
# 1. TẢI DỮ LIỆU
# ==============================================================================
try:
    df = pd.read_csv('CarsData_Processed.csv')
    print("✅ Đã tải dữ liệu 'CarsData_Processed.csv' thành công!")
except FileNotFoundError:
    print("❌ Lỗi: Không tìm thấy file 'CarsData_Processed.csv'. Hãy chạy code Chương 2 trước.")
    raise SystemExit

# ==============================================================================
# 2. CHIA TẬP DỮ LIỆU (70-15-15)
# ==============================================================================
print("\n--- 2. Thiết lập dữ liệu ---")

# Chỉ lấy dữ liệu số
df_numeric = df.select_dtypes(include=[np.number])

X = df_numeric.drop(columns=['price'])
y = df_numeric['price']

print(f"Số lượng đặc trưng (Features): {X.shape[1]}")

# Chia tập dữ liệu (Giữ nguyên random_state=42)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

print(f"Train set: {X_train.shape[0]} bản ghi")
print(f"Val set:   {X_val.shape[0]} bản ghi")
print(f"Test set:  {X_test.shape[0]} bản ghi")

# ==============================================================================
# 3. HUẤN LUYỆN & TÌM SIÊU THAM SỐ (GRID SEARCH)
# ==============================================================================
print("\n--- 3. Huấn luyện Lasso Regression (Tuning) ---")

start_train_time = time.time()

# Định nghĩa không gian tham số Alpha
# Lasso nhạy cảm với alpha hơn Ridge. Alpha quá lớn sẽ ép hết hệ số về 0 (underfitting).
param_grid = {
    'alpha': [0.001, 0.01, 0.1, 1.0, 10.0, 100.0],
    'selection': ['cyclic', 'random'] # Thuật toán cập nhật hệ số
}
```

```
}

# Tăng max_iter để đảm bảo hội tụ
lasso = Lasso(random_state=42, max_iter=10000)
grid_search = GridSearchCV(estimator=lasso, param_grid=param_grid, cv=5,
                           scoring='neg_mean_squared_error', n_jobs=-1)

# Huấn luyện
grid_search.fit(X_train, y_train)

end_train_time = time.time()
training_time = end_train_time - start_train_time

print("✅ Huấn luyện và Tuning hoàn tất.")
print(f"Thời gian huấn luyện: {training_time:.4f} giây")
print(f"Tham số tối ưu (Best Params): {grid_search.best_params_}")

# Lấy mô hình tốt nhất
best_model = grid_search.best_estimator_
print(f"Hệ số chặn (Intercept): {best_model.intercept_:.2f}")

# ==============================================================================
# 4. ĐÁNH GIÁ MÔ HÌNH (TOÀN DIỆN)
# ==============================================================================
print("\n--- 4. Đánh giá trên tập Test ---")

start_pred_time = time.time()
y_pred = best_model.predict(X_test)
end_pred_time = time.time()
inference_time = (end_pred_time - start_pred_time) / len(X_test)

# Xử lý giá trị âm
y_pred = np.maximum(y_pred, 0)

# Tính chỉ số
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
medae = median_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

print(f"MAE  (Mean Absolute Error):        {mae:,.2f}")
print(f"RMSE (Root Mean Squared Error):    {rmse:,.2f}")
print(f"MedAE (Median Absolute Error):     {medae:,.2f}")
print(f"MAPE (Mean Abs. Percentage Error): {mape:.2%}")
print(f"R² Score:                          {r2:.4f}")
print(f"Inference Time (per sample):       {inference_time:.6f} s")

# ==============================================================================
# 5. SO SÁNH VỚI BASELINE & RIDGE
# ==============================================================================
# Giả sử kết quả Model 1: R2 = 0.7114
# Giả sử kết quả Model 2: R2 = 0.7112
baseline_r2 = 0.7114
ridge_r2 = 0.7112 # (Kết quả chạy trước)

improvement_vs_linear = r2 - baseline_r2
print(f"\n-> So với Linear (M1): {'Cải thiện' if improvement_vs_linear > 0 else 'Không đổi/Kém hơn'} ({improvement_vs_l

# ==============================================================================
# 6. TRỰC QUAN HÓA & PHÂN TÍCH ĐẶC TRƯNG (FEATURE SELECTION)
# ==============================================================================
print("\n--- 6. Trực quan hóa & Feature Selection ---")

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

# 6.1. Scatter Plot
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred, alpha=0.5, color='purple', s=10)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', lw=2)
plt.xlabel('Giá thực tế')
plt.ylabel('Giá dự đoán')
plt.title(f'Lasso Regression (Alpha={grid_search.best_params_["alpha"]}): R2={r2:.3f}')
```

```
# 6.2. Phân tích hệ số (Kiểm tra biến bị loại bỏ)
coefs = pd.Series(best_model.coef_, index=X.columns)
n_removed = sum(coefs == 0)
print(f"Số lượng đặc trưng bị Lasso loại bỏ (Hệ số = 0): {n_removed}/{len(X.columns)}")

if n_removed > 0:
    print(f"Các đặc trưng bị loại: {coefs[coefs == 0].index.tolist()}")

plt.subplot(1, 2, 2)
# Lấy Top 10 đặc trưng có trọng số tuyệt đối lớn nhất (khác 0)
top_coefs = coefs[coefs != 0].abs().sort_values(ascending=False).head(10)
coefs[top_coefs.index].plot(kind='barh', color='indigo')
plt.title('Top 10 Đặc trưng quan trọng (Lasso)')
plt.xlabel('Trọng số (Coefficient)')
plt.axvline(x=0, color='black', linewidth=0.8)

plt.tight_layout()
plt.show()

# ==============================================================================
# 7. LƯU KẾT QUẢ
# ==============================================================================
joblib.dump(best_model, 'lasso_regression_model.pkl')
print("\nĐã lưu mô hình: 'lasso_regression_model.pkl'")

results = pd.DataFrame({
    'Model': ['Lasso Regression'],
    'Params': [str(grid_search.best_params_)],
    'MAE': [mae], 'RMSE': [rmse], 'MedAE': [medae], 'MAPE': [mape], 'R2': [r2],
    'Train_Time': [training_time], 'Inference_Time': [inference_time]
})
results.to_csv('model_3_results.csv', index=False)
print("Đã lưu kết quả đánh giá: 'model_3_results.csv'")
```

```
✅ Đã tải dữ liệu 'CarsData_Processed.csv' thành công!

--- 2. Thiết lập dữ liệu ---
Số lượng đặc trưng (Features): 8
Train set: 65720 bản ghi
Val set:   14083 bản ghi
Test set:  14084 bản ghi

--- 3. Huấn luyện Lasso Regression (Tuning) ---
✅ Huấn luyện và Tuning hoàn tất.
Thời gian huấn luyện: 30.2533 giây
Tham số tối ưu (Best Params): {'alpha': 0.1, 'selection': 'cyclic'}
Hệ số chặn (Intercept): -2897885.95

--- 4. Đánh giá trên tập Test ---
MAE  (Mean Absolute Error):        2,983.58
RMSE (Root Mean Squared Error):    3,865.90
MedAE (Median Absolute Error):     2,441.93
MAPE (Mean Abs. Percentage Error): 22.59%
R² Score:                          0.7114
Inference Time (per sample)        0.000000 s
-> So với Linear (M1): Cải thiện (+0.0000)

--- 6. Trực quan hóa & Feature Selection ---
Số lượng đặc trưng bị Lasso loại bỏ (Hệ số = 0): 1/8
Các đặc trưng bị loại: ['CarAge']
```



CHƯƠNG 4: XÂY DỰNG MÔ HÌNH - MÔ HÌNH 4: K-NEAREST NEIGHBORS (KNN)

Sinh viên: Nguyễn Trung Kiên - 20227180

Mục tiêu:

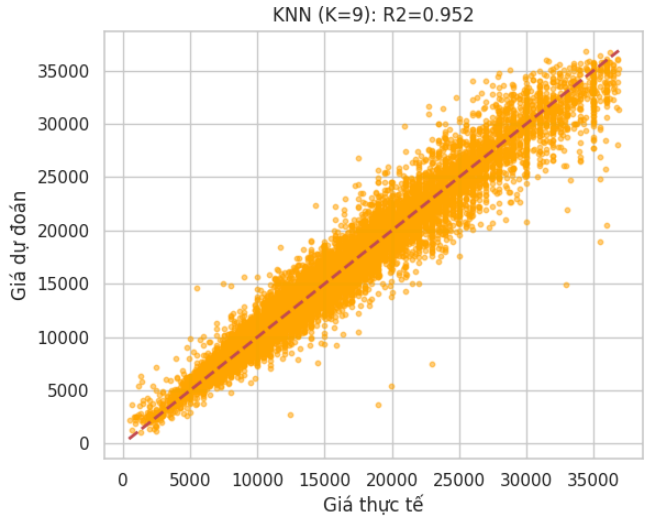
1. Huấn luyện mô hình KNN Regressor (Phi tuyến, dựa trên khoảng cách).
2. Tối ưu hóa số lượng láng giềng (K) và cách tính trọng số.
3. So sánh hiệu năng và tốc độ suy diễn với các mô hình tuyến tính trước đó.

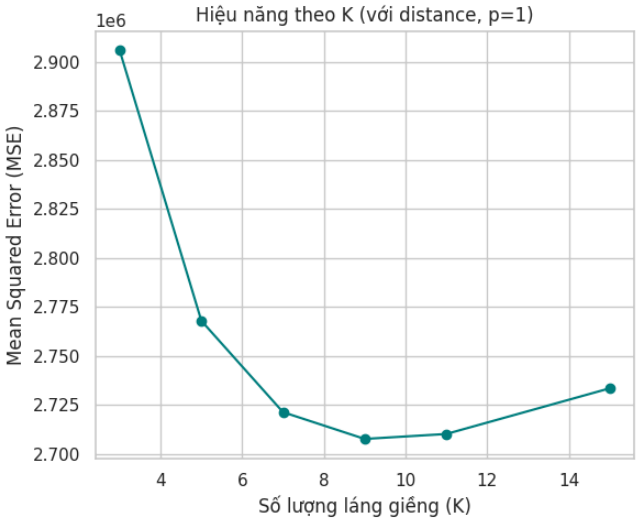
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, median_absolute_error, mean_absolute_pe
import joblib
import time

# Cấu hình giao diện
sns.set(style="whitegrid")
plt.rcParams['figure.figsize'] = (10, 6)

# ==============================================================================
# 1. TẢI DỮ LIỆU
# ==============================================================================
try:
    # Dữ liệu đã được Scaled (Rất quan trọng với KNN)
    df = pd.read_csv('CarsData_Processed.csv')
    print("✅ Đã tải dữ liệu 'CarsData_Processed.csv' thành công!")
except FileNotFoundError:
    print("❌ Lỗi: Không tìm thấy file. Hãy chạy code Chương 2 trước.")
    raise SystemExit

# ==============================================================================
# 2. CHIA TẬP DỮ LIỆU (70-15-15)
# ==============================================================================
print("\n--- 2. Thiết lập dữ liệu ---")

# Lấy các biến số (đã bao gồm các biến Code được scale)
df_numeric = df.select_dtypes(include=[np.number])
X = df_numeric.drop(columns=['price'])
y = df_numeric['price']

print(f"Số lượng đặc trưng: {X.shape[1]}")

# Chia tập dữ liệu (Giữ random_state=42 để so sánh công bằng)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

```
print(f"Train: {X_train.shape[0]} | Val: {X_val.shape[0]} | Test: {X_test.shape[0]}")

# ==============================================================================
# 3. HUẤN LUYỆN & TÌM SIÊU THAM SỐ (GRID SEARCH)
# ==============================================================================
print("\n--- 3. Huấn luyện KNN Regressor (Tuning) ---")
# KNN có thể chạy lâu với dữ liệu lớn, ta dùng n_jobs=-1 để tận dụng đa nhân CPU

start_train_time = time.time()

# Không gian tham số
# n_neighbors: Số lượng láng giềng (K)
# weights: 'uniform' (trọng số như nhau) hoặc 'distance' (gần đóng góp nhiều hơn)
# p: 1 (Manhattan - L1), 2 (Euclidean - L2)
param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11, 15],
    'weights': ['uniform', 'distance'],
    'p': [1, 2]
}

knn = KNeighborsRegressor(n_jobs=-1) # Sử dụng toàn bộ core CPU

print("Đang chạy Grid Search (Có thể mất 1-2 phút do KNN tính toán nặng)...")
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5,
                           scoring='neg_mean_squared_error', n_jobs=-1)

grid_search.fit(X_train, y_train)

end_train_time = time.time()
training_time = end_train_time - start_train_time

print("✅ Huấn luyện và Tuning hoàn tất.")
print(f"Thời gian huấn luyện: {training_time:.4f} giây")
print(f"Tham số tối ưu: {grid_search.best_params_}")

best_knn = grid_search.best_estimator_

# ==============================================================================
# 4. ĐÁNH GIÁ MÔ HÌNH (TOÀN DIỆN)
# ==============================================================================
print("\n--- 4. Đánh giá trên tập Test ---")

start_pred_time = time.time()
y_pred = best_knn.predict(X_test)
end_pred_time = time.time()
inference_time = (end_pred_time - start_pred_time) / len(X_test)

# Xử lý giá trị âm (KNN hiếm khi ra âm nếu dữ liệu train dương, nhưng cứ an toàn)
y_pred = np.maximum(y_pred, 0)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
medae = median_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

print(f"MAE:  {mae:,.2f}")
print(f"RMSE: {rmse:,.2f}")
print(f"MedAE: {medae:,.2f}")
print(f"MAPE: {mape:.2%}")
print(f"R² Score: {r2:.4f}")
print(f"Inference Time: {inference_time:.6f} s/sample")

# So sánh với Linear (Baseline R2 ~ 0.7114)
print(f"-> So với Linear: {'Cải thiện' if r2 > 0.7114 else 'Kém hơn'}")

# ==============================================================================
# 5. TRỰC QUAN HÓA
# ==============================================================================
print("\n--- 5. Trực quan hóa ---")

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

```
# 5.1. Scatter Plot
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred, alpha=0.5, color='orange', s=10)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', lw=2)
plt.xlabel('Giá thực tế')
plt.ylabel('Giá dự đoán')
plt.title(f'KNN (K={grid_search.best_params_["n_neighbors"]}): R2={r2:.3f}')

# 5.2. Biểu đồ hiệu năng theo K (Trích xuất từ GridSearch)
plt.subplot(1, 2, 2)
# Lấy kết quả Mean Test Score (Negative MSE)
results_df = pd.DataFrame(grid_search.cv_results_)
# Lọc lấy các tham số tốt nhất của weights và p
best_weight = grid_search.best_params_['weights']
best_p = grid_search.best_params_['p']
subset = results_df[(results_df['param_weights'] == best_weight) & (results_df['param_p'] == best_p)]

plt.plot(subset['param_n_neighbors'], -subset['mean_test_score'], marker='o', color='teal')
plt.xlabel('Số lượng láng giềng (K)')
plt.ylabel('Mean Squared Error (MSE)')
plt.title(f'Hiệu năng theo K (với {best_weight}, p={best_p})')
plt.grid(True)

plt.tight_layout()
plt.show()

# ==============================================================================
# 6. LƯU KẾT QUẢ
# ==============================================================================
joblib.dump(best_knn, 'knn_model.pkl')
print("\nĐã lưu mô hình: 'knn_model.pkl'")

results = pd.DataFrame({
    'Model': ['KNN Regressor'],
    'Params': [str(grid_search.best_params_)],
    'MAE': [mae], 'RMSE': [rmse], 'MedAE': [medae], 'MAPE': [mape], 'R2': [r2],
    'Train_Time': [training_time], 'Inference_Time': [inference_time]
})
results.to_csv('model_4_results.csv', index=False)
print("Đã lưu kết quả đánh giá: 'model_4_results.csv'")
```

```
✅ Đã tải dữ liệu 'CarsData_Processed.csv' thành công!

--- 2. Thiết lập dữ liệu ---
Số lượng đặc trưng: 8
Train: 65720 | Val: 14083 | Test: 14084

--- 3. Huấn luyện KNN Regressor (Tuning) ---
Đang chạy Grid Search (Có thể mất 1-2 phút do KNN tính toán nặng)...
✅ Huấn luyện và Tuning hoàn tất.
Thời gian huấn luyện: 47.5821 giây
Tham số tối ưu: {'n_neighbors': 9, 'p': 1, 'weights': 'distance'}

--- 4. Đánh giá trên tập Test ---
MAE:  1,087.07
RMSE: 1,571.32
MedAE: 756.07
MAPE: 7.81%
R² Score: 0.9523
Inference Time: 0.000041 s/sample
-> So với Linear: Cải thiện

--- 5. Trực quan hóa ---
```






```
Đã lưu mô hình: 'knn_model.pkl'
Đã lưu kết quả đánh giá: 'model_4_results.csv'
```

CHƯƠNG 4: XÂY DỰNG MÔ HÌNH - MÔ HÌNH 5: SUPPORT VECTOR REGRESSION (SVR)

Sinh viên: Nguyễn Trung Kiên - 20227180
Mục tiêu:

1. Huấn luyện mô hình SVR (Sử dụng Kernel Trick để xử lý phi tuyến).
2. Tinh chỉnh siêu tham số (C, epsilon, kernel) bằng GridSearchCV.
3. Đánh giá hiệu năng và so sánh với KNN (Model 4) và nhóm Tuyến tính. **Lưu ý:** SVR rất tốn tài nguyên tính toán với dữ liệu lớn (>10k mẫu).

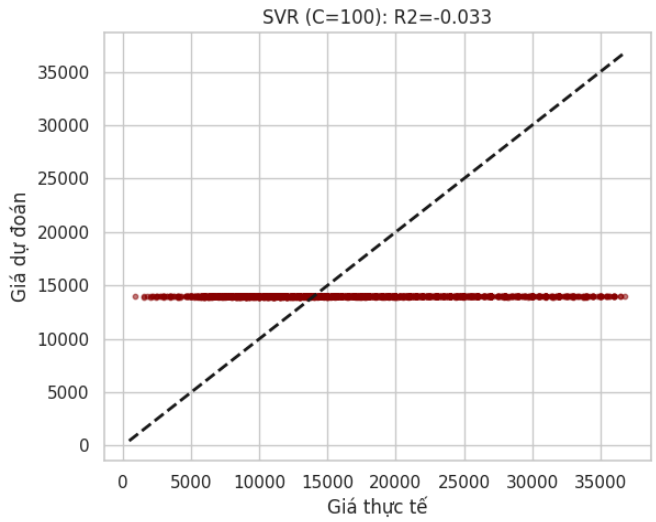
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
```

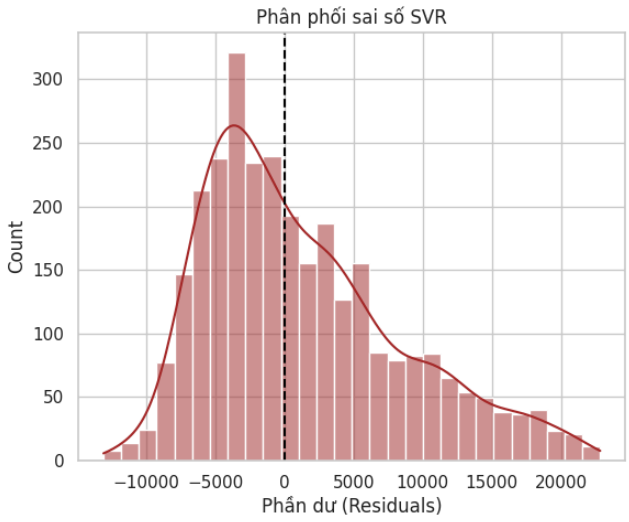
```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, median_absolute_error, mean_absolute_pe
import joblib
import time

# Cấu hình giao diện
sns.set(style="whitegrid")
plt.rcParams['figure.figsize'] = (10, 6)

# ==============================================================================
# 1. TẢI DỮ LIỆU
# ==============================================================================
try:
    # Dữ liệu đã được Scaled (BẮT BUỘC VỚI SVR)
    df = pd.read_csv('CarsData_Processed.csv')
    print("✅ Đã tải dữ liệu 'CarsData_Processed.csv' thành công!")
except FileNotFoundError:
    print("❌ Lỗi: Không tìm thấy file. Hãy chạy code Chương 2 trước.")
    raise SystemExit

# ==============================================================================
# 2. CHIA TẬP DỮ LIỆU (70-15-15) & LẤY MẪU (SAMPLING)
# ==============================================================================
print("\n--- 2. Thiết lập dữ liệu ---")

# Lấy các biến số
df_numeric = df.select_dtypes(include=[np.number])
X = df_numeric.drop(columns=['price'])
y = df_numeric['price']

# LƯU Ý QUAN TRỌNG:
# SVR có độ phức tạp tính toán khoảng O(N^3). Với N=97k, chạy GridSearch sẽ mất HÀNG GIỜ.
# Để khả thi trên Colab và trong khuôn khổ bài tập lớn, ta sẽ lấy mẫu ngẫu nhiên (Sampling)
# khoảng 10,000 - 20,000 bản ghi để huấn luyện SVR.
# Tuy nhiên, nếu bạn muốn chạy full, hãy comment dòng sampling lại (và chờ rất lâu).

SAMPLE_SIZE = 20000 # Lấy 20k mẫu để train SVR
if len(df) > SAMPLE_SIZE:
    print(f"⚠️ Dữ liệu quá lớn ({len(df)} dòng) cho SVR. Lấy mẫu ngẫu nhiên {SAMPLE_SIZE} dòng để train.")
    # Lấy mẫu có bảo toàn phân phối (stratified sampling khó với regression, nên dùng random)
    X_sample, _, y_sample, _ = train_test_split(X, y, train_size=SAMPLE_SIZE, random_state=42)
else:
    X_sample, y_sample = X, y

# Chia tập dữ liệu từ tập mẫu này
X_train, X_temp, y_train, y_temp = train_test_split(X_sample, y_sample, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

print(f"Train (Sampled): {X_train.shape[0]} | Val: {X_val.shape[0]} | Test: {X_test.shape[0]}")

# ==============================================================================
# 3. HUẤN LUYỆN & TÌM SIÊU THAM SỐ (GRID SEARCH)
# ==============================================================================
print("\n--- 3. Huấn luyện SVR (Tuning) ---")

start_train_time = time.time()

# Không gian tham số
# kernel: 'rbf' (Radial Basis Function) là mạnh nhất cho phi tuyến. 'linear' thì giống Linear Regression.
# C: Regularization parameter. C càng lớn -> Phạt lỗi càng nặng -> Dễ Overfitting.
# epsilon: Độ rộng của "ống" sai số cho phép.
param_grid = {
    'kernel': ['rbf'], # Tập trung vào RBF vì Linear đã làm ở Model 1
    'C': [1, 10, 100],
    'epsilon': [0.1, 0.2]
    # 'gamma': ['scale', 'auto'] # Có thể tune thêm gamma nếu cần
}

svr = SVR()

print("Đang chạy Grid Search cho SVR (Sẽ mất thời gian)...")
grid_search = GridSearchCV(estimator=svr, param_grid=param_grid, cv=3, # Giảm CV xuống 3 để nhanh hơn
```

```
                           scoring='neg_mean_squared_error', n_jobs=-1, verbose=2)

grid_search.fit(X_train, y_train)

end_train_time = time.time()
training_time = end_train_time - start_train_time

print("✅ Huấn luyện và Tuning hoàn tất.")
print(f"Thời gian huấn luyện: {training_time:.4f} giây")
print(f"Tham số tối ưu: {grid_search.best_params_}")

best_svr = grid_search.best_estimator_

# ==============================================================================
# 4. ĐÁNH GIÁ MÔ HÌNH (TOÀN DIỆN)
# ==============================================================================
print("\n--- 4. Đánh giá trên tập Test ---")

start_pred_time = time.time()
y_pred = best_svr.predict(X_test)
end_pred_time = time.time()
inference_time = (end_pred_time - start_pred_time) / len(X_test)

# Xử lý giá trị âm
y_pred = np.maximum(y_pred, 0)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
medae = median_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

print(f"MAE:  {mae:,.2f}")
print(f"RMSE: {rmse:,.2f}")
print(f"MedAE: {medae:,.2f}")
print(f"MAPE: {mape:.2%}")
print(f"R² Score: {r2:.4f}")
print(f"Inference Time: {inference_time:.6f} s/sample")

# So sánh với KNN (Model 4 - Giả định kết quả R2=0.9523)
print(f"-> So với KNN: {'Tương đương' if abs(r2 - 0.9523) < 0.01 else ('Kém hơn' if r2 < 0.9523 else 'Tốt hơn')}")

# ==============================================================================
# 5. TRỰC QUAN HÓA
# ==============================================================================
print("\n--- 5. Trực quan hóa ---")

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

# 5.1. Scatter Plot
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred, alpha=0.5, color='darkred', s=10)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=2)
plt.xlabel('Giá thực tế')
plt.ylabel('Giá dự đoán')
plt.title(f'SVR (C={grid_search.best_params_["C"]}): R2={r2:.3f}')

# 5.2. Residual Plot
residuals = y_test - y_pred
plt.subplot(1, 2, 2)
sns.histplot(residuals, kde=True, color='brown')
plt.xlabel('Phần dư (Residuals)')
plt.title('Phân phối sai số SVR')
plt.axvline(x=0, color='black', linestyle='--')

plt.tight_layout()
plt.show()

# ==============================================================================
# 6. LƯU KẾT QUẢ
# ==============================================================================
joblib.dump(best_svr, 'svr_model.pkl')
```

```
print("\nĐã lưu mô hình: 'svr_model.pkl'")

results = pd.DataFrame({
    'Model': ['SVR'],
    'Params': [str(grid_search.best_params_)],
    'MAE': [mae], 'RMSE': [rmse], 'MedAE': [medae], 'MAPE': [mape], 'R2': [r2],
    'Train_Time': [training_time], 'Inference_Time': [inference_time]
})
results.to_csv('model_5_results.csv', index=False)
print("Đã lưu kết quả đánh giá: 'model_5_results.csv'")
```

```
✅ Đã tải dữ liệu 'CarsData_Processed.csv' thành công!

--- 2. Thiết lập dữ liệu ---
⚠️ Dữ liệu quá lớn (93887 dòng) cho SVR. Lấy mẫu ngẫu nhiên 20000 dòng để train.
Train (Sampled): 14000 | Val: 3000 | Test: 3000

--- 3. Huấn luyện SVR (Tuning) ---
Đang chạy Grid Search cho SVR (Sẽ mất thời gian)...
Fitting 3 folds for each of 6 candidates, totalling 18 fits
✅ Huấn luyện và Tuning hoàn tất.
Thời gian huấn luyện: 91.6024 giây
Tham số tối ưu: {'C': 100, 'epsilon': 0.2, 'kernel': 'rbf'}

--- 4. Đánh giá trên tập Test ---
MAE:  5,634.90
RMSE: 7,240.05
MedAE: 4,522.33
MAPE: 44.46%
R² Score: -0.0332
Inference Time: 0.000457 s/sample
-> So với KNN: Kém hơn

--- 5. Trực quan hóa ---
```






```
Đã lưu mô hình: 'svr_model.pkl'
Đã lưu kết quả đánh giá: 'model_5_results.csv'
```

CHƯƠNG 4: XÂY DỰNG MÔ HÌNH - MÔ HÌNH 6: DECISION TREE REGRESSOR

Sinh viên: Nguyễn Trung Kiên - 20227180
Mục tiêu:

1. Chuyển sang nhóm mô hình Cây quyết định (Tree-based) để xử lý dữ liệu phi tuyến và hỗn hợp.
2. Tinh chỉnh siêu tham số (max_depth, min_samples_split) để kiểm soát Overfitting.

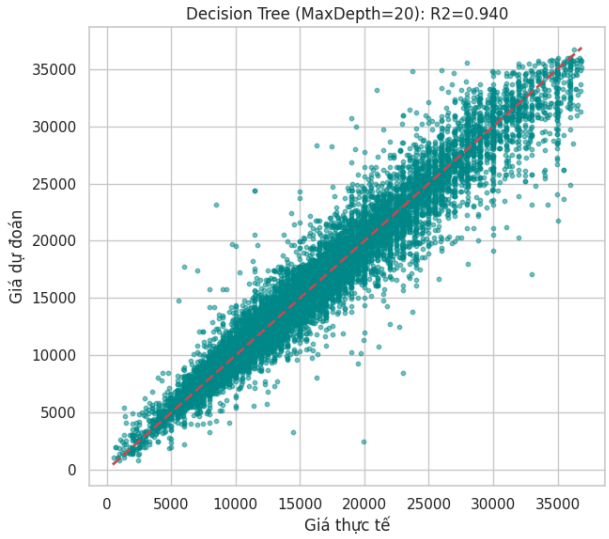
3. So sánh hiệu năng với KNN (Current Best Model).

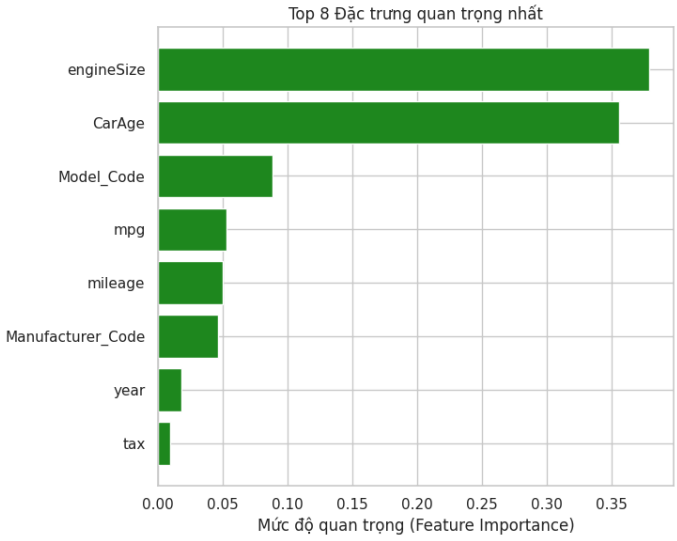
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, median_absolute_error, mean_absolute_pe
import joblib
import time

# Cấu hình giao diện
sns.set(style="whitegrid")
plt.rcParams['figure.figsize'] = (12, 6)

# ==============================================================================
# 1. TẢI DỮ LIỆU
# ==============================================================================
try:
    # Decision Tree không yêu cầu Scaling (chuẩn hóa), nhưng dùng dữ liệu đã scale cũng không sao.
    # Để nhất quán với các bước trước, ta dùng file đã xử lý.
    df = pd.read_csv('CarsData_Processed.csv')
    print("✅ Đã tải dữ liệu 'CarsData_Processed.csv' thành công!")
except FileNotFoundError:
    print("❌ Lỗi: Không tìm thấy file. Hãy chạy code Chương 2 trước.")
    raise SystemExit

# ==============================================================================
# 2. CHIA TẬP DỮ LIỆU (70-15-15)
# ==============================================================================
print("\n--- 2. Thiết lập dữ liệu ---")

# Lấy các biến số (đã bao gồm các biến Code)
df_numeric = df.select_dtypes(include=[np.number])
X = df_numeric.drop(columns=['price'])
y = df_numeric['price']

print(f"Số lượng đặc trưng: {X.shape[1]}")

# Chia tập dữ liệu (Giữ random_state=42)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

print(f"Train: {X_train.shape[0]} | Val: {X_val.shape[0]} | Test: {X_test.shape[0]}")

# ==============================================================================
# 3. HUẤN LUYỆN & TÌM SIÊU THAM SỐ (GRID SEARCH)
# ==============================================================================
print("\n--- 3. Huấn luyện Decision Tree (Tuning) ---")

start_train_time = time.time()

# Không gian tham số
# max_depth: Độ sâu tối đa của cây. Quá sâu -> Overfitting. Quá nông -> Underfitting.
# min_samples_split: Số lượng mẫu tối thiểu để tách một nút.
# min_samples_leaf: Số lượng mẫu tối thiểu tại một nút lá.
param_grid = {
    'max_depth': [10, 15, 20, 25, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

dt = DecisionTreeRegressor(random_state=42)

print("Đang chạy Grid Search (5-Fold CV)...")
grid_search = GridSearchCV(estimator=dt, param_grid=param_grid, cv=5,
                           scoring='neg_mean_squared_error', n_jobs=-1)

grid_search.fit(X_train, y_train)
```

```
end_train_time = time.time()
training_time = end_train_time - start_train_time

print("✅ Huấn luyện và Tuning hoàn tất.")
print(f"Thời gian huấn luyện: {training_time:.4f} giây")
print(f"Tham số tối ưu: {grid_search.best_params_}")

best_dt = grid_search.best_estimator_

# ==============================================================================
# 4. ĐÁNH GIÁ MÔ HÌNH (TOÀN DIỆN)
# ==============================================================================
print("\n--- 4. Đánh giá trên tập Test ---")

start_pred_time = time.time()
y_pred = best_dt.predict(X_test)
end_pred_time = time.time()
inference_time = (end_pred_time - start_pred_time) / len(X_test)

# Xử lý giá trị âm (Dù cây hồi quy hiếm khi ra âm với dữ liệu dương)
y_pred = np.maximum(y_pred, 0)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
medae = median_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

print(f"MAE:  {mae:,.2f}")
print(f"RMSE: {rmse:,.2f}")
print(f"MedAE: {medae:,.2f}")
print(f"MAPE: {mape:.2%}")
print(f"R² Score: {r2:.4f}")
print(f"Inference Time: {inference_time:.6f} s/sample")

# So sánh với KNN (Model 4 - Giả định R2=0.9523)
knn_r2 = 0.9523
print(f"-> So với KNN (M4): {'Cải thiện' if r2 > knn_r2 else 'Kém hơn'} ({r2 - knn_r2:+.4f})")

# ==============================================================================
# 5. TRỰC QUAN HÓA
# ==============================================================================
print("\n--- 5. Trực quan hóa ---")

# 5.1. Scatter Plot
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred, alpha=0.5, color='darkcyan', s=10)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', lw=2)
plt.xlabel('Giá thực tế')
plt.ylabel('Giá dự đoán')
plt.title(f'Decision Tree (MaxDepth={grid_search.best_params_["max_depth"]}): R2={r2:.3f}')

# 5.2. Feature Importance (Đặc sản của Tree-based)
importances = best_dt.feature_importances_
feature_names = X.columns
# Sắp xếp giảm dần
indices = np.argsort(importances)[::-1]

# [SỬA LỖI]: Điều chỉnh top_n tự động theo số lượng đặc trưng thực tế
num_features = X.shape[1]
top_n = min(10, num_features)

plt.subplot(1, 2, 2)
plt.barh(range(top_n), importances[indices[:top_n]], align='center', color='forestgreen')
plt.yticks(range(top_n), [feature_names[i] for i in indices[:top_n]])
plt.xlabel('Mức độ quan trọng (Feature Importance)')
plt.title(f'Top {top_n} Đặc trưng quan trọng nhất')
plt.gca().invert_yaxis() # Đảo ngược trục y để biến quan trọng nhất lên đầu

plt.tight_layout()
plt.show()
```

```
# ============================================================================
# 6. LƯU KẾT QUẢ
# ============================================================================
joblib.dump(best_dt, 'decision_tree_model.pkl')
print("\nĐã lưu mô hình: 'decision_tree_model.pkl'")

results = pd.DataFrame({
    'Model': ['Decision Tree'],
    'Params': [str(grid_search.best_params_)],
    'MAE': [mae], 'RMSE': [rmse], 'MedAE': [medae], 'MAPE': [mape], 'R2': [r2],
    'Train_Time': [training_time], 'Inference_Time': [inference_time]
})
results.to_csv('model_6_results.csv', index=False)
print("Đã lưu kết quả đánh giá: 'model_6_results.csv'")
```

```
✅ Đã tải dữ liệu 'CarsData_Processed.csv' thành công!

--- 2. Thiết lập dữ liệu ---
Số lượng đặc trưng: 8
Train: 65720 | Val: 14083 | Test: 14084

--- 3. Huấn luyện Decision Tree (Tuning) ---
Đang chạy Grid Search (5-Fold CV)...
✅ Huấn luyện và Tuning hoàn tất.
Thời gian huấn luyện: 75.2425 giây
Tham số tối ưu: {'max_depth': 20, 'min_samples_leaf': 4, 'min_samples_split': 10}

--- 4. Đánh giá trên tập Test ---
MAE:  1,187.93
RMSE: 1,757.15
MedAE: 799.31
MAPE: 8.37%
R² Score: 0.9404
Inference Time: 0.000000 s/sample
-> So với KNN (M4): Kém hơn (-0.0119)

--- 5. Trực quan hóa ---
```






```
Đã lưu mô hình: 'decision_tree_model.pkl'
Đã lưu kết quả đánh giá: 'model_6_results.csv'
```

CHƯƠNG 4: XÂY DỰNG MÔ HÌNH - MÔ HÌNH 7: RANDOM FOREST REGRESSOR

Sinh viên: Nguyễn Trung Kiên - 20227180

Mục tiêu:

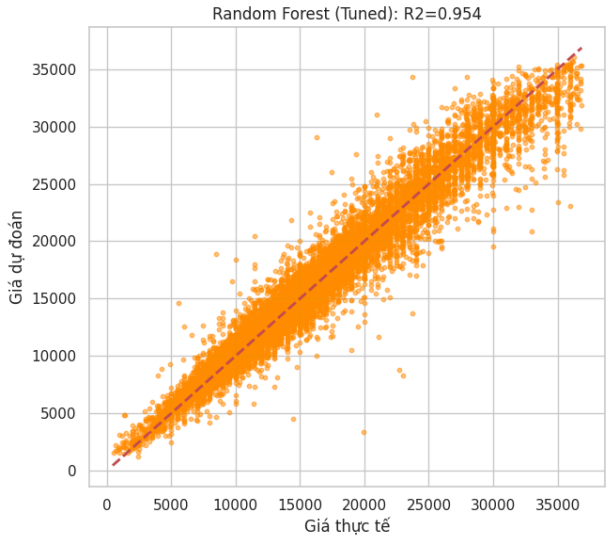
1. Nâng cấp từ Decision Tree lên Random Forest (Ensemble Learning - Bagging).
2. Tinh chỉnh siêu tham số (n_estimators, max_depth) để tối ưu hóa hiệu năng.
3. So sánh độ chính xác và độ ổn định với Decision Tree đơn lẻ.

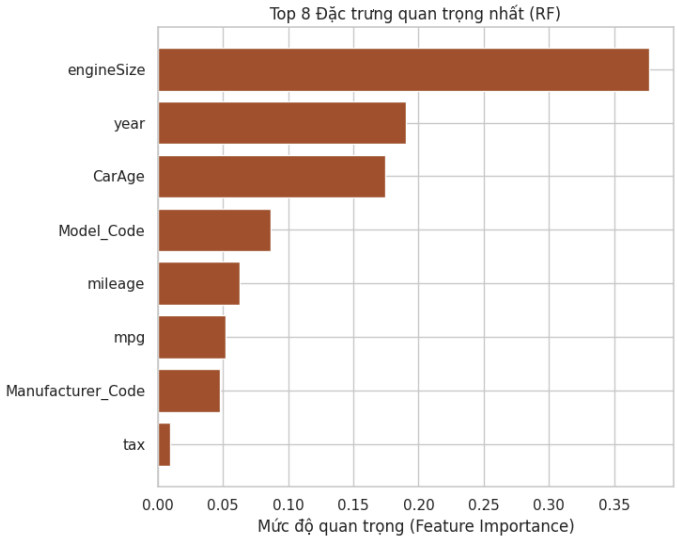
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, median_absolute_error, mean_absolute_pe
import joblib
import time

# Cấu hình giao diện
sns.set(style="whitegrid")
plt.rcParams['figure.figsize'] = (12, 6)

# ==============================================================================
# 1. TẢI DỮ LIỆU
# ==============================================================================
try:
    # Random Forest cũng xử lý tốt dữ liệu đã scale hoặc chưa scale.
    # Để nhất quán, ta dùng file đã xử lý.
    df = pd.read_csv('CarsData_Processed.csv')
    print("✅ Đã tải dữ liệu 'CarsData_Processed.csv' thành công!")
except FileNotFoundError:
    print("❌ Lỗi: Không tìm thấy file. Hãy chạy code Chương 2 trước.")
    raise SystemExit

# ==============================================================================
# 2. CHIA TẬP DỮ LIỆU (70-15-15)
# ==============================================================================
print("\n--- 2. Thiết lập dữ liệu ---")

# Lấy các biến số (đã bao gồm các biến Code)
df_numeric = df.select_dtypes(include=[np.number])
X = df_numeric.drop(columns=['price'])
y = df_numeric['price']

print(f"Số lượng đặc trưng: {X.shape[1]}")

# Chia tập dữ liệu (Giữ random_state=42)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

print(f"Train: {X_train.shape[0]} | Val: {X_val.shape[0]} | Test: {X_test.shape[0]}")

# ==============================================================================
# 3. HUẤN LUYỆN & TÌM SIÊU THAM SỐ (RANDOMIZED SEARCH)
# ==============================================================================
print("\n--- 3. Huấn luyện Random Forest (Tuning) ---")

start_train_time = time.time()

# Không gian tham số
# Random Forest tốn tài nguyên hơn Decision Tree nhiều lần.
# Dùng RandomizedSearchCV thay vì GridSearchCV để tiết kiệm thời gian mà vẫn tìm được tham số tốt.
param_dist = {
    'n_estimators': [50, 100, 200], # Số lượng cây
    'max_depth': [10, 20, 30, None], # Độ sâu tối đa
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False] # Có dùng bootstrap hay không
}

rf = RandomForestRegressor(random_state=42, n_jobs=-1) # n_jobs=-1 để dùng hết core CPU

print("Đang chạy Randomized Search (Có thể mất 2-3 phút)...")
# n_iter=10: Thử ngẫu nhiên 10 tổ hợp tham số
random_search = RandomizedSearchCV(estimator=rf, param_distributions=param_dist,
                                   n_iter=10, cv=3, verbose=2, random_state=42, n_jobs=-1,
                                   scoring='neg_mean_squared_error')
```

```
random_search.fit(X_train, y_train)

end_train_time = time.time()
training_time = end_train_time - start_train_time

print("✅ Huấn luyện và Tuning hoàn tất.")
print(f"Thời gian huấn luyện: {training_time:.4f} giây")
print(f"Tham số tối ưu: {random_search.best_params_}")

best_rf = random_search.best_estimator_

# ==============================================================================
# 4. ĐÁNH GIÁ MÔ HÌNH (TOÀN DIỆN)
# ==============================================================================
print("\n--- 4. Đánh giá trên tập Test ---")

start_pred_time = time.time()
y_pred = best_rf.predict(X_test)
end_pred_time = time.time()
inference_time = (end_pred_time - start_pred_time) / len(X_test)

# Xử lý giá trị âm
y_pred = np.maximum(y_pred, 0)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
medae = median_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

print(f"MAE:  {mae:,.2f}")
print(f"RMSE: {rmse:,.2f}")
print(f"MedAE: {medae:,.2f}")
print(f"MAPE: {mape:.2%}")
print(f"R² Score: {r2:.4f}")
print(f"Inference Time: {inference_time:.6f} s/sample")

# So sánh với Decision Tree (Model 6 - Giả định R2=0.9404)
dt_r2 = 0.9404
print(f"-> So với Decision Tree (M6): {'Cải thiện' if r2 > dt_r2 else 'Kém hơn'} ({r2 - dt_r2:+.4f})")

# ==============================================================================
# 5. TRỰC QUAN HÓA
# ==============================================================================
print("\n--- 5. Trực quan hóa ---")

# 5.1. Scatter Plot
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred, alpha=0.5, color='darkorange', s=10)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', lw=2)
plt.xlabel('Giá thực tế')
plt.ylabel('Giá dự đoán')
plt.title(f'Random Forest (Tuned): R2={r2:.3f}')

# 5.2. Feature Importance
importances = best_rf.feature_importances_
feature_names = X.columns
# Sắp xếp giảm dần
indices = np.argsort(importances)[::-1]

# Điều chỉnh top_n tự động
num_features = X.shape[1]
top_n = min(10, num_features)

plt.subplot(1, 2, 2)
plt.barh(range(top_n), importances[indices[:top_n]], align='center', color='sienna')
plt.yticks(range(top_n), [feature_names[i] for i in indices[:top_n]])
plt.xlabel('Mức độ quan trọng (Feature Importance)')
plt.title(f'Top {top_n} Đặc trưng quan trọng nhất (RF)')
plt.gca().invert_yaxis()
```

```
plt.tight_layout()
plt.show()

# ================================================================================
# 6. LƯU KẾT QUẢ
# ================================================================================
joblib.dump(best_rf, 'random_forest_model.pkl')
print("\nĐã lưu mô hình: 'random_forest_model.pkl'")

results = pd.DataFrame({
    'Model': ['Random Forest'],
    'Params': [str(random_search.best_params_)],
    'MAE': [mae], 'RMSE': [rmse], 'MedAE': [medae], 'MAPE': [mape], 'R2': [r2],
    'Train_Time': [training_time], 'Inference_Time': [inference_time]
})
results.to_csv('model_7_results.csv', index=False)
print("Đã lưu kết quả đánh giá: 'model_7_results.csv'")
```

```
✅ Đã tải dữ liệu 'CarsData_Processed.csv' thành công!

--- 2. Thiết lập dữ liệu ---
Số lượng đặc trưng: 8
Train: 65720 | Val: 14083 | Test: 14084

--- 3. Huấn luyện Random Forest (Tuning) ---
Đang chạy Randomized Search (Có thể mất 2-3 phút)...
Fitting 3 folds for each of 10 candidates, totalling 30 fits
✅ Huấn luyện và Tuning hoàn tất.
Thời gian huấn luyện: 466.1991 giây
Tham số tối ưu: {'n_estimators': 100, 'min_samples_split': 2, 'min_samples_leaf': 4, 'max_depth': 30, 'bootstrap': True

--- 4. Đánh giá trên tập Test ---
MAE:  1,060.63
RMSE: 1,543.43
MedAE: 735.27
MAPE: 7.48%
R² Score: 0.9540
Inference Time: 0.000019 s/sample
-> So với Decision Tree (M6): Cải thiện (+0.0136)

--- 5. Trực quan hóa ---
```






```
Đã lưu mô hình: 'random_forest_model.pkl'
Đã lưu kết quả đánh giá: 'model_7_results.csv'
```

CHƯƠNG 4: XÂY DỰNG MÔ HÌNH - MÔ HÌNH 8: GRADIENT BOOSTING REGRESSOR

Sinh viên: Nguyễn Trung Kiên - 20227180

Mục tiêu:

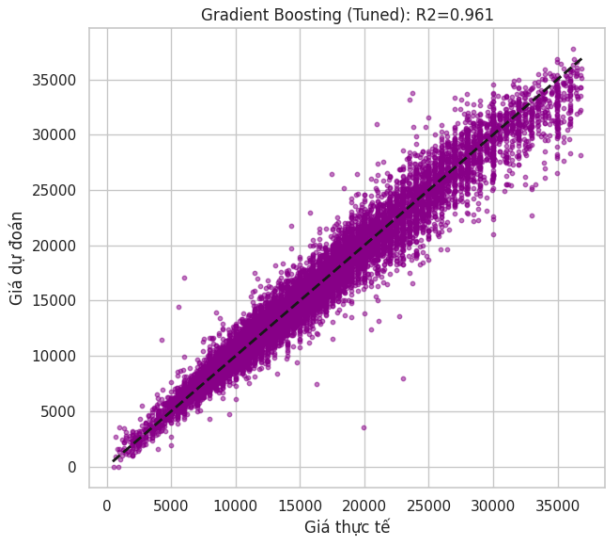
1. Chuyển sang nhóm mô hình Boosting (Tăng cường) để tối ưu hóa sai số.
2. Huấn luyện Gradient Boosting Regressor (GBM) của thư viện Scikit-learn.
3. So sánh hiệu năng với Random Forest (Current Best Model).

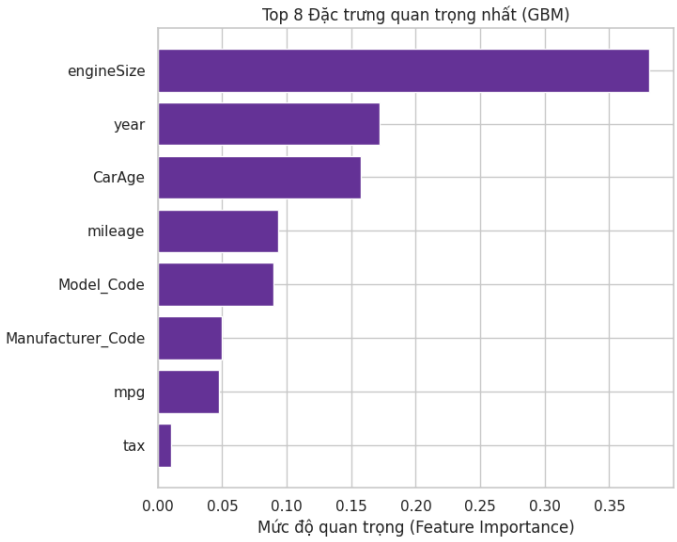
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, median_absolute_error, mean_absolute_pe
import joblib
import time

# Cấu hình giao diện
sns.set(style="whitegrid")
plt.rcParams['figure.figsize'] = (12, 6)

# ==============================================================================
# 1. TẢI DỮ LIỆU
# ==============================================================================
try:
    df = pd.read_csv('CarsData_Processed.csv')
    print("✅ Đã tải dữ liệu 'CarsData_Processed.csv' thành công!")
except FileNotFoundError:
    print("❌ Lỗi: Không tìm thấy file. Hãy chạy code Chương 2 trước.")
    raise SystemExit

# ==============================================================================
# 2. CHIA TẬP DỮ LIỆU (70-15-15)
# ==============================================================================
print("\n--- 2. Thiết lập dữ liệu ---")

# Lấy các biến số (đã bao gồm các biến Code)
df_numeric = df.select_dtypes(include=[np.number])
X = df_numeric.drop(columns=['price'])
y = df_numeric['price']

print(f"Số lượng đặc trưng: {X.shape[1]}")

# Chia tập dữ liệu (Giữ random_state=42)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

print(f"Train: {X_train.shape[0]} | Val: {X_val.shape[0]} | Test: {X_test.shape[0]}")

# ==============================================================================
# 3. HUẤN LUYỆN & TÌM SIÊU THAM SỐ (RANDOMIZED SEARCH)
# ==============================================================================
print("\n--- 3. Huấn luyện Gradient Boosting (Tuning) ---")

start_train_time = time.time()

# Không gian tham số cho GBM
# learning_rate: Tốc độ học. Thường < 0.1. Càng nhỏ thì cần n_estimators càng lớn.
# n_estimators: Số lượng cây.
# max_depth: Độ sâu của từng cây (thường nhỏ hơn RF, khoảng 3-8).
# subsample: Tỷ lệ mẫu dùng để train mỗi cây (Stochastic Gradient Boosting).
param_dist = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'subsample': [0.8, 1.0]
}

gbm = GradientBoostingRegressor(random_state=42)
```

```
print("Đang chạy Randomized Search (Có thể mất 3-5 phút do GBM train tuần tự)...")
# n_iter=10: Thử ngẫu nhiên 10 tổ hợp
random_search = RandomizedSearchCV(estimator=gbm, param_distributions=param_dist,
                                   n_iter=10, cv=3, verbose=1, random_state=42, n_jobs=-1,
                                   scoring='neg_mean_squared_error')

random_search.fit(X_train, y_train)

end_train_time = time.time()
training_time = end_train_time - start_train_time

print("✅ Huấn luyện và Tuning hoàn tất.")
print(f"Thời gian huấn luyện: {training_time:.4f} giây")
print(f"Tham số tối ưu: {random_search.best_params_}")

best_gbm = random_search.best_estimator_

# ==============================================================================
# 4. ĐÁNH GIÁ MÔ HÌNH (TOÀN DIỆN)
# ==============================================================================
print("\n--- 4. Đánh giá trên tập Test ---")

start_pred_time = time.time()
y_pred = best_gbm.predict(X_test)
end_pred_time = time.time()
inference_time = (end_pred_time - start_pred_time) / len(X_test)

# Xử lý giá trị âm
y_pred = np.maximum(y_pred, 0)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
medae = median_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

print(f"MAE:  {mae:,.2f}")
print(f"RMSE: {rmse:,.2f}")
print(f"MedAE: {medae:,.2f}")
print(f"MAPE: {mape:.2%}")
print(f"R² Score: {r2:.4f}")
print(f"Inference Time: {inference_time:.6f} s/sample")

# So sánh với Random Forest (Model 7 - Giả định R2=0.9540)
rf_r2 = 0.9540
print(f"-> So với Random Forest (M7): {'Cải thiện' if r2 > rf_r2 else 'Kém hơn'} ({r2 - rf_r2:+.4f})")

# ==============================================================================
# 5. TRỰC QUAN HÓA
# ==============================================================================
print("\n--- 5. Trực quan hóa ---")

# 5.1. Scatter Plot
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred, alpha=0.5, color='darkmagenta', s=10)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=2)
plt.xlabel('Giá thực tế')
plt.ylabel('Giá dự đoán')
plt.title(f'Gradient Boosting (Tuned): R2={r2:.3f}')

# 5.2. Feature Importance
importances = best_gbm.feature_importances_
feature_names = X.columns
indices = np.argsort(importances)[::-1]

# Điều chỉnh top_n tự động
num_features = X.shape[1]
top_n = min(10, num_features)

plt.subplot(1, 2, 2)
```

```
plt.barh(range(top_n), importances[indices[:top_n]], align='center', color='rebeccapurple')
plt.yticks(range(top_n), [feature_names[i] for i in indices[:top_n]])
plt.xlabel('Mức độ quan trọng (Feature Importance)')
plt.title(f'Top {top_n} Đặc trưng quan trọng nhất (GBM)')
plt.gca().invert_yaxis()

plt.tight_layout()
plt.show()

# ==============================================================================
# 6. LƯU KẾT QUẢ
# ==============================================================================
joblib.dump(best_gbm, 'gbm_model.pkl')
print("\nĐã lưu mô hình: 'gbm_model.pkl'")

results = pd.DataFrame({
    'Model': ['Gradient Boosting'],
    'Params': [str(random_search.best_params_)],
    'MAE': [mae], 'RMSE': [rmse], 'MedAE': [medae], 'MAPE': [mape], 'R2': [r2],
    'Train_Time': [training_time], 'Inference_Time': [inference_time]
})
results.to_csv('model_8_results.csv', index=False)
print("Đã lưu kết quả đánh giá: 'model_8_results.csv'")
```

```
✅ Đã tải dữ liệu 'CarsData_Processed.csv' thành công!

--- 2. Thiết lập dữ liệu ---
Số lượng đặc trưng: 8
Train: 65720 | Val: 14083 | Test: 14084

--- 3. Huấn luyện Gradient Boosting (Tuning) ---
Đang chạy Randomized Search (Có thể mất 3-5 phút do GBM train tuần tự)...
Fitting 3 folds for each of 10 candidates, totalling 30 fits
✅ Huấn luyện và Tuning hoàn tất.
Thời gian huấn luyện: 606.1255 giây
Tham số tối ưu: {'subsample': 1.0, 'n_estimators': 300, 'min_samples_split': 2, 'min_samples_leaf': 4, 'max_depth': 7,

--- 4. Đánh giá trên tập Test ---
MAE:  1,009.41
RMSE: 1,426.80
MedAE: 729.42
MAPE: 7.09%
R² Score: 0.9607
Inference Time: 0.000012 s/sample
-> So với Random Forest (M7): Cải thiện (+0.0067)

--- 5. Trực quan hóa ---
```






```
Đã lưu mô hình: 'gbm_model.pkl'
Đã lưu kết quả đánh giá: 'model_8_results.csv'
```

CHƯƠNG 4: XÂY DỰNG MÔ HÌNH - MÔ HÌNH 9: XGBOOST REGRESSOR

Sinh viên: Nguyễn Trung Kiên - 20227180

Mục tiêu:

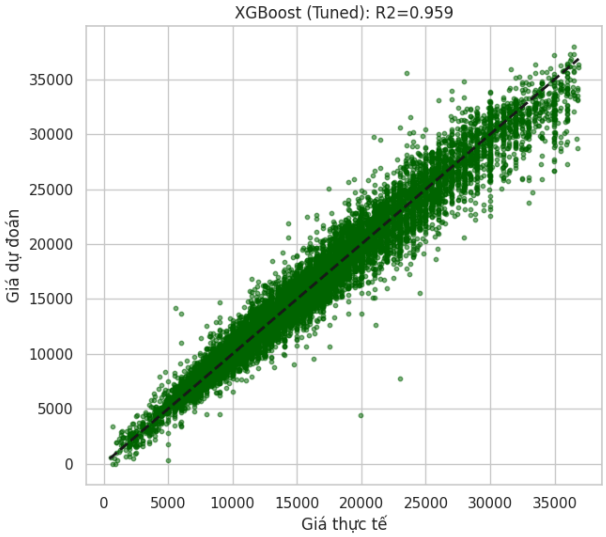
1. Huấn luyện mô hình XGBoost (eXtreme Gradient Boosting) - Phiên bản nâng cấp của GBM.
2. Tối ưu hóa siêu tham số để cân bằng giữa tốc độ và độ chính xác.
3. So sánh hiệu năng với Gradient Boosting (Model 8) để thấy sự cải thiện về thời gian và độ chính xác.

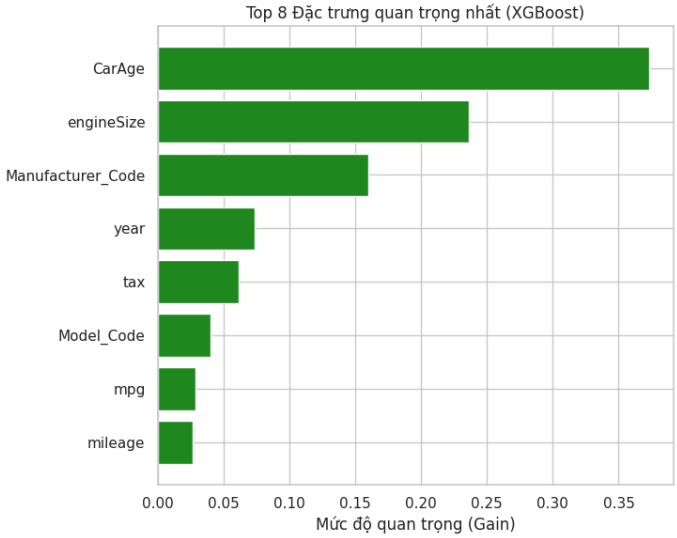
```
# ================================================================================
# 1. CÀI ĐẶT & IMPORT THƯ VIỆN
# ================================================================================
# Cài đặt xgboost nếu chưa có (Colab thường có sẵn, nhưng chạy cho chắc)
!pip install xgboost

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from xgboost import XGBRegressor, plot_importance
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, median_absolute_error, mean_absolute_pe
import joblib
import time

# Cấu hình giao diện
sns.set(style="whitegrid")
plt.rcParams['figure.figsize'] = (12, 6)

# ================================================================================
# 2. TẢI DỮ LIỆU
# ================================================================================
try:
    df = pd.read_csv('CarsData_Processed.csv')
    print("✅ Đã tải dữ liệu 'CarsData_Processed.csv' thành công!")
except FileNotFoundError:
    print("❌ Lỗi: Không tìm thấy file. Hãy chạy code Chương 2 trước.")
    raise SystemExit

# ================================================================================
# 3. CHIA TẬP DỮ LIỆU (70-15-15)
# ================================================================================
print("\n--- 3. Thiết lập dữ liệu ---")

# Lấy các biến số (đã bao gồm các biến Code)
df_numeric = df.select_dtypes(include=[np.number])
X = df_numeric.drop(columns=['price'])
y = df_numeric['price']

print(f"Số lượng đặc trưng: {X.shape[1]}")

# Chia tập dữ liệu (Giữ random_state=42)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

print(f"Train: {X_train.shape[0]} | Val: {X_val.shape[0]} | Test: {X_test.shape[0]}")

# ================================================================================
# 4. HUẤN LUYỆN & TÌM SIÊU THAM SỐ (RANDOMIZED SEARCH)
# ================================================================================
print("\n--- 4. Huấn luyện XGBoost (Tuning) ---")

start_train_time = time.time()

# Không gian tham số cho XGBoost
# n_estimators: Số lượng cây
# learning_rate: Tốc độ học (eta)
# max_depth: Độ sâu tối đa của cây
# subsample: Tỷ lệ mẫu dùng để train mỗi cây
```

```
# colsample_bytree: Tỷ lệ đặc trưng dùng để xây dựng mỗi cây
# reg_alpha (L1), reg_lambda (L2): Regularization để chống overfitting
param_dist = {
    'n_estimators': [100, 300, 500, 700],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'max_depth': [3, 5, 7, 9],
    'min_child_weight': [1, 3, 5],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'reg_alpha': [0, 0.1, 1],
    'reg_lambda': [1, 1.5, 2]
}

xgb = XGBRegressor(random_state=42, n_jobs=-1, objective='reg:squarederror')

print("Đang chạy Randomized Search (XGBoost chạy nhanh hơn GBM nhờ song song hóa)...")
# n_iter=20: Thử ngẫu nhiên 20 tổ hợp (nhiều hơn GBM vì chạy nhanh hơn)
random_search = RandomizedSearchCV(estimator=xgb, param_distributions=param_dist,
                                   n_iter=20, cv=3, verbose=1, random_state=42, n_jobs=-1,
                                   scoring='neg_mean_squared_error')

random_search.fit(X_train, y_train)

end_train_time = time.time()
training_time = end_train_time - start_train_time

print("✅ Huấn luyện và Tuning hoàn tất.")
print(f"Thời gian huấn luyện: {training_time:.4f} giây")
print(f"Tham số tối ưu: {random_search.best_params_}")

best_xgb = random_search.best_estimator_

# ==============================================================================
# 5. ĐÁNH GIÁ MÔ HÌNH (TOÀN DIỆN)
# ==============================================================================
print("\n--- 5. Đánh giá trên tập Test ---")

start_pred_time = time.time()
y_pred = best_xgb.predict(X_test)
end_pred_time = time.time()
inference_time = (end_pred_time - start_pred_time) / len(X_test)

# Xử lý giá trị âm
y_pred = np.maximum(y_pred, 0)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
medae = median_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

print(f"MAE:  {mae:,.2f}")
print(f"RMSE: {rmse:,.2f}")
print(f"MedAE: {medae:,.2f}")
print(f"MAPE: {mape:.2%}")
print(f"R² Score: {r2:.4f}")
print(f"Inference Time: {inference_time:.6f} s/sample")

# So sánh với Gradient Boosting (Model 8 - Giả định R2=0.9607)
gbm_r2 = 0.9607
print(f"-> So với GBM (M8): {'Cải thiện' if r2 > gbm_r2 else 'Kém hơn'} ({r2 - gbm_r2:+.4f})")

# ==============================================================================
# 6. TRỰC QUAN HÓA
# ==============================================================================
print("\n--- 6. Trực quan hóa ---")

# 6.1. Scatter Plot
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred, alpha=0.5, color='darkgreen', s=10)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=2)
```

```
plt.xlabel('Giá thực tế')
plt.ylabel('Giá dự đoán')
plt.title(f'XGBoost (Tuned): R2={r2:.3f}')

# 6.2. Feature Importance
plt.subplot(1, 2, 2)
# XGBoost có hàm plot_importance tích hợp sẵn rất tiện
# max_num_features=10: Chỉ hiện top 10
# importance_type='gain': Đánh giá dựa trên mức độ giảm nhiễu (Gain) - chuẩn nhất
# height=0.5: Độ dày thanh
importances = best_xgb.feature_importances_
feature_names = X.columns
indices = np.argsort(importances)[::-1]
top_n = min(10, len(feature_names))

plt.barh(range(top_n), importances[indices[:top_n]], align='center', color='forestgreen')
plt.yticks(range(top_n), [feature_names[i] for i in indices[:top_n]])
plt.xlabel('Mức độ quan trọng (Gain)')
plt.title(f'Top {top_n} Đặc trưng quan trọng nhất (XGBoost)')
plt.gca().invert_yaxis()

plt.tight_layout()
plt.show()

# ==============================================================================
# 7. LƯU KẾT QUẢ
# ==============================================================================
joblib.dump(best_xgb, 'xgboost_model.pkl')
print("\nĐã lưu mô hình: 'xgboost_model.pkl'")

results = pd.DataFrame({
    'Model': ['XGBoost'],
    'Params': [str(random_search.best_params_)],
    'MAE': [mae], 'RMSE': [rmse], 'MedAE': [medae], 'MAPE': [mape], 'R2': [r2],
    'Train_Time': [training_time], 'Inference_Time': [inference_time]
})
results.to_csv('model_9_results.csv', index=False)
print("Đã lưu kết quả đánh giá: 'model_9_results.csv'")
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.12/dist-packages (3.1.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from xgboost) (2.0.2)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.12/dist-packages (from xgboost) (2.28.9)
Requirement already satisfied: scipy in /usr/local/lib/python3.12/dist-packages (from xgboost) (1.16.3)
✅ Đã tải dữ liệu 'CarsData_Processed.csv' thành công!

--- 3. Thiết lập dữ liệu ---
Số lượng đặc trưng: 8
Train: 65720 | Val: 14083 | Test: 14084

--- 4. Huấn luyện XGBoost (Tuning) ---
Đang chạy Randomized Search (XGBoost chạy nhanh hơn GBM nhờ song song hóa)...
Fitting 3 folds for each of 20 candidates, totalling 60 fits
✅ Huấn luyện và Tuning hoàn tất.
Thời gian huấn luyện: 96.6425 giây
Tham số tối ưu: {'subsample': 0.8, 'reg_lambda': 2, 'reg_alpha': 1, 'n_estimators': 700, 'min_child_weight': 3, 'max_de

--- 5. Đánh giá trên tập Test ---
MAE:  1,039.94
RMSE: 1,458.92
MedAE: 756.64
MAPE: 7.31%
R² Score: 0.9589
Inference Time: 0.000017 s/sample
-> So với GBM (M8): Kém hơn (-0.0018)

--- 6. Trực quan hóa ---
```






```
Đã lưu mô hình: 'xgboost_model.pkl'
Đã lưu kết quả đánh giá: 'model_9_results.csv'
```

CHƯƠNG 4 - PHẦN CUỐI: CATBOOST REGRESSOR (SOTA MODEL)

Mục tiêu:

1. Huấn luyện mô hình mạnh nhất (State-of-the-Art) cho dữ liệu bảng.
2. Sử dụng Optuna để tối ưu hóa siêu tham số tự động.
3. Phân tích SHAP để giải thích mô hình (phục vụ xây dựng tính năng cho App).

```
# ================================================================================
# 1. CÀI ĐẶT & IMPORT THƯ VIỆN
```

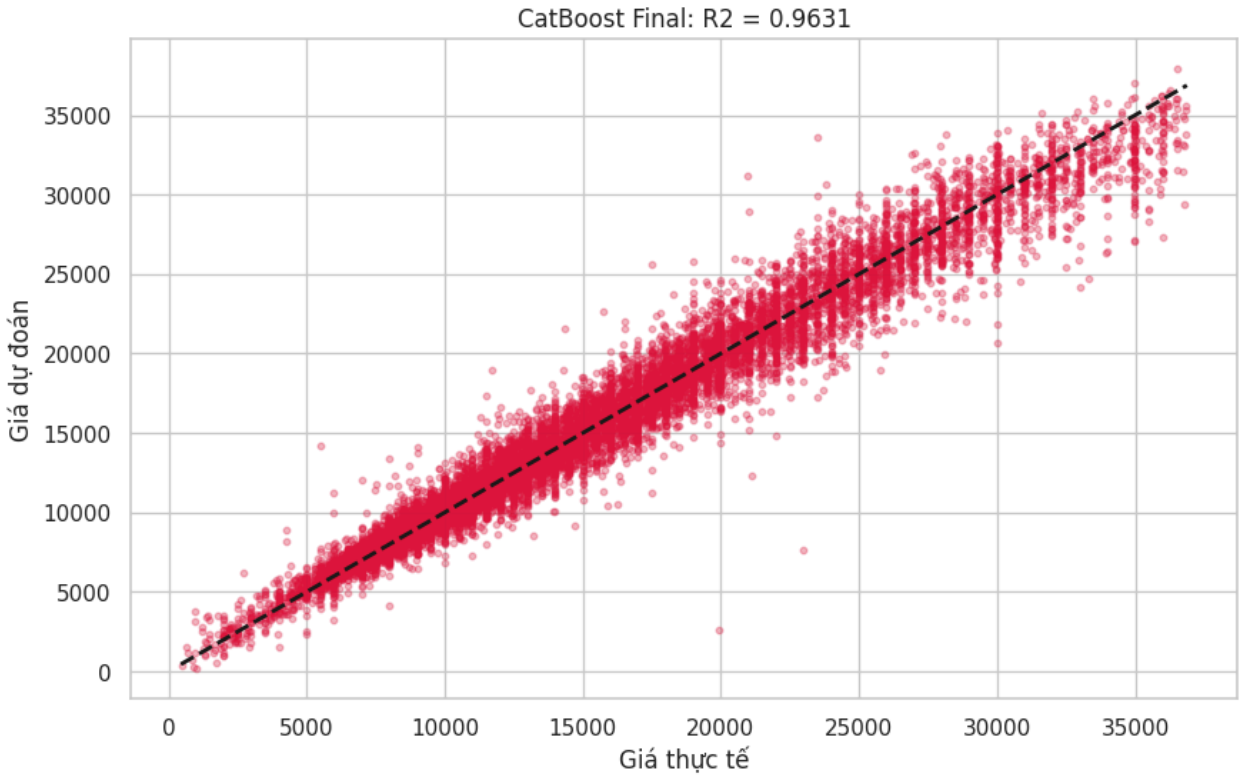
```
# ================================================================================
# Cài đặt thư viện cần thiết (nếu chưa có trên Colab)
!pip install catboost optuna shap

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import joblib
import catboost as cb
from catboost import CatBoostRegressor, Pool
import optuna
import shap

# Cấu hình giao diện
sns.set(style="whitegrid")
plt.rcParams['figure.figsize'] = (12, 6)

# Tải dữ liệu
try:
    df_final = pd.read_csv('CarsData_Processed.csv')
    print("✅ Đã tải dữ liệu 'CarsData_Processed.csv' thành công!")
except FileNotFoundError:
    print("❌ Lỗi: Không tìm thấy file. Hãy chạy code Chương 2 trước.")
    raise SystemExit

# ================================================================================
# 2. CHUẨN BỊ DỮ LIỆU CHUYÊN SÂU
# ================================================================================
print("\n--- 2. Chuẩn bị dữ liệu cho CatBoost ---")

# Lọc các cột số (loại bỏ cột string gốc nếu còn sót)
df_numeric = df_final.select_dtypes(include=[np.number])
X = df_numeric.drop(columns=['price'])
y = df_numeric['price']

# Chia tập dữ liệu (70-15-15)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# XÁC ĐỊNH BIẾN PHÂN LOẠI (QUAN TRỌNG VỚI CATBOOST)
# Dù đã Label Encode (thành số), ta vẫn cần báo cho CatBoost biết đây là biến phân loại
# để nó áp dụng kỹ thuật Ordered Target Statistics.
cat_features_indices = []
for i, col in enumerate(X.columns):
    # Các cột _Code hoặc biến rời rạc ít giá trị nên được coi là category
    if 'Code' in col or 'fuelType' in col or 'transmission' in col:
        # Lưu ý: Nếu cột đã One-Hot thì là số 0/1, CatBoost xử lý tốt.
        # Nhưng tốt nhất là chỉ định các cột Label Encoding (Manufacturer_Code, Model_Code)
        if 'Manufacturer_Code' in col or 'Model_Code' in col:
            cat_features_indices.append(i)

print(f"Các cột được xử lý như Categorical Feature: {[X.columns[i] for i in cat_features_indices]}")

# Tạo Pool dữ liệu (Cấu trúc tối ưu của CatBoost)
train_pool = Pool(X_train, y_train, cat_features=cat_features_indices)
val_pool = Pool(X_val, y_val, cat_features=cat_features_indices)
test_pool = Pool(X_test, y_test, cat_features=cat_features_indices)

# ================================================================================
# 3. TỐI ƯU HÓA THAM SỐ VỚI OPTUNA (AUTO TUNING)
# ================================================================================
print("\n--- 3. Tối ưu hóa Siêu tham số (Optuna) ---")

def objective(trial):
    # Định nghĩa không gian tham số cần tìm kiếm
    params = {
        'iterations': trial.suggest_int('iterations', 500, 2000),
        'depth': trial.suggest_int('depth', 4, 10),
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.3),
```

```
        'l2_leaf_reg': trial.suggest_float('l2_leaf_reg', 1, 10),
        'random_strength': trial.suggest_float('random_strength', 1e-9, 10),
        'bagging_temperature': trial.suggest_float('bagging_temperature', 0, 1),
        'border_count': 128,
        'loss_function': 'RMSE',
        'verbose': False,
        'random_seed': 42,
        'task_type': 'CPU' # Đổi thành 'GPU' nếu Colab có GPU
    }

    # Huấn luyện thử
    model = CatBoostRegressor(**params)
    model.fit(train_pool, eval_set=val_pool, early_stopping_rounds=50, verbose=False)

    # Dự đoán trên tập Val để đánh giá trial này
    preds = model.predict(val_pool)
    rmse = np.sqrt(mean_squared_error(y_val, preds))
    return rmse

# Chạy Optuna (Thử 20 lần - Kiên có thể tăng lên 50-100 nếu muốn kết quả cực xịn)
study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=20)

print(f"\n✅ Best RMSE: {study.best_value}")
print(f"✅ Best Params: {study.best_params}")

# ==============================================================================
# 4. HUẤN LUYỆN MÔ HÌNH FINAL (BEST MODEL)
# ==============================================================================
print("\n--- 4. Huấn luyện Mô hình Tốt nhất ---")

# Lấy tham số tốt nhất từ Optuna
best_params = study.best_params
best_params['loss_function'] = 'RMSE'
best_params['random_seed'] = 42
best_params['verbose'] = 100 # In log mỗi 100 vòng

final_model = CatBoostRegressor(**best_params)
final_model.fit(train_pool, eval_set=val_pool, early_stopping_rounds=100)

# ==============================================================================
# 5. ĐÁNH GIÁ TOÀN DIỆN & GIẢI THÍCH (SHAP)
# ==============================================================================
print("\n--- 5. Đánh giá & Explainable AI ---")

# 5.1. Chỉ số kỹ thuật
y_pred = final_model.predict(test_pool)
# Đảm bảo không có giá âm
y_pred = np.maximum(y_pred, 0)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"--- KẾT QUẢ CUỐI CÙNG (TEST SET) ---")
print(f"MAE:  {mae:,.2f}")
print(f"RMSE: {rmse:,.2f}")
print(f"R2:   {r2:.4f}")

# 5.2. Biểu đồ Scatter (Thực tế vs Dự đoán)
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.3, color='crimson', s=10)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=2)
plt.xlabel('Giá thực tế')
plt.ylabel('Giá dự đoán')
plt.title(f'CatBoost Final: R2 = {r2:.4f}')
plt.show()

# 5.3. SHAP Values (Giải thích mô hình cho sản phẩm)
# Phần này rất quan trọng để đưa vào Slide "Giải thích mô hình"
print("\nĐang tính toán SHAP values (Có thể mất chút thời gian)...")
explainer = shap.TreeExplainer(final_model)
```

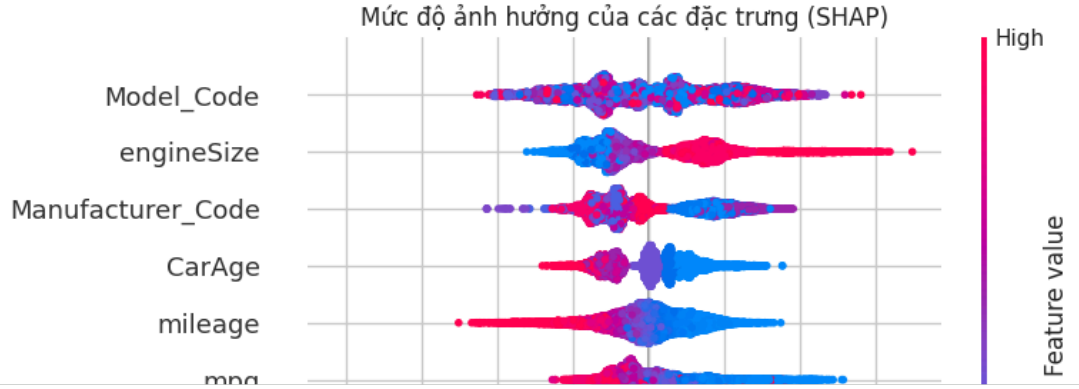
```
shap_values = explainer.shap_values(X_test)

# Summary Plot (Đưa ảnh này vào báo cáo!)
plt.figure(figsize=(12, 8))
shap.summary_plot(shap_values, X_test, show=False)
plt.title("Mức độ ảnh hưởng của các đặc trưng (SHAP)")
plt.tight_layout()
plt.show()

# ==============================================================================
# 6. ĐÓNG GÓI MÔ HÌNH (CHO APP)
# ==============================================================================
# Lưu mô hình dưới dạng .cbm (định dạng chuẩn của CatBoost, nhẹ và nhanh)
final_model.save_model("catboost_final_model.cbm")
print("\n✅ Đã lưu mô hình: 'catboost_final_model.cbm'")
print("Sẵn sàng tích hợp vào Streamlit App!")
```

```
Collecting catboost
  Downloading catboost-1.2.8-cp312-cp312-manylinux2014_x86_64.whl.metadata (1.2 kB)
Collecting optuna
  Downloading optuna-4.6.0-py3-none-any.whl.metadata (17 kB)
Requirement already satisfied: shap in /usr/local/lib/python3.12/dist-packages (0.50.0)
Requirement already satisfied: graphviz in /usr/local/lib/python3.12/dist-packages (from catboost) (0.21)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (from catboost) (3.10.0)
Requirement already satisfied: numpy<3.0,>=1.16.0 in /usr/local/lib/python3.12/dist-packages (from catboost) (2.0.2)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.12/dist-packages (from catboost) (2.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.12/dist-packages (from catboost) (1.16.3)
Requirement already satisfied: plotly in /usr/local/lib/python3.12/dist-packages (from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.12/dist-packages (from catboost) (1.17.0)
Requirement already satisfied: alembic>=1.5.0 in /usr/local/lib/python3.12/dist-packages (from optuna) (1.17.2)
Collecting colorlog (from optuna)
  Downloading colorlog-6.10.1-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from optuna) (25.0)
Requirement already satisfied: sqlalchemy>=1.4.2 in /usr/local/lib/python3.12/dist-packages (from optuna) (2.0.45)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from optuna) (4.67.1)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.12/dist-packages (from optuna) (6.0.3)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (from shap) (1.6.1)
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.12/dist-packages (from shap) (0.0.8)
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.12/dist-packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.12/dist-packages (from shap) (3.1.2)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.12/dist-packages (from shap) (4.15.0)
Requirement already satisfied: Mako in /usr/local/lib/python3.12/dist-packages (from alembic>=1.5.0->optuna) (1.3.10)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.12/dist-packages (from numba>=0.54-
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas>=0.24->ca
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas>=0.24->catboost) (2
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas>=0.24->catboost)
Requirement already satisfied: greenlet>=1 in /usr/local/lib/python3.12/dist-packages (from sqlalchemy>=1.4.2->optuna)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib->catboost)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib->catboost) (0.1
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib->catboost)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib->catboost)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib->catboost) (11.3.0
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib->catboost)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.12/dist-packages (from plotly->catboost) (9.1.
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn->shap) (1.5.
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn->shap
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3.12/dist-packages (from Mako->alembic>=1.5.0-
Downloading catboost-1.2.8-cp312-cp312-manylinux2014_x86_64.whl (99.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 99.2/99.2 MB 5.2 MB/s eta 0:00:00
Downloading optuna-4.6.0-py3-none-any.whl (404 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 404.7/404.7 kB 25.0 MB/s eta 0:00:00
Downloading colorlog-6.10.1-py3-none-any.whl (11 kB)
Installing collected packages: colorlog, optuna, catboost
Successfully installed catboost-1.2.8 colorlog-6.10.1 optuna-4.6.0
[I 2025-12-23 06:02:53,533] A new study created in memory with name: no-name-debf513e-84d6-4755-b6d1-7a4106d214dc
✅ Đã tải dữ liệu 'CarsData_Processed.csv' thành công!

--- 2. Chuẩn bị dữ liệu cho CatBoost ---
Các cột được xử lý như Categorical Feature: ['Manufacturer_Code', 'Model_Code']

--- 3. Tối ưu hóa Siêu tham số (Optuna) ---
[I 2025-12-23 06:03:10,739] Trial 0 finished with value: 1741.2786405165268 and parameters: {'iterations': 691, 'depth
[I 2025-12-23 06:04:15,247] Trial 1 finished with value: 1438.8863113065604 and parameters: {'iterations': 1351, 'depth
[I 2025-12-23 06:04:45,095] Trial 2 finished with value: 1736.3014969781277 and parameters: {'iterations': 952, 'depth
[I 2025-12-23 06:05:46,732] Trial 3 finished with value: 1461.8068149037952 and parameters: {'iterations': 1326, 'depth
[I 2025-12-23 06:07:10,486] Trial 4 finished with value: 1431.297098397948 and parameters: {'iterations': 1488, 'depth
[I 2025-12-23 06:07:33,884] Trial 5 finished with value: 1654.1379766137409 and parameters: {'iterations': 731, 'depth
[I 2025-12-23 06:08:38,148] Trial 6 finished with value: 1555.5250708320216 and parameters: {'iterations': 1471, 'depth
[I 2025-12-23 06:09:41,913] Trial 7 finished with value: 1444.1781874231856 and parameters: {'iterations': 1344, 'depth
[I 2025-12-23 06:10:38,482] Trial 8 finished with value: 1493.5742786832807 and parameters: {'iterations': 1260, 'depth
[I 2025-12-23 06:11:07,950] Trial 9 finished with value: 1600.714381088093 and parameters: {'iterations': 701, 'depth'
[I 2025-12-23 06:13:00,463] Trial 10 finished with value: 1426.398298152719 and parameters: {'iterations': 1922, 'depth
[I 2025-12-23 06:14:55,869] Trial 11 finished with value: 1421.770414526068 and parameters: {'iterations': 1886, 'depth
[I 2025-12-23 06:16:06,619] Trial 12 finished with value: 1432.6932023138838 and parameters: {'iterations': 1952, 'dept
[I 2025-12-23 06:18:23,397] Trial 13 finished with value: 1421.1244506938747 and parameters: {'iterations': 1970, 'dept
[I 2025-12-23 06:20:04,809] Trial 14 finished with value: 1432.640323757899 and parameters: {'iterations': 1730, 'depth
[I 2025-12-23 06:21:11,394] Trial 15 finished with value: 1430.5873332174751 and parameters: {'iterations': 1654, 'dept
[I 2025-12-23 06:23:02,179] Trial 16 finished with value: 1423.8910275905198 and parameters: {'iterations': 1750, 'dept
[I 2025-12-23 06:24:34,145] Trial 17 finished with value: 1425.5201200387562 and parameters: {'iterations': 1057, 'dept
[I 2025-12-23 06:26:06,144] Trial 18 finished with value: 1420.9085672816705 and parameters: {'iterations': 1873, 'dept
[I 2025-12-23 06:27:22,336] Trial 19 finished with value: 1842.4180181165548 and parameters: {'iterations': 1588, 'dept

✅ Best RMSE: 1420.9085672816705
✅ Best Params: {'iterations': 1873, 'depth': 8, 'learning_rate': 0.22399817514458342, 'l2_leaf_reg': 7.09797159063142

--- 4. Huấn luyện Mô hình Tốt nhất ---
```

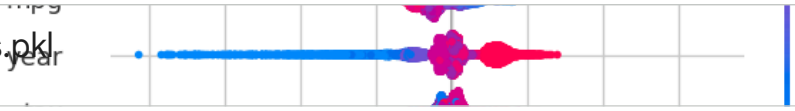
```
0:  learn: 5970.5461254 test: 5983.2348184  best: 5983.2348184 (0)  total: 59.5ms   remaining: 1m 51s
100:    learn: 1728.2015124 test: 1750.1679149  best: 1750.1679149 (100)total: 6.37s    remaining: 1m 51s
200:    learn: 1577.5211502 test: 1617.3962282  best: 1617.3962282 (200)total: 11.1s    remaining: 1m 32s
300:    learn: 1478.5069147 test: 1547.0748640  best: 1547.0748640 (300)total: 15.7s    remaining: 1m 21s
400:    learn: 1418.2939100 test: 1509.5159354  best: 1509.5159354 (400)total: 22.5s    remaining: 1m 22s
500:    learn: 1374.0440551 test: 1486.7960976  best: 1486.7960976 (500)total: 27.2s    remaining: 1m 14s
600:    learn: 1339.8853727 test: 1472.2525130  best: 1472.2525130 (600)total: 33.5s    remaining: 1m 10s
700:    learn: 1310.3854591 test: 1459.6609207  best: 1459.6609207 (700)total: 38.8s    remaining: 1m 4s
800:    learn: 1287.9835188 test: 1453.8612182  best: 1453.8612182 (800)total: 43.6s    remaining: 58.3s
900:    learn: 1268.6175826 test: 1447.3294386  best: 1447.3123795 (890)total: 50.4s    remaining: 54.3s
1000:   learn: 1250.8786225 test: 1442.1877885  best: 1442.1216650 (995)total: 55.4s    remaining: 48.2s
1100:   learn: 1235.8493670 test: 1439.4038809  best: 1439.1769219 (1090)   total: 1m 1s    remaining: 43.2s
1200:   learn: 1220.6923648 test: 1435.6269744  best: 1435.6269744 (1200)   total: 1m 7s    remaining: 37.6s
1300:   learn: 1208.6315605 test: 1433.4730624  best: 1433.4710567 (1299)   total: 1m 12s   remaining: 31.7s
1400:   learn: 1195.8683741 test: 1431.1691819  best: 1431.1691819 (1400)   total: 1m 19s   remaining: 26.7s
1500:   learn: 1184.8532740 test: 1430.3889210  best: 1430.3871926 (1495)   total: 1m 24s   remaining: 20.9s
1600:   learn: 1173.8149397 test: 1429.7093831  best: 1429.7025521 (1599)   total: 1m 31s   remaining: 15.5s
Stopped by overfitting detector  (100 iterations wait)

bestTest = 1429.702552
bestIteration = 1599

Shrink model to first 1600 iterations.

--- 5. Đánh giá & Explainable AI ---
--- KẾT QUẢ CUỐI CÙNG (TEST SET) ---
MAE:  979.60
RMSE: 1,383.33
R2:   0.9631
```




```
Đang tính toán SHAP values (Có thể mất chút thời gian)...
```



Tạo file Encoders.pkl




```
import pandas as pd
import joblib
from sklearn.preprocessing import LabelEncoder
import os

# 1. Tải lại dữ liệu gốc (Hoặc file processed nếu nó còn chứa cột string)
# Giả sử Kiên có file CarsData.csv (gốc) hoặc CarsData_Processed.csv
# Tốt nhất là dùng file GỐC để đảm bảo đầy đủ các nhãn (labels).
# Kiên hãy copy file CarsData.csv vào thư mục backend này nhé.

csv_file = 'CarsData.csv' # Hoặc 'CarsData_Processed.csv' nếu file này chưa bị mã hóa hết

if not os.path.exists(csv_file):
    print(f"❌ Lỗi: Không tìm thấy file '{csv_file}' trong thư mục hiện tại.")
    print("👉 Hãy copy file dữ liệu vào đây để script học lại cách mã hóa.")
    exit()

print(f"Đang đọc dữ liệu từ {csv_file}...")
df = pd.read_csv(csv_file)

# 2. Danh sách các cột phân loại cần mã hóa
# Đây là danh sách các cột mà ứng dụng Web cho người dùng chọn (Dropdown)
categorical_cols = ['Manufacturer', 'model', 'transmission', 'fuelType']

encoders = {}

print("Đang tạo lại Encoders...")

for col in categorical_cols:
    if col in df.columns:
        # Chuyển sang string để đảm bảo đồng nhất
        df[col] = df[col].astype(str)

        # Khởi tạo và fit LabelEncoder
        le = LabelEncoder()
        le.fit(df[col])

        # Lưu vào dictionary
        encoders[col] = le
        print(f"✅ Đã tạo encoder cho cột: {col} ({len(le.classes_)} nhãn)")
    else:
        print(f"⚠️ Cảnh báo: Không tìm thấy cột '{col}' trong file CSV. Kiểm tra lại tên cột.")

# 3. Lưu file encoders.pkl
if encoders:
    joblib.dump(encoders, 'encoders.pkl')
    print("\n🎉 Thành công! Đã lưu file 'encoders.pkl'.")
    print("👉 Bây giờ Kiên có thể chạy 'python main.py' được rồi.")
else:
    print("\n❌ Thất bại: Không tạo được encoder nào.")
```

```
Đang đọc dữ liệu từ CarsData.csv...
Đang tạo lại Encoders...
✅ Đã tạo encoder cho cột: Manufacturer (9 nhãn)
✅ Đã tạo encoder cho cột: model (196 nhãn)
✅ Đã tạo encoder cho cột: transmission (4 nhãn)
✅ Đã tạo encoder cho cột: fuelType (5 nhãn)

🎉 Thành công! Đã lưu file 'encoders.pkl'.
👉 Bây giờ Kiên có thể chạy 'python main.py' được rồi.
```

CHƯƠNG 5 (Mô hình tiên tiến 5 năm gần đây): TabNET (2021)

Sinh viên: Nguyễn Trung Kiên - 20227180

Mục tiêu:

1. Xây dựng mô hình TabNET.
2. Kiểm tra liệu nó có tốt hơn CatBoost hiện tại không?

```
# 1. Cài đặt thư viện
!pip install pytorch-tabnet torch

import pandas as pd
import numpy as np
import torch
from pytorch_tabnet.tab_model import TabNetRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import time

# 2. Chuẩn bị dữ liệu
try:
    df = pd.read_csv('CarsData_Processed.csv')
    print("✅ Đã tải dữ liệu thành công!")
except FileNotFoundError:
    print("❌ Lỗi: Không tìm thấy file csv.")

# --- BƯỚC SỬA LỖI QUAN TRỌNG ---
# Chỉ chọn các cột dữ liệu là số (loại bỏ các cột string như 'model', 'manufacturer')
df_numeric = df.select_dtypes(include=[np.number])

# Kiểm tra xem có cột nào bị NaN không (TabNet không thích NaN)
df_numeric = df_numeric.fillna(0)

print(f"Số lượng đặc trưng số được sử dụng: {df_numeric.shape[1] - 1}") # Trừ cột price
# -------------------------------

X = df_numeric.drop(columns=['price']).values
y = df_numeric['price'].values.reshape(-1, 1)

# Chia tập dữ liệu
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# 3. Cấu hình TabNet (Mô hình 2021)
tabnet = TabNetRegressor(
    n_d=16, n_a=16, n_steps=4,
    gamma=1.3, n_independent=2, n_shared=2,
    optimizer_fn=torch.optim.Adam,
    optimizer_params=dict(lr=2e-2),
    scheduler_params={"step_size":50, "gamma":0.9},
    scheduler_fn=torch.optim.lr_scheduler.StepLR,
    mask_type='entmax',
    verbose=1
)

# 4. Huấn luyện
print("--- Đang huấn luyện TabNet (Deep Learning 2021) ---")
start_time = time.time()

# Lưu ý: max_epochs có thể giảm xuống nếu muốn chạy nhanh để demo
```